drf-yasg Documentation

Release 1.0.3

Cristi V.

Dec 15, 2017

Table of contents:

CHAPTER 1

drf-yasg - Yet another Swagger generator

Generate **real** Swagger/OpenAPI 2.0 specifications from a Django Rest Framework API.

Compatible with

- **Django Rest Framework**: 3.7
- **Django**: 1.11, 2.0
- **Python**: 2.7, 3.4, 3.5, 3.6

Source: https://github.com/axnsan12/drf-yasg/

Documentation: https://drf-yasg.readthedocs.io/en/latest/

1.1 Features

- full support for nested Serializers and Schemas
- response schemas and descriptions
- model definitions compatible with codegen tools
- customization hooks at all points in the spec generation process
- JSON and YAML format for spec
- bundles latest version of swagger-ui and redoc for viewing the generated documentation
- schema view is cacheable out of the box
- generated Swagger schema can be automatically validated by swagger-spec-validator or flex

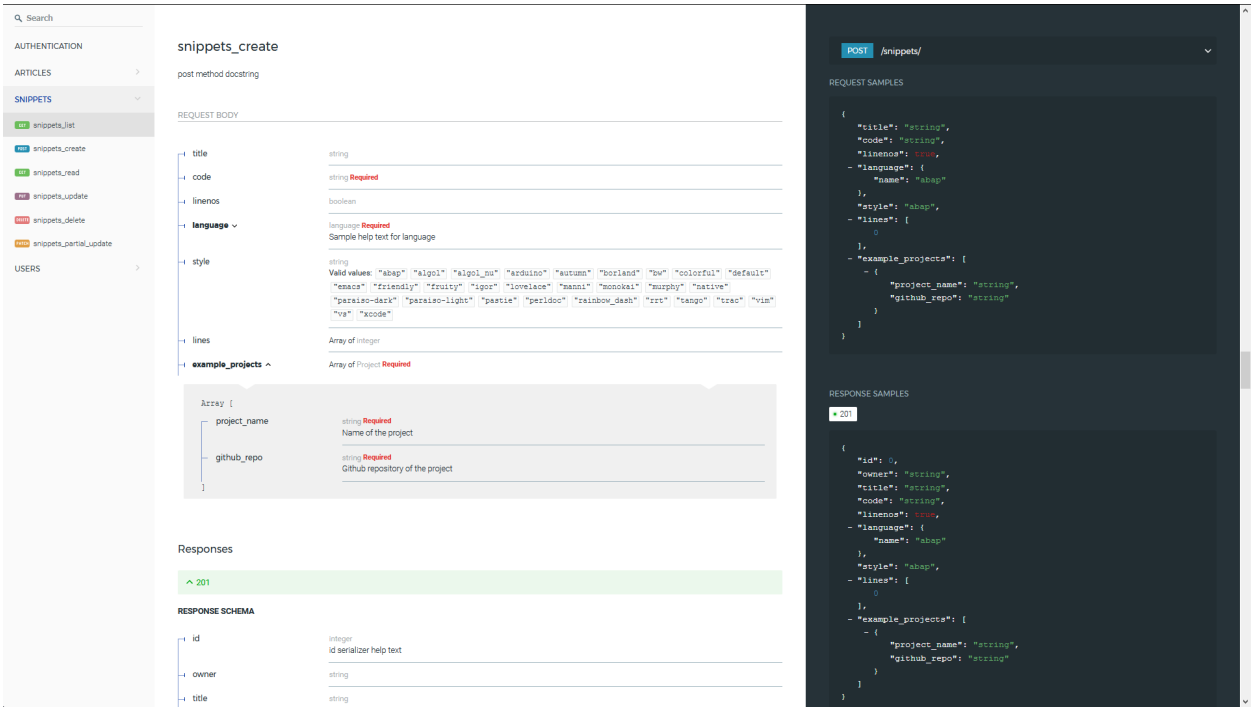

Fig. 1.1: **Fully nested request and response schemas.**

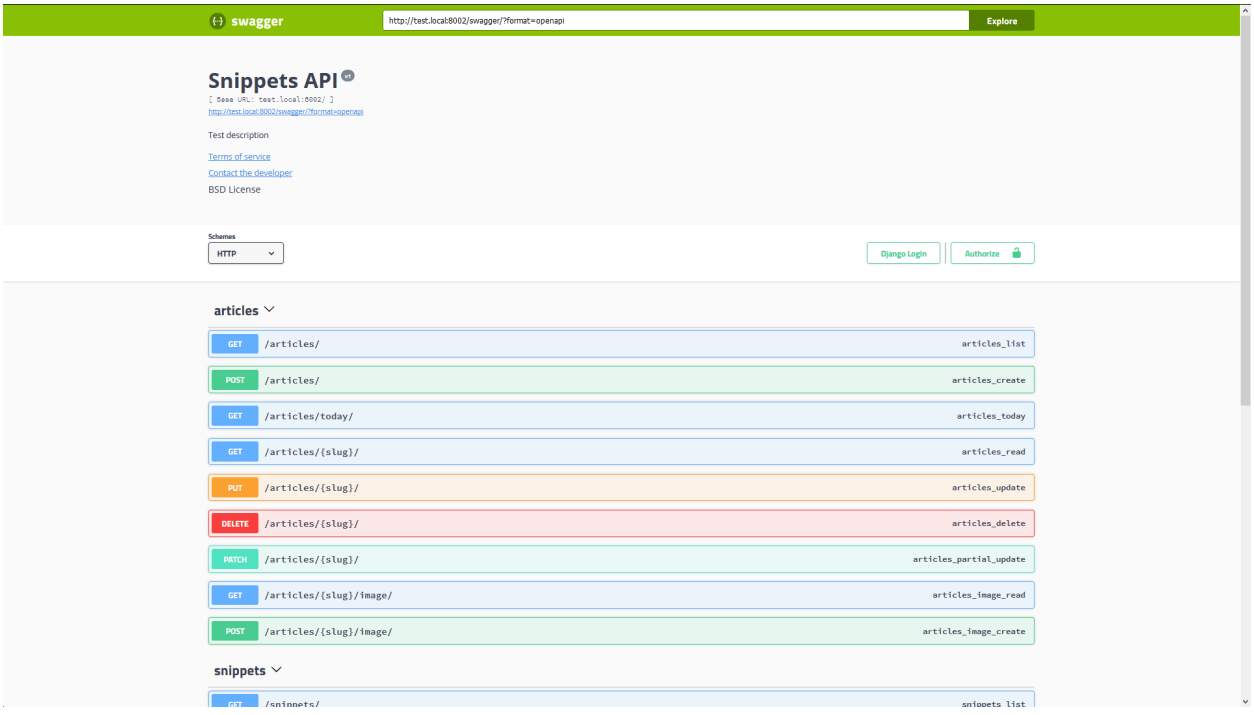

Fig. 1.2: **Choose between redoc and swagger-ui.**

Fig. 1.3: **Real Model definitions.**

1.2 Table of contents

Contents

1.3 Usage

1.3.1 0. Installation

The preferred instalation method is directly from pypi:

```
pip install drf-yasg
```

Additionally, if you want to use the built-in validation mechanisms (see *4. Validation*), you need to install some extra requirements:

```
pip install drf-yasg[validation]
```

