Mongoom Documentation

Release 0.1.1

Dan Bradham

February 16, 2014

Release v0.1.1.

Stay Pythonic while working with MongoDB. Mongoom provides a light-weight api for mapping MongoDB documents to Python objects on top of pymongo.

CHAPTER 1

Features

- Encode and Decode MongoDB documents
- Active Validation
- Document based Events
- Threaded Subscriber

CHAPTER 2

Using Mongoom is simple!

Inherit from `Document` and `EmbeddedDocument` to define a schema.

```
from mongoom import *


class User(Document):
    name = Field(basestring, required=True)
    last_name = Field(basestring, required=True)


class Comment(EmbeddedDocument):
    user = Field(User, required=True)
    text = Field(basestring, required=True)
    created = Field(datetime, default=datetime.utcnow)


class Project(Document):
    name = Field(basestring, required=True)
    user = Field(User, required=True)
    created = Field(datetime, default=datetime.utcnow)
    description = Field(basestring)
    comments = ListField(Comment)
```

Establish a connection and save some `Document` objects.

```
connect("test_db", "localhost", 27017)

edison = User(
    name="Thomas",
    last_name="Edison",
    ).save()

bulb = Project(
    name="Light Bulb",
    user=edison,
    description="Create a commercially viable light bulb.",
    ).save()

naysayer = User(
    name="Anonymous",
    last_name="Naysayer",
    ).save()
```

```
rude_comment = Comment(
    user=naysayer,
    text=("It's impossible to create a viable light bulb. Like all of"
          "Mr. Edison's ideas, this too will be proven impractical."),
    )

bulb.comments.append(rude_comment)
bulb.save()
```

Retrieve and modify a `Document`.

```
bulb = Project.find_one(name="Light Bulb")
edison = User.find_one(last_name="Edison")
rebutt = Comment(
    user=edison,
    text="I'll show you!")
bulb.comments.append(rebutt)
bulb.save()
```

Also included with Mongoom is an `Event` and `Subscriber`. `Event` objects are nothing more than a `Document` object residing in a capped collection. While `Subscriber` objects are tailable cursors awaiting data to be entered into a capped collection. Using these two objects we can easily create a simple event handling system:

```
from mongoom *

class Create(Event):
    '''Create Event'''

class EventHandler(Subscriber):
    def handle(self, document):
        print document
        print document.ref.data

connect("test_db")

fire(Event)  # Fire a blank Event to initialize capped collection

regret = Comment(
    user=User.find_one(name="naysayer"),
    text="I feel like an idiot, the light bulb turned out great."
    )
bulb = Project.find_one(name="Light Bulb")
bulb.append(regret)
bulb.save()
fire(Create, ref=idiot)

ev_handler = EventHandler("Event")
ev_handler.start()
```

For a more elaborate mongorm event-driven system check out EventSubscriber.py in examples.

CHAPTER 3

Installation

```
git clone https://github.com/danbradham/mongoom.git
cd mongoom
python setup.py install
```

CHAPTER 4

API Documentation

4.1 Interface Essentials

One function and five classes, all you need to use Mongoom.

4.1.1 Connection

connect(*database*, *host='localhost'*, *port=27017*, ***kwargs*)

Connect to a database at given host and port. Sets two global attributes, CONNECTION and DATABASE.

Parameters

- **database** – Name of database to use.
- **host** – Host address
- **port** – Host port
- **kwargs** – Extra keyword arguments for `pymongo.mongo_client.MongoClient`

Returns `pymongo.mongo_client.MongoClient` instance.

Usage:

```
c = connect("test_db", "localhost", 27017)
```

4.1.2 Documents

class Document(***data*)

A MongoDB document mapping. A `Document` schema is defined by it's class attributes referencing `Field` instances.

Parameters **data** (*Packed or unpacked dictionary*) – MongoDB document.

Usage:

```
class User(Document):
    name = Field(basestring)

frank = User(name="Frank").save()
```

cache(*_id*)

Cache a Document object by it's _id field.

classmethod collection()

Returns Keyword args used for collection creation.

data

The documents data dictionary.

Getter Returns the objects _data.

Setter Updates the objects fields based on the provided dictionary.

fields

Returns all fields from baseclasses to allow for field inheritence. Collects fields top down ensuring that fields are properly overriden by subclasses.

classmethod find(*decode=True*, ***spec*)

Find objects in a classes collection.

