yadm

Release 1.4.15

Dec 28, 2017

Contents

It's small and simple ODM for use with MongoDB.

CHAPTER 1

Requirements

YAMD support MongoDB version 3.x only. MongoDB 2.x is not supported.

Minimal version of python is 3.5.

Quick start

2.1 Create the models

```python
from datetime import datetime

import pymongo

from yadm import Database
from yadm import Document, EmbeddedDocument
from yadm import fields
from yadm.serialize import to_mongo


class User(Document):
    __collection__ = 'users'

    name = fields.StringField()
    email = fields.EmailField()


class PostLogItem(EmbeddedDocument):
    op = fields.StringField(choices=['created', 'comment_added'])
    at = fields.DatetimeField()
    data = fields.MongoMapField()


class Post(Document):
    __collection__ = 'posts'

    user = fields.ReferenceField(User)
    created_at = fields.DatetimeField(auto_now=True)
    title = fields.StringField()
    body = fields.StringField()
    log = fields.ListField(fields.EmbeddedDocumentField(PostLogItem))
```

```python
class Comment(Document):
    __collection__ = 'comments'

    user = fields.ReferenceField(User)
    created_at = fields.DatetimeField(auto_now=True)
    post = fields.ReferenceField(Post)
    text = fields.StringField()
```

All documents creates from class `Document`. You can use multiple inheritance.

`__collection__` magic attribute setups the collection name for documents of model.

2.2 Connect to database

```python
client = pymongo.MongoClient('mongodb://localhost:27017')
db = Database(client, 'blog')
```

`Database` object is a wrapper about `pymongo` or `motor` `Database`.

2.3 Create documents

2.3.1 User

```python
user = User(name='Bill', email='bill@galactic.hero')
db.insert(user)
```

Just insert document to database.

2.3.2 Post

```python
post = Post()
post.user = user
post.title = 'Small post'
post.body = 'Bla-bla-bla...'
post.log = [PostLogItem(op='created', at=datetime.utcnow())]
db.insert(post)
```

You can fill documents as above.

2.3.3 Comment the post

```python
comment = Comment()
comment.user = user
comment.post = post
comment.text = "RE: Bla-bla-bla..."
db.insert(comment)
db.update_one(post, push={
```

```
    'log': to_mongo(PostLogItem(op='comment_added',
                                at=comment.created_at,
                                data={
                                    'comment': comment.id,
                                    'user': comment.user.id,
                                }))
})
```

We add log item to post's log. This is very usefull case.

2.4 Queries

2.4.1 find

```
qs = db(Post).find({'title': {'$regex': '^S'}})
assert qs.count() > 0
```

1. `db(Post)` creates the `QuerySet` object;

2. `find` method get the raw-query and return new `QuerySet` object with updated criteria;

3. `count` method make the query to database and return value.

```
for post in qs:
    assert post.title.startswith('S')
```

`__iter__` method make the `find`-query and returns the generator of documents.

2.4.2 find_one

```
post = db(Post).find_one({'user': user.id})
```

`find_one` get the first finded document.

```
user = post.user
```

Get attribute with reference makes the query to referred collection. Warning: N+1 problem! We have a cache in `QuerySet` object and get one referred document only once for one queryset.

2.4.3 join

Yes! We have a joins! In userspace. . .

```
comments = db(Comment).find({'post': post.id}).sort(('created_at', 1))
for comment in comments.join('user'):
    print(comment.user.name, comment.text)
```

1. Create the queryset for comments;

2. In `join` we get all documents for qs, get all users with one `$in`-query and bind to documents.

2.4.4 aggregations

```python
agg = (db.aggregate(Comment)
       .match(user=user.id)
       .group(_id='post', count={'$sum': 1})
       .sort(count=-1))

for item in agg:
    print(item)
```

CHANGES

3.1 1.4.15 (2017-12-27)

- Add `projection` argument to `get_document`
- and `reload`;
- Add `Document.__default_projection__` attribute.

3.2 1.4.14 (2017-11-06)

- **Add `EnumField`** for save `enum.Enum`;
- **Add `EnumStateField`** for simple state machines based on `enum.Enum`.

3.3 1.4.13 (2017-10-31)

- **Add `QuerySet.batch_size`** method for setup batch size for cursor;
- Some minor fixes.

3.4 1.4.10 (2017-07-07)

- **`ReferenceField.from_mongo`** try to get document from primary if not found by default.

3.5 1.4.9 (2017-07-06)

- **Add `QuerySet.read_primary` method** for simple setup `pymongo.read_preference.Primary`.

