apispec

Release 6.6.1

unknown

Jun 24, 2024

CONTENTS

Release v6.6.1 (*Changelog*)

A pluggable API specification generator. Currently supports the OpenAPI Specification (f.k.a. the Swagger specification).

CHAPTER

ONE

FEATURES

- Supports the OpenAPI Specification (versions 2 and 3)
- Framework-agnostic
- Built-in support for marshmallow
- Utilities for parsing docstrings

CHAPTER

TWO

EXAMPLE APPLICATION

```
import uuid

from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin
from apispec_webframeworks.flask import FlaskPlugin
from flask import Flask
from marshmallow import Schema, fields


# Create an APISpec
spec = APISpec(
    title="Swagger Petstore",
    version="1.0.0",
    openapi_version="3.0.2",
    plugins=[FlaskPlugin(), MarshmallowPlugin()],
)


# Optional marshmallow support
class CategorySchema(Schema):
    id = fields.Int()
    name = fields.Str(required=True)


class PetSchema(Schema):
    categories = fields.List(fields.Nested(CategorySchema))
    name = fields.Str()


# Optional security scheme support
api_key_scheme = {"type": "apiKey", "in": "header", "name": "X-API-Key"}
spec.components.security_scheme("ApiKeyAuth", api_key_scheme)


# Optional Flask support
app = Flask(__name__)


@app.route("/random")
def random_pet():
```

(continues on next page)

(continued from previous page)

```
    """A cute furry animal endpoint.
    ---
    get:
      description: Get a random pet
      security:
        - ApiKeyAuth: []
      responses:
        200:
          description: Return a pet
          content:
            application/json:
              schema: PetSchema
    """
    # Hardcoded example data
    pet_data = {
        "name": "sample_pet_" + str(uuid.uuid1()),
        "categories": [{"id": 1, "name": "sample_category"}],
    }
    return PetSchema().dump(pet_data)


# Register the path and the entities within it
with app.test_request_context():
    spec.path(view=random_pet)
```

2.1 Generated OpenAPI Spec

```
import json

print(json.dumps(spec.to_dict(), indent=2))
# {
#   "info": {
#     "title": "Swagger Petstore",
#     "version": "1.0.0"
#   },
#   "openapi": "3.0.2",
#   "components": {
#     "schemas": {
#       "Category": {
#         "type": "object",
#         "properties": {
#           "id": {
#             "type": "integer",
#             "format": "int32"
#           },
#           "name": {
#             "type": "string"
#           }
#         },
#         "required": [
```

(continues on next page)

(continued from previous page)

```
#           "name"
#         ]
#       },
#       "Pet": {
#         "type": "object",
#         "properties": {
#           "categories": {
#             "type": "array",
#             "items": {
#               "$ref": "#/components/schemas/Category"
#             }
#           },
#           "name": {
#             "type": "string"
#           }
#         }
#       },
#     }
#   },
#   "securitySchemes": {
#      "ApiKeyAuth": {
#        "type": "apiKey",
#        "in": "header",
#        "name": "X-API-Key"
#     }
#   },
#   "paths": {
#     "/random": {
#       "get": {
#         "description": "Get a random pet",
#         "security": [
#           {
#             "ApiKeyAuth": []
#           }
#         ],
#         "responses": {
#           "200": {
#             "description": "Return a pet",
#             "content": {
#               "application/json": {
#                 "schema": {
#                   "$ref": "#/components/schemas/Pet"
#                 }
#               }
#             }
#           }
#         }
#       }
#     }
#   },
# }
```

(continues on next page)

(continued from previous page)

```
print(spec.to_yaml())
# info:
#   title: Swagger Petstore
#   version: 1.0.0
# openapi: 3.0.2
# components:
#   schemas:
#     Category:
#       properties:
#         id:
#           format: int32
#           type: integer
#         name:
#           type: string
#       required:
#       - name
#       type: object
#     Pet:
#       properties:
#         categories:
#           items:
#             $ref: '#/components/schemas/Category'
#           type: array
#         name:
#           type: string
#       type: object
#   securitySchemes:
#     ApiKeyAuth:
#       in: header
#       name: X-API-KEY
#       type: apiKey
# paths:
#   /random:
#     get:
#       description: Get a random pet
#       responses:
#         '200':
#           content:
#             application/json:
#               schema:
#                 $ref: '#/components/schemas/Pet'
#           description: Return a pet
#       security:
#       - ApiKeyAuth: []
```

CHAPTER

THREE

USER GUIDE

3.1 Install

3.1.1 From the PyPI

To install the latest version from the PyPI:

```
pip install -U apispec
```

To install with validation support:

```
pip install -U 'apispec[validation]'
```

To install with YAML support:

```
pip install -U 'apispec[yaml]'
```

3.1.2 Get the Bleeding Edge Version

To install the latest development version:

```
pip install -U git+https://github.com/marshmallow-code/apispec@dev
```

3.2 Quickstart

3.2.1 Basic Usage

First, create an *APISpec* object, passing basic information about your API.

```
from apispec import APISpec

spec = APISpec(
    title="Gisty",
    version="1.0.0",
    openapi_version="3.0.2",
    info=dict(description="A minimal gist API"),
)
```

Add schemas to your spec using *spec.components.schema*.

```
spec.components.schema(
    "Gist",
    {
        "properties": {
            "id": {"type": "integer", "format": "int64"},
            "name": {"type": "string"},
        }
    },
)
```

Add paths to your spec using *path*.

```
spec.path(
    path="/gist/{gist_id}",
    operations=dict(
        get=dict(
            responses={"200": {"content": {"application/json": {"schema": "Gist"}}}}
        )
    ),
)
```

The API is chainable, allowing you to combine multiple method calls in one statement:

```
spec.path(...).path(...).tag(...)

spec.components.schema(...).parameter(...)
```

To output your OpenAPI spec, invoke the `to_dict` method.

```
from pprint import pprint

pprint(spec.to_dict())
# {'components': {'parameters': {},
#                 'responses': {},
#                 'schemas': {'Gist': {'properties': {'id': {'format': 'int64',
#                                                            'type': 'integer'},
#                                                     'name': {'type': 'string'}}}}},
#  'info': {'description': 'A minimal gist API',
#           'title': 'Gisty',
#           'version': '1.0.0'},
#  'openapi': '3.0.2',
#  'paths': {'/gist/{gist_id}':
#               {'get': {'responses': {'200': {'content': {'application/json': {'schema': {'$ref
↪': '#/definitions/Gist'}}}}}}}},
#  'tags': []}
```

Use *to_yaml* to export your spec to YAML.

```
print(spec.to_yaml())
# components:
#   parameters: {}
#   responses: {}
```

(continues on next page)

(continued from previous page)

```
#   schemas:
#     Gist:
#       properties:
#         id: {format: int64, type: integer}
#         name: {type: string}
# info: {description: A minimal gist API, title: Gisty, version: 1.0.0}
# openapi: 3.0.2
# paths:
#   /gist/{gist_id}:
#     get:
#       responses:
#         '200':
#           content:
#             application/json:
#               schema: {$ref: '#/definitions/Gist'}
# tags: []
```

See also:

For a full reference of the *APISpec* class, see the *Core API Reference*.

3.2.2 Next Steps

We've learned how to programmatically construct an OpenAPI spec, but defining our entities was verbose.

In the next section, we'll learn how to let plugins do the dirty work: *Using Plugins*.

3.3 Using Plugins

3.3.1 What is an apispec "plugin"?

An apispec *plugin* is an object that provides helper methods for generating OpenAPI entities from objects in your application.

A plugin may modify the behavior of *APISpec* methods so that they can take your application's objects as input.

3.3.2 Enabling Plugins

To enable a plugin, pass an instance to the constructor of *APISpec*.

```
from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin

spec = APISpec(
    title="Gisty",
    version="1.0.0",
    openapi_version="3.0.2",
    info=dict(description="A minimal gist API"),
    plugins=[MarshmallowPlugin()],
)
```

3.3.3 Example: Flask and Marshmallow Plugins

The bundled marshmallow plugin (*apispec.ext.marshmallow.MarshmallowPlugin*) provides helpers for generating OpenAPI schema and parameter objects from marshmallow schemas and fields.

The apispec-webframeworks package includes a Flask plugin with helpers for generating path objects from view functions.

Let's recreate the spec from the *Quickstart guide* using these two plugins.

First, ensure that `apispec-webframeworks` is installed:

```
$ pip install apispec-webframeworks
```

Also, ensure that a compatible `marshmallow` version is used:

```
$ pip install -U apispec[marshmallow]
```

We can now use the marshmallow and Flask plugins.

```
from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin
from apispec_webframeworks.flask import FlaskPlugin

spec = APISpec(
    title="Gisty",
    version="1.0.0",
    openapi_version="3.0.2",
    info=dict(description="A minimal gist API"),
    plugins=[FlaskPlugin(), MarshmallowPlugin()],
)
```

Our application will have a marshmallow `Schema` for gists.

```
from marshmallow import Schema, fields


class GistParameter(Schema):
    gist_id = fields.Int()


class GistSchema(Schema):
    id = fields.Int()
    content = fields.Str()
```

The marshmallow plugin allows us to pass this `Schema` to *spec.components.schema*.

```
spec.components.schema("Gist", schema=GistSchema)
```

The schema is now added to the spec.

```
from pprint import pprint

pprint(spec.to_dict())
# {'components': {'parameters': {}, 'responses': {}, 'schemas': {}},
#  'info': {'description': 'A minimal gist API',
```

(continues on next page)

(continued from previous page)

```
#           'title': 'Gisty',
#           'version': '1.0.0'},
#  'openapi': '3.0.2',
#  'paths': {},
#  'tags': []}
```

Our application will have a Flask route for the gist detail endpoint.

We'll add some YAML in the docstring to add response information.

```
from flask import Flask

app = Flask(__name__)


# NOTE: Plugins may inspect docstrings to gather more information for the spec
@app.route("/gists/<gist_id>")
def gist_detail(gist_id):
    """Gist detail view.
    ---
    get:
      parameters:
      - in: path
        schema: GistParameter
      responses:
        200:
          content:
            application/json:
              schema: GistSchema
    """
    return "details about gist {}".format(gist_id)
```

The Flask plugin allows us to pass this view to *spec.path*.

```
# Since path inspects the view and its route,
# we need to be in a Flask request context
with app.test_request_context():
    spec.path(view=gist_detail)
```

Our OpenAPI spec now looks like this:

```
pprint(spec.to_dict())
# {'components': {'parameters': {},
#                 'responses': {},
#                 'schemas': {'Gist': {'properties': {'content': {'type': 'string'},
#                                                     'id': {'format': 'int32',
#                                                            'type': 'integer'}},
#                                      'type': 'object'}}},
#  'info': {'description': 'A minimal gist API',
#           'title': 'Gisty',
#           'version': '1.0.0'},
#  'openapi': '3.0.2',
#  'paths': {'/gists/{gist_id}': {'get': {'parameters': [{'in': 'path',
```

(continues on next page)

(continued from previous page)

```
#                                                       'name': 'gist_id',
#                                                       'required': True,
#                                                       'schema': {'format': 'int32',
#                                                                  'type': 'integer'}}],
#                                          'responses': {200: {'content': {'application/json
↪': {'schema': {'$ref': '#/components/schemas/Gist'}}}}}}},
#  'tags': []}
```

If your API uses method-based dispatching, the process is similar. Note that the method no longer needs to be included in the docstring.

```
from flask.views import MethodView


class GistApi(MethodView):
    def get(self):
        """Gist view
        ---
        description: Get a gist
        responses:
          200:
            content:
              application/json:
                schema: GistSchema
        """
        pass

    def post(self):
        pass


method_view = GistApi.as_view("gist")
app.add_url_rule("/gist", view_func=method_view)
with app.test_request_context():
    spec.path(view=method_view)
pprint(dict(spec.to_dict()["paths"]["/gist"]))
# {'get': {'description': 'get a gist',
#          'responses': {200: {'content': {'application/json': {'schema': {'$ref': '#/
↪components/schemas/Gist'}}}}}},
#  'post': {}}
```

3.3.4 Marshmallow Plugin

Nested Schemas

By default, Marshmallow `Nested` fields are represented by a JSON Reference object. If the schema has been added to the spec via *`spec.components.schema`*, the user-supplied name will be used in the reference. Otherwise apispec will add the nested schema to the spec using an automatically resolved name for the nested schema. The default *`resolver`* function will resolve a name based on the schema's class `__name__`, dropping a trailing "Schema" so that `class PetSchema(Schema)` resolves to "Pet".

To change the behavior of the name resolution simply pass a function accepting a `Schema` class, `Schema` instance or

a string that resolves to a `Schema` class and returning a string to the plugin's constructor. To easily work with these argument types the marshmallow plugin provides *`resolve_schema_cls`* and *`resolve_schema_instance`* functions. If the `schema_name_resolver` function returns a value that evaluates to `False` in a boolean context the nested schema will not be added to the spec and instead defined in-line.

