

---

# **Marshmallow-Mongoengine Documentation**

***Release 0.31.2***

**Emmanuel Leblond**

**Aug 24, 2023**



---

## Contents

---

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Tutorial . . . . .	3
1.2	API Reference . . . . .	6
<b>2</b>	<b>Indices and tables</b>	<b>9</b>
	<b>Python Module Index</b>	<b>11</b>
	<b>Index</b>	<b>13</b>



Mongoengine integration with the `marshmallow` (de)serialization library.



# CHAPTER 1

---

## Contents

---

***Tutorial*** A quick tutorial to start with the project

***API Reference*** The complete API documentation

## 1.1 Tutorial

Marshmallow-Mongoengine is about bringing together a Mongoengine Document with a Marshmallow Schema.

### 1.1.1 Warming up

First we need a Mongoengine Document:

```
import mongoengine as me

class Task(me.EmbeddedDocument):
    content = me.StringField(required=True)
    priority = me.IntField(default=1)

class User(me.Document):
    name = me.StringField()
    password = me.StringField(required=True)
    email = me.StringField()
    tasks = me.ListField(me.EmbeddedDocumentField(Task))
```

Great ! Now it's time for the Marshmallow Schema. To keep things DRY, we use marshmallow-mongoengine to do the mapping:

```
import marshmallow_mongoengine as ma
```

(continues on next page)

(continued from previous page)

```
class UserSchema(ma.ModelSchema):
    class Meta:
        model = User
```

Finally it's time to use our schema to load/dump documents:

```
>>> user_schema = UserSchema()
>>> u = user_schema.load({
...     "name": "John Doe", "email": "jdoe@example.com", "password": "123456",
...     "tasks": [{"content": "Find a proper password"}]})
>>> u.save()
<User: User object>
>>> u.name
"John Doe"
>>> u.tasks
[<Task: Task object>]
>>> user_schema.dump(u)
{"name": "John Doe", "email": "jdoe@example.com", "password": "123456", "tasks": [{
  ↪ "content": "Find a proper password", "priority": 1}]}
```

If the document already exists, we can update it using *update*

```
>>> u
>>> u2 = user.schema.update(u, {"name": "Jacques Faite"})
>>> u2 is u
True
>>> u2.name
"Jacques Faite"
```

---

**Note:** *required* argument in the fields is not taken into account when using *update*

---

## 1.1.2 Configuring the schema

Let say we use *user\_schema.dump* to send data to a client through HTTP. In this case, returning the *password* field seems a pretty bad idea !

We could solve this by using marshmallow's *Meta.exclude* list, but this means the field would be also excluded loading.

The solution is to use the *Model.model\_fields\_kwargs* option for customizing the field (de)serializers:

```
class UserSchemaNoPassword(ma.ModelSchema):
    class Meta:
        model = User
        model_fields_kwargs = {'password': {'load_only': True}}
```

Now consider the loading process: For the moment we directly put the password in the password field. This is not a good idea - it is also a really poor idea to use "123456" as the password ;-) - we should first hash and salt it.

To do that, we need to disable the build of the Mongoengine document by specifying *Model.model\_build\_obj*

```
class UserSchemaJSON(ma.ModelSchema):
    class Meta:
```

(continues on next page)



(continued from previous page)

```
model = User
model_build_obj = False # default is True
```

Now the schema will do all the integrity checks, but after that will stop and return a dict:

```
>>> user_schema = UserSchemaJSON()
>>> data = user_schema.load({"name": "John Doe", "email": "jdoe@example.com",
↳ "password": "123456"})
>>> data
{"name": "John Doe", "email": "jdoe@example.com", "password": "123456"}
>>> data["password"] = hash_and_salt(data["password"]) # Alter the data
>>> User(**data) # Finally build the Mongoengine document from the data
<User: User object>
```

### 1.1.3 Customizing the schema

Now let's say we want to customize the way the tasks are dumped. For example we want to return the field *priority* in a more understandable way than just a number (1 => "High", 2 => "Medium", 3 => "Will see tomorrow").

