# audoma - API Automatic Documentation Maker

***Release 0.1***

**Iteo**

**Aug 24, 2022**

# CONTENTS:

Audoma is a Python library that was designed to make Django Rest Framework documentation easier and more automatic. It's build on the top of *drf-spectacular* and extends some of its functionalities.

CHAPTER

ONE

# INSTALLATION

## 1.1 Python version

**Audoma supports and was tested for the following versions of python:**

- python 3.7
- python 3.8
- python 3.9

## 1.2 Dependencies

**These distributions will be installed automatically when installing audoma (if you don't have them installed already):**

- Django
- django-rest-framework
- drf-spectacular
- exrex - a regular expression generator
- django-phonenumber-field - a phone number field package for django
- django-macaddress - a mac address field package for django
- phonenumbers - python package for parsing phone numbers - we use it for generating examples of phone numbers
- django-filter - django package that allows to filter down a queryset based on a model's fields
- django-money - django package that allows to store money and currency values in the database
- lorem - generator for random text that looks like Latin.

## 1.3 Install and configure Audoma

Using *pip*:

```
$ pip install git+https://github.com/Iteo/audoma.git
```

After successful installation, set *AudomaAutoSchema* as a default schema for your Django Rest Framework project:

```
REST_FRAMEWORK = {
# YOUR SETTINGS
'DEFAULT_SCHEMA_CLASS': 'audoma.drf.openapi.AudomaAutoSchema',
}
```

Audoma allows you to add path prefixes that should be included in schema exclusively. All you need to do is declare a variable *SCHEMA_PATTERN_PREFIX* ` in your project's *settings.py* file and add preprocessing function *preprocess_include_path_format* as preprocessing hook in *SPECTACULAR_SETTINGS dictionary* ` as in the example below.

```
SCHEMA_PATTERN_PREFIX = 'api'

SPECTACULAR_SETTINGS = {
    ...
    'PREPROCESSING_HOOKS': ['audoma.hooks.preprocess_include_path_format'],
    # OTHER SETTINGS
}
```

CHAPTER

TWO

# USAGE

## 2.1 Viewset defined serializers

By, default Django Rest Framework provides two methods to get a serializer:
*get_serializer* and *get_serializer_class*.
Those methods check if *self* has the property called *serializer_class*
and returns it or its instance.

Audoma Extends this behavior by, first of all, enabling to
define *collect* and *result* serializer for given views, for all actions.

Example:

```
class MyViewSet(viewsets.ModelViewSet):
    serializer_class = MySerializer
    collect_serializer_class = MyCollectSerializer
    result_serializer_class = MyResultSerializer
```

It is also possible to define a custom serializer for each action in the viewset.

Example:

```
class MyViewSet(viewsets.ModelViewSet):
    create_serializer_class = MyCreateSerializer
```

In some specific cases, you may want your action to serve more than one HTTP method.
With audoma, it is possible to define serializer per action and method.

Example:

```
class MyViewSet(viewsets.ModelViewSet):
    get_list_serializer_class = MyListSerializer
    post_list_serializer_class = MyBulkCreateSerializer
```

Each action also may want to use different collect
and result serializers, it is also possible to define this on viewset.

```
class MyViewSet(viewsets.ModelViewSet):
    create_collect_serializer = MyModelCreateSerializer
    create_result_serializer = MyModelSerializer
```

Audoma also allows combining all three variables determining serializer_class usage.
You may want to have different collect/result serializers for action which serve multiple HTTP methods.

Example:

```
class MyViewSet(viewsets.ModelViewSet):
    get_list_serializer_class = MyListSerializer
    post_list_serializer_class = MyBulkCreateSerializer
```

All of the above can be defined in one viewset:

```
class MyViewSet(viewsets.ModelViewSet):
    serializer_class = MySerializer
    collect_serializer_class = MyCollectSerializer
    result_serializer_class = MyResultSerializer
    create_serializer_class = MyCreateSerializer
    get_list_serializer_class = MyListSerializerfeatures
```

In case there are multiple name conventions used, serializer will be discovered in following order:

- *{method}_{action}_{type}_serializer_class*
- *{action}_{type}_serializer_class*
- *{method}_{action}_serializer_class*
- *{action}_serializer_class*
- *common_{type}_serializer_class*
- *serializer_class*

What's important all of the serializers defined this way, will be documented properly.

## 2.2 Permissions

By default, in the *drf-spectacular* viewset permissions were not documented at all. Currently, permissions are being documented for each viewset separately.