Note: A `schema_name_resolver` function must return a string name when working with circular-referencing schemas in order to avoid infinite recursion.

Schema Modifiers

apispec will respect schema modifiers such as `exclude` and `partial` in the generated schema definition. If a schema is initialized with modifiers, apispec will treat each combination of modifiers as a unique schema definition.

Custom DateTime formats

apispec supports all four basic formats of `marshmallow.fields.DateTime`: `"rfc"` (for RFC822), `"iso"` (for ISO8601), `"timestamp"`, `"timestamp_ms"` (for a POSIX timestamp).

If you are using a custom DateTime format you should pass a regex string to the `pattern` parameter in your field `metadata` so that it is included as documentation.

```
class SchemaWithCustomDate(Schema):
    french_date = ma.DateTime(
        format="%d-%m%Y %H:%M:%S",
        metadata={"pattern": r"^\d{2}-\d{2}-\d{4} \d{2}:\d{2}:\d{2}$"},
    )
```

Custom Fields

apispec maps standard marshmallow fields to OpenAPI types and formats. If your custom field subclasses a standard marshmallow `Field` class then it will inherit the default mapping. If you want to override the OpenAPI type and format for custom fields, use the *`map_to_openapi_type`* method. It can be invoked with either a pair of strings providing the OpenAPI type and format, or a marshmallow `Field` that has the desired target mapping.

```
from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin
from marshmallow.fields import Integer, Field

ma_plugin = MarshmallowPlugin()

spec = APISpec(
    title="Demo", version="0.1", openapi_version="3.0.0", plugins=(ma_plugin,)
)


# Inherits Integer mapping of ('integer', None)
class CustomInteger(Integer):
    pass
```

(continues on next page)

(continued from previous page)

```
# Override Integer mapping
class Int32(Integer):
    pass


ma_plugin.map_to_openapi_type(Int32, "string", "int32")


# Map to ('integer', None) like Integer
class IntegerLike(Field):
    pass


ma_plugin.map_to_openapi_type(IntegerLike, Integer)
```

In situations where greater control of the properties generated for a custom field is desired, users may add custom logic to the conversion of fields to OpenAPI properties through the use of the *add_attribute_function* method. Continuing from the example above:

```
def my_custom_field2properties(self, field, **kwargs):
    """Add an OpenAPI extension flag to MyCustomField instances"""
    ret = {}
    if isinstance(field, MyCustomField):
        if self.openapi_version.major > 2:
            ret["x-customString"] = True
    return ret


ma_plugin.converter.add_attribute_function(my_custom_field2properties)
```

The function passed to `add_attribute_function` will be bound to the converter. It must accept the converter instance as first positional argument.

In some rare cases, typically with container fields such as fields derived from `List`, documenting the parameters using this field require some more customization. This can be achieved using the *add_parameter_attribute_function* method.

For instance, when documenting webargs's `DelimitedList` field, one may register this function:

```
def delimited_list2param(self, field, **kwargs):
    ret: dict = {}
    if isinstance(field, DelimitedList):
        if self.openapi_version.major < 3:
            ret["collectionFormat"] = "csv"
        else:
            ret["explode"] = False
            ret["style"] = "form"
    return ret


ma_plugin.converter.add_parameter_attribute_function(delimited_list2param)
```

Enum Fields

When using `marshmallow.fields.Enum` fields to (de)serialize `enum.Enum` values, we recommend passing a marshmallow field to `by_value`. This ensures the correct `type` property is included in the generated OAI spec.

```python
from enum import Enum
from apispec import APISpec

from apispec.ext.marshmallow import MarshmallowPlugin
from marshmallow import Schema, fields

spec = APISpec(
    title="Gisty",
    version="1.0.0",
    openapi_version="3.0.2",
    info=dict(description="A minimal gist API"),
    plugins=[MarshmallowPlugin()],
)


class GistVisibility(Enum):
    PRIVATE = "private"
    PUBLIC = "public"


class GistSchema(Schema):
    id = fields.Int()
    visibility = fields.Enum(GistVisibility, by_value=fields.String())


spec.components.schema("Gist", schema=GistSchema)
print(spec.to_yaml())
# info:
#   description: A minimal gist API
#   title: Gisty
#   version: 1.0.0
# paths: {}
# openapi: 3.0.2
# components:
#   schemas:
#     Gist:
#       type: object
#       properties:
#         id:
#           type: integer
#         visibility:
#           type: string
#           enum:
#           - private
#           - public
```

3.3.5 Next Steps

You now know how to use plugins. The next section will show you how to write plugins: *Writing Plugins*.

3.4 Writing Plugins

A plugins is a subclass of *apispec.plugin.BasePlugin*.

3.4.1 Helper Methods

Plugins provide "helper" methods that augment the behavior of *apispec.APISpec* methods.

There are five types of helper methods:

- Schema helpers
- Parameter helpers
- Response helpers
- Path helpers
- Operation helpers

Helper functions modify *apispec.APISpec* methods. For example, path helpers modify *apispec.APISpec.path*.

A plugin with a path helper function may look something like this:

```
from apispec import BasePlugin
from apispec.yaml_utils import load_operations_from_docstring


class MyPlugin(BasePlugin):
    def path_helper(self, path, operations, func, **kwargs):
        """Path helper that parses docstrings for operations. Adds a
        ``func`` parameter to `apispec.APISpec.path`.
        """
        operations.update(load_operations_from_docstring(func.__doc__))
```

All plugin helpers must accept extra **kwargs, allowing custom plugins to define new arguments if required.

A plugin with an operation helper that adds `deprecated` flag may look like this

```
# deprecated_plugin.py

from apispec import BasePlugin
from apispec.yaml_utils import load_operations_from_docstring


class DeprecatedPlugin(BasePlugin):
    def operation_helper(self, path, operations, **kwargs):
        """Operation helper that add `deprecated` flag if in `kwargs`"""
        if kwargs.pop("deprecated", False) is True:
            for key, value in operations.items():
                value["deprecated"] = True
```

Using this plugin

```
import json
from apispec import APISpec
from deprecated_plugin import DeprecatedPlugin

spec = APISpec(
    title="Gisty",
    version="1.0.0",
    openapi_version="3.0.2",
    plugins=[DeprecatedPlugin()],
)

# path will call operation_helper on operations
spec.path(
    path="/gists/{gist_id}",
    operations={"get": {"responses": {"200": {"description": "standard response"}}}},
    deprecated=True,
)
print(json.dumps(spec.to_dict()["paths"]))
# {"/gists/{gist_id}": {"get": {"responses": {"200": {"description": "standard response"}
↪}, "deprecated": true}}}
```

3.4.2 The `init_spec` Method

`BasePlugin` has an `init_spec` method that `APISpec` calls on each plugin at initialization with the spec object itself as parameter. It is no-op by default, but a plugin may override it to access and store useful information on the spec object.

A typical use case is conditional code depending on the OpenAPI version, which is stored as `openapi_version` on the `spec` object. See source code for apispec.ext.marshmallow.MarshmallowPlugin for an example.

3.4.3 Example: Docstring-parsing Plugin

Here's a plugin example involving conditional processing depending on the OpenAPI version:

```
# docplugin.py

from apispec import BasePlugin
from apispec.yaml_utils import load_operations_from_docstring


class DocPlugin(BasePlugin):
    def init_spec(self, spec):
        super(DocPlugin, self).init_spec(spec)
        self.openapi_major_version = spec.openapi_version.major

    def operation_helper(self, operations, func, **kwargs):
        """Operation helper that parses docstrings for operations. Adds a
        ``func`` parameter to `apispec.APISpec.path`.
        """
        doc_operations = load_operations_from_docstring(func.__doc__)
```

(continues on next page)

(continued from previous page)

```
        # Apply conditional processing
        if self.openapi_major_version < 3:
            "...Mutating doc_operations for OpenAPI v2..."
        else:
            "...Mutating doc_operations for OpenAPI v3+..."
        operations.update(doc_operations)
```

To use the plugin:

```
from apispec import APISpec
from docplugin import DocPlugin

spec = APISpec(
    title="Gisty", version="1.0.0", openapi_version="3.0.2", plugins=[DocPlugin()]
)


def gist_detail(gist_id):
    """Gist detail view.
    ---
    get:
      responses:
        200:
          content:
            application/json:
              schema: '#/definitions/Gist'
    """
    pass


spec.path(path="/gists/{gist_id}", func=gist_detail)
print(dict(spec.to_dict()["paths"]))
# {'/gists/{gist_id}': {'get': {'responses': {200: {'content': {'application/json': {'schema': '#/
↪definitions/Gist'}}}}}}}
```

3.4.4 Next Steps

To learn more about how to write plugins:

- Consult the *Core API docs* for `BasePlugin`
- View the source for an existing apispec plugin, e.g. FlaskPlugin.
- Check out some projects using apispec: https://github.com/marshmallow-code/apispec/wiki/Ecosystem

3.5 Special Topics

Solutions to specific problems are documented here.

3.5.1 Adding Additional Fields To Schema Objects

To add additional fields (e.g. `"discriminator"`) to Schema objects generated from *spec.components.schema* , pass them to the `component` parameter. If your'e using `MarshmallowPlugin`, the `component` properties will get merged with the autogenerated properties.

```
properties = {
    "id": {"type": "integer", "format": "int64"},
    "name": {"type": "string", "example": "doggie"},
}

spec.components.schema("Pet", component={"discriminator": "petType"}, schema=PetSchema)
```

Note: Be careful about the input that you pass to `component`. `apispec` will not guarantee that the passed fields are valid against the OpenAPI spec.

3.5.2 Rendering to YAML or JSON

YAML

```
spec.to_yaml()
```

Note: *to_yaml* requires `PyYAML` to be installed. You can install apispec with YAML support using:

```
pip install 'apispec[yaml]'
```

JSON

```
import json

json.dumps(spec.to_dict())
```

3.5.3 Documenting Top-level Components

The `APISpec` object contains helpers to add top-level components:

Component type	Helper method	OpenAPI version
Schema (f.k.a. "definition" in OAS v2)	*spec.components.schema*	2, 3
Parameter	*spec.components.parameter*	2, 3
Response	*spec.components.response*	2, 3
Header	*spec.components.response*	3
Example	*spec.components.response*	3
Security scheme	*spec.components.response*	2, 3

Most component registration methods provide a `lazy` keyword argument, allowing to define a component but only publish it in the generated documentation if it is actually referenced.

To add other top-level objects, pass them to the `APISpec` as keyword arguments.

Here is an example that includes a Server Object.

```
import yaml
from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin
from apispec.utils import validate_spec

OPENAPI_SPEC = """
openapi: 3.0.2
info:
  description: Server API document
  title: Server API
  version: 1.0.0
servers:
- url: http://localhost:{port}/
  description: The development API server
  variables:
    port:
      enum:
      - '3000'
      - '8888'
      default: '3000'
"""

settings = yaml.safe_load(OPENAPI_SPEC)
# retrieve  title, version, and openapi version
title = settings["info"].pop("title")
spec_version = settings["info"].pop("version")
openapi_version = settings.pop("openapi")

spec = APISpec(
    title=title,
    version=spec_version,
    openapi_version=openapi_version,
    plugins=(MarshmallowPlugin(),),
    **settings
```

(continues on next page)

(continued from previous page)

```
)

validate_spec(spec)
```

3.5.4 Documenting Security Schemes

Use *spec.components.security_scheme* to document Security Scheme Objects.

```
from pprint import pprint
from apispec import APISpec

spec = APISpec(title="Swagger Petstore", version="1.0.0", openapi_version="3.0.2")

api_key_scheme = {"type": "apiKey", "in": "header", "name": "X-API-Key"}
jwt_scheme = {"type": "http", "scheme": "bearer", "bearerFormat": "JWT"}

spec.components.security_scheme("api_key", api_key_scheme)
spec.components.security_scheme("jwt", jwt_scheme)

pprint(spec.to_dict()["components"]["securitySchemes"], indent=2)
# { 'api_key': {'in': 'header', 'name': 'X-API-Key', 'type': 'apiKey'},
#   'jwt': {'bearerFormat': 'JWT', 'scheme': 'bearer', 'type': 'http'}}
```

3.5.5 Referencing Top-level Components

On OpenAPI, top-level component are meant to be referenced using a `$ref`, as in `{$ref: '#/components/schemas/Pet'}` (OpenAPI v3) or `{$ref: '#/definitions/Pet'}` (OpenAPI v2).

APISpec automatically resolves references in paths and in components themselves when a string is provided while a dict is expected. Passing a fully-resolved reference is not supported. In other words, rather than passing `{"schema": {$ref: '#/components/schemas/Pet'}}`, the user must pass `{"schema": "Pet"}`. APISpec assumes a schema reference named `"Pet"` has been defined and builds the reference using the components location corresponding to the OpenAPI version.