3.6 1.4.4 (2017-05-17)

- Add `TimedeltaField` for stores durations;

- **Add *SimpleEmbeddedDocumentField*** for simply create embedded documents.

```
class Doc(Document):
    embedded = SimpleEmbeddedDocumentField({
        'i': IntegerField(),
        's': StringField(),
    })
```

3.7 1.4.3 (2017-05-14)

- Add *StaticField* for static data.

3.8 1.4.2 (2017-04-09)

- Additional arguments (like `write_concern`) for write operations;

- `create_fake` save the documents with write concern "majority" by default.

3.9 1.4.0 (2017-04-05)

- Drop pymongo 2 support;

- Additional options for databases and collections;

- Add `Database.get_document`;

- Add *TypedEmbeddedDocumentField*;

- **reload argument of Database.update_one <yadm.database.Database.update_one>**
  must be keyword (may be backward incompotable).

3.10 1.3.1 (2017-02-21)

- Change raw data for `Money`;

3.11 1.3.0 (2017-02-19)

- **Add currency support to Money:**

  - Totaly rewrite `Money` type. Now it is not subclass of `Decimal`;

  - Add storage for currencies: `yadm.fields.money.currency.DEFAULT_CURRENCY_STORAGE`;

3.12 1.2.1 (2017-01-19)

- Add `QuerySet.find_in` for `$in` queries with specified order;

3.13 1.2.0 (2016-12-27)

- Drop MongoDB 2.X support;
- Objects for update and remove results;
- Use Faker instead fake-factory;

3.14 1.1.4 (2016-08-20)

- **Add some features to :py:module:'Bulk <yadm.bulk>':**
    - *Bulk.update_one(document, **kw)*: method for add update one document in bulk;
    - `Bulk.find(query).update(**kw)`: update many documents by query;
    - `Bulk.find(query).upsert().update(**kw)`: upsert document;
    - `Bulk.find(query).remove(**kw)`: remove documents;

3.15 1.1.3 (2016-07-23)

- **Add `QuerySet.ids`** method for get only documents id's from queryset;
- **Add `Money.total_cents`** method and `Money.total_cents` classmethod;

3.16 1.1 (2016-04-26)

- **Add cacheing on queryset level and use it for** *ReferenceField*;
- Add mongo aggregation framework support;
- **Add `exc` argument to** `QuerySet.find_one` for raise specified exception if not found;
- **Add `multi` argument to** `QuerySet.remove`;
- Deprecate `QuerySet.find_one`
- Refactoring.

3.17 1.0 (2015-11-14)

- **Change document structure. No more bad `BaseDocument.__data__` attribute:**
    - *BaseDocument.__raw__*: raw data from mongo;
    - *BaseDocument.__cache__*: cached objects, casted with fields;

- – *BaseDocument.__changed__*: changed objects.
- **Changes api for custom fields:**
    - – Not more need create field descriptors for every field;
    - – *prepare_value* called only for setattr;
    - – *to_mongo* called only for save objects to mongo;
    - – *from_mongo* called only for load values from *BaseDocument.__raw__*;
    - – Remove *Field.default* attribute. Use *Field.get_default* method;
    - – Add *get_if_not_loaded* and *get_if_attribute_not_set* method;
    - – By default raise *NotLoadedError* if field not loaded from projection;
- **Changes in *ReferenceField*:**
    - – Raise *BrokenReference* if link is bloken;
    - – Raise *NotBindingToDatabase* if document not saved to database;
- *smart_null* keyword for *Field*;
- Fields in document must be instances (not classes!);
- Remove *ArrayContainer* and *ArrayContainerField*;
- Remove old *MapIntKeysField* and *MapObjectIdKeysField*. Use new *MapCustomKeysField*;
- Add *Database.update_one* method for run simple update query with specified document;
- Add QuerySet.distinct;
- *serialize.from_mongo* now accept *not_loaded* sequence with filed names who must mark as not loaded, *parent* and *name*;
- *serialize.to_mongo* do not call *FieldDescriptor.__set__*;
- Fakers! Subsystem for generate test objects;
- Tests now use pytest;
- And more, and more. . .

API documentation

4.1 API

API documentation

4.1.1 Database

This module for provide work with MongoDB database.

```python
import pymongo
from yadm.database import Database

from mydocs import Doc

client = pymongo.MongoClient("localhost", 27017)
db = Database(self.client, 'test')

doc = Doc()
db.insert(doc)

doc.arg = 13
db.save(doc)

qs = db.get_queryset(Doc).find({'arg': {'$gt': 10}})
for doc in qs:
    print(doc)
```

class yadm.database.**Database**(*client*, *name*)
   Main object who provide work with database.

   Parameters

   - **client** (*pymongo.Client*) – database connection
   - **name** (*str*) – database name

aggregate (*document_class*, ***, *pipeline=None*, ***collection_params*)
>   Return aggregator for use aggregation framework.