Parameters

- **decode** – If True, return `Document` objects.
- **spec** – Key, Value pairs to match in mongodb documents.

classmethod find_one(*decode=True*, ***spec*)

Find one object in a classes collection.

Parameters

- **decode** – If True, return `Document` object.
- **spec** – Key, Value pairs to match in mongodb documents.

classmethod generate_objects(*cursor*)

Generator that returns all documents from a pymongo cursor as their equivalent python class.

Parameters **cursor** – A `pymongo.cursor.Cursor`

classmethod get_cache(*_id*)

Returns a python object if the _id is in __cache__.

classmethod index()

Returns Keyword args used for collection index.

ref

Returns a DBRef, saves before returning if _id not in data.

remove()

Remove Document from database.

save(**args*, ***kwargs*)

Write _data dict to database.

Accepts the same parameters as `pymongo.collection.insert()` and `pymongo.collection.update()`

validate()

Ensure all required fields are in _data.

class EmbeddedDocument(***data*)

Baseclass for all embedded documents. Unlike `Document`, `EmbeddedDocument` does not:

•have a cache

•have an objectid

•have save, find or remove methods...*yet*

•map to a collection

Parameters

- **data** (*Packed or unpacked dictionary*) – MongoDB document.
- **use_data** – If provided it is assigned to the _data attribute of the new instance, ensuring that the object passed in is exactly the same object that EmbeddedDocument is acting on. Typically only passed when decoding an embedded document as in `BaseField`'s `from_dict()` method.

Usage:

```
class Comment(EmbeddedDocument):
    user = Field("Frank")
    text = Field(basestring)

my_comment = Comment(user="Frank", text="Hello there.")
```

fields
Returns all fields from baseclasses to allow for field inheritence. Collects fields top down ensuring that fields are properly overriden by subclasses.

validate()
Ensure all required fields are in data.

4.1.3 Fields

class Field(**types*, ***kwargs*)
A multipurpose field supporting python standard types. This is the go to field for basestring, boolean, int, float, dict and also supports `Document`, and `EmbeddedDocument`. `Document` objects are automatically stored as `bson.dbref.DBRef` and decoded back to `Document`. Similarly `EmbeddedDocument` objects are stored as dicts and decoded back to `EmbeddedDocument`.

Parameters

- **types** – all args are types for validation
- **default** – default values are copied to inst._data on instantiation, can be a callable.
- **required** – is the field requried?
- **name** – name of the attribute that field is assigned to. (When used in classes inheriting from Document, you don't need to set the name parameter.)

class ListField(**types*, ***kwargs*)
A `ListField`! Supports multiple types like a `Field` descriptor, and the same automatic encoding and decoding of `bson.dbref.DBRef` and `EmbeddedDocument`.

Parameters

- **types** – all args are types for validation
- **default** – default values are copied to inst._data on instantiation, can be a callable.
- **required** – is the field requried?
- **name** – name of the attribute that field is assigned to. (When used in classes inheriting from Document, you don't need to set the name parameter.)

class `ObjectIdField`(***kwargs*)
Exactly the same as :class:‘Field‘(ObjectId)

4.2 Document Based Events

Insert Event objects into a capped collection using fire.

class `Event`(***data*)
`Event` documents live in a capped mongodb collection. Allowing people to subscribe to events using a tailable cursor. Inherit from `Event` to create custom events.

Parameters

- **ref** – `Document` that the `Event` refers to.
- **user** – User that fired the `Event`.
- **created** – Date that the `Event` was fired.

4.3 Subscribers

Subscribe to a capped collection using a Subscriber.

class `Subscriber`(*collection*, **args*, ***kwargs*)
Watch a database collection by using a tailable cursor. Collection must be initialized with capped=True prior to invoking a subscriber thread.

Parameters

- **collection** – Database collection to be subscribed to.
- ***args** – standard thread arguments.
- ****kwargs** – standard thread keyword arguments.

Usage:

```
mySubscriber = Subscriber("Event")
mySubscriber.start()
```

`decode`(*doc*)
Decode event document on receiving a doc from tailable cursor. Requires all `Document` subclasses to be imported into the “__main__” module.

`handle`(*document*)
What do you want to do with the `Document`?

`run`()
Start watching a database collection.

m