You don't have to define anything extra, this is being handled just out of the box. The only thing it is required is to define permissions on your viewset.

Example:

```
class ExampleModelViewSet(
mixins.ActionModelMixin,
mixins.CreateModelMixin,
mixins.RetrieveModelMixin,
mixins.DestroyModelMixin,
mixins.ListModelMixin,
viewsets.GenericViewSet,
):
    permission_classes = [
        IsAuthenticated,
        ViewAndDetailPermission,
        DetailPermission,
        ViewPermission,
        AlternatePermission1 | AlternatePermission2,
    ]
    ...
```

## 2.3 Custom choices

Audoma provides a convenient method of creating choices, which allows
referring to choices by their name.

Example:

```
if body_type == BODY_TYPE_CHOICES.SEDAN:
    ...
```

To create custom choices you have to use the *make_choices* method.

```
from audoma.choices import make_choices
...

class ExampleModel(models.Model):

    EXAMPLE_CHOICES = make_choices(
        "CHOICES",
        (
```

(continues on next page)

(continued from previous page)

```
            (1, "EX_1", "example 1"),
            (2, "EX_2", "example 2"),
            (3, "EX_3", "example 3"),
        ),
    )

    ....
choices = models.IntegerField(choices=EXAMPLE_CHOICES.get_choices())
```

As you may see if, you are passing those choices into a model field you should use the *get_choices* method.
This will return the choices known from Django.

## 2.4 Filters

### 2.4.1 Default Filters

In *drf*, it's possible to define *filterset_fields* and *filterset_class*.
By default, *drf-spectacular`* supports *django-filters*. Which are being documented.
Audoma has been tested with default drfs filter backend and *DjangoFilterBackend*.
For more accurate documentation, we recommend using`DjangoFilterBackend` as the default one.
Filters and search fields are being documented out of the box automatically:

```
class CarViewSet(
mixins.ActionModelMixin,
mixins.RetrieveModelMixin,
mixins.ListModelMixin,
viewsets.GenericViewSet,
):
    queryset = Car.objects.none()
    serializer_class = CarModelSerializer

    filter_backends = [SearchFilter, df_filters.DjangoFilterBackend]

    filterset_fields = ["engine_type"]
    search_fields = ["=manufacturer", "name"]
```

It is also possible to define the *filterset* class which will also be documented
without any additional steps.

The main extension of this feature in audoma is additional enum documentation.
In *drf-spectacular*, enums are being shown only as values possible to pass to the filter.
With audoma you also get a display field of a choice, this may be useful to show display value
in a drop-down for example.

Additionally enum fields get extension value in OpenApi schema, which is not visible in redoc/swagger frontend. This value is *x-choices*, you may read about it here.

## 2.5 Validators

### 2.5.1 ExclusiveFieldsValidator

This is an additional validator, which allows for defining mutually exclusive fields in the serializer.
It validates if any of the fields have been given and if not all exclusive fields have been given.

This validator takes params:

- fields - list or a tuple of field names
- message - string message, which will replace default validator message
- required - boolean which determines if any of the fields must be given
- **message_required - a message which will be displayed if one of the fields is required,** and none has been passed

Usage is quite simple:

```
class MutuallyExclusiveExampleSerializer(serializers.Serializer):
    class Meta:

        validators = [
            ExclusiveFieldsValidator(
                fields=[
                    "example_field",
                    "second_example_field",
                ]
            ),
        ]

    example_field = serializers.CharField(required=False)
    second_example_field = serializers.CharField(required=False)
```

## 2.6 Decorators

### 2.6.1 @extend_schema_field

This decorator is a basic *drf-spectacular* decorator, but its behavior has been changed.
It allows passing the example to the field without using information about the field.
Data is not overridden, it's updated.

```
from audoma.drf.fields import FloatField

from drf_spectacular.utils import extend_schema_field

@extend_schema_field(
    field={
        "example": 10.00
    }
)
class CustomExampleFloatField(FloatField):
    ...
```

This decorator also allows passing all used by *drf-spectacular* parameters.

