TxMongo Documentation

Release 16.1.0

Alexandre Fiori, Bret Curtis

Apr 26, 2017

Contents

CHAPTER 1

What is TxMongo

TxMongo is a pure-Python Twisted MongoDB client library that implements:

- Asynchronous client driver to MongoDB

Get it from PyPI, find out what's new in the *Changelog*!

CHAPTER 2

Quick Usage Example

```
from OpenSSL import SSL
from txmongo.connection import ConnectionPool
from twisted.internet import defer, reactor, ssl

class ServerTLSContext(ssl.DefaultOpenSSLContextFactory):
    def __init__(self, *args, **kw):
        kw['sslmethod'] = SSL.TLSv1_METHOD
        ssl.DefaultOpenSSLContextFactory.__init__(self, *args, **kw)

@defer.inlineCallbacks
def example():
    tls_ctx = ServerTLSContext(privateKeyFileName='./mongodb.key',␣
↪certificateFileName='./mongodb.crt')
    mongodb_uri = "mongodb://localhost:27017"

    mongo = yield ConnectionPool(mongodb_uri, ssl_context_factory=tls_ctx)

    foo = mongo.foo  # `foo` database
    test = foo.test  # `test` collection

    # fetch some documents
    docs = yield test.find(limit=10)
    for doc in docs:
        print doc

if __name__ == '__main__':
    example().addCallback(lambda ign: reactor.stop())
    reactor.run()
```

CHAPTER 3

User's Guide

txmongo package

Submodules

txmongo.collection module

class txmongo.collection.**Collection**(*database*, *name*, *write_concern=None*, *codec_options=None*)

Bases: object

Creates new *Collection* object

Parameters

- **database** – the Database instance to get collection from
- **name** – the name of the collection to get
- **write_concern** – An instance of WriteConcern. If None, database.write_concern is used.
- **codec_options** – An instance of CodecOptions. If None, database.codec_options is used.

aggregate(*pipeline*, *full_response=False*)

bulk_write(*requests*, *ordered=True*)

codec_options

Read only access to the CodecOptions of this instance.

Use coll.with_options(codec_options=CodecOptions(...)) to change codec options.

count(*filter=None*, ***kwargs*)

Get the number of documents in this collection.

Parameters

- **filter** – argument is a query document that selects which documents to count in the collection.
- **hint** – *(keyword only)* `hint` instance specifying index to use.
- **limit** (*int*) – *(keyword only)* The maximum number of documents to count.
- **skip** (*int*) – *(keyword only)* The number of matching documents to skip before returning results.

Returns a `Deferred` that called back with a number of documents matching the criteria.

create_index(*sort_fields*, ***kwargs*)

database
The `Database` that this `Collection` is a part of.

delete_many(*filter*)

delete_one(*filter*)

distinct(*key*, *filter=None*)

drop()

drop_index(*index_identifier*)

drop_indexes()

ensure_index(*sort_fields*, ***kwargs*)

filemd5(*spec*, ***kwargs*)

find(*filter=None*, *projection=None*, *skip=0*, *limit=0*, *sort=None*, ***kwargs*)
Find documents in a collection.

Ordering, indexing hints and other query parameters can be set with *sort* argument. See `txmongo.filter` for details.

Parameters

- **filter** – MongoDB query document. To return all documents in a collection, omit this parameter or pass an empty document (`{}`). You can pass `{"key": "value"}` to select documents having `key` field equal to `"value"` or use any of MongoDB's query selectors.
- **projection** – a list of field names that should be returned for each document in the result set or a dict specifying field names to include or exclude. If *projection* is a list `_id` fields will always be returned. Use a dict form to exclude fields: `projection={"_id": False}`.
- **skip** – the number of documents to omit from the start of the result set.
- **limit** – the maximum number of documents to return. All documents are returned when *limit* is zero.
- **sort** – query filter. You can specify ordering, indexing hints and other query parameters with this argument. See `txmongo.filter` for details.

Returns an instance of `Deferred` that called back with a list with all documents found.

find_and_modify(*query=None*, *update=None*, *upsert=False*, ***kwargs*)

find_one(*filter=None*, *projection=None*, ***kwargs*)
Get a single document from the collection.

All arguments to `find()` are also valid for `find_one()`, although *limit* will be ignored.