>   **Parameters**

>   - **document_class** – *yadm.documents.Document*

>   - **pipeline** (*list*) – initial pipeline

>   - ****collection_params** – params for get_collection

bulk (*document_class*, ***, *ordered=False*, *raise_on_errors=True*, ***collection_params*)
>   Return Bulk.

>   **Parameters**

>   - **document_class** (*MetaDocument*) – class of documents fo bulk

>   - **ordered** (*bool*) – create ordered bulk (default *False*)

>   - **raise_on_errors** (*bool*) – raise BulkWriteError exception if write errors (default *True*)

>   - ****collection_params** – params for get_collection

>   Context manager:

>   > **with db.bulk(Doc) as bulk:** bulk.insert(doc_1) bulk.insert(doc_2)

get_document (*document_class*, *_id*, ***, *projection=None*, *exc=None*, *read_preference=PrimaryPreferred(tag_sets=None*, *max_staleness=-1)*, ***collection_params*)
>   Get document for it _id.

>   **Parameters**

>   - **document_class** – *yadm.documents.Document*

>   - **_id** – document's _id

>   - **projection** (*dict*) – projection for query

>   - **exc** (*Exception*) – raise given exception if not found

>   - ****collection_params** – params for get_collection

>   Default ReadPreference is PrimaryPreferred.

get_queryset (*document_class*, ***, *projection=None*, *cache=None*, ***collection_params*)
>   Return queryset for document class.

>   **Parameters**

>   - **document_class** – *yadm.documents.Document*

>   - **cache** – cache for share with other querysets

>   - ****collection_params** – params for get_collection

>   This create instance of yadm.queryset.QuerySet with presetted document's collection information.

insert (*document*, ***collection_params*)
>   Insert document to database.

>   **Parameters document** (*Document*) – document instance for insert to database

>   It's bind new document to database set *_id*. :param **collection_params: params for get_collection

reload(*document*, *new_instance=False*, *, *projection=None*, *read_preference=PrimaryPreferred(tag_sets=None*, *max_staleness=-1)*, ***collection_params*)
    Reload document.

> **Parameters**
>
> - **document** ([Document](#)) – instance for reload
>
> - **new_instance** (*bool*) – if *True* return new instance of document, else change data in given document (default: *False*)
>
> - **projection** (*dict*) – projection for query
>
> - ****collection_params** – params for get_collection

remove(*document*, ***collection_params*)
    Remove document from database.

> **Parameters**
>
> - **document** ([Document](#)) – instance for remove from database
>
> - ****collection_params** – params for get_collection

save(*document*, *full=False*, *upsert=False*, ***collection_params*)
    Save document to database.

> **Parameters**
>
> - **document** ([Document](#)) – document instance for save
>
> - **full** (*bool*) – fully resave document (default: *False*)
>
> - **upsert** (*bool*) – see documentation for MongoDB's *update* (default: *False*)
>
> - ****collection_params** – params for get_collection

If document has no *_id* [insert](#) new document.

update_one(*document*, *, *reload=True*, *set=None*, *unset=None*, *inc=None*, *push=None*, *pull=None*, ***collection_params*)
    Update one document.

> **Parameters**
>
> - **document** ([Document](#)) – document instance for update
>
> - **reload** (*bool*) – if True, reload document
>
> - ****collection_params** – params for get_collection

4.1.2 Documents

Basic documents classes for build models.

```python
class User(Document):
    __collection__ = 'users'

    first_name = fields.StringField()
    last_name = fields.StringField()
    age = fields.IntegerField()
```

All fields placed in [yadm.fields](#) package.

class yadm.documents.**MetaDocument**(*cls*, *name*, *bases*, *cls_dict*)
    Metaclass for documents.

class yadm.documents.**BaseDocument**(***kwargs*)
    Base class for all documents.

    __raw__
        Dict with raw data from mongo

    __cache__
        Dict with cached objects, casted with fields

    __changed__
        Dict with changed objects

    __data__
        Deprecated! For backward compatibility only!

        Old way to storing data in documents. Now equal to *__raw__*.

    __debug_print__()
        Print debug information.

    __fake__(*values*, *faker*, *depth*)
        Fake data customizer.

class yadm.documents.**Document**(***kwargs*)
    Class for build first level documents.

    __collection__
        Name of MongoDB collection

    __default_projection__
        Default projection for querysets

    _id
        Mongo object id (bson.ObjectId)

    id
        Alias for *_id* for simply use

    __db__
        Internal attribute contain instance of *yadm.database.Database* for realize yadm.fields.
        references.ReferenceField. It bind in *yadm.database.Database* or yadm.queryset.
        QuerySet.