CHAPTER

FOUR

API REFERENCE

4.1 Core API

4.1.1 apispec

Contains main apispec classes: *APISpec* and *BasePlugin*

class apispec.**APISpec**(*title: str*, *version: str*, *openapi_version: str*, *plugins: Sequence[*BasePlugin*] = ()*, ***options: Any*)

Stores metadata that describes a RESTful API using the OpenAPI specification.

Parameters

- **title** (*str*) – API title
- **version** (*str*) – API version
- **plugins** (*list* | *tuple*) – Plugin instances. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#infoObject
- **openapi_version** (*str*) – OpenAPI Specification version. Should be in the form '2.x' or '3.x.x' to comply with the OpenAPI standard.
- **options** – Optional top-level keys See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#openapi-object

path(*path: str | None = None*, ***, *operations: dict[str, Any] | None = None*, *summary: str | None = None*, *description: str | None = None*, *parameters: list[dict] | None = None*, ***kwargs: Any*) → *APISpec*

Add a new path object to the spec.

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

Parameters

- **path** (*str* | *None*) – URL path component
- **operations** (*dict* | *None*) – describes the http methods and options for *path*
- **summary** (*str*) – short summary relevant to all operations in this path
- **description** (*str*) – long description relevant to all operations in this path
- **parameters** (*list* | *None*) – list of parameters relevant to all operations in this path
- **kwargs** – parameters used by any path helpers see `register_path_helper()`

tag(*tag: dict*) → *APISpec*

Store information about a tag.

Parameters
tag (*dict*) – the dictionary storing information about the tag.

to_yaml(*yaml_dump_kwargs: Any | None = None*) → str

Render the spec to YAML. Requires PyYAML to be installed.

Parameters
yaml_dump_kwargs (*dict*) – Additional keyword arguments to pass to `yaml.dump`

class apispec.**BasePlugin**

Base class for APISpec plugin classes.

header_helper(*header: dict*, ***kwargs: Any*) → dict | None

May return header component description as a dict.

Parameters

- **header** (*dict*) – Header fields
- **kwargs** – All additional keywords arguments sent to `APISpec.header()`

init_spec(*spec:* APISpec) → None

Initialize plugin with APISpec object

Parameters
spec (APISpec) – APISpec object this plugin instance is attached to

operation_helper(*path: str | None = None*, *operations: dict | None = None*, ***kwargs: Any*) → None

May mutate operations.

Parameters

- **path** (*str*) – Path to the resource
- **operations** (*dict*) – A `dict` mapping HTTP methods to operation object. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#operationObject
- **kwargs** – All additional keywords arguments sent to *APISpec.path()*

parameter_helper(*parameter: dict*, ***kwargs: Any*) → dict | None

May return parameter component description as a dict.

Parameters

- **parameter** (*dict*) – Parameter fields
- **kwargs** – All additional keywords arguments sent to `APISpec.parameter()`

path_helper(*path: str | None = None*, *operations: dict | None = None*, *parameters: list[dict] | None = None*, ***kwargs: Any*) → str | None

May return a path as string and mutate operations dict and parameters list.

Parameters

- **path** (*str*) – Path to the resource
- **operations** (*dict*) – A `dict` mapping HTTP methods to operation object. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#operationObject

- **parameters** (*list*) – A `list` of parameters objects or references for the path. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#parameterObject and https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#referenceObject
- **kwargs** – All additional keywords arguments sent to *APISpec.path()*

Return value should be a string or None. If a string is returned, it is set as the path.

The last path helper returning a string sets the path value. Therefore, the order of plugin registration matters. However, generally, registering several plugins that return a path does not make sense.

response_helper(*response: dict*, ***kwargs: Any*) → dict | None

May return response component description as a dict.

Parameters

- **response** (*dict*) – Response fields
- **kwargs** – All additional keywords arguments sent to `APISpec.response()`

schema_helper(*name: str*, *definition: dict*, ***kwargs: Any*) → dict | None

May return definition as a dict.

Parameters

- **name** (*str*) – Identifier by which schema may be referenced
- **definition** (*dict*) – Schema definition
- **kwargs** – All additional keywords arguments sent to `APISpec.schema()`

4.1.2 apispec.core

Core apispec classes and functions.

class apispec.core.**Components**(*plugins: Sequence[BasePlugin]*, *openapi_version: Version*)

Stores OpenAPI components

Components are top-level fields in OAS v2. They became sub-fields of "components" top-level field in OAS v3.

example(*component_id: str*, *component: dict*, ***, *lazy: bool = False*) → *Components*

Add an example which can be referenced

Parameters

- **component_id** (*str*) – identifier by which example may be referenced
- **component** (*dict*) – example fields
- **lazy** (*bool*) – register component only when referenced in the spec

https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md#exampleObject

get_ref(*obj_type: str*, *obj_or_component_id: dict | str*) → dict

Return object or reference

If obj is a dict, it is assumed to be a complete description and it is returned as is. Otherwise, it is assumed to be a reference name as string and the corresponding $ref string is returned.

Parameters

- **subsection** (*str*) – "schema", "parameter", "response" or "security_scheme"

- **obj** (*dict | str*) – object in dict form or as ref_id string

header(*component_id: str*, *component: dict*, ***, *lazy: bool = False*, ***kwargs: Any*) → *Components*

Add a header which can be referenced.

Parameters

- **component_id** (*str*) – identifier by which header may be referenced
- **component** (*dict*) – header fields
- **lazy** (*bool*) – register component only when referenced in the spec
- **kwargs** – plugin-specific arguments

https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md#headerObject

parameter(*component_id: str*, *location: str*, *component: dict | None = None*, ***, *lazy: bool = False*, ***kwargs: Any*) → *Components*

Add a parameter which can be referenced.

Parameters

- **component_id** (*str*) – identifier by which parameter may be referenced
- **location** (*str*) – location of the parameter
- **component** (*dict*) – parameter fields
- **lazy** (*bool*) – register component only when referenced in the spec
- **kwargs** – plugin-specific arguments

response(*component_id: str*, *component: dict | None = None*, ***, *lazy: bool = False*, ***kwargs: Any*) → *Components*

Add a response which can be referenced.

Parameters

- **component_id** (*str*) – ref_id to use as reference
- **component** (*dict*) – response fields
- **lazy** (*bool*) – register component only when referenced in the spec
- **kwargs** – plugin-specific arguments

schema(*component_id: str*, *component: dict | None = None*, ***, *lazy: bool = False*, ***kwargs: Any*) → *Components*

Add a new schema to the spec.

Parameters

- **component_id** (*str*) – identifier by which schema may be referenced
- **component** (*dict*) – schema definition
- **lazy** (*bool*) – register component only when referenced in the spec
- **kwargs** – plugin-specific arguments

Note: If you are using *apispec.ext.marshmallow*, you can pass fields' metadata as additional keyword arguments.

For example, to add `enum` and `description` to your field:

```
status = fields.String(
    required=True,
    metadata={
        "description": "Status (open or closed)",
        "enum": ["open", "closed"],
    },
)
```

---

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

security_scheme(*component_id: str*, *component: dict*) → *Components*

Add a security scheme which can be referenced.

Parameters

- **component_id** (*str*) – component_id to use as reference
- **component** (*dict*) – security scheme fields

4.1.3 apispec.exceptions

Exception classes.

exception apispec.exceptions.**APISpecError**

Base class for all apispec-related errors.

exception apispec.exceptions.**DuplicateComponentNameError**

Raised when registering two components with the same name

exception apispec.exceptions.**DuplicateParameterError**

Raised when registering a parameter already existing in a given scope

exception apispec.exceptions.**InvalidParameterError**

Raised when parameter doesn't contains required keys

exception apispec.exceptions.**OpenAPIError**

Raised when a OpenAPI spec validation fails.

exception apispec.exceptions.**PluginMethodNotImplementedError**

Raised when calling an unimplemented helper method in a plugin

4.1.4 apispec.utils

Various utilities for parsing OpenAPI operations from docstrings and validating against the OpenAPI spec.

apispec.utils.**build_reference**(*component_type: str*, *openapi_major_version: int*, *component_name: str*) → dict[str, str]

Return path to reference

Parameters

- **component_type** (*str*) – Component type (schema, parameter, response, security_scheme)
- **openapi_major_version** (*int*) – OpenAPI major version (2 or 3)
- **component_name** (*str*) – Name of component to reference

apispec.utils.**dedent**(*content: str*) → str

Remove leading indent from a block of text. Used when generating descriptions from docstrings. Note that python's `textwrap.dedent` doesn't quite cut it, as it fails to dedent multiline docstrings that include unindented text on the initial line.

apispec.utils.**deepupdate**(*original: dict*, *update: dict*) → dict

Recursively update a dict.

Subdict's won't be overwritten but also updated.

apispec.utils.**trim_docstring**(*docstring: str*) → str

Uniformly trims leading/trailing whitespace from docstrings.

Based on http://www.python.org/peps/pep-0257.html#handling-docstring-indentation

4.2 Built-in Plugins

4.2.1 apispec.ext.marshmallow

marshmallow plugin for apispec. Allows passing a marshmallow `Schema` to *`spec.components.schema`*, *`spec.components.parameter`*, *`spec.components.response`* (for response and headers schemas) and *`spec.path`* (for responses and response headers).

Requires marshmallow>=3.13.0.

`MarshmallowPlugin` maps marshmallow `Field` classes with OpenAPI types and formats.

It inspects field attributes to automatically document properties such as read/write-only, range and length constraints, etc.

OpenAPI properties can also be passed as metadata to the `Field` instance if they can't be inferred from the field attributes (`description`,...), or to override automatic documentation (`readOnly`,...). A metadata attribute is used in the documentation either if it is a valid OpenAPI property, or if it starts with `"x-"` (vendor extension).

Warning: `MarshmallowPlugin` infers the `default` property from the `load_default` attribute of the `Field` (unless `load_default` is a callable). Since default values are entered in deserialized form, the value displayed in the doc is serialized by the `Field` instance. This may lead to inaccurate documentation in very specific cases. The default value to display in the documentation can be specified explicitly by passing `default` as field metadata.

```
from pprint import pprint
import datetime as dt

from apispec import APISpec
from apispec.ext.marshmallow import MarshmallowPlugin
from marshmallow import Schema, fields

spec = APISpec(
    title="Example App",
    version="1.0.0",
    openapi_version="3.0.2",
    plugins=[MarshmallowPlugin()],
)
```

(continues on next page)

(continued from previous page)

```
class UserSchema(Schema):
    id = fields.Int(dump_only=True)
    name = fields.Str(metadata={"description": "The user's name"})
    created = fields.DateTime(
        dump_only=True,
        dump_default=dt.datetime.utcnow,
        metadata={"default": "The current datetime"}
    )


spec.components.schema("User", schema=UserSchema)
pprint(spec.to_dict()["components"]["schemas"])
# {'User': {'properties': {'created': {'default': 'The current datetime',
#                                      'format': 'date-time',
#                                      'readOnly': True,
#                                      'type': 'string'},
#                          'id': {'readOnly': True,
#                                 'type': 'integer'},
#                          'name': {'description': "The user's name",
#                                   'type': 'string'}},
#           'type': 'object'}}
```

class apispec.ext.marshmallow.**MarshmallowPlugin**(*schema_name_resolver: Callable[[type[Schema]], str] | None = None*)

APISpec plugin for translating marshmallow schemas to OpenAPI/JSONSchema format.

Parameters

schema_name_resolver (*callable*) – Callable to generate the schema definition name. Receives the `Schema` class and returns the name to be used in refs within the generated spec. When working with circular referencing this function must must not return `None` for schemas in a circular reference chain.

Example:

```
from apispec.ext.marshmallow.common import resolve_schema_cls

def schema_name_resolver(schema):
    schema_cls = resolve_schema_cls(schema)
    return schema_cls.__name__
```

Converter

alias of *OpenAPIConverter*

Resolver

alias of *SchemaResolver*

header_helper(*header: dict*, ***kwargs: Any*)

Header component helper that allows using a marshmallow `Schema` in header definition.

Parameters

header (*dict*) – header fields. May contain a marshmallow Schema class or instance.

init_spec(*spec:* APISpec) → None

Initialize plugin with APISpec object

Parameters
spec (APISpec) – APISpec object this plugin instance is attached to

map_to_openapi_type(*field_cls*, **args*)

Set mapping for custom field class.

Parameters
field_cls (*type*) – Field class to set mapping for.

*args can be:

- a pair of the form `(type, format)`
- a core marshmallow field type (in which case we reuse that type's mapping)

Examples:

```
# Override Integer mapping
class Int32(Integer):
    # ...

ma_plugin.map_to_openapi_type(Int32, 'string', 'int32')

# Map to ('integer', None) like Integer
class IntegerLike(Integer):
    # ...

ma_plugin.map_to_openapi_type(IntegerLike, Integer)
```

operation_helper(*path: str | None = None*, *operations: dict | None = None*, ***kwargs: Any*) → None

May mutate operations.