### 2.6.2 @audoma_action

This is one of the most complex features offered by audoma.
In fact this is an extension of action decorator, which by default is Django Rest Framewok functionality.
It also allows registering custom action for viewset.
In the case of *audoma_action*, it is also possible to define additional parameters, such as:

#### collectors

This param allows defining serializer class which will collect and process request data.
To define this, action must serve POST/PATCH or PUT method.
Collectors may be defined in a few ways:

```
@audoma_action(
    detail=False,
    methods=["post"],
    results=ExampleOneFieldSerializer,
    collectors=ExampleOneFieldSerializer,
)
```

As defined above, simply as a serializer class, which must inherit from *serializers.BaseSerializer*.

```
@audoma_action(
    detail=True,
    methods=["post"],
    collectors={"post": ExampleModelCreateSerializer},
    results={
        "post": {201: ExampleModelSerializer, 202: ExampleOneFieldSerializer}
    },
)
```

(continues on next page)

(continued from previous page)

```
def detail_action(self, request, collect_serializer, pk=None):
    ...
```

It also may be defined as a dictionary with given http methods, than
the collectors, will be used for each http method. For Example, we may define
different collectors for POST and PATCH.

```
@audoma_action(
    detail=True,
    methods=["post", "patch"],
    collectors={
        "post": ExampleModelCreateSerializer,
         "patch": ExampleModelUpdateSerializer
    },
    results={
        "post": {
            201: ExampleModelSerializer,
            202: ExampleOneFieldSerializer
        },
        "patch": {
            200: ExampleModelSerializer,
            202: ExampleOneFieldSerializer
        }
    },
)
def detail_action(self, request, collect_serializer, pk=None):
    ...
```

This parameter is optional, so you don't have to pass collectors. If collectors won't be passed, and
request method will be in *[PUT, POST, PATCH]* then by default, audoma_action fill fallback to default
*get_serializer_class* method for audoma.

**Important**

If you are using collectors it is important to remember,
that your method should tak additional kwarg *collect_serializer* which will be
validated collector instance.

### results

This param allows defining custom results for each method and each response status code.
Results may be defined in three possible forms:

```
@audoma_action(
    detail=True,
    methods=["put", "patch"],
    collectors=ExampleModelCreateSerializer,
    results=ExampleModelSerializer,
)
def example_update_action(self, request, collect_serializer, pk=None):
    ...
```

As a serializer class, which must inherit from the *serializers.BaseSerializer*.
This will be used to the serializer, returned instance

```
@audoma_action(
    detail=True,
    methods=["post"],
    collectors={"post": ExampleModelCreateSerializer},
    results={"post": {201: ExampleModelSerializer, 202: ExampleOneFieldSerializer}},
)
def detail_action(self, request, collect_serializer, pk=None):
    ...

@audoma_action(
    detail=False,
    methods=["get"],
    results={"get": {200: "This is a test view", 404: "Not found"}},
)
def non_detail_action(self, request):
    ...
```

As a dictionary with http methods and status code, where dict values, may be serializer
classes or text messages. If values will be serializers,
view should return alongside status code, an instnace which may be serialized.
If those are messages, the view should return None as an instance,
or an overriding message for a given status code.

Results param is not mandatory, if you won't pass the results
param into audoma_action, then there will be a fallback to default
*Viewset defined serializers*.

### errors

This param may be a list of classes and instances of exceptions, which are
allowed to rise in this action. Such behavior prevents rising, not defined exceptions, and allows
to document such exceptions properly in OpenApi schema.

The main difference between passing exception class and exception instance, is that
if you pass exception instance, audoma will not only check if exception
type matches, it'll also validate its content.
We presume that if you pass, the exception class, you want to accept all exceptions of this class.

In case the risen exception is not defined in audoma_action errors, there will be another
exception risen: AudomaActionException, in case the settings.DEBUG = False, this exception
will be handled silently by logging it, but the code will pass. In the case of settings.DEBUG = True,
then the exception won't be silent.

By default audoma accepts some exceptions, which are defined globally.
Those exceptions are:

- NotFound
- NotAuthenticated
- AuthenticationFailed
- ParseError
- PermissionDenied

If you want to extend this list of globally accepted exceptions, you can do it by
defining *COMMON_API_ERRORS* in your settings, for example:

```
COMMON_API_ERRORS = [
    myexceptions.SomeException
]
```

### ignore_view_collectors

Boolean variable which tells if audoma_action should fallback to
default way of retrieving collector from view, if the collector has not been passed
and action use method which allows collecting serializer usage.

## 2.7 Examples

### 2.7.1 Define example for field

Above we described *@extend_schema_field* decorator which allows defining example for field.
For all fields defined in audoma, there are being examples generated automatically,
but you may also pass your example as a field parameter.