    __qs__
        Documents gets from this queryset

class yadm.documents.**DocumentItemMixin**
    Mixin for custom all fields values, such as *EmbeddedDocument*, *yadm.fields.containers.*
    Container.

    __parent__
        Parent object.

```
assert doc.embedded_doc.__parent__ is doc
assert doc.list[13].__parent__ is doc.list
```

    __name__

```
assert doc.list.__name__ == 'list'
assert doc.list[13].__name__ == 13
```

__db__
    Database object.

```
assert doc.f.l[0].__db__ is doc.__db__
```

__document__
    Root document.

```
assert doc.f.l[0].__document__ is doc
```

__field_name__
    Dotted field name for MongoDB opperations, like as $set, $push and other. . .

```
assert doc.f.l[0].__field_name__ == 'f.l.0'
```

__get_value__ (*document*)
    Get value from document with path to self.

__path__
    Path to root generator.

```
assert list(doc.f.l[0].__path__) == [doc.f.l[0], doc.f.l, doc.f]
```

__path_names__
    Path to root generator.

```
assert list(doc.f.l[0].__path__) == [0, 'l', 'f']
```

__qs__
    Queryset object.

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

class yadm.documents.**EmbeddedDocument**(***kwargs*)
    Class for build embedded documents.

4.1.3 Serializers and deserializers

Functions for serialize and deserialize data.

yadm.serialize.**from_mongo**(*document_class*, *data*, *not_loaded=()*, *parent=None*, *name=None*)
    Deserialize MongoDB data to document.

>   **Parameters**

>   - **document_class** – document class

>   - **data** (*dict*) – data from MongoDB

>   - **not_loaded** ([*list*]) – fields, who marked as not loaded

>   - **parent** – parent for new document

>   - **name** (*str*) – name for new document

`yadm.serialize.`**`to_mongo`**(*document*, *exclude=()*, *include=None*)
> Serialize document to MongoDB data.

> > **Parameters**

> > > - **document** (`BaseDocument`) – document for serializing

> > > - **exclude** (`list`) – exclude fields

> > > - **include** (`list`) – include only fields (all by default)

4.1.4 Queryset

class `yadm.queryset.`**`BaseQuerySet`**(*db*, *document_class*, ***, *cache=None*, *criteria=None*, *projection=None*, *sort=None*, *slice=None*, *batch_size=None*, *collection_params=None*)
> Query builder.

> > **Parameters**

> > > - **db** –

> > > - **document_class** –

> > > - **cache** –

> > > - **criteria** (*dict*) –

> > > - **projection** (*dict*) –

> > > - **sort** (`list`) –

> > > - **slice** (*slice*) –

> > > - **collection_params** (*dict*) –

`batch_size`(*batch_size*)
> Setup batch size to cursor for this queryset.

`cache`
> Queryset cache object.

`copy`(***, *cache=None*, *criteria=None*, *projection=None*, *sort=None*, *slice=None*, *batch_size=None*, *collection_params=None*)
> Copy queryset with new parameters.

> Only keywords arguments is alowed. Parameters simply replaced with given arguments.

> > **Parameters**

> > > - **cache** –

> > > - **criteria** (*dict*) –

> > > - **projection** (*dict*) –

> > > - **sort** (`list`) –

> > > - **slice** (*slice*) –

> > > - **collection_params** (*dict*) –

> > **Returns** new `yadm.queryset.QuerySet` object

fields(**fields*)

> Get only setted fields.
>
> Update projection with fields.
>
> > **Parameters fields** (`str`) –
> >
> > **Returns** new `yadm.queryset.QuerySet`

```
qs('field', 'field2')
```

fields_all()

> Clear projection.

find(*criteria=None*, *projection=None*)

> Return queryset copy with new criteria and projection.
>
> > **Parameters**
> >
> > - **criteria** (`dict`) – update queryset's criteria
> >
> > - **projection** (`dict`) – set queryset's projection
> >
> > **Returns** new `yadm.queryset.QuerySet`

```
qs({'field': {'$gt': 3}}, {'field': True})
```

read_preference(*read_preference*)

> Setup readPreference.
>
> Return new QuerySet instance.
>
> Deprecated since 1.4.0. Use *collection_params* argument in *copy*.

read_primary(*preferred=False*)

> Return queryset with setupd read concern for primary.
>
> If *preferred* argument is *True*, *PrimaryPreferred* is used else *Primary*.

sort(**sort*)

> Return queryset with sorting.
>
> > **Parameters sort** (`tuples`) – tuples with two items: *('field_name', sort_order_as_int)*.

```
qs.sort(('field_1', 1), ('field_2', -1))
```

class `yadm.queryset.`**NotFoundBehavior**

> An enumeration.