Returns a `Deferred` that called back with single document or `None` if no matching documents is found.

find_one_and_delete(*filter*, *projection=None*, *sort=None*, ***kwargs*)

find_one_and_replace(*filter*, *replacement*, *projection=None*, *sort=None*, *upsert=False*, *return_document=ReturnDocument.BEFORE*)

find_one_and_update(*filter*, *update*, *projection=None*, *sort=None*, *upsert=False*, *return_document=ReturnDocument.BEFORE*)

find_with_cursor(*filter=None*, *projection=None*, *skip=0*, *limit=0*, *sort=None*, ***kwargs*)

Find documents in a collection and return them in one batch at a time.

Arguments are the same as for *find()*.

Returns

an instance of `Deferred` that fires with tuple of `(docs, dfr)`, where `docs` is a partial result, returned by MongoDB in a first batch and `dfr` is a `Deferred` that fires with next `(docs, dfr)`. Last result will be `([], None)`. You can iterate over the result set with code like that:

```
@defer.inlineCallbacks
def query():
    docs, dfr = yield coll.find(query, cursor=True)
    while docs:
        for doc in docs:
            do_something(doc)
        docs, dfr = yield dfr
```

full_name

Full name of this *Collection*, i.e. *db_name.collection_name*

group(*keys*, *initial*, *reduce*, *condition=None*, *finalize=None*, ***kwargs*)

index_information()

insert(*docs*, *safe=None*, *flags=0*, ***kwargs*)

Insert a document(s) into this collection.

Please consider using new-style *insert_one()* *or* *insert_many()* *methods instead.*

If document doesn't have `"_id"` field, *insert()* will generate new `ObjectId` and set it to `"_id"` field of the document.

Parameters

- **docs** – Document or a list of documents to insert into a collection.
- **safe** – `True` or `False` forces usage of respectively acknowledged or unacknowledged Write Concern. If `None`, *write_concern* is used.
- **flags** – If zero (default), inserting will stop after the first error encountered. When `flags` set to `txmongo.protocol.INSERT_CONTINUE_ON_ERROR`, MongoDB will try to insert all documents passed even if inserting some of them will fail (for example, because of duplicate `_id`). Not that *insert()* won't raise any errors when this flag is used.

Returns `Deferred` that fires with single `_id` field or a list of `_id` fields of inserted documents.

insert_many(*documents*, *ordered=True*)

Insert an iterable of documents into collection

Parameters

- **documents** – An iterable of documents to insert (`list`, `tuple`, ...)
- **ordered** – If `True` (the default) documents will be inserted on the server serially, in the order provided. If an error occurs, all remaining inserts are aborted. If `False`, documents will be inserted on the server in arbitrary order, possibly in parallel, and all document inserts will be attempted.

Returns `Deferred` that called back with `pymongo.results.InsertManyResult`

insert_one(*document*)
Insert a single document into collection

Parameters **document** – Document to insert

Returns `Deferred` that called back with `pymongo.results.InsertOneResult`

map_reduce(*map*, *reduce*, *full_response=False*, ***kwargs*)

name
Name of this *Collection* (without database name).

options()
Get the options set on this collection.

Returns `Deferred` that called back with dictionary of options and their values or with empty dict if collection doesn't exist.

remove(*spec*, *safe=None*, *single=False*, *flags=0*, ***kwargs*)

rename(*new_name*)

replace_one(*filter*, *replacement*, *upsert=False*)
Replace a single document matching the filter.

Raises

- **ValueError** – if *update* document is empty
- **ValueError** – if *update* document has fields that starts with *$* sign. This method only allows *replacing* document completely. Use *update_one()* for modifying existing document.

Parameters

- **filter** – A query that matches the document to replace.
- **replacement** – The new document to replace with.
- **upsert** – If `True`, perform an insert if no documents match the filter.

Returns deferred instance of `pymongo.results.UpdateResult`.

save(*doc*, *safe=None*, ***kwargs*)

update(*spec*, *document*, *upsert=False*, *multi=False*, *safe=None*, *flags=0*, ***kwargs*)
Update document(s) in this collection

Please consider using new-style *update_one()*, *update_many()* and *replace_one()* *methods instead.*

Raises **TypeError** – if *spec* or *document* are not instances of *dict* or *upsert* is not an instance of *bool*.

Parameters

- **spec** – query document that selects documents to be updated

- **document** – update document to be used for updating or upserting. See MongoDB Update docs for the format of this document and allowed operators.
- **upsert** – perform an upsert if `True`
- **multi** – update all documents that match *spec*, rather than just the first matching document. The default value is `False`.
- **safe** – `True` or `False` forces usage of respectively acknowledged or unacknowledged Write Concern. If `None`, *`write_concern`* is used.