Example:

```
class ExampleSerializer(serializers.Serializer):
    ...
    phone_number_example = serializers.PhoneNumberField(example="+48 123 456 789")
    ...
```

### 2.7.2 Define custom fields with auto-generated examples

If you want to define your field with auto example generation,
it is possible, that your field class should inherit from the base *ExampleMixin* class,
set proper example class.

```
from rest_framework import fields
from audoma.mixins import ExampleMixin
from audoma.examples import NumericExample,


class SomeExampleField(ExampleMixin, fields.Field):
    audoma_example_class = NumericExample
```

### 2.7.3 Define custom example classes

It is possible to define your custom example classes, by default audio has defined
two specific example classes inside the *audoma.examples* module:

- *NumericExample*
- *RegexExample*

And one general class: * *Example*

To define your example class, you should inherit from the *Example* class
and override the *generate_value* method

```
from audoma.examples import Example

class MyExample(Example):
    def generate_value(self):
        return "My example value"
```

## 2.8 Extra Fields

### 2.8.1 Money Field

Our money field is an extension of the *MoneyField* known from *django_money*.
This field is defined as one field in the model, but it creates two fields in the database.

It creates a separate fielfield
There is nothing complex in this field usage, simply define it in your model:

```
from audoma.django.db import models

class ExamplePerson(models.Model):
    ...
    savings = models.MoneyField(max_digits=14, decimal_places=2, default_currency="PLN")
    ...
```

### 2.8.2 PhoneNumberField

Audoma provides a *PhoneNumberField* which is an extension of the *django-phonenumber-field*. You can use it in your models straight away, just as the original PhoneNumberField, and what we added here is an automatically generated example in documentation, based on country code.

Example:

```
from audoma.django.db import models

class ExamplePerson(models.Model):
    ...
    phone_number = models.PhoneNumberField(region="GB")
    ...
```

Above will result in the following example in the documentation:

```
{
    ...
    "phone_number": "+44 20 7894 5678",
    ...
}
```

## 2.9 Serializer Field links

Audoma allows defining links for serializer fields, which values
are related to other endpoints. This is useful if you want to limit value choices to
other filtered endpoint lists.

Such link won't be visible in redoc/swagger frontend.
It'll be included in OpenApi schema as *x-choices*.

Link definition:

```
class CarModelSerializer(serializers.ModelSerializer):

    choices_options_links = {
        "manufacturer": {
            "viewname": "manufacturer_viewset-list",
            "value_field": "id",
            "display_field": "name",
        }
    }

    manufacturer = serializers.IntegerField()

    class Meta:
        model = Car
        fields = "__all__"
```

- viewname - the name of a view from which variables should be retrieved
- value_field - field name from which value should be retrieved
- display_field - field name from which display value should be retrieved

## 2.10 Schema Extensions

### 2.10.1 x-choices

This extension is being added to all fields which have limited choice to some range.
All fields which have defined choices as enum will have this included in their schema.
If the filter field is also limited to choices this also will be included.

x-choices may have two different forms.
The first one when it's just a representation of choices enum.
Then it'll be a mapping:

```
{
    "x-choices": {
        "choices": {
            "value1": "displayValue1",
            "value2": "displayValue2",
            "value3": "displayValue3",
            "value4": "displayValue4",
        }
    }
}
```

This is simplay a mapping of values to display values.
This may be useful during displaying choices in for example drop-down.

The second form of x-choices is:

```
{
    "x-choices": {
        "operationRef": "#/paths/manufacturer_viewset~1",
        "value": "$response.body#results/*/id",
        "display": "$response.body#results/*/name"
    }
}
```

This x-choices is a reference to a different endpoint.
This may be used to read limited choices from the related endpoint.
* operationRef - is a JSON pointer to ther related endpoint which should be accesible in this chema
* value - shows which field should be taken as a field value
* display - shows which field should be taken as field display value (be shown at frontend)

CHAPTER

THREE

# TESTING AND EXAMPLE APPLICATION

## 3.1 Running example application

You can easily test audoma functionalities with our example application.
From the root folder, go to *docker/* and run *docker-compose up example_app*.
You can explore the possibilities of audoma documentation maker as it shows all functionalities.

## 3.2 Running unit tests

Go to *docker/* and run *docker-compose run –rm example_app bash -c "cd audoma_examples/drf_example && python manage.py test"*

## 3.3 Modyfying Example Application

Example Application should illustrate every feature which has been introduced into audoma.
This means that any change made in the audoma code should have its example here.
This allows a better understanding of audoma features and will allow testing more cautiously.