Parameters

- **path** (*str*) – Path to the resource
- **operations** (*dict*) – A `dict` mapping HTTP methods to operation object. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#operationObject
- **kwargs** – All additional keywords arguments sent to `APISpec.path()`

parameter_helper(*parameter*, ***kwargs*)

Parameter component helper that allows using a marshmallow `Schema` in parameter definition.

Parameters
parameter (*dict*) – parameter fields. May contain a marshmallow Schema class or instance.

response_helper(*response*, ***kwargs*)

Response component helper that allows using a marshmallow `Schema` in response definition.

Parameters
parameter (*dict*) – response fields. May contain a marshmallow Schema class or instance.

schema_helper(*name*, *_*, *schema=None*, ***kwargs*)

Definition helper that allows using a marshmallow `Schema` to provide OpenAPI metadata.

Parameters
schema (*type | Schema*) – A marshmallow Schema class or instance.

warn_if_schema_already_in_spec(*schema_key: tuple*) → None

Method to warn the user if the schema has already been added to the spec.

apispec.ext.marshmallow.**resolver**(*schema: type[Schema]*) → str

Default schema name resolver function that strips 'Schema' from the end of the class name.

apispec.ext.marshmallow.schema_resolver

class apispec.ext.marshmallow.schema_resolver.**SchemaResolver**(*openapi_version*, *converter*)

Resolve marshmallow Schemas in OpenAPI components and translate to OpenAPI schema objects, parameter objects or reference objects.

resolve_callback(*callbacks*)

Resolve marshmallow Schemas in a dict mapping callback name to OpenApi **`Callback Object https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#callbackObject`_**.

This is done recursively, so it is possible to define callbacks in your callbacks.

Example:

```
#Input
{
    "userEvent": {
        "https://my.example/user-callback": {
            "post": {
                "requestBody": {
                    "content": {
                        "application/json": {
                            "schema": UserSchema
                        }
                    }
                }
            },
        }
    }
}

#Output
{
    "userEvent": {
        "https://my.example/user-callback": {
            "post": {
                "requestBody": {
                    "content": {
                        "application/json": {
                            "schema": {
                                "$ref": "#/components/schemas/User"
                            }
                        }
                    }
                }
            },
        }
    }
}
```

resolve_operations(*operations*, ***kwargs*)

Resolve marshmallow Schemas in a dict mapping operation to OpenApi `Operation Object https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#operationObject`_

resolve_parameters(*parameters*)

Resolve marshmallow Schemas in a list of OpenAPI Parameter Objects. Each parameter object that contains a Schema will be translated into one or more Parameter Objects.

If the value of a `schema` key is marshmallow Schema class, instance or a string that resolves to a Schema Class each field in the Schema will be expanded as a separate Parameter Object.

Example:

```
#Input
class UserSchema(Schema):
    name = fields.String()
    id = fields.Int()

[
    {"in": "query", "schema": "UserSchema"}
]

#Output
[
    {"in": "query", "name": "id", "required": False, "schema": {"type": "integer
↪"}},
    {"in": "query", "name": "name", "required": False, "schema": {"type":
↪"string"}}
]
```

If the Parameter Object contains a `content` key a single Parameter Object is returned with the Schema translated into a Schema Object or Reference Object.

Example:

```
#Input
[{"in": "query", "name": "pet", "content":{"application/json": {"schema":
↪"PetSchema"}} }]

#Output
[
    {
        "in": "query",
        "name": "pet",
        "content": {
            "application/json": {
                "schema": {"$ref": "#/components/schemas/Pet"}
            }
        }
    }
]
```

Parameters

parameters (*list*) – the list of OpenAPI parameter objects to resolve.

resolve_response(*response*)

Resolve marshmallow Schemas in OpenAPI Response Objects. Schemas may appear in either a Media Type Object or a Header Object.

Example:

```
#Input
{
    "content": {"application/json": {"schema": "PetSchema"}},
    "description": "successful operation",
    "headers": {"PetHeader": {"schema": "PetHeaderSchema"}},
}

#Output
{
    "content": {
        "application/json":{"schema": {"$ref": "#/components/schemas/Pet"}}
    },
    "description": "successful operation",
    "headers": {
        "PetHeader": {"schema": {"$ref": "#/components/schemas/PetHeader"}}
    },
}
```

Parameters

response (*dict*) – the response object to resolve.

resolve_schema(*data*)

Resolve marshmallow Schemas in an OpenAPI component or header - modifies the input dictionary to translate marshmallow Schemas to OpenAPI Schema Objects or Reference Objects.

OpenAPIv3 Components:

```
#Input
{
    "description": "user to add to the system",
    "content": {
        "application/json": {
            "schema": "UserSchema"
        }
    }
}

#Output
{
    "description": "user to add to the system",
    "content": {
        "application/json": {
            "schema": {
                "$ref": "#/components/schemas/User"
            }
        }
    }
}
```

Parameters
data (*dict | str*) – either a parameter or response dictionary that may contain a schema, or a reference provided as string

resolve_schema_dict(*schema*)

Resolve a marshmallow Schema class, object, or a string that resolves to a Schema class or a schema reference or an OpenAPI Schema Object containing one of the above to an OpenAPI Schema Object or Reference Object.

If the input is a marshmallow Schema class, object or a string that resolves to a Schema class the Schema will be translated to an OpenAPI Schema Object or Reference Object.

Example:

```
#Input
"PetSchema"

#Output
{"$ref": "#/components/schemas/Pet"}
```

If the input is a dictionary representation of an OpenAPI Schema Object recursively search for a marshmallow Schemas to resolve. For `"type": "array"`, marshmallow Schemas may appear as the value of the `items` key. For `"type": "object"` Marshmalow Schemas may appear as values in the `properties` dictionary.

Examples:

```
#Input
{"type": "array", "items": "PetSchema"}

#Output
{"type": "array", "items": {"$ref": "#/components/schemas/Pet"}}

#Input
{"type": "object", "properties": {"pet": "PetSchcema", "user": "UserSchema"}}

#Output
{
    "type": "object",
    "properties": {
        "pet": {"$ref": "#/components/schemas/Pet"},
        "user": {"$ref": "#/components/schemas/User"}
    }
}
```

Parameters
schema (*string | Schema | dict*) – the schema to resolve.

apispec.ext.marshmallow.openapi

Utilities for generating OpenAPI Specification (fka Swagger) entities from marshmallow `Schemas` and `Fields`.

Warning: This module is treated as private API. Users should not need to use this module directly.

class apispec.ext.marshmallow.openapi.**OpenAPIConverter**(*openapi_version: Version | str*, *schema_name_resolver*, *spec:* APISpec)

Adds methods for generating OpenAPI specification from marshmallow schemas and fields.

Parameters

- **openapi_version** (*Version | str*) – The OpenAPI version to use. Should be in the form '2.x' or '3.x.x' to comply with the OpenAPI standard.
- **schema_name_resolver** (*callable*) – Callable to generate the schema definition name. Receives the `Schema` class and returns the name to be used in refs within the generated spec. When working with circular referencing this function must must not return `None` for schemas in a circular reference chain.
- **spec** (`APISpec`) – An initalied spec. Nested schemas will be added to the spec

add_parameter_attribute_function(*func*) → None

Method to add a field parameter function to the list of field parameter functions that will be called on a field to convert it to a field parameter.

Parameters

func (*func*) – the field parameter function to add The attribute function will be bound to the *OpenAPIConverter* instance. It will be called for each field in a schema with *self* and a `field` instance positional arguments and `ret` keyword argument. May mutate `ret`. User added field parameter functions will be called after all built-in field parameter functions in the order they were added.

field2required(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI parameter attributes for a required field.

Parameters

field (*Field*) – A marshmallow field.

Return type

dict

fields2jsonschema(*fields*, ***, *partial=None*)

Return the JSON Schema Object given a mapping between field names and `Field` objects.

Parameters

- **fields** (*dict*) – A dictionary of field name field object pairs
- **partial** (*bool | tuple*) – Whether to override a field's required flag. If `True` no fields will be set as required. If an iterable fields in the iterable will not be marked as required.

Return type

dict, a JSON Schema Object

get_ref_dict(*schema*)

Method to create a dictionary containing a JSON reference to the schema in the spec

list2param(*field: Field*, ***kwargs: Any*) → dict

Return a dictionary of parameter properties from `List <marshmallow.fields.List` fields.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

resolve_nested_schema(*schema*)

Return the OpenAPI representation of a marshmallow Schema.

Adds the schema to the spec if it isn't already present.

Typically will return a dictionary with the reference to the schema's path in the spec unless the `schema_name_resolver` returns `None`, in which case the returned dictionary will contain a JSON Schema Object representation of the schema.

Parameters
schema – schema to add to the spec

schema2jsonschema(*schema*)

Return the JSON Schema Object for a given marshmallow `Schema` instance. Schema may optionally provide the `title` and `description` class Meta options.

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

Parameters
schema (*Schema*) – A marshmallow Schema instance

Return type
dict, a JSON Schema Object

schema2parameters(*schema*, ***, *location*, *name: str = 'body'*, *required: bool = False*, *description: str | None = None*)

Return an array of OpenAPI parameters given a given marshmallow `Schema`. If `location` is "body", then return an array of a single parameter; else return an array of a parameter for each included field in the `Schema`.

In OpenAPI 3, only "query", "header", "path" or "cookie" are allowed for the location of parameters. "requestBody" is used when fields are in the body.

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

apispec.ext.marshmallow.field_converter

Utilities for generating OpenAPI Specification (fka Swagger) entities from `Fields`.

Warning: This module is treated as private API. Users should not need to use this module directly.

class apispec.ext.marshmallow.field_converter.**FieldConverterMixin**

Adds methods for converting marshmallow fields to an OpenAPI properties.

add_attribute_function(*func*)

Method to add an attribute function to the list of attribute functions that will be called on a field to convert it from a field to an OpenAPI property.

Parameters
func (*func*) – the attribute function to add The attribute function will be bound to the *OpenAPIConverter* instance. It will be called for each field in a schema with *self* and a `field` instance positional arguments and `ret` keyword argument. Must return a dictionary of OpenAPI properties that will be shallow merged with the return values of all other attribute functions called on the field. User added attribute functions will be called after all built-in attribute functions in the order they were added. The merged results of all previously called attribute functions are accessible via the `ret` argument.

datetime2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `DateTime <marshmallow.fields.DateTime` fields.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

dict2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `Dict` fields.

Only applicable for Marshmallow versions greater than 3. Will provide an `additionalProperties` property based on the field's `value_field` attribute

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

enum2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `Enum <marshmallow.fields.Enum` fields.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2choices(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI field attributes for valid choices definition.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2default(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary containing the field's default value.

Will first look for a `default` key in the field's metadata and then fall back on the field's `missing` parameter. A callable passed to the field's missing parameter will be ignored.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2length(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI field attributes for a set of `Length` validators.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2nullable(*field: Field*, *ret*) → dict

Return the dictionary of OpenAPI field attributes for a nullable field.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2pattern(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI field attributes for a `Regexp` validator.

If there is more than one such validator, only the first is used in the output spec.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2property(*field: Field*) → dict

Return the JSON Schema property definition given a marshmallow `Field`.

Will include field metadata that are valid properties of OpenAPI schema objects (e.g. "description", "enum", "example").

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

Parameters
field (*Field*) – A marshmallow field.

Return type
dict, a Property Object

field2range(*field: Field*, *ret*) → dict

Return the dictionary of OpenAPI field attributes for a set of `Range` validators.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2read_only(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI field attributes for a dump_only field.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2type_and_format(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI type and format based on the field type.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

field2write_only(*field: Field*, ***kwargs: Any*) → dict

Return the dictionary of OpenAPI field attributes for a load_only field.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

list2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `List` fields.

Will provide an `items` property based on the field's `inner` attribute

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

map_to_openapi_type(*field_cls*, **args*)

Set mapping for custom field class.

Parameters
field_cls (*type*) – Field class to set mapping for.

`*args` can be:

- a pair of the form `(type, format)`
- a core marshmallow field type (in which case we reuse that type's mapping)

metadata2properties(*field: Field*, ***kwargs: Any*) → dict

Return a dictionary of properties extracted from field metadata.

Will include field metadata that are valid properties of OpenAPI schema objects (e.g. "description", "enum", "example").

In addition, specification extensions are supported. Prefix `x_` to the desired extension when passing the keyword argument to the field constructor. apispec will convert `x_` to `x-` to comply with OpenAPI.

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

nested2properties(*field: Field*, *ret*) → dict

Return a dictionary of properties from `Nested <marshmallow.fields.Nested` fields.

Typically provides a reference object and will add the schema to the spec if it is not already present If a custom `schema_name_resolver` function returns `None` for the nested schema a JSON schema object will be returned

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

pluck2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `Pluck <marshmallow.fields.Pluck` fields.

Pluck effectively trans-includes a field from another schema into this, possibly wrapped in an array (`many=True`).

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

timedelta2properties(*field*, ***kwargs: Any*) → dict

Return a dictionary of properties from `TimeDelta` fields.