Given that we can shadow the auto-generated fields by defining our own in the schema, we only have to redefine the *property* field and we're done !

```
class UserSchemaCustomPriority(ma.ModelSchema):
    class Meta:
        model = User

    priority = ma.fields.Method(serialize="_priority_serializer", deserialize="_
↳ priority_deserializer")

    def _priority_serializer(self, obj):
        if obj.priority == 1:
            return "High"
        elif obj.priority == 2:
            return "Medium"
        else:
            return "Will do tomorrow"
```

```
>>> user_schema = UserSchemaCustomPriority()
>>> user = User(name="John Doe", email="jdoe@example.com",
...             tasks=[{"content": "Find a proper password"},
...                     {"content": "Learn to cook", "priority": 2},
...                     {"content": "Fix issues", "priority": 3}])
>>> dump = user_schema.dump(user)
>>> dump
{"name": "John Doe", "email": "jdoe@example.com", "tasks": [{"content": "Find a
↳ proper password", "priority": "High"}, {"content": "Learn to cook", "priority":
↳ "Medium"}, {"content": "Fix issues", "priority": "Will do tomorrow"}]}
```

## 1.2 API Reference

```
class marshmallow_mongoengine.ModelSchema(*, only: types.StrSequenceOrSet | None =
None, exclude: types.StrSequenceOrSet = (),
many: bool = False, context: dict | None =
None, load_only: types.StrSequenceOrSet = (), dump_only: types.StrSequenceOrSet = (), partial: bool | types.StrSequenceOrSet | None =
None, unknown: str | None = None)
```

Base class for Mongoengine model-based Schemas.

Example:

```
from marshmallow_mongoengine import ModelSchema
from mymodels import User

class UserSchema(ModelSchema):
    class Meta:
        model = User
```

### OPTIONS\_CLASS

alias of *SchemaOpts*

### update(obj, data)

Helper function to update an already existing document instead of creating a new one. :param obj: Mongoengine Document to update :param data: incoming payload to deserialize :return: an :class: UnmarshalledResult:

Example:

```
from marshmallow_mongoengine import ModelSchema
from mymodels import User

class UserSchema(ModelSchema):
    class Meta:
        model = User

def update_obj(id, payload):
    user = User.objects(id=id).first()
    result = UserSchema().update(user, payload)
    result.data is user # True
```

Note:

Given the update is done on a existing object, the required param on the fields is ignored

```
class marshmallow_mongoengine.SchemaOpts(meta, *args, **kwargs)
```

Options class for *ModelSchema*. Adds the following options:

- **model:** The Mongoengine Document model to generate the *Schema* from (required).
- **model\_fields\_kwargs:** Dict of {field: kwargs} to provide as additional argument during fields creation.
- **model\_build\_obj:** If true, :Schema load: returns a :model: objects instead of a dict (default: True).
- **model\_converter:** *ModelConverter* class to use for converting the Mongoengine Document model to marshmallow fields.

- **model\_dump\_only\_pk:** If the document autogenerate it primary\_key (default behaviour in Mongoengine), ignore it from the incoming data (default: False)
- **model\_skip\_values:** Skip the field if it contains one of the given values (default: None, [] and {})

**class** `marshmallow_mongoengine.ModelConverter`

Class that converts a mongoengine Document into a dictionary of corresponding marshmallow *Fields* <marshmallow.fields.Field>.

**exception** `marshmallow_mongoengine.ModelConversionError`

Raised when an error occurs in converting a Mongoengine construct to a marshmallow object.

`marshmallow_mongoengine.register_field_builder(mongo_field_cls, builder)`

Register a :class MetaFieldBuilder: to a given Mongoengine Field :param mongo\_field\_cls: Mongoengine Field :param build: field\_builder to register

`marshmallow_mongoengine.register_field(mongo_field_cls, marshmallow_field_cls, available_params=())`

Bind a marshmallow field to it corresponding mongoengine field :param mongo\_field\_cls: Mongoengine Field :param marshmallow\_field\_cls: Marshmallow Field :param available\_params: List of :class marshmallow\_mongoengine.conversion.params.MetaParam:

instances to import the mongoengine field config to marshmallow



## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### m

`marshmallow_mongoengine`, [6](#)





## M

`marshmallow_mongoengine` (*module*), 6  
`ModelConversionError`, 7  
`ModelConverter` (*class in marshmallow\_mongoengine*), 7  
`ModelSchema` (*class in marshmallow\_mongoengine*), 6

## O

`OPTIONS_CLASS` (*marshmallow\_mongoengine.ModelSchema attribute*), 6

## R

`register_field()` (*in module marshmallow\_mongoengine*), 7  
`register_field_builder()` (*in module marshmallow\_mongoengine*), 7

## S

`SchemaOpts` (*class in marshmallow\_mongoengine*), 6

## U

`update()` (*marshmallow\_mongoengine.ModelSchema method*), 6