CHAPTER

# FOUR

# OVERVIEW

## 4.1 audoma.choices

audoma.choices.**make_choices**(*name: str*, *choices_tuple: Tuple[Any, str, str]*) → audoma.choices._T

## 4.2 audoma.decorators

audoma_action.**__init__**(*collectors: Optional[Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]]] = None*, *results: Optional[Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]]] = None*, *errors: Optional[List[Union[Exception, Type[Exception]]]] = None*, *many: bool = False*, *ignore_view_collectors: bool = False*, ***kwargs*) → None

audoma_action.**__call__**(*func: Callable*) → Callable

” Call of audoma_action decorator. This is where the magic happends.

**Args:** func - decorated function

**Returns:** wrapper callable.

**class** audoma.decorators.**AudomaActionException**

**class** audoma.decorators.**AudomaArgs**(*results: Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]]*, *collectors: Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]]*, *errors: List[Union[Exception, Type[Exception]]]*, *many: bool*)

## 4.3 audoma.examples

**class** audoma.examples.**Example**(*field*, *example=<class 'audoma.examples.DEFAULT'>*)
Class that represents an example for a field. It allows to add example to the field during initialization.

**generate_value**() → Type[audoma.examples.DEFAULT]

**get_value**() → Any

**to_representation**(*value*) → Any

**class** audoma.examples.**NumericExample**(*field*, *example=<class 'audoma.examples.DEFAULT'>*)

**generate_value**() → float
Extracts information from the field and generates a random value based on min_value and max_value.

**Returns:** Random value between min_value and max_value

**class** audoma.examples.**RegexExample**(*field*, *example=<class 'audoma.examples.DEFAULT'>*)

**generate_value**() → str
Extracts information from the field and generates a random value based on field's RegexValidators.

**Returns:** Generated regex string if regex is found, otherwise returns None

## 4.4 audoma.hooks

audoma.hooks.**postprocess_common_errors_section**(*result: dict*, *request*, ***kwargs*) → dict
Postprocessing hook which adds COMMON_API_ERRORS description to the API description.

audoma.hooks.**preprocess_include_path_format**(*endpoints: Tuple[str, str, str, Callable]*, ***kwargs*) → List[Tuple[str, str, str, Callable]]
Preprocessing hook that filters {format} prefdixed paths, in case schema pattern prefix is used and {format} path params are wanted.

## 4.5 audoma.mixins

**class** audoma.mixins.**ExampleMixin**(**args*, *example=<class 'audoma.examples.DEFAULT'>*, ***kwargs*)
A mixin class that adds an example to the field in documentation by overriding *field* parameter in *_spectacular_annotation*.

**Args:**

**audoma_example_class** [Type[Example]] The class that will be used to create the example. Depends on the type of field

**audoma_example_class**
alias of *audoma.examples.Example*

**class** audoma.mixins.**NumericExampleMixin**(**args*, *example=<class 'audoma.examples.DEFAULT'>*, ***kwargs*)
A mixin class that adds an example to the field in documentation for numeric fields

**audoma_example_class**
alias of *audoma.examples.NumericExample*

**class** audoma.mixins.**RegexExampleMixin**(**args*, *example=<class 'audoma.examples.DEFAULT'>*, ***kwargs*)

A mixin class that adds an example to the field in documentation for regex fields

**audoma_example_class**

alias of *audoma.examples.RegexExample*

## 4.6 audoma.openapi

**class** audoma.openapi.**AudomaAutoSchema**

**_build_exclusive_fields_schema**(*schema: dict*, *exclusive_fields: List[str]*) → List[dict]

**_extract_action_function**(*view*) → Callable

**_extract_audoma_action_operations**(*view: django.views.generic.base.View*, *serializer_type: str*) → Union[Dict[str, drf_spectacular.utils.OpenApiResponse], Dict[str, rest_framework.serializers.BaseSerializer], Dict[str, Type[rest_framework.serializers.BaseSerializer]], Dict[str, Dict[int, rest_framework.serializers.BaseSerializer]], Dict[int, drf_spectacular.utils.OpenApiResponse]]

Extracts the audoma action operations from the view

**_get_enum_choices_for_field**(*field*)

**_get_link_choices_for_field**(*field*, *serializer*)

**_get_permissions_description**() → str

**_get_request_for_media_type**(*serializer*)

**_get_response_for_code**(*serializer*, *status_code*, *media_types=None*)

**_get_serializer**(*serializer_type: str = 'collect'*) → Union[rest_framework.serializers.BaseSerializer, Type[rest_framework.serializers.BaseSerializer]]