Adds a `x-unit` vendor property based on the field's `precision` attribute

Parameters
field (*Field*) – A marshmallow field.

Return type
dict

apispec.ext.marshmallow.field_converter.**make_min_max_attributes**(*validators*, *min_attr*, *max_attr*) → dict

Return a dictionary of minimum and maximum attributes based on a list of validators. If either minimum or maximum values are not present in any of the validator objects that attribute will be omitted.

Parameters

- **list** (*validators*) – A list of `Marshmallow` validator objects. Each objct is inspected for a minimum and maximum values
- **string** (*max_attr*) – The OpenAPI attribute for the minimum value
- **string** – The OpenAPI attribute for the maximum value

apispec.ext.marshmallow.field_converter.**make_type_list**(*types*)

Return a list of types from a type attribute

Since OpenAPI 3.1.0, "type" can be a single type as string or a list of types, including 'null'. This function takes a "type" attribute as input and returns it as a list, be it an empty or single-element list. This is useful to factorize type-conditional code or code adding a type.

apispec.ext.marshmallow.common

Utilities to get schema instances/classes

apispec.ext.marshmallow.common.**filter_excluded_fields**(*fields: dict[str, Field]*, *Meta*, **, exclude_dump_only: bool*) → dict[str, Field]

Filter fields that should be ignored in the OpenAPI spec.

Parameters

- **fields** (*dict*) – A dictionary of fields name field object pairs

- **Meta** – the schema's Meta class
- **exclude_dump_only** (*bool*) – whether to filter dump_only fields

apispec.ext.marshmallow.common.**get_fields**(*schema: type[Schema] | Schema*, ***, *exclude_dump_only: bool = False*) → dict[str, Field]

Return fields from schema.

Parameters

- **schema** (*Schema*) – A marshmallow Schema instance or a class object
- **exclude_dump_only** (*bool*) – whether to filter fields in Meta.dump_only

Return type
dict, of field name field object pairs

apispec.ext.marshmallow.common.**get_unique_schema_name**(*components: Components*, *name: str*, *counter: int = 0*) → str

Function to generate a unique name based on the provided name and names already in the spec. Will append a number to the name to make it unique if the name is already in the spec.

Parameters

- **components** (*Components*) – instance of the components of the spec
- **name** (*string*) – the name to use as a basis for the unique name
- **counter** (*int*) – the counter of the number of recursions

Returns
the unique name

apispec.ext.marshmallow.common.**resolve_schema_cls**(*schema: type[Schema] | str | Schema*) → type[Schema] | list[type[Schema]]

Return schema class for given schema (instance or class).

Parameters
type|Schema|str – instance, class or class name of marshmallow.Schema

Returns
schema class of given schema (instance or class)

apispec.ext.marshmallow.common.**resolve_schema_instance**(*schema: type[Schema] | Schema | str*) → Schema

Return schema instance for given schema (instance or class).

Parameters
schema (*type | Schema | str*) – instance, class or class name of marshmallow.Schema

Returns
schema instance of given schema (instance or class)

apispec.ext.marshmallow.common.**warn_if_fields_defined_in_meta**(*fields: dict[str, Field]*, *Meta*)

Warns user that fields defined in Meta.fields or Meta.additional will be ignored.

Parameters

- **fields** (*dict*) – A dictionary of fields name field object pairs
- **Meta** – the schema's Meta class

CHAPTER

FIVE

PROJECT LINKS

- apispec @ GitHub
- Issue Tracker

CHAPTER

SIX

PROJECT INFO

See also:

Need help upgrading to a newer version? Check out the *upgrading guide*.

6.1 Changelog

6.1.1 6.6.1 (2024-04-22)

Bug fixes:

- `MarshmallowPlugin`: Fix handling of `Nested` fields with `allow_none=True` (#833). Thanks @jc-harrison for reporting and @kolditz-senec for the PR.

6.1.2 6.6.0 (2024-03-15)

Features:

- Add IP fields to `DEFAULT_FIELD_MAPPING (:pr:`892`)` to document format. Thanks @cjproud for the PR.

6.1.3 6.5.0 (2024-02-26)

Bug fixes:

- Include `null` as a value when using `validate.OneOf` or `fields.Enum` when `allow_none` is `True` for a field (#812). Thanks @pmdarrow for reporting and @kolditz-senec for the PR.

Other changes:

- Deprecate the `__version__` attribute. Use feature detection, or `importlib.metadata.version("apispec")`, instead (#878).

6.1.4 6.4.0 (2024-01-09)

Features:

- `MarshmallowPlugin`: Support different datetime formats for `marshmallow.fields.DateTime` fields (#814). Thanks @TheBigRoomXXL for the suggestion and PR.
- `MarshmallowPlugin`: Handle resolving names of schemas with spaces in the name (#856). Thanks @duchuyvp for the PR.
- Various typing improvements (#873).

Other changes:

- Support Python 3.12.
- Drop support for Python 3.7, which is EOL.
- Remove `[validation]` from extras, as it is no longer used.

6.1.5 6.3.1 (2023-12-21)

Bug fixes:

- Fix conversion of deprecated flag on parameters (#850).

  Thanks @tsokalski for the PR.

6.1.6 6.3.0 (2023-03-10)

Features:

- Resolve schema references in parameters content (#830). Thanks @codectl for the PR.

6.1.7 6.2.0 (2023-03-06)

Features:

- Resolve references in callbacks (#827). Thanks @codectl for the PR.

6.1.8 6.1.0 (2023-03-03)

Bug fixes:

- Serialize min/max values in `field2range` (#825).

Other changes:

- Test against Python 3.11 (#809).

6.1.9 6.0.2 (2022-11-10)

Bug fixes:

- Allow passing `openapi_version` as string in `marshmallow OpenAPIConverter` (#810). Thanks @paradoxxxzero for the PR.

6.1.10 6.0.1 (2022-11-05)

Bug fixes:

- Document `fields.Enum` as list of values, not string (#806). Thanks @tadams42 for reporting.

6.1.11 6.0.0 (2022-10-15)

Features:

- Support `fields.Enum` (#802).
- *Backwards-incompatible*: Change `MarshmallowPlugin.map_to_openapi_type` from a decorator to a classic function, taking a field as first argument (#804).
- *Backwards-incompatible*: Remove `validate_spec` from public API. Users may call their validator of choice directly (#803).

Other changes:

- Drop support for marshmallow < 3.18.0 (#802).

6.1.12 6.0.0b1 (2022-10-04)

Features:

- Add `OpenAPIConverter.add_parameter_attribute_function` to allow documentation of custom list fields such as webargs `DelimitedList` (#778).
- *Backwards-incompatible*: Remove `OpenAPIVersion` and use `packaging.Version` instead (#801).

6.1.13 5.2.2 (2022-05-13)

Bug fixes:

- Fix schema property ordering regression in `ApiSpec.to_yaml()` (#768). Thanks @vorticity for the PR.

6.1.14 5.2.1 (2022-05-01)

Bug fixes:

- Fix type hints for `APISpec.path` and `BasePlugin` methods (#765).

6.1.15 5.2.0 (2022-04-29)

Features:

- Use `raise from` whenever possible (#763).

Refactoring:

- Use a `tuple` rather than a `namedtuple` for "schema key" (#725).

Other changes:

- Add type hints (#747). Thanks @kasium for the PR.
- Test against Python 3.10 (#724).
- Drop support for Python 3.6 (#727).
- Switch to Github Actions for CI (#751).

6.1.16 5.1.1 (2021-09-27)

Bug fixes:

- Fix field ordering in "ordered" schema classes documentation (#714).

Other changes:

- Don't build universal wheels. We don't support Python 2 anymore. (#705)
- Make the build reproducible (#669).

6.1.17 5.1.0 (2021-08-10)

Features:

- Add `lazy` option to component registration methods. This allows to add components to the spec only if they are actually referenced. (#702)
- Add `BasePlugin.header_helper` and `MarshmallowPlugin.header_helper` (#703).

Bug fixes:

- Ensure plugin helpers get component copies. Avoids issues if a plugin helper mutates its inputs. (#704)

6.1.18 5.0.0 (2021-07-29)

Features:

- Rename `doc_default` to `default`. Since schema metadata is namespaced in a single `metadata` parameter, there is no name collision with `default` parameter anymore (#687).
- Don't build schema component reference in `OpenAPIConverter.resolve_nested_schema`. This is done later in `Components` (#700).
- `MarshmallowPlugin`: resolve schemas in `allOf`, `oneOf`, `anyOf` and `not` (#701). Thanks @stefanv for the initial work on this.

Other changes:

- Refactor `Components` methods to make them consistent. Use `component_id` rather than `name`, remove `**kwargs` when unused. (#696)

6.1.19 5.0.0b1 (2021-07-22)

Features:

- Resolve all component references in paths and components. All references must be passed as strings, not as a `{$ref: '...'}` dict (#671).

Other changes:

- Don't use deprecated `missing` marshmallow field attribute but use `load_default` instead (#692).
- Refactor references resolution. `get_ref` method is moved from `APISpec` to `Components` (#655). `APISpec.clean_parameters` and `APISpec.clean_parameters` are now private methods (#695).
- Drop support for marshmallow < 3.13.0 (#692).

6.1.20 4.7.1 (2021-07-06)

Bug fixes:

- Correct spelling of `'null'`: remove extra quotes (#689). Thanks @mjpieters for the PR.

6.1.21 4.7.0 (2021-06-28)

Features:

- Document `deprecated` property from field metadata (#686). Thanks @greyli for the PR.
- Document `writeOnly` and `nullable` properties from field metadata (#684). Thanks @greyli for the PR.

6.1.22 4.6.0 (2021-06-14)

Features:

- Support `Pluck` field (#677). Thanks @mjpieters for the PR.
- Support `TimeDelta` field (#678).

6.1.23 4.5.0 (2021-06-04)

Features:

- Support OpenAPI 3.1.0 (#579).

Bug fixes:

- Fix `get_fields` to avoid crashing when a field is named `fields` (#673). Thanks @Reskov for reporting.

Other changes:

- Don't pass field metadata as keyword arguments in the tests. This is deprecated since marshmallow 3.10. apispec is still compatible with marshmallow >=3,<3.10 but tests now require marshmallow >=3.10. (#675)

6.1.24 4.4.2 (2021-05-24)

Bug fixes:

- Respect `partial` marshmallow schema parameter: don't document the field as required. (#627). Thanks @Anti-Distinctlyminty for the PR.

6.1.25 4.4.1 (2021-05-07)

Bug fixes:

- Don't set `additionalProperties` if `Meta.unknown` is `EXCLUDE` (#659). Thanks @kupuguy for the PR.

6.1.26 4.4.0 (2021-03-31)

Features:

- Populate `additionalProperties` from `Meta.unknown` (#635). Thanks @timsilvers for the PR.
- Allow `to_yaml` to pass kwargs to `yaml.dump` (#648).
- Resolve header references in responses (#650).
- Resolve example references in parameters, request bodies and responses (#651).

6.1.27 4.3.0 (2021-02-10)

Features:

- Add *apispec.core.Components.header* to register header components (#637).

6.1.28 4.2.0 (2021-02-06)

Features:

- Make components public attributes of `Components` class (#634).

6.1.29 4.1.0 (2021-01-26)

Features:

- Resolve schemas in callbacks (#544). Thanks @kortsi for the PR.

Bug fixes:

- Fix docstrings documenting kwargs type as dict (#534).
- Use `x-minimum` and `x-maximum` extensions to document ranges that are not of number type (e.g. datetime) (#614).

Other changes:

- Test against Python 3.9.

6.1.30 4.0.0 (2020-09-30)

Features:

- *Backwards-incompatible*: Automatically generate references for schemas passed as strings in responses and request bodies. When using `MarshmallowPlugin`, if a schema is passed as string, the marshmallow registry is looked up for this schema name and if none is found, the name is assumed to be a reference to a manually created schema and a reference is generated. No exception is raised anymore if the schema name can't be found in the registry. (#554)

6.1.31 4.0.0b1 (2020-09-06)

Features:

- *Backwards-incompatible*: Ignore `location` field metadata. This attribute was used in webargs but it has now been dropped. A `Schema` can now only have a single location. This simplifies the logic in `OpenAPIConverter` methods, where `default_in` argument now becomes `location`. (#526)
- *Backwards-incompatible*: Don't document `int` format as `"int32"` and `float` format as `"float"`, as those are platform-dependent (#595).

Refactoring:

- `OpenAPIConverter.field2parameters` and `OpenAPIConverter.property2parameter` are removed. `OpenAPIConverter.field2parameter` becomes private. (#581)

Other changes:

- Drop support for marshmallow 2. Marshmallow 3.x is required. (#583)
- Drop support for Python 3.5. Python 3.6+ is required. (#582)

6.1.32 3.3.2 (2020-08-29)

Bug fixes:

- Fix crash when field metadata contains non-string keys (#596). Thanks @sanzoghenzo for the fix.