Returns `Deferred` that is called back when request is sent to MongoDB or confirmed by MongoDB (depending on selected Write Concern).

update_many (*filter*, *update*, *upsert=False*)
Update one or more documents that match the filter.

Raises

- **ValueError** – if *update* document is empty.
- **ValueError** – if *update* document has fields that don't start with *$* sign. This method only allows *modification* of document (with *$set*, *$inc*, etc.), not *replacing* it. For replacing use *`replace_one()`* instead.

Parameters

- **filter** – A query that matches the documents to update.
- **update** – update document to be used for updating or upserting. See MongoDB Update docs for allowed operators.
- **upsert** – If `True`, perform an insert if no documents match the *filter*.

Returns deferred instance of `pymongo.results.UpdateResult`.

update_one (*filter*, *update*, *upsert=False*)
Update a single document matching the filter.

Raises

- **ValueError** – if *update* document is empty.
- **ValueError** – if *update* document has any fields that don't start with *$* sign. This method only allows *modification* of document (with *$set*, *$inc*, etc.), not *replacing* it. For replacing use *`replace_one()`* instead.

Parameters

- **filter** – A query that matches the document to update.
- **update** – update document to be used for updating or upserting. See MongoDB Update docs for allowed operators.
- **upsert** – If `True`, perform an insert if no documents match the *filter*.

Returns deferred instance of `pymongo.results.UpdateResult`.

with_options (*, *write_concern=None*, *codec_options=None*)
Get a clone of collection changing the specified settings.

Parameters

- **write_concern** – *(keyword only)* new `WriteConcern` to use.
- **codec_options** – *(keyword only)* new `CodecOptions` to use.

write_concern
Read only access to the `WriteConcern` of this instance.

Use `coll.with_options(write_concern=WriteConcern(...))` to change the Write Concern.

txmongo.connection module

class `txmongo.connection.`**ConnectionPool**(*uri='mongodb://127.0.0.1:27017'*, *pool_size=1*, *ssl_context_factory=None*, *ping_interval=10*, *ping_timeout=10*, ***kwargs*)

Bases: `object`

authenticate(*database*, *username*, *password*, *mechanism='DEFAULT'*)

codec_options

disconnect()

drop_database(*name_or_database*)

get_default_database()

getprotocol()

getprotocols()

uri

write_concern

class `txmongo.connection.`**MongoConnection**(*host='127.0.0.1'*, *port=27017*, *pool_size=1*, ***kwargs*)

Bases: *txmongo.connection.ConnectionPool*

`txmongo.connection.`**MongoConnectionPool**
alias of *MongoConnection*

`txmongo.connection.`**lazyMongoConnection**
alias of *MongoConnection*

`txmongo.connection.`**lazyMongoConnectionPool**
alias of *MongoConnection*

txmongo.database module

class `txmongo.database.`**Database**(*factory*, *database_name*, *write_concern=None*, *codec_options=None*)

Bases: `object`

authenticate(*name*, *password*, *mechanism='DEFAULT'*)
Send an authentication command for this database. mostly stolen from pymongo

codec_options

collection_names()

command(*command*, *value=1*, *check=True*, *allowable_errors=None*, *codec_options=DEFAULT_CODEC_OPTIONS*)

connection

create_collection(*name*, *options=None*, *write_concern=None*, *codec_options=None*, ***kwargs*)

drop_collection(*name_or_collection*)

name

write_concern

txmongo.filter module

txmongo.filter.**ASCENDING**(*keys*)
Ascending sort order

txmongo.filter.**DESCENDING**(*keys*)
Descending sort order

txmongo.filter.**GEO2D**(*keys*)
Two-dimensional geospatial index http://www.mongodb.org/display/DOCS/Geospatial+Indexing

txmongo.filter.**GEO2DSPHERE**(*keys*)
Two-dimensional geospatial index http://www.mongodb.org/display/DOCS/Geospatial+Indexing

txmongo.filter.**GEOHAYSTACK**(*keys*)
Bucket-based geospatial index http://www.mongodb.org/display/DOCS/Geospatial+Haystack+Indexing

txmongo.filter.**TEXT**(*keys*)
Text-based index https://docs.mongodb.com/manual/core/index-text/

class txmongo.filter.**comment**(*comment*)
Bases: txmongo.filter._QueryFilter

class txmongo.filter.**explain**
Bases: txmongo.filter._QueryFilter

Returns an explain plan for the query.

class txmongo.filter.**hint**(*index_list_or_name*)
Bases: txmongo.filter._QueryFilter

Adds a *hint*, telling Mongo the proper index to use for the query.

class txmongo.filter.**snapshot**
Bases: txmongo.filter._QueryFilter

class txmongo.filter.**sort**(*key_list*)
Bases: txmongo.filter._QueryFilter

Sorts the results of a query.

txmongo.gridfs module

txmongo.protocol module

Low level connection to Mongo.