**_handle_permission**(*permission_class: Union[rest_framework.permissions.OperandHolder, rest_framework.permissions.SingleOperandHolder, rest_framework.permissions.BasePermission]*, *operations: list*, *current_operation: Type = <class 'rest_framework.permissions.AND'>*) → dict

**_map_serializer**(*serializer: Union[drf_spectacular.types.OpenApiTypes, rest_framework.serializers.BaseSerializer, Type[rest_framework.serializers.BaseSerializer]]*, *direction: str*, *bypass_extensions: bool = False*) → dict

**_map_serializer_field**(*field: rest_framework.fields.Field*, *direction: str*, *bypass_extensions=False*) → dict

Allows to use @extend_schema_field with *field* dict so that it gets updated instead of being overriden

**_parse_action_errors**(*action_errors*) → Dict[int, drf_spectacular.utils.OpenApiResponse]

**_parse_action_result_serializer**(*serializer: Union[Type[rest_framework.serializers.BaseSerializer], str]*, *many: bool = False*) → Union[rest_framework.serializers.BaseSerializer, str]

**_parse_action_result_serializers**(*action_serializers: Union[Dict[str, Dict[int, Union[str, Type[rest_framework.serializers.BaseSerializer]]]], Dict[str, Union[str, Type[rest_framework.serializers.BaseSerializer]]], str, Type[rest_framework.serializers.BaseSerializer]]*, *many: bool = False*) → Union[Dict[str, drf_spectacular.utils.OpenApiResponse], Dict[str, rest_framework.serializers.BaseSerializer], Dict[str, Dict[int, rest_framework.serializers.BaseSerializer]]]

**choice_link_schema_generator = <audoma.links.ChoicesOptionsLinkSchemaGenerator object>**

**get_description**() → str
override this for custom behaviour

**get_operation**(*path*, *path_regex*, *path_prefix*, *method*, *registry: drf_spectacular.plumbing.ComponentRegistry*)

**get_operation_id**()
override this for custom behaviour

**get_response_serializers**() → Union[rest_framework.serializers.BaseSerializer, Type[rest_framework.serializers.BaseSerializer]]
overrides this for custom behaviour

## 4.7 audoma.django.db.fields

Audoma Django model Fields This module contains all the fields from Django models with additional functionality. By inheriting from Audoma Mixins, an example is generated for each field (i.e. FloatField will have example generated based on field's min and max values). We can define custom example by simply passing *example* as an argument to the field.

**class** audoma.django.db.fields.**AutoField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**BigAutoField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**BigIntegerField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**BinaryField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**BooleanField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**CharField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**CommaSeparatedIntegerField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**CurrencyField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**DateField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**DateTimeField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**DecimalField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**DurationField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**EmailField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**Field**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**FilePathField**(**args*, ***kwargs*)

**class** audoma.django.db.fields.**FloatField**(**args*, ***kwargs*)

class audoma.django.db.fields.**GenericIPAddressField**(**args*, ***kwargs*)

class audoma.django.db.fields.**IPAddressField**(**args*, ***kwargs*)

class audoma.django.db.fields.**IntegerField**(**args*, ***kwargs*)

class audoma.django.db.fields.**JSONField**(**args*, ***kwargs*)

class audoma.django.db.fields.**MACAddressField**(**args*, ***kwargs*)

class audoma.django.db.fields.**MoneyField**(**args*, ***kwargs*)

**add_currency_field**(*cls*, *name*)
Adds CurrencyField instance to a model class and creates example in documentation.

**formfield**(***kwargs*)
Return a django.forms.Field instance for this field.

class audoma.django.db.fields.**NullBooleanField**(**args*, ***kwargs*)

class audoma.django.db.fields.**PhoneNumberField**(**args*, *region=None*, ***kwargs*)

class audoma.django.db.fields.**PositiveBigIntegerField**(**args*, ***kwargs*)

class audoma.django.db.fields.**PositiveIntegerField**(**args*, ***kwargs*)

class audoma.django.db.fields.**PositiveSmallIntegerField**(**args*, ***kwargs*)

class audoma.django.db.fields.**SlugField**(**args*, ***kwargs*)

class audoma.django.db.fields.**SmallAutoField**(**args*, ***kwargs*)

class audoma.django.db.fields.**SmallIntegerField**(**args*, ***kwargs*)

class audoma.django.db.fields.**TextField**(**args*, ***kwargs*)

class audoma.django.db.fields.**TimeField**(**args*, ***kwargs*)

class audoma.django.db.fields.**URLField**(**args*, ***kwargs*)

class audoma.django.db.fields.**UUIDField**(**args*, ***kwargs*)