1.3.2 1. Quickstart

In `settings.py`:

```
INSTALLED_APPS = [
    ...
    'drf_yasg',
    ...
]
```

In `urls.py`:

```
...
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

...

schema_view = get_schema_view(
    openapi.Info(
        title="Snippets API",
        default_version='v1',
        description="Test description",
        terms_of_service="https://www.google.com/policies/terms/",
        contact=openapi.Contact(email="contact@snippets.local"),
        license=openapi.License(name="BSD License"),
    ),
    validators=['ssv', 'flex'],
    public=True,
    permission_classes=(permissions.AllowAny,),
)
```

```
urlpatterns = [
    url(r'^swagger(?P<format>.json|.yaml)$', schema_view.without_ui(cache_
↪timeout=None), name='schema-json'),
    url(r'^swagger/$', schema_view.with_ui('swagger', cache_timeout=None), name=
↪'schema-swagger-ui'),
    url(r'^redoc/$', schema_view.with_ui('redoc', cache_timeout=None), name='schema-
↪redoc'),
    ...
]
```

This exposes 4 cached, validated and publicly available endpoints:

- A JSON view of your API specification at `/swagger.json`
- A YAML view of your API specification at `/swagger.yaml`
- A swagger-ui view of your API specification at `/swagger/`
- A ReDoc view of your API specification at `/redoc/`

1.3.3 2. Configuration

a. `get_schema_view` parameters

- `info` - Required. Swagger API Info object
- `url` - API base url; if left blank will be deduced from the location the view is served at
- `patterns` - passed to SchemaGenerator
- `urlconf` - passed to SchemaGenerator
- `public` - if False, includes only endpoints the current user has access to
- `validators` - a list of validator names to apply on the generated schema; allowed values are `flex`, `ssv`
- `authentication_classes` - authentication classes for the schema view itself
- `permission_classes` - permission classes for the schema view itself

b. `SchemaView` options

- `SchemaView.with_ui(renderer, cache_timeout, cache_kwargs)` - get a view instance using the specified UI renderer; one of `swagger`, `redoc`
- `SchemaView.without_ui(cache_timeout, cache_kwargs)` - get a view instance with no UI renderer; same as `as_cached_view` with no kwargs
- `SchemaView.as_cached_view(cache_timeout, cache_kwargs, **initkwargs)` - same as `as_view`, but with optional caching
- you can, of course, call `as_view` as usual

All of the first 3 methods take two optional arguments, `cache_timeout` and `cache_kwargs`; if present, these are passed on to Django's `cached_page` decorator in order to enable caching on the resulting view. See *3. Caching*.

c. SWAGGER_SETTINGS and REDOC_SETTINGS

Additionally, you can include some more settings in your `settings.py` file. The possible settings and their default values are as follows:

```
SWAGGER_SETTINGS = {
    'USE_SESSION_AUTH': True,  # add Django Login and Django Logout buttons, CSRF␣
↪token to swagger UI page
    'LOGIN_URL': getattr(django.conf.settings, 'LOGIN_URL', None),  # URL for the␣
↪login button
    'LOGOUT_URL': getattr(django.conf.settings, 'LOGOUT_URL', None),  # URL for the␣
↪logout button

    # Swagger security definitions to include in the schema;
    # see https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md
↪#security-definitions-object
    'SECURITY_DEFINITIONS': {
        'basic': {
            'type': 'basic'
        }
    },

    # url to an external Swagger validation service; defaults to 'http://online.
↪swagger.io/validator/'
    # set to None to disable the schema validation badge in the UI
    'VALIDATOR_URL': '',

    # swagger-ui configuration settings, see https://github.com/swagger-api/swagger-
↪ui/blob/112bca906553a937ac67adc2e500bdeed96d067b/docs/usage/configuration.md
↪#parameters
    'OPERATIONS_SORTER': None,
    'TAGS_SORTER': None,
    'DOC_EXPANSION': 'list',
    'DEEP_LINKING': False,
    'SHOW_EXTENSIONS': True,
    'DEFAULT_MODEL_RENDERING': 'model',
    'DEFAULT_MODEL_DEPTH': 2,
}
```

```
REDOC_SETTINGS = {
    # ReDoc UI configuration settings, see https://github.com/Rebilly/ReDoc#redoc-tag-
↪attributes
    'LAZY_RENDERING': True,
    'HIDE_HOSTNAME': False,
    'EXPAND_RESPONSES': 'all',
    'PATH_IN_MIDDLE': False,
}
```

1.3.4 3. Caching

Since the schema does not usually change during the lifetime of the django process, there is out of the box support for caching the schema view in-memory, with some sane defaults:

- caching is enabled by the cache_page decorator, using the default Django cache backend, can be changed using the `cache_kwargs` argument
- HTTP caching of the response is blocked to avoid confusing situations caused by being shown stale schemas

- if *public* is set to `False` on the SchemaView, the cached schema varies on the `Cookie` and `Authorization` HTTP headers to enable filtering of visible endpoints according to the authentication credentials of each user; note that this means that every user accessing the schema will have a separate schema cached in memory.

1.3.5 4. Validation

Given the numerous methods to manually customzie the generated schema, it makes sense to validate the result to ensure it still conforms to OpenAPI 2.0. To this end, validation is provided at the generation point using python swagger libraries, and can be activated by passing `validators=['ssv', 'flex']` to `get_schema_view`; if the generated schema is not valid, a `SwaggerValidationError` is raised by the handling codec.

Warning: This internal validation can slow down your server. Caching can mitigate the speed impact of validation.

The provided validation will catch syntactic errors, but more subtle violations of the spec might slip by them. To ensure compatibility with code generation tools, it is recommended to also employ one or more of the following methods:

`swagger-ui` validation badge

Online

If your schema is publicly accessible, *swagger-ui* will automatically validate it against the official swagger online validator and display the result in the bottom-right validation badge.

Offline

If your schema is not accessible from the internet, you can run a local copy of swagger-validator and set the *VALIDATOR_URL* accordingly:

```
SWAGGER_SETTINGS = {
    ...
    'VALIDATOR_URL': 'http://localhost:8189',
    ...
}
```

```
$ docker run --name swagger-validator -d -p 8189:8080 --add-host test.local:10.0.75.1␣
↪swaggerapi/swagger-validator
84dabd52ba967c32ae6b660934fa6a429ca6bc9e594d56e822a858b57039c8a2
$ curl http://localhost:8189/debug?url=http://test.local:8002/swagger/?format=openapi
{}
```

Using `swagger-cli`

https://www.npmjs.com/package/swagger-cli

```
$ npm install -g swagger-cli
[...]
$ swagger-cli validate http://test.local:8002/swagger.yaml
http://test.local:8002/swagger.yaml is valid
```

Manually on editor.swagger.io

Importing the generated spec into https://editor.swagger.io/ will automatically trigger validation on it. This method is currently the only way to get both syntactic and semantic validation on your specification. The other validators only provide JSON schema-level validation, but miss things like duplicate operation names, improper content types, etc

1.3.6 5. Code generation

You can use the specification outputted by this library together with swagger-codegen to generate client code in your language of choice:

```
$ docker run --rm -v ${PWD}:/local swaggerapi/swagger-codegen-cli generate -i /local/
↪tests/reference.yaml -l javascript -o /local/.codegen/js
```

See the github page linked above for more details.

1.4 Background

`OpenAPI 2.0`/`Swagger` is a format designed to encode information about a Web API into an easily parsable schema that can then be used for rendering documentation, generating code, etc.

More details are available on swagger.io and on the OpenAPI 2.0 specification page.

From here on, the terms "OpenAPI" and "Swagger" are used interchangeably.