This module contains the wire protocol implementation for txmongo. The various constants from the protocol are available as constants.

This implementation requires pymongo so that as much of the implementation can be shared. This includes BSON encoding and decoding as well as Exception types, when applicable.

class txmongo.protocol.**Delete**
Bases: *txmongo.protocol.Delete*

class txmongo.protocol.**Getmore**
Bases: *txmongo.protocol.Getmore*

class txmongo.protocol.**Insert**
Bases: *txmongo.protocol.Insert*

class txmongo.protocol.**KillCursors**
Bases: *txmongo.protocol.KillCursors*

exception txmongo.protocol.**MongoAuthenticationError**
Bases: Exception

class txmongo.protocol.**MongoClientProtocol**
Bases: twisted.internet.protocol.Protocol

get_request_id()

send(*request*)

send_DELETE(*request*)

send_GETMORE(*request*)

send_INSERT(*request*)

send_KILL_CURSORS(*request*)

send_MSG(*request*)

send_QUERY(*request*)

send_REPLY(*request*)

send_UPDATE(*request*)

class txmongo.protocol.**MongoDecoder**
Bases: object

dataBuffer = None

static decode(*message_data*)

feed(*data*)

next()

class txmongo.protocol.**MongoProtocol**
Bases: *txmongo.protocol.MongoServerProtocol*, *txmongo.protocol.MongoClientProtocol*

authenticate(*database_name*, *username*, *password*, *mechanism*)

authenticate_mongo_cr(*database_name*, *username*, *password*)

authenticate_mongo_x509(*database_name*, *username*, *password*)

authenticate_scram_sha1(*database_name*, *username*, *password*)

connectionLost(*reason=<twisted.python.failure.Failure twisted.internet.error.ConnectionDone: Connection was closed cleanly.>*)

connectionMade()

connectionReady()

fail(*reason*)

get_last_error(*db*, ***options*)

handle_REPLY(*request*)

inflight()

max_wire_version = None

min_wire_version = None

send_GETMORE(*request*)

send_QUERY(*request*)

set_wire_versions(*min_wire_version*, *max_wire_version*)

class txmongo.protocol.**MongoServerProtocol**
Bases: twisted.internet.protocol.Protocol

dataReceived(*data*)

handle(*request*)

handle_DELETE(*request*)

handle_GETMORE(*request*)

handle_INSERT(*request*)

handle_KILL_CURSORS(*request*)

handle_MSG(*request*)

handle_QUERY(*request*)

handle_REPLY(*request*)

handle_UPDATE(*request*)

class txmongo.protocol.**Msg**(*len*, *request_id*, *response_to*, *opcode*, *message*)
Bases: tuple

len
Alias for field number 0

message
Alias for field number 4

opcode
Alias for field number 3

request_id
Alias for field number 1

response_to
Alias for field number 2

class txmongo.protocol.**Query**
Bases: *txmongo.protocol.Query*

class txmongo.protocol.**Reply**
Bases: *txmongo.protocol.Reply*

class txmongo.protocol.**Update**
Bases: *txmongo.protocol.Update*

Module contents

txmongo._gridfs package

Submodules

txmongo._gridfs.errors module

Exceptions raised by the `gridfs` package

exception `txmongo._gridfs.errors.`**`CorruptGridFile`**
Bases: *txmongo._gridfs.errors.GridFSError*

Raised when a file in `GridFS` is malformed.

exception `txmongo._gridfs.errors.`**`GridFSError`**
Bases: `Exception`

Base class for all GridFS exceptions.

New in version 1.5.

exception `txmongo._gridfs.errors.`**`NoFile`**
Bases: *txmongo._gridfs.errors.GridFSError*

Raised when trying to read from a non-existent file.

New in version 1.6.

exception `txmongo._gridfs.errors.`**`UnsupportedAPI`**
Bases: *txmongo._gridfs.errors.GridFSError*

Raised when trying to use the old GridFS API.

In version 1.6 of the PyMongo distribution there were backwards incompatible changes to the GridFS API. Upgrading shouldn't be difficult, but the old API is no longer supported (with no deprecation period). This exception will be raised when attempting to use unsupported constructs from the old API.