## 4.8 audoma.django.forms.fields

class audoma.django.forms.fields.**MoneyField**(***, *max_value=None*, *min_value=None*, *max_digits=None*, *decimal_places=None*, ***kwargs*)

**prepare_value**(*value*) → decimal.Decimal

## 4.9 audoma.drf.decorators

audoma.drf.decorators.**document_and_format**(*serializer_or_field: Any*) → Callable

## 4.10 audoma.drf.fields

This module is an override for default drf's field module. Most of those fields, are providing additional example functionality, and also has defined schema type.

**class** audoma.drf.fields.**BooleanField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**CharField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**ChoiceField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**DateField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**DateTimeField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**DecimalField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**DictField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**DurationField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**EmailField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**FileField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**FilePathField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**FloatField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**HStoreField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**IPAddressField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**ImageField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**IntegerField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**JSONField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**ListField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**MACAddressField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**MoneyField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**MultipleChoiceField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**NullBooleanField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**PhoneNumberField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**ReadOnlyField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**RegexField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**SerializerMethodField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**SlugField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**TimeField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**URLField**(**args*, ***kwargs*)

**class** audoma.drf.fields.**UUIDField**(**args*, ***kwargs*)

## 4.11 audoma.drf.filters

**class** audoma.drf.filters.**DocumentedTypedChoiceFilter**(*full_choices: NamedTuple*, *parameter_name: str*, ***kwargs*)

Extended TypedChoiceFilter to generate documentation automatically

## 4.12 audoma.drf.generics

**class** audoma.drf.generics.**GenericAPIView**(***kwargs*)

Extended GenericAPIView known from rest_framework. This class extends *get_serializer* and *get_serializer_class* methods. Also provides *get_result_serializer*, which is a shourtcut for *get_serializer* with proper param.

**get_result_serializer**(**args*, ***kwargs*) → rest_framework.serializers.BaseSerializer

Shortuct for get_serializer. Simply has serializer_type set to *result*

**get_serializer**(**args*, ***kwargs*) → rest_framework.serializers.BaseSerializer

Passes additional param to *get_serializer_class*.

**kwargs:**

**serializer_type - defines if serializer is collect or result serializer.** result serializer will be used to produce response, collect to process incoming data.

**serializer_class - it is possible to pass serializer_class to get_serializer, this will** ends with returning passed serializer_class object.

**Returns:** Object of obtained serializer class.

**get_serializer_class**(*type: str = 'collect'*, *many: bool = False*) → Type[rest_framework.serializers.BaseSerializer]

Extends defuault *get_serializer_class* method. This returns proper serializer_class for current request.

**Args:** type - type of serializer to be returned, it may be collect or result serializer.

**Returns:** This returns serializer_class

## 4.13 audoma.drf.mixins

This module overwrites basic mixins provided bu django rest framework. Mixins defined here should be used instead of default drf's mixins. Those mixins should be used to allow usage of extended *get_serializer* method>

Example:

from audoma.drf import mixins from audoma.drf import viewsets

**class ExampleModelViewSet(** mixins.ActionModelMixin, mixins.CreateModelMixin, viewsets.GenericViewSet,

**):** serializer_class = ExampleModelSerializer queryset = ExampleModel.objects.all()

**class** audoma.drf.mixins.**ActionModelMixin**

**get_success_headers**(*data: dict*) → dict

**perform_action**(*request: rest_framework.request.Request*, *success_status: int = 200*, *instance: Optional[Any] = None*, *partial: bool = False*, ***kwargs*) → rest_framework.response.Response

**retrieve_instance**(*request: rest_framework.request.Request*, *instance: Optional[Any] = None*, *success_status: int = 200*, ***kwargs*) → rest_framework.response.Response

class audoma.drf.mixins.**BulkCreateModelMixin**

Either create a single or many model instances in bulk by using the Serializers `many=True` ability from Django REST >= 2.2.5. .. note:

```
This mixin uses the same method to create model instances
as ``CreateModelMixin`` because both non-bulk and bulk
requests will use ``POST`` request method.
```

**create**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

**perform_bulk_create**(*serializer: rest_framework.serializers.BaseSerializer*) → None

class audoma.drf.mixins.**BulkUpdateModelMixin**

Update model instances in bulk by using the Serializers `many=True` ability from Django REST >= 2.2.5.