1.4.1 Swagger in Django Rest Framework

Since Django Rest 3.7, there is now built in support for automatic OpenAPI 2.0 schema generation. However, this generation is based on the coreapi standard, which for the moment is vastly inferior to OpenAPI in both features and tooling support. In particular, the OpenAPI codec/compatibility layer provided has a few major problems:

- there is no support for documenting response schemas and status codes
- nested schemas do not work properly
- does not handle more complex fields such as `FileField`, `ChoiceField`, ...

In short this makes the generated schema unusable for code generation, and mediocre at best for documentation.

1.4.2 Other libraries

There are currently two decent Swagger schema generators that I could find for django-rest-framework:

- django-rest-swagger
- drf-openapi

Out of the two, `django-rest-swagger` is just a wrapper around DRF 3.7 schema generation with an added UI, and thus presents the same problems. `drf-openapi` is a bit more involved and implements some custom handling for response schemas, but ultimately still falls short in code generation because the responses are plain of lacking support for named schemas.

Both projects are also currently unmantained.

1.4.3 Documentation, advanced usage

https://drf-yasg.readthedocs.io/en/latest/

CHAPTER 2

Serving the schema

2.1 `get_schema_view` and the `SchemaView` class

The *`get_schema_view()`* function and the *`SchemaView`* class it returns (click links for documentation) are intended to cover the majority of use cases one might want to configure. The class returned by *`get_schema_view()`* can be used to obtain view instances via *`SchemaView.with_ui()`*, *`SchemaView.without_ui()`* and *`SchemaView.as_cached_view()`* - see *1. Quickstart* in the README for a usage example.

You can also subclass *`SchemaView`* by extending the return value of *`get_schema_view()`*, e.g.:

```
SchemaView = get_schema_view(info, ...)

class CustomSchemaView(SchemaView):
    generator_class = CustomSchemaGenerator
    renderer_classes = (CustomRenderer1, CustomRenderer2,)
```

2.2 Renderers and codecs

If you need to modify how your Swagger spec is presented in views, you might want to override one of the renderers in *`renderers`* or one of the codecs in *`codecs`*. The codec is the last stage where the Swagger object arrives before being transformed into bytes, while the renderer is the stage responsible for tying toghether the codec and the view.

You can use your custom renderer classes as kwargs to *`SchemaView.as_cached_view()`* or by subclassing *`SchemaView`*.

CHAPTER 3

Custom schema generation

If the default spec generation does not quite match what you were hoping to achieve, `drf-yasg` provides some custom behavior hooks by default.

3.1 Swagger spec overview

This library generates OpenAPI 2.0 documents. The authoritative specification for this document's structure will always be the official documentation over at swagger.io and the OpenAPI 2.0 specification page.

Beause the above specifications are a bit heavy and convoluted, here is a general overview of how the specification is structured, starting from the root `Swagger` object.

- ***`Swagger`* object**
  - `info`, `schemes`, `securityDefinitions` and other informative attributes
  - **`paths`: *`Paths`* object** A list of all the paths in the API in the form of a mapping
    - **`{path}`: *`PathItem`* - each *`PathItem`* has multiple operations keyed by method**
      - **`{http_method}`: *`Operation`*** Each operation is thus uniquely identified by its `(path, http_method)` combination, e.g. `GET /articles/`, `POST /articles/`, etc.
      - `parameters`: [*`Parameter`*] - and a list of path parameters
  - **`definitions`: named Models** A list of all the named models in the API in the form of a mapping
    - `{ModelName}`: *`Schema`*
- ***`Operation`* contains the following information about each operation:**
  - **`parameters`: [*`Parameter`*]** A list of all the *query*, *header* and *form* parameters accepted by the operation.
    - there can also be **at most one** body parameter whose structure is represented by a *`Schema`* or a reference to one (*`SchemaRef`*)

  - **responses:** ***Responses*** A list of all the possible responses the operation is expected to return. Each response can optionally have a *Schema* which describes the structure of its body.
    * `{status_code}`: *Response* - mapping of status code to response definition
  - `operationId` - should be unique across all operations
  - `tags` - used to group operations in the listing

It is interesting to note the main differences between *Parameter* and *Schema* objects:

Schema	*Parameter*
Can nest other Schemas	Cannot nest other Parameters Can only nest a Schema if the parameter is `in: body`
Cannot describe file uploads - `file` is not permitted as a value for `type`	Can describe file uploads via `type` = `file`, but only as part of a form *Operation*[1]
Can be used in *Response*s	Cannot be used in *Response*s
Cannot be used in form *Operation*s[1]	Can be used in form *Operation*s[1]

3.2 The `@swagger_auto_schema` decorator

You can use the *@swagger_auto_schema* decorator on view functions to override some properties of the generated *Operation*. For example, in a `ViewSet`,

```
@swagger_auto_schema(operation_description="partial_update description override",␣
↪responses={404: 'slug not found'})
def partial_update(self, request, *args, **kwargs):
    """partial_update method docstring"""
    ...
```

will override the description of the `PATCH /article/{id}/` operation, and document a 404 response with no body and the given description.

Where you can use the *@swagger_auto_schema* decorator depends on the type of your view:

- for function based `@api_view`s, because the same view can handle multiple methods, and thus represent multiple operations, you have to add the decorator multiple times if you want to override different operations:

```
test_param = openapi.Parameter('test', openapi.IN_QUERY, description=
↪"test manual param", type=openapi.TYPE_BOOLEAN)
user_response = openapi.Response('response description', UserSerializer)

@swagger_auto_schema(method='get', manual_parameters=[test_param],␣
↪responses={200: user_response})
@swagger_auto_schema(methods=['put', 'post'], request_
↪body=UserSerializer)
@api_view(['GET', 'PUT', 'POST'])
def user_detail(request, pk):
    ...
```

- for class based `APIView`, `GenericAPIView` and non-`ViewSet` derivatives, you have to decorate the respective method of each operation:

[1] a form Operation is an *Operation* that consumes `multipart/form-data` or `application/x-www-form-urlencoded`
- a form Operation cannot have `body` parameters
- a non-form operation cannot have `form` parameters

```
class UserList(APIView):
    @swagger_auto_schema(responses={200: UserSerializer(many=True)})
    def get(self, request):
        ...

    @swagger_auto_schema(operation_description="description")
    def post(self, request):
        ...
```

- for `ViewSet`, `GenericViewSet`, `ModelViewSet`, because each viewset corresponds to multiple **paths**, you have to decorate the *action methods*, i.e. `list`, `create`, `retrieve`, etc. Additionally, `@list_routes` or `@detail_routes` defined on the viewset, like function based api views, can respond to multiple HTTP methods and thus have multiple operations that must be decorated separately:

```
class ArticleViewSet(viewsets.ModelViewSet):
    @swagger_auto_schema(operation_description='GET /articles/today/')
    @list_route(methods=['get'])
    def today(self, request):
        ...

    @swagger_auto_schema(method='get', operation_description="GET /
↪articles/{id}/image/")
    @swagger_auto_schema(method='post', operation_description="POST /
↪articles/{id}/image/")
    @detail_route(methods=['get', 'post'], parser_
↪classes=(MultiPartParser,))
    def image(self, request, id=None):
        ...

    @swagger_auto_schema(operation_description="PUT /articles/{id}/")
    def update(self, request, *args, **kwargs):
        ...

    @swagger_auto_schema(operation_description="PATCH /articles/{id}/")
    def partial_update(self, request, *args, **kwargs):
        ...
```

3.3 Subclassing and extending

For more advanced control you can subclass *SwaggerAutoSchema* - see the documentation page for a list of methods you can override.

You can put your custom subclass to use by setting it on a view method using the *@swagger_auto_schema* decorator described above.