New in version 1.6.

txmongo._gridfs.grid_file module

Tools for representing files stored in GridFS.

class `txmongo._gridfs.grid_file.`**`GridIn`**(*root_collection*, ***kwargs*)
Bases: `object`

Class to write data to GridFS.

`chunk_size`
Chunk size for this file.

This attribute is read-only.

`close`()
Flush the file and close it.

A closed file cannot be written any more. Calling *close()* more than once is allowed.

`closed`
Is this file closed?

content_type
Mime-type for this file.

This attribute can only be set before *close()* has been called.

filename
Name of this file.

This attribute can only be set before *close()* has been called.

length
Length (in bytes) of this file.

This attribute is read-only and can only be read after *close()* has been called.

md5
MD5 of the contents of this file (generated on the server).

This attribute is read-only and can only be read after *close()* has been called.

upload_date
Date that this file was uploaded.

This attribute is read-only and can only be read after *close()* has been called.

write(*data*)
Write data to the file. There is no return value.

data can be either a string of bytes or a file-like object (implementing `read()`).

Due to buffering, the data may not actually be written to the database until the *close()* method is called. Raises `ValueError` if this file is already closed. Raises `TypeError` if *data* is not an instance of `str` or a file-like object.

Parameters

- *data*: string of bytes or file-like object to be written to the file

writelines(*sequence*)
Write a sequence of strings to the file.

Does not add separators.

class txmongo._gridfs.grid_file.**GridOut**(*root_collection*, *doc*)
Bases: `object`

Class to read data out of GridFS.

aliases
List of aliases for this file.

This attribute is read-only.

chunk_size
Chunk size for this file.

This attribute is read-only.

close()

content_type
Mime-type for this file.

This attribute is read-only.

length
Length (in bytes) of this file.

This attribute is read-only.

md5
MD5 of the contents of this file (generated on the server).

This attribute is read-only.

metadata
Metadata attached to this file.

This attribute is read-only.

name
Name of this file.

This attribute is read-only.

read(*size=-1*)
Read at most *size* bytes from the file (less if there isn't enough data).

The bytes are returned as an instance of `str`. If *size* is negative or omitted all data is read.

Parameters

- *size* (optional): the number of bytes to read

seek(*pos*, *whence=0*)
Set the current position of this file.

Parameters

- *pos*: the position (or offset if using relative positioning) to seek to
- *whence* (optional): where to seek from. `os.SEEK_SET` (`0`) for absolute file positioning, `os.SEEK_CUR` (`1`) to seek relative to the current position, `os.SEEK_END` (`2`) to seek relative to the file's end.

tell()
Return the current position of this file.

upload_date
Date that this file was first uploaded.

This attribute is read-only.

class txmongo._gridfs.grid_file.**GridOutIterator**(*grid_out*, *chunks*)
Bases: object

next()

Module contents

GridFS is a specification for storing large objects in Mongo.

The `gridfs` package is an implementation of GridFS on top of `pymongo`, exposing a file-like interface.

class txmongo._gridfs.**GridFS**(*database*, *collection='fs'*)
Bases: object

An instance of GridFS on top of a single Database.

count (*filename*)
Count the number of versions of a given file. Returns an integer number of versions of the file in GridFS whose filename matches *filename*, or raises NoFile if the file doesn't exist. :Parameters:

•*filename*: `"filename"` of the file to get version count of

delete (*file_id*)
Delete a file from GridFS by `"_id"`.

Removes all data belonging to the file with `"_id"`: *file_id*.

> **Warning:** Any processes/threads reading from the file while this method is executing will likely see an invalid/corrupt file. Care should be taken to avoid concurrent reads to a file while it is being deleted.

Parameters

- *file_id*: `"_id"` of the file to delete

New in version 1.6.

get (*file_id*)
Get a file from GridFS by `"_id"`.

Returns an instance of `GridOut`, which provides a file-like interface for reading.

Parameters

- *file_id*: `"_id"` of the file to get

New in version 1.6.

get_last_version (*filename*)
Get a file from GridFS by `"filename"`.

Returns the most recently uploaded file in GridFS with the name *filename* as an instance of `GridOut`. Raises `NoFile` if no such file exists.

An index on `{filename: 1, uploadDate: -1}` will automatically be created when this method is called the first time.