6.1.33 3.3.1 (2020-06-06)

Bug fixes:

- Fix `MarshmallowPlugin` crash when `resolve_schema_dict` is passed a schema as string and `schema_name_resolver` returns `None` (#566). Thanks @black3r for reporting and thanks @Bangertm for the PR.

6.1.34 3.3.0 (2020-02-14)

Features:

- Instantiate `Components` before calling plugins' `init_spec` (#539). Thanks @Colin-b for the PR.

6.1.35 3.2.0 (2019-12-22)

Features:

- Add `match_info` to `__location_map__` (#517). Thanks @fedorfo for the PR.

6.1.36 3.1.1 (2019-12-17)

Bug fixes:

- Don't emit a warning when passing "default" as response status code in OASv2 (#521).

6.1.37 3.1.0 (2019-11-04)

Features:

- Add *apispec.core.Components.example* for adding Example Objects (#515). Thanks @codeasashu for the PR.

Support:

- Test against Python 3.8 (#510).

3.0.0 (2019-09-17)

Features:

- Add support for generating user-defined OpenAPI properties for custom field classes via an `add_attribute_function` method (#478 and #498).
- [apispec.ext.marshmallow]: *Backwards-incompatible* `fields.Raw` and `fields.Field` are now represented by OpenAPI Any Type (#495).
- [apispec.ext.marshmallow]: *Backwards-incompatible*: The `schema_name_resolver` function now receives a `Schema` class, a `Schema` instance or a string that resolves to a `Schema` class. This allows a custom resolver to generate different names depending on schema modifiers used in a `Schema` instance (#476).

Bug fixes:

- [apispec.ext.marshmallow]: With marshmallow 3, the default value of a field in the documentation is the serialized value of the `missing` attribute, not `missing` itself (#490).

Refactoring:

- `clean_parameters` and `clean_operations` are now `APISpec` methods (#489).
- [apispec.ext.marshmallow]: `Schema` resolver methods are extracted from `MarshmallowPlugin` into a `SchemaResolver` class member (#496).
- [apispec.ext.marshmallow]: `OpenAPIConverter` is now a class member of `MarshmallowPlugin` (#493).
- [apispec.ext.marshmallow]: `Field` to properties conversion logic is extracted from `OpenAPIConverter` into `FieldConverterMixin` (#478).

Other changes:

- Drop support for Python 2 (#491). Thanks @hugovk for the PR.
- Drop support for marshmallow pre-releases. Only stable 2.x and 3.x versions are supported (#485).

2.0.2 (2019-07-04)

Bug fixes:

- Fix compatibility with marshmallow 3.0.0rc8 (#469).

Other changes:

- Switch to Azure Pipelines (#468).

2.0.1 (2019-06-26)

Bug fixes:

- Don't mutate `operations` and `parameters` in `APISpec.path` to avoid issues when calling it twice with the same `operations` or `parameters` (#464).

2.0.0 (2019-06-18)

Features:

- Add support for path level parameters (#453). Thanks @karec for the PR.
- *Backwards-incompatible*: A `apispec.exceptions.DuplicateParameterError` is raised when two parameters with same name and location are passed to a path or an operation (#455).
- *Backwards-incompatible*: A `apispec.exceptions.InvalidParameterError` is raised when a parameter is missing required `name` and `in` attributes after helpers have been executed (#455).

Other changes:

- *Backwards-incompatible*: All plugin helpers must accept extra `**kwargs` (#453).
- *Backwards-incompatible*: Components must be referenced by ID, not full path (#463).

1.3.3 (2019-05-05)

Bug fixes:

- marshmallow 3.0.0rc6 compatibility (#445).

1.3.2 (2019-05-02)

Bug fixes:

- Fix handling of OpenAPI v3 components content without schema in `MarshmallowPlugin` (#443).

1.3.1 (2019-04-29)

Bug fixes:

- Fix handling of `http.HTTPStatus` objects (#426). Thanks @DStape.
- [apispec.ext.marshmallow]: Ensure make_schema_key returns a unique key on unhashable iterables (#416, #439). Thanks @zedrdave.

1.3.0 (2019-04-24)

Features:

- [apispec.ext.marshmallow]: Use class hierarchy to infer `type` and `format` properties (#433, #250). Thanks @andrjohn for the PR.

1.2.1 (2019-04-18)

Bug fixes:

- Fix error in `MarshmallowPlugin` when passing `exclude` and `dump_only` as `class Meta` attributes mixing `list` and `tuple` (#431). Thanks @blagasz for the PR.

1.2.0 (2019-04-08)

Features:

- Strip empty sections (components, tags) from generated documentation (#421 and #425).

1.1.2 (2019-04-07)

Bug fixes:

- Fix behavior when using "2xx", 3xx", etc. for response keys (#422). Thanks @zachmullen for reporting.

1.1.1 (2019-04-02)

Bug fixes:

- Fix passing references for parameters/responses when using `MarshmallowPlugin` (#414).

1.1.0 (2019-03-17)

Features:

- Resolve `Schema` classes in response headers (#409).

1.0.0 (2019-02-08)

Features:

- Expanded support for OpenAPI Specification version 3 (#165).
- Add `summary` and `description` parameters to `APISpec.path` (#227). Thanks @timakro for the suggestion.
- Add *apispec.core.Components.security_scheme* for adding Security Scheme Objects (#245).
- [apispec.ext.marshmallow]: Add support for outputting field patterns from `Regexp` validators (#364). Thanks @DStape for the PR.

Bug fixes:

- [apispec.ext.marshmallow]: Fix automatic documentation of schemas when using `Nested(MySchema, many==True)` (#383). Thanks @whoiswes for reporting.

Other changes:

- *Backwards-incompatible*: Components properties are now passed as dictionaries rather than keyword arguments (#381).

```
# <1.0.0
spec.components.schema("Pet", properties={"name": {"type": "string"}})
spec.components.parameter("PetId", "path", format="int64", type="integer")
spec.components.response("NotFound", description="Pet not found")

# >=1.0.0
spec.components.schema("Pet", {"properties": {"name": {"type": "string"}}})
spec.components.parameter("PetId", "path", {"format": "int64", "type": "integer"})
spec.components.response("NotFound", {"description": "Pet not found"})
```

Deprecations/Removals:

- *Backwards-incompatible*: The `ref` argument passed to fields is no longer used (#354). References for nested `Schema` are stored automatically.
- *Backwards-incompatible*: The `extra_fields` argument of *apispec.core.Components.schema* is removed. All properties may be passed in the `component` argument.

```
# <1.0.0
spec.definition("Pet", schema=PetSchema, extra_fields={"discriminator": "name"})

# >=1.0.0
spec.components.schema("Pet", schema=PetSchema, component={"discriminator": "name"})
```

1.0.0rc1 (2018-01-29)

Features:

- Automatically generate references to nested schemas with a computed name, e.g. `fields.Nested(PetSchema())` -> `#components/schemas/Pet`.
- Automatically generate references for `requestBody` using the above mechanism.
- Ability to opt out of the above behavior by passing a `schema_name_resolver` function that returns `None` to `api.ext.MarshmallowPlugin`.
- References now respect Schema modifiers, including `exclude` and `partial`.

- *Backwards-incompatible*: A *apispec.exceptions.DuplicateComponentNameError* is raised when registering two components with the same name (#340).

1.0.0b6 (2018-12-16)

Features:

- *Backwards-incompatible*: `basePath` is not removed from paths anymore. Paths passed to `APISpec.path` should not contain the application base path (#345).
- Add `apispec.ext.marshmallow.openapi.OpenAPIConverter.resolve_schema_class` (#346). Thanks @buxx.

1.0.0b5 (2018-11-06)

Features:

- `apispec.core.Components` is added. Each `APISpec` instance has a `Components` object used to define components such as schemas, parameters or responses. "Components" is the OpenAPI v3 terminology for those reusable top-level objects.
- `apispec.core.Components.parameter` and `apispec.core.Components.response` are added.
- *Backwards-incompatible*: `apispec.APISpec.add_path` and `apispec.APISpec.add_tag` are renamed to `apispec.APISpec.path` and `apispec.APISpec.tag`.
- *Backwards-incompatible*: `apispec.APISpec.definition` is moved to the `Components` class and renamed to `apispec.core.Components.schema`.

```
# apispec<1.0.0b5
spec.add_tag({'name': 'Pet', 'description': 'Operations on pets'})
spec.add_path('/pets/', operations=...)
spec.definition('Pet', properties=...)

# apispec>=1.0.0b5
spec.tag({'name': 'Pet', 'description': 'Operations on pets'})
spec.path('/pets/', operations=...)
spec.components.schema('Pet', properties=...)
```

- Plugins can define `parameter_helper` and `response_helper` to modify parameter and response components definitions.
- `MarshmallowPlugin` resolves schemas in parameters and responses components.
- Components helpers may return `None` as a no-op rather than an empty `dict` (#336).

Bug fixes:

- `MarshmallowPlugin.schema_helper` does not crash when no schema is passed (#336).

Deprecations/Removals:

- The legacy `response_helper` feature is removed. The same can be achieved from `operation_helper`.

1.0.0b4 (2018-10-28)

- *Backwards-incompatible*: `apispec.ext.flask`, `apispec.ext.bottle`, and `apispec.ext.tornado` are moved to a separate package, apispec-webframeworks. (#302).

If you use these plugins, install `apispec-webframeworks` and update your imports like so:

```
# apispec<1.0.0b4
from apispec.ext.flask import FlaskPlugin

# apispec>=1.0.0b4
from apispec_webframeworks.flask import FlaskPlugin
```

Thanks @ergo for the suggestion and the PR.

1.0.0b3 (2018-10-08)

Features:

- [apispec.core]: *Backwards-incompatible*: `openapi_version` parameter of `APISpec` class does not default to `'2.0'` anymore and `info` parameter is merged with `**options` kwargs.

Bug fixes:

- [apispec.ext.marshmallow]: Exclude `load_only` fields when documenting responses (#119). Thanks @luisin-crespo for reporting.
- [apispec.ext.marshmallow]: Exclude `dump_only` fields when documenting request body parameter schema.

1.0.0b2 (2018-09-09)

- Drop deprecated plugin interface. Only plugin classes are now supported. This includes the removal of `APISpec`'s `register_*_helper` methods, as well as its `schema_name_resolver` parameter. Also drop deprecated `apispec.utils.validate_swagger`. (#259)
- Use `yaml.safe_load` instead of `yaml.load` when reading docstrings (#278). Thanks @lbeaufort for the suggestion and the PR.

1.0.0b1 (2018-07-29)

Features:

- [apispec.core]: *Backwards-incompatible*: Remove `Path` class. Plugins' `path_helper` methods should now return a path as a string and optionally mutate the `operations` dictionary (#238).
- [apispec.core]: *Backwards-incompatible*: YAML support is optional. To install with YAML support, use `pip install 'apispec[yaml]'`. You will need to do this if you use `FlaskPlugin`, `BottlePlugin`, or `TornadoPlugin` (#251).
- [apispec.ext.marshmallow]: Allow overriding the documentation for a field's default. This is especially useful for documenting callable defaults (#196).

0.39.0 (2018-06-28)

Features:

- [apispec.core]: *Backwards-incompatible*: Change plugin interface. Plugins are now child classes of `apispec.BasePlugin`. Built-in plugins are still usable with the deprecated legacy interface. However, the new class interface is mandatory to pass parameters to plugins or to access specific methods that used to be accessed as module level functions (typically in `apispec.ext.marshmallow.swagger`). Also, `schema_name_resolver` is now a parameter of `apispec.ext.marshmallow.MarshmallowPlugin`. It can still be passed to `APISpec` while using the legacy interface. (#207)
- [apispec.core]: *Backwards-incompatible*: `APISpec.openapi_version` is now an `apispec.utils.OpenAPIVersion` instance.

0.38.0 (2018-06-10)

Features:

- [apispec.core]: *Backwards-incompatible*: Rename `apispec.utils.validate_swagger` to `apispec.utils.validate_spec` and `apispec.exceptions.SwaggerError` to `apispec.exceptions.OpenAPIError`. Using `validate_swagger` will raise a `DeprecationWarning` (#224).
- [apispec.core]: `apispec.utils.validate_spec` no longer relies on the `check_api` NPM module. `prance` and `openapi-spec-validator` are required for validation, and can be installed using `pip install 'apispec[validation]'` (#224).
- [apispec.core]: Deep update components instead of overwriting components for OpenAPI 3 (#222). Thanks @Guoli-Lyu.

Bug fixes:

- [apispec.ext.marshmallow]: Fix description for parameters in OpenAPI 3 (#223). Thanks again @Guoli-Lyu.

Other changes:

- Drop official support for Python 3.4. Only Python 2.7 and >=3.5 are supported.

0.37.1 (2018-05-28)

Features:

- [apispec.ext.marshmallow]: Fix OpenAPI 3 conversion of schemas in parameters (#217). Thanks @Guoli-Lyu for the PR.