If you need to control things at a higher level than *Operation* objects (e.g. overall document structure, vendor extensions in metadata) you can also subclass *OpenAPISchemaGenerator* - again, see the documentation page for a list of its methods.

This custom generator can be put to use by setting it as the *generator_class* of a *SchemaView* using *get_schema_view()*.

CHAPTER 4

Customizing the web UI

There is currently no pluggable way of customizing the web UI apart from the settings available in *Swagger UI settings* and *ReDoc UI settings*. If you really need to, you can override one of the `drf-yasg/swagger-ui.html` or `drf-yasg/redoc.html` templates that are used for rendering.

CHAPTER 5

Settings

Settings are configurable in `settings.py` by defining `SWAGGER_SETTINGS` or `REDOC_SETTINGS`.

Example:

settings.py

```
SWAGGER_SETTINGS = {
    'SECURITY_DEFINITIONS': {
        'basic': {
            'type': 'basic'
        }
    },
    ...
}

REDOC_SETTINGS = {
   'LAZY_RENDERING': True,
   ...
}
```

The possible settings and their default values are as follows:

5.1 SWAGGER_SETTINGS

5.1.1 Authorization

USE_SESSION_AUTH

Enable/disable Django login as an authentication/authorization mechanism. If True, a login/logout button will be displayed in Swagger UI.

Default: `True`

LOGIN_URL

URL for the Django Login action when using *USE_SESSION_AUTH*.

Default: `django.conf.settings.LOGIN_URL`

LOGOUT_URL

URL for the Django Logout action when using *USE_SESSION_AUTH*.

Default: `django.conf.settings.LOGOUT_URL`

SECURITY_DEFINITIONS

Swagger security definitions to be included in the specification. See https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#security-definitions-object.

Default:

```
'basic': {
   'type': 'basic'
}
```

5.1.2 Swagger UI settings

Swagger UI configuration settings. See https://github.com/swagger-api/swagger-ui/blob/112bca906553a937ac67adc2e500bdeed96d067b/docs/usage/configuration.md#parameters.

VALIDATOR_URL

URL pointing to a swagger-validator instance; used for the validation badge shown in swagger-ui. Can be modified to point to a local install of swagger-validator or set to `None` to remove the badge.

Default: `'http://online.swagger.io/validator/'` *Maps to parameter*: `validatorUrl`

OPERATIONS_SORTER

Sorting order for the operation list of each tag.

- `None`: show in the order returned by the server
- `alpha`: sort alphabetically by path
- `method`: sort by HTTP method

Default: `None` *Maps to parameter*: `operationsSorter`

TAGS_SORTER

Sorting order for tagged operation groups.

- `None`: Swagger UI default ordering
- `alpha`: sort alphabetically

Default: `None` *Maps to parameter*: `tagsSorter`

DOC_EXPANSION

Controls the default expansion setting for the operations and tags.

- `None`: everything is collapsed
- `list`: only tags are expanded
- `full`: all operations are expanded

Default: `'list'` *Maps to parameter*: `docExpansion`

DEEP_LINKING

Automatically update the fragment part of the URL with permalinks to the currently selected operation.

Default: `False` *Maps to parameter*: `deepLinking`

SHOW_EXTENSIONS

Show vendor extension (`x-..`) fields.

Default: `True` *Maps to parameter*: `showExtensions`

DEFAULT_MODEL_RENDERING

Controls whether operations show the model structure or the example value by default.

- `model`: show the model fields by default
- `example`: show the example value by default

Default: `'model'` *Maps to parameter*: `defaultModelRendering`

DEFAULT_MODEL_DEPTH

Controls how many levels are expaned by default when showing nested models.

Default: `2` *Maps to parameter*: `defaultModelExpandDepth`

5.2 REDOC_SETTINGS

5.2.1 ReDoc UI settings

ReDoc UI configuration settings. See https://github.com/Rebilly/ReDoc#redoc-tag-attributes.

LAZY_RENDERING

Default: `True` *Maps to attribute*: `lazy-rendering`

HIDE_HOSTNAME

Default: `False` *Maps to attribute*: `hide-hostname`

EXPAND_RESPONSES

Default: `'all'` *Maps to attribute*: `expand-responses`

PATH_IN_MIDDLE

Default: `False` *Maps to attribute*: `path-in-middle-panel`

CHAPTER 6

Contributing

Contributions are always welcome and appreciated! Here are some ways you can contribut.

6.1 Issues

You can and should open an issue for any of the following reasons:

- you found a bug; steps for reproducing, or a pull request with a failing test case will be greatly appreciated
- you wanted to do something but did not find a way to do it after reading the documentation
- you believe the current way of doing something is more complicated or less elegant than it can be
- a related feature that you want is missing from the package

Please always check for existing issues before opening a new issue.

6.2 Pull requests

You want to contribute some code? Great! Here are a few steps to get you started:

1. Fork the repository on GitHub
2. Clone your fork and create a branch for the code you want to add
3. Create a new virtualenv and install the package in development mode

```
$ virtualenv venv
$ source venv/bin/activate
(venv) $ pip install -e .[validation]
(venv) $ pip install -rrequirements/dev.txt -rrequirements/test.txt
```

4. Make your changes and check them against the test project

```
(venv) $ cd testproj
(venv) $ python manage.py runserver
(venv) $ curl localhost:8000/swagger.yaml
```

5. Update the tests if necessary

   You can find them in the `tests` directory.

   If your change modifies the expected schema output, you should download the new generated `swagger.yaml`, diff it against the old reference output in `tests/reference.yaml`, and replace it after checking that no unexpected changes appeared.

6. Run tests. The project is setup to use tox and pytest for testing

```
# run tests in the current environment, faster than tox
(venv) $ pytest --cov
# (optional) run tests for other python versions in separate environments
(venv) $ tox
```

7. Update documentation

   If the change modifies behaviour or adds new features, you should update the documentation and `README.rst` accordingly. Documentation is written in reStructuredText and built using Sphinx. You can find the sources in the `docs` directory.

   To build and check the docs, run

```
(venv) $ tox -e docs
```

8. Push your branch and submit a pull request to the master branch on GitHub

   Incomplete/Work In Progress pull requrests are encouraged, because they allow you to get feedback and help more easily.

9. Your code must pass all the required travis jobs before it is merged. As of now, this includes running on Python 2.7, 3.4, 3.5 and 3.6, and building the docs succesfully.

CHAPTER 7

License

7.1 BSD 3-Clause License

CHAPTER 8

Changelog

8.1 1.0.3

- **FIX:** fixed bug that caused schema views returned from cache to fail (#14)
- **FIX:** disabled automatic generation of response schemas for form operations to avoid confusing errors caused by attempting to shove file parameters into Schema objects

8.2 1.0.2

- First published version

CHAPTER 9

Source code documentation

- genindex
- modindex
- search

9.1 drf_yasg package

9.1.1 drf_yasg.app_settings

class drf_yasg.app_settings.**AppSettings**(*user_settings*, *defaults*, *import_strings=None*)
Bases: object

Stolen from Django Rest Framework, removed caching for easier testing

user_settings

9.1.2 drf_yasg.codecs

class drf_yasg.codecs.**OpenAPICodecJson**(*validators*)
Bases: *drf_yasg.codecs._OpenAPICodec*

_dump_dict(*spec*)
Dump spec into JSON.

media_type = 'application/json'

class drf_yasg.codecs.**OpenAPICodecYaml**(*validators*)
Bases: *drf_yasg.codecs._OpenAPICodec*

_dump_dict(*spec*)
Dump spec into YAML.

media_type = 'application/yaml'

class drf_yasg.codecs.**_OpenAPICodec**(*validators*)

Bases: object

_dump_dict(*spec*)

Dump the given dictionary into its string representation.