Parameters

- *filename*: `"filename"` of the file to get

New in version 1.6.

get_version (*filename=None*, *version=-1*)
Get a file from GridFS by `"filename"`. Returns a version of the file in GridFS whose filename matches *filename* and whose metadata fields match the supplied keyword arguments, as an instance of `GridOut`. Version numbering is a convenience atop the GridFS API provided by MongoDB. If more than one file matches the query (either by *filename* alone, by metadata fields, or by a combination of both), then version `-1` will be the most recently uploaded matching file, `-2` the second most recently uploaded, etc. Version `0` will be the first version uploaded, `1` the second version, etc. So if three versions have been uploaded, then version `0` is the same as version `-3`, version `1` is the same as version `-2`, and version `2` is the same as version `-1`. Note that searching by random (unindexed) meta data is not supported here. Raises `NoFile` if no such version of that file exists. :Parameters:

•*filename*: `"filename"` of the file to get, or *None*

•*version* (optional): version of the file to get (defaults to -1, the most recent version uploaded)

indexes_created()
Returns a defer on the creation of this GridFS instance's indexes

list()
List the names of all files stored in this instance of *GridFS*.

Changed in version 1.6: Removed the *collection* argument.

new_file(***kwargs*)
Create a new file in GridFS.

Returns a new `GridIn` instance to which data can be written. Any keyword arguments will be passed through to `GridIn()`.

Parameters

- ***kwargs* (optional): keyword arguments for file creation

New in version 1.6.

put(*data*, ***kwargs*)
Put data in GridFS as a new file.

Equivalent to doing:

```
>>> f = new_file(**kwargs)
>>> try:
>>>     f.write(data)
>>> finally:
>>>     f.close()
```

data can be either an instance of `str` or a file-like object providing a `read()` method. Any keyword arguments will be passed through to the created file - see `GridIn()` for possible arguments. Returns the `"_id"` of the created file.

Parameters

- *data*: data to be written as a file.
- ***kwargs* (optional): keyword arguments for file creation

New in version 1.6.

CHAPTER 4

Meta

Changelog

Release 17.1.0 (UNRELEASED)

Bugfixes

- Memory leak fixed in *Collection.bulk_write()*
- Use authSource as auth database if specify in connect uri

Release 16.3.0 (2016-11-25)

Features

- Full-text indexes can be used with new `filter.TEXT()`
- Client authentication by X509 certificates. Use your client certificate when connecting to MongoDB and then call `Database.authenticate` with certificate subject as username, empty password and `mechanism="MONGODB-X509"`.
- `get_version()` to approximate the behaviour of get_version in PyMongo. One noteable exception is the omission of searching by random (unindexed) meta-data which should be considered a bad idea as it may create *very* variable conditions in terms of loading and timing.
- New `ConnectionPool.drop_database()` method for easy and convenient destruction of all your precious data.
- `count()` to return the number of versions of any given file in GridFS.

API Changes

- `find()`, `find_one()`, `find_with_cursor()`, `count()` and `distinct()` signatures changed to more closely match PyMongo's counterparts. New signatures are:
  - `find(filter=None, projection=None, skip=0, limit=0, sort=None, **kwargs)`
  - `find_with_cursor(filter=None, projection=None, skip=0, limit=0, sort=None, **kwargs)`
  - `find_one(filter=None, projection=None, **kwargs)`
  - `count(filter=None, **kwargs)`
  - `distinct(key, filter=None, **kwargs)`

  Old signatures are now deprecated and will be supported in this and one subsequent releases. After that only new signatures will be valid.
- `cursor` argument to `find()` is deprecated. Please use `find_with_cursor()` directly if you need to iterate over results by batches. `cursor` will be supported in this and one subsequent releases.
- `as_class` argument to `find()`, `find_with_cursor()` and `find_one()` is deprecated. Please use ‘‘collection.with_options(codec_options=CodecOptions(document_class=...)).find()‘ instead. It is lengthty, but it is more generic and this is how you do it with current PyMongo.
- `Database.command()` now takes `codec_options` argument.
- `watchdog_interval` and `watchdog_timeout` arguments of `ConnectionPool` renamed to `ping_interval` and `ping_timeout` correspondingly along with internal change of connection aliveness checking mechanism.

Bugfixes

- `GridFS.get_last_version()` was creating redundant index

Release 16.2.0 (2016-10-02)

Features

- `Collection.bulk_write()` that maches behavior of corresponding PyMongo's method. It accepts an iterable of `InsertOne`, `UpdateOne`, ... from `pymongo.operations`, packs them into batches and returns aggregated response from MongoDB.
- `codec_options` properties for `ConnectionPool`, `Database` and `Collection`. `Collection.with_options(codec_options=CodecOptions(document_class=...))` is now preferred over `Collection.find(..., as_class=...)`.