0.37.0 (2018-05-14)

Features:

- [apispec.ext.marshmallow]: Resolve an array of schema objects in parameters (#209). Thanks @cvlong for reporting and implementing this.

0.36.0 (2018-05-07)

Features:

- [apispec.ext.marshmallow]: Document `values` parameter of `Dict` field as `additionalProperties` (#201). Thanks @UrKr.

0.35.0 (2018-04-10)

Features:

- [apispec.ext.marshmallow]: Recurse over properties when resolving schemas (#186). Thanks @lphuberdeau.
- [apispec.ext.marshmallow]: Support `writeOnly` and `nullable` in OpenAPI 3 (fall back to `x-nullable` for OpenAPI 2) (#165). Thanks @lafrech.

Bug fixes:

- [apispec.ext.marshmallow]: Always use `field.missing` instead of `field.default` when introspecting fields (#32). Thanks @lafrech.

Other changes:

- [apispec.ext.marshmallow]: Refactor some of the internal functions in `apispec.ext.marshmallow.swagger` for consistent API (#199). Thanks @lafrech.

0.34.0 (2018-04-04)

Features:

- [apispec.core]: Maintain order in which methods are added to an endpoint (#189). Thanks @lafrech.

Other changes:

- [apispec.core]: `Path` no longer inherits from `dict` (#190). Thanks @lafrech.

0.33.0 (2018-04-01)

Features:

- [apispec.ext.marshmallow]: Respect `data_key` argument on fields (in marshmallow 3). Thanks @lafrech.

0.32.0 (2018-03-24)

Features:

- [apispec.ext.bottle]: Allow `app` to be passed to `spec.add_path` (#188). Thanks @dtaniwaki for the PR.

Bug fixes:

- [apispec.ext.marshmallow]: Fix issue where "body" and "required" were getting overwritten when passing a `Schema` to a parameter (#168, #184). Thanks @dlopuch and @mathewmarcus for reporting and thanks @mathewmarcus for the PR.

0.31.0 (2018-01-30)

- [apispec.ext.marshmallow]: Use `dump_to` for name even if `load_from` does not match it (#178). Thanks @LeonAgmonNacht for reporting and thanks @lafrech for the fix.

0.30.0 (2018-01-12)

Features:

- [apispec.core]: Add `Spec.to_yaml` method for serializing to YAML (#161). Thanks @jd.

0.29.0 (2018-01-04)

Features:

- [apispec.core and apispec.ext.marshmallow]: Add limited support for OpenAPI v3. Pass `openapi_version='3.0.0'` to `Spec` to use it (#165). Thanks @Bangertm.

0.28.0 (2017-12-09)

Features:

- [apispec.core and apispec.ext.marshmallow]: Add `schema_name_resolver` param to `APISpec` for resolving ref names for marshmallow Schemas. This is useful when a self-referencing schema is nested within another schema (#167). Thanks @buxx for the PR.

0.27.1 (2017-12-06)

Bug fixes:

- [apispec.ext.flask]: Don't document view methods that aren't included in `app.add_url_rule(..., methods=[...])` (#173). Thanks @ukaratay.

0.27.0 (2017-10-30)

Features:

- [apispec.core]: Add `register_operation_helper`.

Bug fixes:

- Order of plugins does not matter (#136).

Thanks @yoichi for these changes.

0.26.0 (2017-10-23)

Features:

- [apispec.ext.marshmallow]: Generate "enum" property with single entry when the `validate.Equal` validator is used (#155). Thanks @Bangertm for the suggestion and PR.

Bug fixes:

- Allow OPTIONS to be documented (#162). Thanks @buxx for the PR.
- Fix regression from 0.25.3 that caused a `KeyError` (#163). Thanks @yoichi.

0.25.4 (2017-10-09)

Bug fixes:

- [apispec.ext.marshmallow]: Fix swagger location mapping for `default_in` param in fields2parameters (#156). Thanks @decaz.

0.25.3 (2017-09-27)

Bug fixes:

- [apispec.ext.marshmallow]: Correctly handle multiple fields with `location=json` (#75). Thanks @shaicantor for reporting and thanks @yoichi for the patch.

0.25.2 (2017-09-05)

Bug fixes:

- [apispec.ext.marshmallow]: Avoid AttributeError when passing non-dict items to path objects (#151). Thanks @yoichi.

0.25.1 (2017-08-23)

Bug fixes:

- [apispec.ext.marshmallow]: Fix `use_instances` when `many=True` is set (#148). Thanks @theirix.

0.25.0 (2017-08-15)

Features:

- [apispec.ext.marshmallow]: Add `use_instances` parameter to `fields2paramters` (#144). Thanks @theirix.

Other changes:

- Don't swallow `YAMLError` when YAML parsing fails (#135). Thanks @djanderson for the suggestion and the PR.

0.24.0 (2017-08-15)

Features:

- [apispec.ext.marshmallow]: Add `swagger.map_to_swagger_field` decorator to support custom field classes (#120). Thanks @frol for the suggestion and thanks @dradetsky for the PR.

0.23.1 (2017-08-08)

Bug fixes:

- [apispec.ext.marshmallow]: Fix swagger location mapping for `default_in` param in `property2parameter` (#142). Thanks @decaz.

0.23.0 (2017-08-03)

- Pass `operations` constructed by plugins to downstream marshmallow plugin (#138). Thanks @yoichi.
- [apispec.ext.marshmallow] Generate parameter specification from marshmallow Schemas (#127). Thanks @ewalker11 for the suggestion thanks @yoichi for the PR.
- [apispec.ext.flask] Add support for Flask MethodViews (#85, #125). Thanks @lafrech and @boosh for the suggestion. Thanks @djanderson and @yoichi for the PRs.

0.22.3 (2017-07-16)

- Release wheel distribution.

0.22.2 (2017-07-12)

Bug fixes:

- [apispec.ext.marshmallow]: Properly handle callable `default` values in output spec (#131). Thanks @Night-Blues.

0.22.1 (2017-06-25)

Bug fixes:

- [apispec.ext.marshmallow]: Include `default` in output spec when `False` is the default for a `Boolean` field (#130). Thanks @nebularazer.

0.22.0 (2017-05-30)

Features:

- [apispec.ext.bottle] Added bottle plugin (#128). Thanks @lucasrc.

0.21.0 (2017-04-21)

Features:

- [apispec.ext.marshmallow] Sort list of required field names in generated spec (#124). Thanks @dradetsky.

0.20.1 (2017-04-18)

Bug fixes:

- [apispec.ext.tornado]: Fix compatibility with Tornado>=4.5.
- [apispec.ext.tornado]: Fix adding paths for handlers with coroutine methods in Python 2 (#99).

0.20.0 (2017-03-19)

Features:

- [apispec.core]: Definition helper functions receive the `definition` keyword argument, which is the current state of the definition (#122). Thanks @martinlatrille for the PR.

Other changes:

- [apispec.ext.marshmallow] *Backwards-incompatible*: Remove `dump` parameter from `schema2parameters`, `fields2parameters`, and `field2parameter` (#114). Thanks @lafrech and @frol for the feedback and @lafrech for the PR.

0.19.0 (2017-03-05)

Features:

- [apispec.core]: Add `extra_fields` parameter to `APISpec.definition` (#110). Thanks @lafrech for the PR.
- [apispec.ext.marshmallow]: Preserve the order of `choices` (#113). Thanks @frol for the PR.

Bug fixes:

- [apispec.ext.marshmallow]: 'discriminator' is no longer valid as field metadata. It should be defined by passing `extra_fields={'discriminator': '...'}` to `APISpec.definition`. Thanks for reporting, @lafrech.
- [apispec.ext.marshmallow]: Allow additional properties when translating `Nested` fields using `allOf` (#108). Thanks @lafrech for the suggestion and the PR.
- [apispec.ext.marshmallow]: Respect `dump_only` and `load_only` specified in `class Meta` (#84). Thanks @lafrech for the fix.

Other changes:

- Drop support for Python 3.3.

0.18.0 (2017-02-19)

Features:

- [apispec.ext.marshmallow]: Translate `allow_none` on `Fields` to `x-nullable` (#66). Thanks @lafrech.

0.17.4 (2017-02-16)

Bug fixes:

- [apispec.ext.marshmallow]: Fix corruption of `Schema._declared_fields` when serializing an APISpec (#107). Thanks @serebrov for the catch and patch.

0.17.3 (2017-01-21)

Bug fixes:

- [apispec.ext.marshmallow]: Fix behavior when passing `Schema` instances to `APISpec.definition`. The `Schema's` class will correctly be registered as a an available `ref` (#84). Thanks @lafrech for reporting and for the PR.

0.17.2 (2017-01-03)

Bug fixes:

- [apispec.ext.tornado]: Remove usage of `inspect.getargspec` for Python >= 3.3 (#102). Thanks @mati-jabesednik.

0.17.1 (2016-11-19)

Bug fixes:

- [apispec.ext.marshmallow]: Prevent unnecessary warning when generating specs for marshmallow Schema's with autogenerated fields (#95). Thanks @khorolets reporting and for the PR.
- [apispec.ext.marshmallow]: Correctly translate `Length` validator to `minItems` and `maxItems` for array-type fields (`Nested` and `List`) (#97). Thanks @YuriHeupa for reporting and for the PR.

0.17.0 (2016-10-30)

Features:

- [apispec.ext.marshmallow]: Add support for properties that start with `x-`. Thanks @martinlatrille for the PR.

0.16.0 (2016-10-12)

Features:

- [apispec.core]: Allow `description` to be passed to `APISpec.definition` (#93). Thanks @martinlatrille.

0.15.0 (2016-10-02)

Features:

- [apispec.ext.marshmallow]: Allow `'query'` to be passed as a field location (#89). Thanks @lafrech.

Bug fixes:

- [apispec.ext.flask]: Properly strip off `basePath` when `APPLICATION_ROOT` is set on a Flask app's config (#78). Thanks @deckar01 for reporting and @asteinlein for the PR.

0.14.0 (2016-08-14)

Features:

- [apispec.core]: Maintain order in which paths are added to a spec (#87). Thanks @ranjanashish for the PR.
- [apispec.ext.marshmallow]: Maintain order of fields when `ordered=True` on Schema. Thanks again @ranjanashish.

0.13.0 (2016-07-03)

Features:

- [apispec.ext.marshmallow]: Add support for `Dict` field (#80). Thanks @ericb for the PR.
- [apispec.ext.marshmallow]: `dump_only` fields add `readOnly` flag in OpenAPI spec (#79). Thanks @itajaja for the suggestion and PR.

Bug fixes:

- [apispec.ext.marshmallow]: Properly exclude nested dump-only fields from parameters (#82). Thanks @incognick for the catch and patch.

Support:

- Update tasks.py for compatibility with invoke>=0.13.0.

0.12.0 (2016-05-22)

Features:

- [apispec.ext.marshmallow]: Inspect validators to set additional attributes (#66). Thanks @deckar01 for the PR.

Bug fixes:

- [apispec.ext.marshmallow]: Respect `partial` parameters on `Schemas` (#74). Thanks @incognick for reporting.

0.11.1 (2016-05-02)

Bug fixes:

- [apispec.ext.flask]: Flask plugin respects `APPLICATION_ROOT` from app's config (#69). Thanks @deckar01 for the catch and patch.
- [apispec.ext.marshmallow]: Fix support for plural schema instances (#71). Thanks again @deckar01.

0.11.0 (2016-04-12)

Features:

- Support vendor extensions on paths (#65). Thanks @lucascosta for the PR.
- *Backwards-incompatible*: Remove support for old versions (<=0.15.0) of webargs.

Bug fixes:

- Fix error message when plugin does not have a `setup()` function.
- [apispec.ext.marshmallow] Fix bug in introspecting self-referencing marshmallow fields, i.e. `fields.Nested('self')` (#55). Thanks @whoiswes for reporting.
- [apispec.ext.marshmallow] `field2property` no longer pops off `location` from a field's metadata (#67).

Support:

- Lots of new docs, including a User Guide and improved extension docs.

0.10.1 (2016-04-09)

Note: This version is a re-upload of 0.10.0. There is no 0.10.0 release on PyPI.

Features:

- Add Tornado extension (#62).

Bug fixes:

- Compatibility fix with marshmallow>=2.7.0 (#64).
- Fix bug that raised error for Swagger parameters that didn't include the `in` key (#63).

Big thanks @lucascosta for all these changes.

0.9.1 (2016-03-17)

Bug fixes:

- Fix generation of metadata for `Nested` fields (#61). Thanks @martinlatrille.

0.9.0 (2016-03-13)

Features:

- Add `APISpec.add_tags` method for adding Swagger tags. Thanks @martinlatrille.

Bug fixes:

- Fix bug in marshmallow extension where metadata was being lost when converting marshmallow `Schemas` when `many=False`. Thanks again @martinlatrille.

Other changes:

- Remove duplicate `SWAGGER_VERSION` from `api.ext.marshmallow.swagger`.