Parameters **spec** (*dict*) – a python dict

Returns string representation of spec

Return type str

encode(*document*)

Transform an *Swagger* object to a sequence of bytes.

Also performs validation and applies settings.

Parameters **document** (openapi.Swagger) – Swagger spec object as generated by *OpenAPISchemaGenerator*

Returns binary encoding of document

Return type bytes

encode_error(*err*)

Dump an error message into an encoding-appropriate sequence of bytes

generate_swagger_object(*swagger*)

Generates the root Swagger object.

Parameters **swagger** (openapi.Swagger) – Swagger spec object as generated by *OpenAPISchemaGenerator*

Returns swagger spec as dict

Return type OrderedDict

media_type = None

validators

List of validator names to apply

drf_yasg.codecs.**_validate_flex**(*spec*, *codec*)

drf_yasg.codecs.**_validate_swagger_spec_validator**(*spec*, *codec*)

drf_yasg.codecs.**yaml_sane_dump**(*data*, *binary*)

Dump the given data dictionary into a sane format:

- OrderedDicts are dumped as regular mappings instead of non-standard !!odict
- multi-line mapping style instead of json-like inline style
- list elements are indented into their parents

Parameters

- **data** (*dict*) – the data to be serializers
- **binary** (*bool*) – True to return a utf-8 encoded binary object, False to return a string

Returns the serialized YAML

Return type str,bytes

9.1.3 drf_yasg.errors

exception drf_yasg.errors.**SwaggerError**
Bases: Exception

exception drf_yasg.errors.**SwaggerGenerationError**
Bases: *drf_yasg.errors.SwaggerError*

exception drf_yasg.errors.**SwaggerValidationError**(*msg*, *validator_name*, *spec*, *source_codec*, **args*)
Bases: *drf_yasg.errors.SwaggerError*

9.1.4 drf_yasg.generators

class drf_yasg.generators.**OpenAPISchemaGenerator**(*info*, *version*, *url=None*, *patterns=None*, *urlconf=None*)
Bases: object

This class iterates over all registered API endpoints and returns an appropriate OpenAPI 2.0 compliant schema. Method implementations shamelessly stolen and adapted from rest_framework SchemaGenerator.

Parameters

- **info** (*Info*) – information about the API
- **version** (*str*) – API version string, takes preedence over the version in *info*
- **url** (*str*) – API
- **patterns** – if given, only these patterns will be enumerated for inclusion in the API spec
- **urlconf** – if patterns is not given, use this urlconf to enumerate patterns; if not given, the default urlconf is used

create_view(*callback*, *method*, *request=None*)
Create a view instance from a view callback as registered in urlpatterns.

Parameters

- **callback** (*callable*) – view callback registered in urlpatterns
- **method** (*str*) – HTTP method
- **request** (*rest_framework.request.Request*) – request to bind to the view

Returns the view instance

get_endpoints(*request=None*)
Iterate over all the registered endpoints in the API.

Parameters **request** (*rest_framework.request.Request*) – used for returning only endpoints available to the given request

Returns {path: (view_class, list[(http_method, view_instance)])

Return type dict

get_operation_keys(*subpath*, *method*, *view*)
Return a list of keys that should be used to group an operation within the specification.

```
/users/                         ("users", "list"), ("users", "create")
/users/{pk}/                    ("users", "read"), ("users", "update"), ("users",
↪"delete")
/users/enabled/                 ("users", "enabled")  # custom viewset list action
/users/{pk}/star/               ("users", "star")     # custom viewset detail␣
↪action
/users/{pk}/groups/             ("users", "groups", "list"), ("users", "groups",
↪"create")
/users/{pk}/groups/{pk}/        ("users", "groups", "read"), ("users", "groups",
↪"update")
```

Parameters

- **subpath** (*str*) – path to the operation with any common prefix/base path removed
- **method** (*str*) – HTTP method
- **view** – the view associated with the operation

Return type tuple

get_overrides(*view*, *method*)
Get overrides specified for a given operation.

Parameters

- **view** – the view associated with the operation
- **method** (*str*) – HTTP method

Returns a dictionary containing any overrides set by *@swagger_auto_schema*

Return type dict

get_path_parameters(*path*, *view_cls*)
Return a list of Parameter instances corresponding to any templated path variables.

Parameters

- **path** (*str*) – templated request path
- **view_cls** (*type*) – the view class associated with the path

Returns path parameters

Return type list[*openapi.Parameter*]

get_paths(*endpoints*, *components*)
Generate the Swagger Paths for the API from the given endpoints.

Parameters

- **endpoints** (*dict*) – endpoints as returned by get_endpoints
- **components** (ReferenceResolver) – resolver/container for Swagger References

Return type *openapi.Paths*

get_schema(*request=None*, *public=False*)
Generate an *Swagger* representing the API schema.

Parameters

- **request** (*rest_framework.request.Request*) – the request used for filtering accesible endpoints and finding the spec URI
- **public** (*bool*) – if True, all endpoints are included regardless of access through *request*

Returns the generated Swagger specification

Return type *openapi.Swagger*

9.1.5 drf_yasg.inspectors

class drf_yasg.inspectors.**SwaggerAutoSchema**(*view*, *path*, *method*, *overrides*, *components*)

Bases: object

Inspector class responsible for providing *Operation* definitions given a

Parameters

- **view** – the view associated with this endpoint
- **path** (*str*) – the path component of the operation URL
- **method** (*str*) – the http method of the operation
- **overrides** (*dict*) – manual overrides as passed to *@swagger_auto_schema*
- **components** (openapi.ReferenceResolver) – referenceable components

add_manual_parameters(*parameters*)

Add/replace parameters from the given list of automatically generated request parameters.

Parameters **parameters** (*list[openapi.Parameter]*) – genereated parameters

Returns modified parameters

Return type list[*openapi.Parameter*]

coreapi_field_to_parameter(*field*)

Convert an instance of *coreapi.Field* to a swagger *Parameter* object.

Parameters **field** (*coreapi.Field*) –

Return type *openapi.Parameter*

field_to_parameter(*field*, *name*, *in_*)

Convert a DRF serializer Field to a swagger *Parameter* object.

Parameters

- **field** (*coreapi.Field*) –
- **name** (*str*) – the name of the parameter
- **in** (*str*) – the location of the parameter, one of the *openapi.IN_** constants

Return type *openapi.Parameter*

get_consumes()

Return the MIME types this endpoint can consume.

Return type list[str]

get_default_responses()

Get the default responses determined for this view from the request serializer and request method.

Type dict[str, openapi.Schema]

get_description()
Return an operation description determined as appropriate from the view's method and class docstrings.

Returns the operation description

Return type str

get_filter_backend_parameters(*filter_backend*)
Get the filter parameters for a single filter backend **instance**.

Parameters **filter_backend** (*BaseFilterBackend*) – the filter backend

Return type list[*openapi.Parameter*]

get_filter_parameters()
Return the parameters added to the view by its filter backends.

Return type list[*openapi.Parameter*]

get_operation(*operation_keys*)
Get an *Operation* for the given API endpoint (path, method). This includes query, body parameters and response schemas.

Parameters **operation_keys** (*tuple[str]*) – an array of keys describing the hierarchical layout of this view in the API; e.g. `('snippets', 'list')`, `('snippets', 'retrieve')`, etc.

Return type *openapi.Operation*

get_paged_response_schema(*response_schema*)
Add appropriate paging fields to a response *Schema*.