4.1.5 Bulk queries

class `yadm.bulk.`**Bulk**(*db*, *document_class*, *ordered*, *raise_on_errors*, *collection_params*)

> Bulk object.
>
> > **Parameters**
> >
> > - **db** (`Database`) – Database instance
> >
> > - **document_class** (`MetaDocument`) – document class for collection
> >
> > - **ordered** (`bool`) – create ordered bulk (default *False*)

- **raise_on_errors** (*bool*) – raise BulkWriteError exception if write errors (default *True*)

Context manager example:

> **with db.bulk(Doc, ordered=True) as bulk:** bulk.insert(doc_1) bulk.insert(doc_2) bulk.update_one(doc_3, inc={'incr_key': 1}) bulk.find({'key': 'value'}).update(set={'key': 'new_value'}) bulk.find({'key': 'new_value'}).remove()

error
> True for executed errors.

execute()
> Execute the bulk query.

> > **Returns** *BulkResult* instance

find(*query*)
> Start "find" query in bulk.

> > **Parameters query** (*dict*) –

> > **Returns** BulkQuery instance

insert(*document*)
> Add insert document to bulk.

> > **Parameters document** (*Document*) – document for insert

> > ---
> > **Warning:** This unlike *Database.insert*! Currently, it is not bind objects to database and set id.
> > ---

result
> A BulkResult instance or rise RuntimeError if not executed.

update_one(*document*, *, *set=None*, *unset=None*, *inc=None*, *push=None*, *pull=None*)
> Add update one document to bulk.

class yadm.bulk.**BulkResult**(*bulk*, *raw*)
> Object who provide result of *Bulk.execute()*.

> **n_inserted**
> > Provide *nInserted* from raw result.

> **n_modified**
> > Provide *nModified* from raw result.

> **n_removed**
> > Provide *nRemoved* from raw result.

> **n_upserted**
> > Provide *nUpserted* from raw result.

> **write_errors**
> > Provide *writeErrors* from raw result.

4.1.6 Mongo Aggregation Framework

Mongo Aggregation Framework helper.

```
cur = db.aggregate(Doc).match({'i': {'$gt': 13}}).project(a='$i').limit(8)
```

4.1.7 Join

class yadm.join.**Join**(*qs*)

Helper for build client-side joins.

```
# Doc.ref is instance of ReferenceField
qs = db(Doc).find({'k': 1})  # queryset filter
join = qs.join('ref')  # create join query in this place
for doc in join:
    print(doc.ref)  # do not create query to database
```

get_queryset(*field_name*)

Return queryset for joined objects.

join(**field_names*)

Do manual join.

4.1.8 Fields

This package contain all fields.

4.1.8.1 Base fields

Base classes for build database fields.

class yadm.fields.base.**NotLoadedError**

Raise if value marked as not loaded.

```
doc = db(Doc).fields('a').find_one()
try:
    doc.b
except NotLoadedError:
    print("raised!")
```

class yadm.fields.base.**FieldDescriptor**(*name*, *field*)

Base desctiptor for fields.

name

Name of field

field

Field instance for this desctiptor

__delete__(*instance*)

Mark document's key as not set.

__get__(*instance*, *owner*)

Get python value from document.

1. Lookup in __changed__;

2. Lookup in __cache__;

3. Lookup in __raw__:

   - if AttributeNotSet – call Field.get_if_attribute_not_set;

   - if NotLoaded – call Field.get_if_not_loaded;

   - call Field.from_mongo;

   - set __name__ and __parent__

   - save to __cache__

4. Call Field.get_default;

5. If AttributeNotSet – call Field.get_if_attribute_not_set;

6. Return value.

__set__(*instance*, *value*)

Set value to document.

1. Call Field.prepare_value for cast value;

2. Save in Document.__changed__;

3. Call Field.set_parent_changed.

class yadm.fields.base.**Field**(*smart_null=False*)

Base field for all database fields.

> **Parameters** **smart_null** (*bool*) – If it *True*, access to not exists fields return *None* instead *AttributeError* exception. You will not be able to distinguish null value from not exist. Use with care.

descriptor_class

Class of desctiptor for work with field

document_class

Class of document. Set in *contribute_to_class()*.

name

Name of field in document. Set in *contribute_to_class()*.

contribute_to_class(*document_class*, *name*)

Add field for document_class.

> **Parameters** **document_class** (*MetaDocument*) – document class for add

copy()

Return copy of field.

descriptor_class

alias of *FieldDescriptor*

from_mongo(*document*, *value*)

Convert mongo value to python value.