**bulk_update**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

**get_object**() → Any

**partial_bulk_update**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

**perform_bulk_update**(*serializer: rest_framework.serializers.BaseSerializer*) → None

**perform_update**(*serializer: rest_framework.serializers.BaseSerializer*) → None

class audoma.drf.mixins.**CreateModelMixin**

**create**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

class audoma.drf.mixins.**DestroyModelMixin**

**destroy**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

class audoma.drf.mixins.**ListModelMixin**

**get_paginated_response**(*data: List[Dict]*) → rest_framework.response.Response

**list**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

class audoma.drf.mixins.**RetrieveModelMixin**

**retrieve**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

class audoma.drf.mixins.**UpdateModelMixin**

**update**(*request: rest_framework.request.Request*, **args*, ***kwargs*) → rest_framework.response.Response

## 4.14 audoma.drf.serializers

class audoma.drf.serializers.**BulkListSerializer**(**args*, ***kwargs*)

class audoma.drf.serializers.**ListSerializer**(**args*, ***kwargs*)

class audoma.drf.serializers.**ModelSerializer**(**args*, ***kwargs*)

Extends default ModelSerializer, modifies serializer_field_mapping (replaces some fields with audoma fields). Adds support for generating audoma example for field.

**build_standard_field**(*field_name*, *model_field*) → Tuple[Union[Type[rest_framework.fields.Field], dict]]

Adds support for mapping example from model fields to model serializer fields.

**serializer_choice_field**

alias of *audoma.drf.fields.ChoiceField*

class audoma.drf.serializers.**ResultSerializerClassMixin**

Allows to define wrap for serializer result.

class audoma.drf.serializers.**Serializer**(**args*, ***kwargs*)

audoma.drf.serializers.**result_serializer_class**(*SerializerClass: Type[rest_framework.serializers.BaseSerializer]*, *many: bool = False*) → Type[rest_framework.serializers.BaseSerializer]

Helper function which wraps the serializer result if necessary.

**Args:**

- SerializerClass - serializer class which result should be wrapped

Returns: ResultSerializer class

## 4.15 audoma.drf.validators

class audoma.drf.validators.**ExclusiveFieldsValidator**(*fields: Union[List[str], Tuple[str]]*, *message: Optional[str] = None*, *required: bool = True*, *message_required: Optional[str] = None*)

This is extra validator defined in audoma. This validator allows to define mutually exclusvie fields.

**Attributes:** fields - list or a tuple of mutually exclusive field names message - string validation error message required - boolean value, determines if fields are required message_reqiured - string message if one of fields is required and none has been passed

**Args:** fields - list or a tuple of mutually exclusive field names message - string validation error message required - boolean value, determines if fields are required message_reqiured - string message if one of fields is required and none has been passed

**message = 'The fields {field_names} are mutually exclusive arguments.'**

**message_required = 'One of the fields {field_names} is required.'**

## 4.16 audoma.drf.viewsets

**class** audoma.drf.viewsets.**AudomaPagination**

A simple page number based style that supports page numbers as query parameters.

**Note:** If this won't be used it'll cause less explicit pagination documentation. This class does not provide any additional functionality.

**Args:** page_size (int) - number of items per page - by default this is set to 25 max_page_size (int) - maximum number of items per page - by default this is set to 2000

**get_paginated_response_schema**(*schema: List[dict]*) → dict

Simple method to add pagination information to the schema.

**Args:** schema (List[dict]) - list of schema items

**Returns:** Dictionary with pagination information including examples

**max_page_size = 2000**

**page_size = 25**

**class** audoma.drf.viewsets.**GenericViewSet**(***kwargs*)

**_parse_response_data**(*response_data: List[Any]*) → List[str]

**handle_exception**(*exc: Exception*) → rest_framework.response.Response

Handle any exception that occurs, by returning an appropriate response, or re-raising the error.

**pagination_class**

alias of *audoma.drf.viewsets.AudomaPagination*

CHAPTER

FIVE

# LICENSE

MIT License

Copyright (c) 2022 iteo

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.

CHAPTER

SIX

# CHANGELOG

CHAPTER

SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a

# INDEX

## Symbols



## A

## B



## C



## D



## E



## F



## G

## R



## S



## T



## U