Parameters **response_schema** (`openapi.Schema`) – the response schema that must be paged.

Return type *openapi.Schema*

get_pagination_parameters()
Return the parameters added to the view by its paginator.

Return type list[*openapi.Parameter*]

get_paginator_parameters(*paginator*)
Get the pagination parameters for a single paginator **instance**.

Parameters **paginator** (*BasePagination*) – the paginator

Return type list[*openapi.Parameter*]

get_query_parameters()
Return the query parameters accepted by this view.

Return type list[*openapi.Parameter*]

get_request_body_parameters(*consumes*)
Return the request body parameters for this view. This is either:

- a list with a single object Parameter with a *Schema* derived from the request serializer
- a list of primitive Parameters parsed as form data

Parameters **consumes** (*list[str]*) – a list of accepted MIME types as returned by *get_consumes()*

Returns a (potentially empty) list of *Parameter*s either `in: body` or `in: formData`

Return type list[*openapi.Parameter*]

get_request_body_schema(*serializer*)
Return the *Schema* for a given request's body data. Only applies to PUT, PATCH and POST requests.

Parameters **serializer** – the view's request serializer as returned by *get_request_serializer()*

Return type *openapi.Schema*

get_request_form_parameters(*serializer*)
Given a Serializer, return a list of `in: formData` *Parameter*s.

Parameters **serializer** – the view's request serializer as returned by *get_request_serializer()*

Return type list[*openapi.Parameter*]

get_request_serializer()
Return the request serializer (used for parsing the request payload) for this endpoint.

Returns the request serializer, or one of *Schema*, *SchemaRef*, `None`

get_response_schemas(*response_serializers*)
Return the *openapi.Response* objects calculated for this view.

Parameters **response_serializers** (*dict*) – response serializers as returned by *get_response_serializers()*

Returns a dictionary of status code to *Response* object

Return type dict[str, *openapi.Response*]

get_response_serializers()
Return the response codes that this view is expected to return, and the serializer for each response body. The return value should be a dict where the keys are possible status codes, and values are either strings, `Serializers`, *Schema*, *SchemaRef* or *Response* objects. See *@swagger_auto_schema* for more details.

Returns the response serializers

Return type dict

get_responses()
Get the possible responses for this view as a swagger *Responses* object.

Returns the documented responses

Return type *openapi.Responses*

make_body_parameter(*schema*)
Given a *Schema* object, create an `in: body` *Parameter*.

Parameters **schema** (`openapi.Schema`) – the request body schema

Return type *openapi.Parameter*

serializer_to_schema(*serializer*)
Convert a DRF Serializer instance to an *openapi.Schema*.

Parameters **serializer** (*serializers.BaseSerializer*) –

Return type *openapi.Schema*

should_filter()
Determine whether filter backend parameters should be included for this request.

Return type bool

should_page()
Determine whether paging parameters and structure should be added to this operation's request and response.

Return type bool

drf_yasg.inspectors.**force_serializer_instance**(*serializer*)
Force *serializer* into a `Serializer` instance. If it is not a `Serializer` class or instance, raises an assertion error.

Parameters **serializer** – serializer class or instance

Returns serializer instance

9.1.6 drf_yasg.middleware

class drf_yasg.middleware.**SwaggerExceptionMiddleware**(*get_response*)
Bases: `object`

process_exception(*request*, *exception*)

9.1.7 drf_yasg.openapi

class drf_yasg.openapi.**Contact**(*name=None*, *url=None*, *email=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Swagger Contact object

At least one of the following fields is required:

Parameters

- **name** (*str*) – contact name
- **url** (*str*) – contact url
- **email** (*str*) – contact e-mail

class drf_yasg.openapi.**Info**(*title*, *default_version*, *description=None*, *terms_of_service=None*, *contact=None*, *license=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Swagger Info object

Parameters

- **title** (*str*) – Required. API title.
- **default_version** (*str*) – Required. API version string (not to be confused with Swagger spec version)
- **description** (*str*) – API description; markdown supported
- **terms_of_service** (*str*) – API terms of service; should be a URL
- **contact** (`Contact`) – contact object
- **license** (`License`) – license object

class drf_yasg.openapi.**Items**(*type=None*, *format=None*, *enum=None*, *pattern=None*, *items=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Used when defining an array *Parameter* to describe the array elements.

Parameters

- **type** (*str*) – type of the array elements; must not be `object`
- **format** (*str*) – value format, see OpenAPI spec
- **enum** (*list*) – restrict possible values
- **pattern** (*str*) – pattern if type is `string`
- **items** (Items) – only valid if *type* is `array`

class drf_yasg.openapi.**License**(*name*, *url=None*, ***extra*)

Bases: *drf_yasg.openapi.SwaggerDict*

Swagger License object

Parameters

- **name** (*str*) – Required. License name
- **url** (*str*) – link to detailed license information

class drf_yasg.openapi.**Operation**(*operation_id*, *responses*, *parameters=None*, *consumes=None*, *produces=None*, *description=None*, *tags=None*, ***extra*)

Bases: *drf_yasg.openapi.SwaggerDict*

Information about an API operation (path + http method combination)

Parameters

- **operation_id** (*str*) – operation ID, should be unique across all operations
- **responses** (Responses) – responses returned
- **parameters** (*list[Parameter]*) – parameters accepted
- **consumes** (*list[str]*) – content types accepted
- **produces** (*list[str]*) – content types produced
- **description** (*str*) – operation description
- **tags** (*list[str]*) – operation tags

class drf_yasg.openapi.**Parameter**(*name*, *in_*, *description=None*, *required=None*, *schema=None*, *type=None*, *format=None*, *enum=None*, *pattern=None*, *items=None*, ***extra*)

Bases: *drf_yasg.openapi.SwaggerDict*

Describe parameters accepted by an *Operation*. Each parameter should be a unique combination of (*name*, *in_*). `body` and `form` parameters in the same operation are mutually exclusive.

Parameters

- **name** (*str*) – parameter name
- **in** (*str*) – parameter location
- **description** (*str*) – parameter description
- **required** (*bool*) – whether the parameter is required for the operation
- **schema** (Schema, SchemaRef) – required if *in_* is `body`
- **type** (*str*) – parameter type; required if *in_* is not `body`; must not be `object`
- **format** (*str*) – value format, see OpenAPI spec
- **enum** (*list*) – restrict possible values
- **pattern** (*str*) – pattern if type is `string`

- **items** (Items) – only valid if *type* is `array`

class drf_yasg.openapi.**PathItem**(*get=None*, *put=None*, *post=None*, *delete=None*, *options=None*, *head=None*, *patch=None*, *parameters=None*, ***extra*)

Bases: *drf_yasg.openapi.SwaggerDict*

Information about a single path

Parameters

- **get** (Operation) – operation for GET
- **put** (Operation) – operation for PUT
- **post** (Operation) – operation for POST
- **delete** (Operation) – operation for DELETE
- **options** (Operation) – operation for OPTIONS
- **head** (Operation) – operation for HEAD
- **patch** (Operation) – operation for PATCH
- **parameters** (*list[Parameter]*) – parameters that apply to all operations

class drf_yasg.openapi.**Paths**(*paths*, ***extra*)

Bases: *drf_yasg.openapi.SwaggerDict*

A listing of all the paths in the API.

Parameters **paths** (*dict[str,PathItem]*) –

class drf_yasg.openapi.**ReferenceResolver**(**scopes*)

Bases: `object`

A mapping type intended for storing objects pointed at by Swagger Refs. Provides support and checks for different refernce scopes, e.g. 'definitions'.