> **Parameters**
>
> - **document** (*BaseDocument*) – document
>
> - **value** – mongo value
>
> **Returns** python value

get_default(*document*)

Return default value.

get_fake(*document*, *faker*, *deep*)
>   Return fake data for testing.

get_if_attribute_not_set(*document*)
>   Call if key not exist in document.

get_if_not_loaded(*document*)
>   Call if field data marked as not loaded.

prepare_value(*document*, *value*)
>   The method is called when value is assigned for the attribute.

>>   **Parameters**

>>   - **document** (`BaseDocument`) – document

>>   - **value** – raw value

>>   **Returns** prepared value

>   It must be accept *value* argument and return processed (e.g. casted) analog. Also it is called once for the default value.

to_mongo(*document*, *value*)
>   Convert python value to mongo value.

>>   **Parameters**

>>   - **document** (`BaseDocument`) – document

>>   - **value** – python value

>>   **Returns** mongo value

4.1.8.2 Simple fields

Fields for basic data types.

class yadm.fields.simple.**BooleanField**(*default=<class 'yadm.markers.AttributeNotSet'>*, ***, *choices=None*, ***kwargs*)
>   Field for boolean values.

>   **type**
>>   alias of `bool`

class yadm.fields.simple.**FloatField**(*default=<class 'yadm.markers.AttributeNotSet'>*, ***, *choices=None*, ***kwargs*)
>   Field for float.

>   **type**
>>   alias of `float`

class yadm.fields.simple.**IntegerField**(*default=<class 'yadm.markers.AttributeNotSet'>*, ***, *choices=None*, ***kwargs*)
>   Field for integer.

>   **type**
>>   alias of `int`

class yadm.fields.simple.**ObjectIdField**(*default_gen=False*)
>   Field for ObjectId.

>>   **Parameters default_gen** (`bool`) – generate default value if not set

type
    alias of `ObjectId`

class `yadm.fields.simple.`**`SimpleField`**(*default=<class 'yadm.markers.AttributeNotSet'>*, *, *choices=None*, ***kwargs*)
    Base field for simple types.

    Parameters

    - **`default`** – default value

    - **`choices`** (`set`) – set of possible values

class `yadm.fields.simple.`**`StaticField`**(*data*)
    Field for static data.

class `yadm.fields.simple.`**`StringField`**(*default=<class 'yadm.markers.AttributeNotSet'>*, *, *choices=None*, ***kwargs*)
    Field for string.

    type
        alias of `str`

4.1.8.3 Datetime field

class `yadm.fields.datetime.`**`DatetimeField`**(*, *auto_now=False*, ***kwargs*)
    Field for time stamp.

        Parameters **`auto_now`** (`bool`) – datetime.now as default (default: False)

4.1.8.4 Decimal field

Field for decimal numbers

This code save to MongoDB document:

class `yadm.fields.decimal.`**`DecimalField`**(*, *context=None*, ***kwargs*)
    Field for work with `decimal.Decimal`.

    Parameters

    - **`context`** (`decimal.Context`) – context for decimal operations (default: run `decimal.getcontext()` when need)

    - **`default`** (`decimal.Decimal`) –

TODO: context in copy()

context
    Context.

        Returns `decimal.Context` for values

prepare_value(*document*, *value*)
    Cast value to `decimal.Decimal`.

4.1.8.5 Embedded documents fields

Work with embedded documents.

```python
class EDoc(EmbeddedDocument):
    i = fields.IntegerField()

class Doc(Document):
    __collection__ = 'docs'
    edoc = EmbeddedDocumentField(EDoc)

doc = Doc()
doc.edoc = EDoc()
doc.edoc.i = 13
db.insert(doc)
```

class yadm.fields.embedded.**EmbeddedDocumentField**(*embedded_document_class*, *, *auto_create=True*, ***kwargs*)

Field for embedded objects.

> **Parameters**
>
> - **embedded_document_class** ([EmbeddedDocument](#)) – class for embedded document
> - **auto_create** (*bool*) – automatic creation embedded document from access

copy()

Return copy of field.

get_if_attribute_not_set(*document*)

Call if key not exist in document.

If auto_create is True, create and return new embedded document. Else AttributeError is raised.

class yadm.fields.embedded.**SimpleEmbeddedDocumentField**(*fields*, *, *auto_create=True*, ***kwargs*)

Field for simply create embedded documents.

Usage:

> **class Doc(Document):**
>
> > embedded = SimpleEmbeddedDocumentField({ 'i': IntegerField(), 's': StringField(),
> >
> > })

class yadm.fields.embedded.**TypedEmbeddedDocumentField**(*type_field=None*, *types=None*, ***kwargs*)

Field for embedded document with variable types.