Bugfixes

- Fixed bug in *find()* that can cause undefined ordering of the results when sorting on multiple fields is requested.

Release 16.1.0 (2016-06-15)

API Changes

- `insert_many()` raises `BulkWriteError` instead `WriteError`/`DuplicateKeyError` to match PyMongo's behavior. This is also allows to extract multiple duplicate key errors from exception object when `insert_many` is used with `ordered=False`.
- `fields` parameter removed for `Collection.count()`.
- `ConnectionPool` has two new parameters: `watchdog_interval` which is how many seconds before testing a connection to see if it is stale, and ‘‘watchdog_timeout‘‘is how long the check takes before dropping the stale connection and try to reconnect.

Features

- Stale connections are now dropped after failing to contact mongodb longer than `watchdog_timeout`.
- `insert_many()` is now able to insert more than 1000 documents and more than 16Mb of documents at once.
- GridFS's default `chunkSize` changed to 255kB, to avoid the overhead with usePowerOf2Sizes option.
- Add `GridFS.indexes_created` to obtain a defer on the creation of the current GridFS instance's indexes
- GridFS create indexes for the `files` collection in addition to the `chunks` one

Release 16.0.1 (2016-03-03)

Features

- Make existing logging more verbose, indicate that it is TxMongo raising the error or sending the message.
- Add additional logging.

Release 16.0.0 (2016-02-25)

Bugfixes

- Memory leak fixed in *find_with_cursor* that affected almost all query methods

Release 15.3.1 (2015-10-26)

API Changes

- `connection.ConnectionPool` exposes *max_delay* which is used to set the maximum number of seconds between connection attempts. The default is set to 60.

Features

- Updated and simplified setup.py, enforce minimal versions of PyMongo and Twisted necessary to install TxMongo.

Release 15.3.0 (2015-09-29)

API Changes

- `NotMaster` instead of `AutoReconnect` error will be returned when a call can be safely retried.

Features

- Added `deadline` to `collection` methods, this will raise a `DeadlineExceeded` when the deadline, a unix timestamp in seconds, is exceeded. This happens only in methods with `getprotocol()` and methods that reference them.
- Added `timeout` to `collection` methods, this will raise a `TimeoutExceeded` when the timeout, in seconds, is exceeded. This happens only in methods with `getprotocol()` and methods that reference them.

Bugfixes

- Fixed `collection.count()` to return an int instead of float, this matches how count in with PyMongo.

Release 15.2.2 (2015-09-15)

Bugfix release to handle str assert that wasn't passing unicode properly in python 2.6, used Twisted compat library StringType.

Release 15.2.1 (2015-09-07)

Bugfix release to handle uncaught exceptions in logging and to remove support for python 2.6 and since it was removed in latest Twisted.

Release 15.2 (2015-09-05)

This release makes TxMongo fully Python3 compatible and has an API change that breaks older TxMongo compatibility by bringing it inline with PyMongo.

API Changes

- `txmongo.dbref` removed. Use `bson.dbref` instead. **Incompatibility note:** `bson.dbref.DBRef` takes collection name as string while `txmongo.dbref.DBRef` was able to accept `Collection` instance. Please use `collection.name` instead.
- Added `timeout` parameter for `connection.ConnectionPool` that can passed on to Twisted's `connectTCP` and `connectSSL` methods.

Features

- `name`, `full_name` and `database` properties of `Collection`
- Python3 compatible.

Release 15.1 (2015-06-08)

This is a major release in that while increasing code coverage to 95% (see https://coveralls.io/builds/2749499), we've also caught several bugs, added features and changed functionality to be more inline with PyMongo.

This is no small thanks to travis-ci and coveralls while using tox to cover all iterations that we support.

We can officially say that we are Python 2.6, 2.7 and PyPy compatible.