For example:

```
> components = ReferenceResolver('definitions', 'parameters')
> definitions = ReferenceResolver.with_scope('definitions')
> definitions.set('Article', Schema(...))
> print(components)
{'definitions': OrderedDict([('Article', Schema(...)]), 'parameters':␣
↪OrderedDict()}
```

Parameters **scopes** (*str*) – an enumeration of the valid scopes this resolver will contain

_check_scope(*scope*)

get(*name*, *scope=None*)

Get an object from the given scope, raise an error if it does not exist.

Parameters

- **name** (*str*) – reference name
- **scope** (*str*) – reference scope

Returns the object

getdefault(*name*, *default=None*, *scope=None*)

Get an object from the given scope or a default value if it does not exist.

Parameters

- **name** (*str*) – reference name
- **default** – the default value
- **scope** (*str*) – reference scope

Returns the object or *default*

has(*name*, *scope=None*)
Check if an object exists in the given scope.

Parameters

- **name** (*str*) – reference name
- **scope** (*str*) – reference scope

Returns True if the object exists

Return type bool

keys()

scopes

set(*name*, *obj*, *scope=None*)
Set an object in the given scope, raise an error if it already exists.

Parameters

- **name** (*str*) – reference name
- **obj** – referenced object
- **scope** (*str*) – reference scope

setdefault(*name*, *maker*, *scope=None*)
Set an object in the given scope only if it does not exist.

Parameters

- **name** (*str*) – reference name
- **maker** (*callable*) – object factory, called only if necessary
- **scope** (*str*) – reference scope

with_scope(*scope*)
Return a new *ReferenceResolver* whose scope is defaulted and forced to *scope*.

Parameters **scope** (*str*) – target scope, must be in this resolver's *scopes*

Returns the bound resolver

Return type *ReferenceResolver*

class drf_yasg.openapi.**Response**(*description*, *schema=None*, *examples=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Describes the structure of an operation's response.

Parameters

- **description** (*str*) – response description
- **schema** (Schema, SchemaRef) – sturcture of the response body
- **examples** (*dict*) – example bodies mapped by mime type

class drf_yasg.openapi.**Responses**(*responses*, *default=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Describes the expected responses of an *Operation*.

Parameters

- **responses** (*dict[(str,int),Response]*) – mapping of status code to response definition
- **default** (Response) – description of the response structure to expect if another status code is returned

class drf_yasg.openapi.**Schema**(*description=None*, *required=None*, *type=None*, *properties=None*, *additional_properties=None*, *format=None*, *enum=None*, *pattern=None*, *items=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Describes a complex object accepted as parameter or returned as a response.

Parameters

- **description** – schema description
- **required** (*list[str]*) – list of requried property names
- **type** (*str*) – value type; required
- **properties** (*list[Schema,SchemaRef]*) – object properties; required if *type* is `object`
- **additional_properties** (*bool,Schema,SchemaRef*) – allow wildcard properties not listed in *properties*
- **format** (*str*) – value format, see OpenAPI spec
- **enum** (*list*) – restrict possible values
- **pattern** (*str*) – pattern if type is `string`
- **items** (Schema,SchemaRef) – only valid if *type* is `array`

OR_REF = (<class 'drf_yasg.openapi.Schema'>, <class 'drf_yasg.openapi.SchemaRef'>)

class drf_yasg.openapi.**SchemaRef**(*resolver*, *schema_name*)
Bases: *drf_yasg.openapi._Ref*

Adds a reference to a named Schema defined in the `#/definitions/` object.

Parameters

- **resolver** (ReferenceResolver) – component resolver which must contain the definition
- **schema_name** (*str*) – schema name

class drf_yasg.openapi.**Swagger**(*info=None*, *_url=None*, *_version=None*, *paths=None*, *definitions=None*, ***extra*)
Bases: *drf_yasg.openapi.SwaggerDict*

Root Swagger object.

Parameters

- **info** (Info) – info object
- **_url** (*str*) – URL used for guessing the API host, scheme and basepath
- **_version** (*str*) – version string to override Info
- **paths** (Paths) – paths object

- **definitions** (*dict[str, Schema]*) – named models

class drf_yasg.openapi.**SwaggerDict**(***attrs*)
Bases: collections.OrderedDict

A particular type of OrderedDict, which maps all attribute accesses to dict lookups using *make_swagger_name()*. Attribute names starting with _ are set on the object as-is and are not included in the specification output.

Used as a base class for all Swagger helper models.

static _as_odict(*obj*)

_insert_extras__()
From an ordering perspective, it is desired that extra attributes such as vendor extensions stay at the bottom of the object. However, python2.7's OrderdDict craps out if you try to insert into it before calling init. This means that subclasses must call super().__init__ as the first statement of their own __init__, which would result in the extra attributes being added first. For this reason, we defer the insertion of the attributes and require that subclasses call ._insert_extras__ at the end of their __init__ method.

as_odict()

class drf_yasg.openapi.**_Ref**(*resolver*, *name*, *scope*, *expected_type*)
Bases: *drf_yasg.openapi.SwaggerDict*

Base class for all reference types. A reference object has only one property, $ref, which must be a JSON reference to a valid object in the specification, e.g. #/definitions/Article to refer to an article model.

Parameters

- **resolver** (ReferenceResolver) – component resolver which must contain the referneced object
- **name** (*str*) – referenced object name, e.g. "Article"
- **scope** (*str*) – reference scope, e.g. "definitions"
- **expected_type** (*type[SwaggerDict]*) – the expected type that will be asserted on the object found in resolver

drf_yasg.openapi.**make_swagger_name**(*attribute_name*)
Convert a python variable name into a Swagger spec attribute name.
In particular,

- if name starts with x_, return x-{camelCase}
- if name is ref, return $ref
- else return the name converted to camelCase, with trailing underscores stripped

Parameters **attribute_name** (*str*) – python attribute name

Returns swagger name

9.1.8 drf_yasg.renderers

class drf_yasg.renderers.**OpenAPIRenderer**
Bases: *drf_yasg.renderers._SpecRenderer*

Renders the schema as a JSON document with the application/openapi+json specific mime type.

codec_class
alias of OpenAPICodecJson

format = 'openapi'

media_type = 'application/openapi+json'

class drf_yasg.renderers.**ReDocRenderer**
Bases: *drf_yasg.renderers._UIRenderer*

Renders a ReDoc web interface for schema browisng. Also requires *OpenAPIRenderer* as an available renderer on the same view.

format = 'redoc'

template = 'drf-yasg/redoc.html'

class drf_yasg.renderers.**SwaggerJSONRenderer**
Bases: *drf_yasg.renderers._SpecRenderer*

Renders the schema as a JSON document with the generic `application/json` mime type.

codec_class
alias of `OpenAPICodecJson`

format = '.json'

media_type = 'application/json'

class drf_yasg.renderers.**SwaggerUIRenderer**
Bases: *drf_yasg.renderers._UIRenderer*

Renders a swagger-ui web interface for schema browisng. Also requires *OpenAPIRenderer* as an available renderer on the same view.

format = 'swagger'

template = 'drf-yasg/swagger-ui.html'

class drf_yasg.renderers.**SwaggerYAMLRenderer**
Bases: *drf_yasg.renderers._SpecRenderer*

Renders the schema as a YAML document.

codec_class
alias of `OpenAPICodecYaml`

format = '.yaml'

media_type = 'application/yaml'

class drf_yasg.renderers.**_SpecRenderer**
Bases: `rest_framework.renderers.BaseRenderer`

Base class for text renderers. Handles encoding and validation.

charset = None

codec_class = None

render(*data*, *media_type=None*, *renderer_context=None*)

validators = ['ssv', 'flex']

classmethod with_validators(*validators*)

class drf_yasg.renderers.**_UIRenderer**
Bases: `rest_framework.renderers.BaseRenderer`

Base class for web UI renderers. Handles loading an passing settings to the appropriate template.

charset = 'utf-8'

get_auth_urls()

get_redoc_settings()

get_swagger_ui_settings()

media_type = 'text/html'

render(*swagger*, *accepted_media_type=None*, *renderer_context=None*)

set_context(*renderer_context*, *swagger*)

template = ''

9.1.9 drf_yasg.utils

drf_yasg.utils.**find_regex**(*regex_field*)

Given a `Field`, look for a `RegexValidator` and try to extract its pattern and return it as a string.