> **Parameters**
>
> - **type_field** (*str*) – name of field in embedded document for select type
> - **types** (*dict*) – map of type names to embedded document classes

4.1.8.6 Reference field

Work with references.

```python
class RDoc(Document):
    i = fields.IntegerField()

class Doc(Document):
    rdoc = fields.ReferenceField(RDoc)
```

```
rdoc = RDoc()
rdoc.i = 13
db.insert(rdoc)

doc = Doc()
doc.rdoc = rdoc
db.insert(doc)

doc = db.get_queryset(Doc).find_one(doc.id)  # reload doc
assert doc.rdoc.id == rdoc.id
assert doc.rdoc.i == 13
```

Or asynchronous:

```
rdoc = await doc.rdoc
assert rdoc.id == rdoc.id
assert rdoc.i == 13
assert doc.rdoc == rdoc.id
```

exception `yadm.fields.reference.`**`BrokenReference`**
> Raise if referrenced document is not found.

exception `yadm.fields.reference.`**`NotBindingToDatabase`**
> Raise if set ObjectId insted referenced document to new document, who not binded to database.

class `yadm.fields.reference.`**`Reference`**(*_id: bson.objectid.ObjectId, parent: yadm.documents.DocumentItemMixin, field: yadm.fields.reference.ReferenceField*)

> Reference object.
>
> This is awaitable:
>
> > doc = await reference

class `yadm.fields.reference.`**`ReferenceField`**(*reference_document_class, **kwargs*)
> Field for work with references.
>
> > **Parameters** **`reference_document_class`** – class for refered documents
>
> **`get_fake`**(*document, faker, depth*)
> > Try create referenced document.

4.1.8.7 Containers fields

Base classes for containers.

class `yadm.fields.containers.`**`Container`**(*field, parent, value*)
> Base class for containers.
>
> **`reload`**()
> > Reload all object from database.

class `yadm.fields.containers.`**`ContainerField`**(*item_field=None, *, auto_create=True, **kwargs*)
> Base class for container fields.
>
> **`container`**
> > alias of `Container`
>
> **`from_mongo`**(*document, value*)

> **get_default**(*document*)
>
> **get_default_value**()
>
> **prepare_item**(*container*, *item*, *value*)
>
> **prepare_value**(*document*, *value*)
>
> **to_mongo**(*document*, *value*)

4.1.8.8 List fields

List of objects.

```python
class Doc(Document):
    __collection__ = 'docs'
    integers = fields.ListField(fields.IntegerField)

doc = Doc()
doc.integers.append(1)
doc.integers.append(2)
assert doc.integers == [1, 2]

db.insert(doc)
doc = db.get_queryset(Doc).find_one(doc.id)  # reload

doc.integers.append(3)   # do not save
assert doc.integers == [1, 2, 3]
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.integers == [1, 2]

doc.integers.remove(2)   # do not save too
assert doc.integers == [1]
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.integers == [1, 2]

doc.integers.push(3)   # $push query
assert doc.integers == [1, 2, 3]
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.integers == [1, 2, 3]

doc.integers.pull(2)   # $pull query
assert doc.integers == [1, 3]
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.integers == [1, 3]
```

class yadm.fields.list.**List**(*field*, *parent*, *value*)
  Container for list.

  append(*item*)
    Append item to list.

        Parameters **item** – item for append

    This method does not save object!

  insert(*index*, *item*)
    Append item to list.

        Parameters

- **index** (*int*) –

- **item** – item for insert

This method does not save object!

pull (*query*, *reload=True*)
    Pull item from database.

    ### Parameters

    - **query** – query for *$pull* on this field

    - **reload** (*bool*) – automatically reload all values from database

    See *$pull* in MongoDB's *update*.

push (*item*, *reload=True*)
    Push item directly to database.

    ### Parameters

    - **item** – item for *$push*

    - **reload** (*bool*) – automatically reload all values from database

    See *$push* in MongoDB's *update*.

remove (*item*)
    Remove item from list.

    Parameters item – item for remove

    This method does not save object!

replace (*query*, *item*, *reload=True*)
    Replace list elements.

    ### Parameters

    - **query** – query for *update*. Keys of this query is relative.

    - **item** – embedded document or dict

    - **reload** (*bool*) – automatically reload all values from database

update (*query*, *values*, *reload=True*)
    Update fields in embedded documents.

    ### Parameters

    - **query** – query for *update*. Keys of this query is relative.

    - **values** – dict of new values

    - **reload** (*bool*) – automatically reload all values from database

class yadm.fields.list.**ListField** (*item_field=None*, *, *auto_create=True*, ***kwargs*)
    Field for list values.

    For example, document with list of integers:

```python
class TestDoc(Document):
    __collection__ = 'testdoc'
    li = fields.ListField(fields.IntegerField())
```

    container
        alias of *List*

4.1.8.9 Set field

Field with sets.