API Changes

- **TxMongo now requires PyMongo 3.x**, if you need PyMongo 2.x support, please use 15.0, otherwise it is highgly recommend to use PyMongo 3.x which still support MongoDB 2.6.
- Better handling of replica-sets, we now raise an `autoreconnect` when master is unreachable.
- Changed the behaviour of `find_one` to return `None` instead of an empty dict `{}` when no result is found.
- New-style query methods: `insert_one/many`, `update_one/many`, `delete_one/many`, `replace_one` and `find_one_and_update/replace`

Features

- Added `db.command` function, just like PyMongo.
- Added support for named indexes in `filter`.
- `insert()`, `update()`, `save()` and `remove()` now support write-concern options via named args: `w`, `wtimeout`, `j`, `fsync`. `safe` argument is still supported for backward compatibility.
- Default write-concern can be specified for `Connection` using named arguments in constructor or by URI options.
- Write-concern options can also be set for `Database` and `Collection` with `write_concern` named argument of their constructors. In this case write-concern is specified by instance of `pymongo.write_concern.WriteConcern`
- `txmongo.protocol.INSERT_CONTINUE_ON_ERROR` flag defined for using with `insert()`
- Replaced all traditional deferred callbacks (and errbacks) to use @defer.inlineCallbacks

Bugfixes

- Fixed typo in `map_reduce()` when returning results.
- Fixed hang in `create_collection()` in case of error.
- Fixed typo in `rename()` that wasn't using the right factory.
- Fixed exception in `drop_index` that was being thrown when dropping a non-existent collection. This makes the function idempotent.
- Fixed URI prefixing when "mongodb://" is not present in URI string in `connection`.
- Fixed fail-over when using replica-sets in `connection`. It now raises `autoreconnect` when there is a problem with the existing master. It is then up to the client code to reconnect to the new master.
- Fixed number of cursors in protocol so that it works with py2.6, py2.6 and pypy.

Release 15.0 (2015-05-04)

This is the first release using the Twisted versioning method.

API Changes

- `collections.index_information` now mirrors PyMongo's method.
- `getrequestid` is now `get_request_id`

Features

- Add support for 2dsphere indexes, see http://docs.mongodb.org/manual/tutorial/build-a-2dsphere-index/
- PEP8 across files as we work through them.
- Authentication reimplemented for ConnectionPool support with multiple DBs.
- Add support for MongoDB 3.0

Bugfixes

- Fixed failing tests due to changes in Python in 2.6
- Fixed limit not being respected, which should help performance.
- Find now closes MongoDB cursors.
- Fixed 'hint' filter to correctly serialize with double dollar signs.

Improved Documentation

- Added, updated and reworked documentation using Sphinx.
- The documentation is now hosted on https://txmongo.readthedocs.org/.

Release 0.6 (2015-01-23)

This is the last release in this version scheme, we'll be switching to the Twisted version scheme in the next release.

API Changes

- TxMongo: None

Features

- Added SSL support using Twisted SSLContext factory
- Added "find with cursor" like pymongo
- Test coverage is now measured. We're currently at around 78%.

Bugfixes

- Fixed import in database.py

Release 0.5 (2014-10-02)

Code review and cleanup

Bugfixes

- Bug fixes

Release 0.4 (2013-01-07)

Significant performance improvements.

API Changes

- TxMongo: None

Features

- Support AutoReconnect to connect to fail-over master.
- Use pymongo instead of in-tree copy.

Bugfixes

- Bug fixes

Release 0.3 (2010-09-13)

Initial release.

License

- Apache 2.0

Status and History

TxMongo was created by Alexandre Fiori who developed it during the years 2009-2010. From 2010 and onwards mainly bug fixes were added, as Alexandre entered maintance mode with many contributions being made by others. Development picked back up in 2014 by Bret Curtis with Alexandre's consent and was migrated to Twisted where it is a first-party Twisted library. TxMongo can be found here:

https://github.com/twisted/txmongo

The MongoDB client library functionality is in active use. It is stable and works very well.

Contributions

How to Contribute

Head over to: https://github.com/twisted/TxMongo and submit your bugs or feature requests. If you wish to contribute code, just fork it, make a branch and send us a pull request. We'll review it, and push back if necessary.

TxMongo generally follows the coding and documentation standards of the Twisted project.

Contributors

- 10gen, Inc
- Alexandre Fiori
- Alexey Palazhchenko (AlekSi)
- Amplidata
- Andre Ferraz
- Bret Curtis
- Carl D'Halluin (Amplidata)
- Christian Hergert
- Dave Peticolas
- Gleicon Moraes
- Ilya Skriblovsky
- Jonathan Stoppani
- Mark L
- Mike Dirolf (mdirolf)
- Renzo Sanchez-Silva (rnz0)
- Runar Petursson
- Silas Sewell
- Stiletto
- Toby Padilla
- Tryggvi Björgvinsson
- Vanderson Mota (chunda)
- flanked
- renzo
- shylent
- Gareth Bult (oddjobz)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

N

O

P

Q

R

S

T

U

W