Support:

- Update docs to reflect rename of Swagger to OpenAPI.

0.8.0 (2016-03-06)

Features:

- `apispec.ext.marshmallow.swagger.schema2jsonschema` properly introspects `Schema` instances when `many=True` (#53). Thanks @frol for the PR.

Bug fixes:

- Fix error reporting when an invalid object is passed to `schema2jsonschema` or `schema2parameters` (#52). Thanks again @frol.

0.7.0 (2016-02-11)

Features:

- `APISpec.add_path` accepts `Path` objects (#49). Thanks @Trii for the suggestion and the implementation.

Bug fixes:

- Use correct field name in "required" array when `load_from` and `dump_to` are used (#48). Thanks @benbeadle for the catch and patch.

0.6.0 (2016-01-04)

Features:

- Add `APISpec#add_parameter` for adding common Swagger parameter objects. Thanks @jta.
- The field name in a spec will be adjusted if a `Field's load_from` and `dump_to` attributes are the same. #43. Thanks again @jta.

Bug fixes:

- Fix bug that caused a stack overflow when adding nested Schemas to an `APISpec` (#31, #41). Thanks @alapshin and @itajaja for reporting. Thanks @itajaja for the patch.

0.5.0 (2015-12-13)

- `schema2jsonschema` and `schema2parameters` can introspect a marshmallow `Schema` instance as well as a `Schema` class (#37). Thanks @frol.
- *Backwards-incompatible*: The first argument to `schema2jsonschema` and `schema2parameters` was changed from `schema_cls` to `schema`.

Bug fixes:

- Handle conflicting signatures for plugin helpers. Thanks @AndrewPashkin for the catch and patch.

0.4.2 (2015-11-23)

- Skip dump-only fields when `dump=False` is passed to `schema2parameters` and `fields2parameters`. Thanks @frol.

Bug fixes:

- Raise `SwaggerError` when `validate_swagger` fails. Thanks @frol.

0.4.1 (2015-10-19)

- Correctly pass `dump` parameter to `field2parameters`.

0.4.0 (2015-10-18)

- Add `dump` parameter to `field2property` (#32).

0.3.0 (2015-10-02)

- Rename and repackage as "apispec".
- Support `enum` field of JSON Schema based on `OneOf` and `ContainsOnly` validators.

0.2.0 (2015-09-27)

- Add `schema2parameters`, `fields2parameters`, and `field2parameters`.
- Removed `Fixed` from `swagger.FIELD_MAPPING` for compatibility with marshmallow>=2.0.0.

0.1.0 (2015-09-13)

- First release.

6.2 Upgrading to Newer Releases

This section documents migration paths to new releases.

6.2.1 Upgrading to 5.0.0

6.2.2 Upgrading to 4.0.0

location is ignored in field metadata

`location` parameter is ignored in `Field` metadata. A `Schema` can only have a single location.

A `Schema` with fields from different locations must be split into multiple ``Schema``s.

6.2.3 Upgrading to 3.0.0

6.2.4 Upgrading to 2.0.0

plugin helpers must accept extra `**kwargs`

Since custom plugins helpers may define extra kwargs and those kwargs are passed to all plugin helpers by `APISpec.path`, all plugins should accept unknown kwargs.

The example plugin below defines an additional `func` argument and accepts extra `**kwargs`.

```
class MyPlugin(BasePlugin):
    def path_helper(self, path, func, **kwargs):
        """Path helper that parses docstrings for operations. Adds a
        ``func`` parameter to `apispec.APISpec.path`.
        """
        operations.update(load_operations_from_docstring(func.__doc__))
```

Components must be referenced by ID, not full path

While apispec 1.x would let the user reference components by path or ID, apispec 2.x only accepts references by ID.

```
# apispec<2.0.0
spec.path(
    path="/gist/{gist_id}",
    operations=dict(
        get=dict(
            responses={
                "200": {
                    "content": {
                        "application/json": {"schema": {"$ref": "#/definitions/Gist"}}
                    }
                }
            }
        )
    ),
```

(continues on next page)

(continued from previous page)

```
)

# apispec>=2.0.0
spec.path(
    path="/gist/{gist_id}",
    operations=dict(
        get=dict(
            responses={"200": {"content": {"application/json": {"schema": "Gist"}}}}
        )
    ),
)
```

References by ID are accepted by both apispec 1.x ad 2.x and are a better choice because they delegate the creation of the full component path to apispec. This allows more flexibility as apispec creates the component path according to the OpenAPI version.

6.2.5 Upgrading to 1.0.0

`openapi_version` Is Required

`openapi_version` no longer defaults to `"2.0"`. It is now a required argument.

```
spec = APISpec(
    title="Swagger Petstore",
    version="1.0.0",
    openapi_version="2.0",  # or "3.0.2"
    plugins=[MarshmallowPlugin()],
)
```

Web Framework Plugins Packaged Separately

`apispec.ext.flask`, `apispec.ext.bottle`, and `apispec.ext.tornado` have been moved to a a separate package, apispec-webframeworks.

If you use these plugins, install `apispec-webframeworks` with `pip`:

```
$ pip install apispec-webframeworks
```

Then, update your imports:

```
# apispec<1.0.0
from apispec.ext.flask import FlaskPlugin

# apispec>=1.0.0
from apispec_webframeworks.flask import FlaskPlugin
```

YAML Support Is Optional

YAML functionality is now optional. To install with YAML support:

```
$ pip install 'apispec[yaml]'
```

You will need to do this if you use `apispec-webframeworks` or call *APISpec.to_yaml* in your code.

Registering Entities

Methods for registering OAS entities are changed to the noun form for internal consistency and for consistency with OAS v3 terminology.

```
# apispec<1.0.0
spec.add_tag({"name": "Pet", "description": "Operations on pets"})
spec.add_path("/pets/", operations={...})
spec.definition("Pet", properties={...})
spec.add_parameter("PetID", "path", {...})

# apispec>=1.0.0
spec.tag({"name": "Pet", "description": "Operations on pets"})
spec.path("/pets/", operations={...})
spec.components.schema("Pet", {"properties": {...}})
spec.components.parameter("PetID", "path", {...})
```

Adding Additional Fields to Schemas

The `extra_fields` parameter to `schema` is removed. It is no longer necessary. Pass all fields in to the component `dict`.

```
# <1.0.0
spec.definition("Pet", schema=PetSchema, extra_fields={"discriminator": "name"})

# >=1.0.0
spec.components.schema("Pet", schema=PetSchema, component={"discriminator": "name"})
```

Nested Schemas Are Referenced

When using the *MarshmallowPlugin*, nested Schema classes are referenced (with `"$ref"`) in the output spec. By default, the name in the spec will be the class name with the "Schema" suffix removed, e.g. `fields.Nested(PetSchema())` -> `"#components/schemas/Pet"`.

The `ref` argument to fields.Nested is no longer respected.

```
# apispec<1.0.0
class PetSchema(Schema):
    owner = fields.Nested(
        HumanSchema,
        # `ref` has no effect in 1.0.0. Remove.
        ref="#components/schemas/Human",
    )
```

(continues on next page)

(continued from previous page)

```
# apispec>=1.0.0
class PetSchema(Schema):
    owner = fields.Nested(HumanSchema)
```

See also:

This behavior is customizable. See *Nested Schemas*.

6.3 Ecosystem

A list of apispec-related projects can be found at the GitHub wiki here:

https://github.com/marshmallow-code/apispec/wiki/Ecosystem

6.4 Authors

6.4.1 Leads

- Steven Loria @sloria
- Jérôme Lafréchoux @lafrech

6.4.2 Contributors (chronological)

- Josh Johnston @Trii
- Vlad Frolov @frol
- Josh Carp @jmcarp
- Andrew Pashkin @AndrewPashkin
- João Taveira Araújo @jta
- Giacomo Tagliabue @itajaja
- Ben Beadle @benbeadle
- Martin Latrille @martinlatrille
- Lucas Costa @lucascosta
- Jared Deckard @deckar01
- Eric Bobbitt @ericb
- Nick Phillips @incognick
- Ashish Ranjan @ranjanashish
- Jérôme Lafréchoux @lafrech
- Anders Steinlein @asteinlein
- Yuri Heupa @YuriHeupa

- Matija Besednik @matijabesednik
- Boris Serebrov @serebrov
- Daniel Radetsky @dradetsky
- Lucas Coutinho @lucasrc
- @lamiskin
- Florian Scheffler @nebularazer
- Yoichi NAKAYAMA @yoichi
- Vadim Radovel @NightBlues
- Douglas Anderson @djanderson
- Marat Sharafutdinov @decaz
- Daniel Radetsky @dradetsky
- Evgeny Seliverstov @theirix
- Michael Bangert @Bangertm
- Bastien Sevajol @buxx
- Durmus Karatay @ukaratay
- Julien Danjou @jd
- Daisuke Taniwaki @dtaniwaki
- @mathewmarcus
- Louis-Philippe Huberdeau @lphuberdeau
- Urban @UrKr
- Christina Long @cvlong
- Felix Yan @felixonmars
- Guoli Lyu @Guoli-Lyu
- Laura Beaufort @lbeaufort
- Marcin Lulek @ergo
- Jonathan Beezley @jbeezley
- David Stapleton @dstape
- Szabolcs Blága @blagasz
- Andrew Johnson @andrjohn
- Dave @zedrdave
- Emmanuel Valette @karec
- Hugo van Kemenade @hugovk
- Bastien Gerard @bagerard
- Ashutosh Chaudhary @codeasashu
- Fedor Fominykh @fedorfo
- Colin Bounouar @Colin-b

- Mikko Kortelainen @kortsi
- David Bishop @teancom
- Andrea Ghensi @sanzoghenzo
- @timsilvers
- Kangwook Lee @pbzweihander
- Martijn Pieters @mjpieters
- Duncan Booth @kupuguy
- Luke Whitehorn https://github.com/Anti-Distinctlyminty
- François Magimel https://github.com/Linkid
- Stefan van der Walt https://github.com/stefanv
- https://github.com/kasium
- Edwin Erdmanis @vorticity
- Mounier Florian @paradoxxxzero
- Renato Damas @codectl
- Tayler Sokalski @tsokalski
- Sebastien Lovergne @TheBigRoomXXL
- Luna Lovegood @duchuyvp
- Tobias Kolditz @kolditz-senec
- Christian Proud @cjproud
- **`<https://github.com/kolditz-senec >`_**

6.5 Contributing Guidelines

6.5.1 Security Contact Information

To report a security vulnerability, please use the Tidelift security contact. Tidelift will coordinate the fix and disclosure.

6.5.2 Questions, Feature Requests, Bug Reports, and Feedback…

…should all be reported on the GitHub Issue Tracker .

6.5.3 Contributing Code

Setting Up for Local Development

1. Fork apispec on GitHub.

```
$ git clone https://github.com/marshmallow-code/apispec.git
$ cd apispec
```

2. Install development requirements. **It is highly recommended that you use a virtualenv.** Use the following command to install an editable version of apispec along with its development requirements.

```
# After activating your virtualenv
$ pip install -e '.[dev]'
```

3. Install the pre-commit hooks, which will format and lint your git staged files.

```
# The pre-commit CLI was installed above
$ pre-commit install
```

Git Branch Structure

apispec abides by the following branching model:

dev
: Current development branch. **New features should branch off here**.

X.Y-line
: Maintenance branch for release `X.Y`. **Bug fixes should be sent to the most recent release branch.** The maintainer will forward-port the fix to `dev`. Note: exceptions may be made for bug fixes that introduce large code changes.

Always make a new branch for your work, no matter how small. Also, **do not put unrelated changes in the same branch or pull request**. This makes it more difficult to merge your changes.

Pull Requests

1. Create a new local branch.

```
# For a new feature
$ git checkout -b name-of-feature dev

# For a bugfix
$ git checkout -b fix-something 1.2-line
```

2. Commit your changes. Write good commit messages.

```
$ git commit -m "Detailed commit message"
$ git push origin name-of-feature
```

3. Before submitting a pull request, check the following:

- If the pull request adds functionality, it is tested and the docs are updated.
- You've added yourself to `AUTHORS.rst`.

4. **Submit a pull request to `marshmallow-code:dev` or the appropriate maintenance branch.**
   The CI build must be passing before your pull request is merged.

Running Tests

To run all tests:

```
$ pytest
```

To run syntax checks:

```
$ tox -e lint
```

(Optional) To run tests in all supported Python versions in their own virtual environments (must have each interpreter installed):

```
$ tox
```

Documentation

Contributions to the documentation are welcome. Documentation is written in reStructuredText (rST). A quick rST reference can be found here. Builds are powered by Sphinx.

To build the docs in "watch" mode:

```
$ tox -e watch-docs
```

Changes in the `docs/` directory will automatically trigger a rebuild.

6.6 License

```
Copyright Steven Loria, Jérôme Lafréchoux, and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

PYTHON MODULE INDEX

a

INDEX

S

T

W