Similar as *yadm.fields.list*.

class yadm.fields.set.**Set**(*field*, *parent*, *value*)
> Container for set.

> **add**(*item*)
> > Append item to set.

> > > **Parameters item** – item for add

> > This method does not save object!

> **add_to_set**(*item*, *reload=True*)
> > Add item directly to database.

> > > **Parameters**

> > > > • **item** – item for *$addToSet*

> > > > • **reload**(*bool*) – automatically reload all values from database

> > See *$addToSet* in MongoDB's *update*.

> **discard**(*item*)
> > Remove item from the set if it is present.

> > > **Parameters item** – item for discard

> > This method does not save object!

> **pull**(*query*, *reload=True*)
> > Pull item from database.

> > > **Parameters**

> > > > • **query** – query for *$pull* on this field

> > > > • **reload**(*bool*) – automatically reload all values from database

> > See *$pull* in MongoDB's *update*.

> **remove**(*item*)
> > Remove item from set.

> > > **Parameters item** – item for remove

> > This method does not save object!

class yadm.fields.set.**SetField**(*item_field=None*, ***, *auto_create=True*, ***kwargs*)
> Field for set values.

> **container**
> > alias of *Set*

4.1.8.10 Map field

Map.

```python
class Doc(Document):
    __collection__ = 'docs'
    map = fields.MapField(fields.IntegerField)

doc = Doc()
doc.map['a'] = 1
doc.map['b'] = 2
assert doc.map == {'a': 1, 'b': 2}

db.insert(doc)
doc = db.get_queryset(Doc).find_one(doc.id)  # reload

doc.map['c'] = 3  # do not save
assert doc.map == {'a': 1, 'b': 2, 'c': 3}
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.map == {'a': 1, 'b': 2}

del doc.map['b']  # do not save too
assert doc.map == {'a': 1}
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.map == {'a': 1, 'b': 2}

doc.map.set('d', 3)  # $set query
assert doc.map == {'a': 1, 'b': 2, 'c': 3}
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.map == {'a': 1, 'b': 2, 'c': 3}

doc.map.unset('d', 3)  # $unset query
assert doc.map == {'a': 1, 'b': 2}
doc = db.get_queryset(Doc).find_one(doc.id)  # reload
assert doc.map == {'a': 1, 'b': 2}
```

class yadm.fields.map.**Map**(*field*, *parent*, *value*)

> Map.

> **set**(*key*, *value*, *reload=True*)
>> Set key directly in database.
>>
>>> **Parameters**
>>>
>>> - **key** – key
>>> - **value** – value for *$set*
>>
>> See *$set* in MongoDB's *set*.

> **unset**(*key*, *reload=True*)
>> Unset key directly in database.
>>
>>> **Parameters key** – key
>>
>> See *$unset* in MongoDB's *unset*.

class yadm.fields.map.**MapCustomKeysField**(*item_field*, *key_factory*, ***, *key_to_str=<class 'str'>*, *auto_create=True*, ***kwargs*)

> Field for maps with custom key type.

>> **Parameters**
>>
>> - **item_field**(*field*) –
>> - **key_factory**(*func*) – function, who return thue key from raw string key

- **key_to_str**(*func*) –

- **auto_create**(*bool*) –

class yadm.fields.map.**MapField**(*item_field=None*, *, *auto_create=True*, **kwargs*)
    Field for maps.

    container
        alias of *Map*

4.1.8.11 Geo fields

Fields for geo data

See: http://docs.mongodb.org/manual/applications/geospatial-indexes/

GeoJSON: http://geojson.org/geojson-spec.html

class yadm.fields.geo.**Geo**
    Base class for GeoJSON data.

class yadm.fields.geo.**GeoCoordinates**
    Base class for GeoJSON data with coordinates.

class yadm.fields.geo.**GeoField**(*types=[<class 'yadm.fields.geo.Point'>, <class 'yadm.fields.geo.MultiPoint'>], **kwargs*)
    Base field for GeoJSON objects.

class yadm.fields.geo.**GeoOneTypeField**(**kwargs*)
    Base field for GeoJSON objects with one acceptable type.

class yadm.fields.geo.**MultiPoint**(*points*)
    Class for GeoJSON MultiPoint objects.

    See: http://geojson.org/geojson-spec.html#id5

class yadm.fields.geo.**MultiPointField**(**kwargs*)
    Field for MultiPoint.

    type
        alias of *MultiPoint*

class yadm.fields.geo.**Point**(*longitude*, *latitude*)
    Class for GeoJSON Point objects.

    See: http://geojson.org/geojson-spec.html#id2

class yadm.fields.geo.**PointField**(**kwargs*)
    Field for Point.

    type
        alias of *Point*

y

Symbols

A

B

C

D