Parameters **regex_field** (*serializers.Field*) – the field instance

Returns the extracted pattern, or `None`

Return type str

drf_yasg.utils.**is_list_view**(*path*, *method*, *view*)

Check if the given path/method appears to represent a list view (as opposed to a detail/instance view).

Parameters

- **path** (*str*) – view path
- **method** (*str*) – http method
- **view** (*APIView*) – target view

Return type bool

drf_yasg.utils.**serializer_field_to_swagger**(*field*, *swagger_object_type*, *definitions=None*, ***kwargs*)

Convert a drf Serializer or Field instance into a Swagger object.

Parameters

- **field** (*rest_framework.serializers.Field*) – the source field
- **swagger_object_type** (*type*[openapi.SwaggerDict]) – should be one of Schema, Parameter, Items
- **definitions** (ReferenceResolver) – used to serialize Schemas by reference
- **kwargs** – extra attributes for constructing the object; if swagger_object_type is Parameter, `name` and `in_` should be provided

Returns the swagger object

Return type *openapi.Parameter*, *openapi.Items*, *openapi.Schema*

drf_yasg.utils.**swagger_auto_schema**(*method=None*, *methods=None*, *auto_schema=None*, *request_body=None*, *manual_parameters=None*, *operation_description=None*, *responses=None*)

Decorate a view method to customize the *Operation* object generated from it.

method and *methods* are mutually exclusive and must only be present when decorating a view method that accepts more than one HTTP request method.

The *auto_schema* and *operation_description* arguments take precendence over view- or method-level values.

Parameters

- **method** (*str*) – for multi-method views, the http method the options should apply to
- **methods** (*list[str]*) – for multi-method views, the http methods the options should apply to
- **auto_schema** (SwaggerAutoSchema) – custom class to use for generating the Operation object
- **request_body** (Schema, SchemaRef, *Serializer*) – custom request body, or no_body. The value given here will be used as the schema property of a *Parameter* with in: 'body'.

  A Schema or SchemaRef is not valid if this request consumes form-data, because form and body parameters are mutually exclusive in an *Operation*. If you need to set custom form parameters, you can use the *manual_parameters* argument.

  If a Serializer class or instance is given, it will be automatically converted into a *Schema* used as a body *Parameter*, or into a list of form *Parameter*s, as appropriate.
- **manual_parameters** (*list[Parameter]*) – a list of manual parameters to override the automatically generated ones

  *Parameter*s are identified by their (name, in) combination, and any parameters given here will fully override automatically generated parameters if they collide.

  It is an error to supply form parameters when the request does not consume form-data.
- **operation_description** (*str*) – operation description override
- **responses** (*dict[str, (Schema, SchemaRef, Response, str, Serializer)]*) – a dict of documented manual responses keyed on response status code. If no success (2xx) response is given, one will automatically be generated from the request body and http method. If any 2xx response is given the automatic response is suppressed.
  - if a plain string is given as value, a *Response* with no body and that string as its description will be generated
  - if a *Schema*, *SchemaRef* is given, a *Response* with the schema as its body and an empty description will be generated
  - a Serializer class or instance will be converted into a *Schema* and treated as above
  - a *Response* object will be used as-is; however if its schema attribute is a Serializer, it will automatically be converted into a *Schema*

9.1.10 drf_yasg.views

class drf_yasg.views.**SchemaView**(***kwargs*)

Bases: rest_framework.views.APIView

Constructor. Called in the URLconf; can contain helpful extra keyword arguments, and other things.

_ignore_model_permissions = True

classmethod as_cached_view(*cache_timeout=0*, *cache_kwargs=None*, ***initkwargs*)

Calls .as_view() and wraps the result in a cache_page decorator. See https://docs.djangoproject.com/en/1.11/topics/cache/

Parameters

- **cache_timeout** (*int*) – same as cache_page; set to 0 for no cache
- **cache_kwargs** (*dict*) – dictionary of kwargs to be passed to cache_page
- **initkwargs** – kwargs for .as_view()

Returns a view instance

authentication_classes = [<class 'rest_framework.authentication.SessionAuthentication'

generator_class
alias of `OpenAPISchemaGenerator`

get(*request*, *version=''*, *format=None*)

permission_classes = [<class 'rest_framework.permissions.AllowAny'>]

public = False

renderer_classes = (<class 'drf_yasg.renderers.SwaggerYAMLRenderer'>, <class 'drf_yasg

schema = None

classmethod with_ui(*renderer='swagger'*, *cache_timeout=0*, *cache_kwargs=None*)
Instantiate this view with a Web UI renderer, optionally wrapped with cache_page. See https://docs.djangoproject.com/en/1.11/topics/cache/.

Parameters

- **renderer** (*str*) – UI renderer; allowed values are `swagger`, `redoc`
- **cache_timeout** (*int*) – same as cache_page; set to 0 for no cache
- **cache_kwargs** (*dict*) – dictionary of kwargs to be passed to cache_page

Returns a view instance

classmethod without_ui(*cache_timeout=0*, *cache_kwargs=None*)
Instantiate this view with just JSON and YAML renderers, optionally wrapped with cache_page. See https://docs.djangoproject.com/en/1.11/topics/cache/.

Parameters

- **cache_timeout** (*int*) – same as cache_page; set to 0 for no cache
- **cache_kwargs** (*dict*) – dictionary of kwargs to be passed to cache_page

Returns a view instance

drf_yasg.views.**deferred_never_cache**(*view_func*)
Decorator that adds headers to a response so that it will never be cached.

drf_yasg.views.**get_schema_view**(*info*, *url=None*, *patterns=None*, *urlconf=None*, *public=False*, *validators=None*, *generator_class=<class 'drf_yasg.generators.OpenAPISchemaGenerator'>*, *authentication_classes=[<class 'rest_framework.authentication.SessionAuthentication'>, <class 'rest_framework.authentication.BasicAuthentication'>]*, *permission_classes=[<class 'rest_framework.permissions.AllowAny'>]*)
Create a SchemaView class with default renderers and generators.

Parameters

- **info** (`Info`) – Required. Swagger API Info object
- **url** (*str*) – API base url; if left blank will be deduced from the location the view is served at

- **patterns** – passed to SchemaGenerator
- **urlconf** – passed to SchemaGenerator
- **public** (*bool*) – if False, includes only endpoints the current user has access to
- **validators** (*list*) – a list of validator names to apply; allowed values are `flex`, `ssv`
- **generator_class** (*type*) – schema generator class to use; should be a subclass of *OpenAPISchemaGenerator*
- **authentication_classes** (*tuple*) – authentication classes for the schema view itself
- **permission_classes** (*tuple*) – permission classes for the schema view itself

Returns SchemaView class

Return type type[*SchemaView*]

Python Module Index

Index

Symbols

A

C

D

E

F

G

H

I

K

L

M