ctypes-ejdb Documentation

Release 0.4.7

Tzu-ping Chung

August 30, 2016

Contents

CHAPTER 1

Contents

1.1 Installation

You can install ctypes-ejdb with `pip`:

```
pip install ctypes-ejdb
```

The EJDB library should be installed to make database access possible. See EJDB installation guide for details.

Python 2.7 or 3.3+ is required.

1.2 Usage

This chapter describes how ctypes-ejdb can be used to manipulate an EJDB instance.

1.2.1 Tutorial

Before we start, make sure you have both ctypes-ejdb and EJDB installed. See Installation for instructions. The following should run without an exception:

```
import ejdb
```

Getting a Database

When working with ctypes-ejdb, the first step is to create a *Database* instance to a new or existing database file.

Todo

- Finish this tutorial.

1.3 API References

exception ejdb.**CollectionDoesNotExist**

exception ejdb.**DatabaseError**

exception ejdb.**OperationError**

exception ejdb.**TransactionError**

class ejdb.**Collection**(*database*, *wrapped*)
Representation of a collection inside a database.

You generally should not instantiate a collection directly. Call *Database.get_collection()* to get a collection inside a database instead.

abort_transaction()
Abort a transaction, discarding all un-committed operations.

begin_transaction(*allow_nested=False*)
Begin a transaction on this collection.

This can be used directly, with the user calling *commit_transaction()* or *abort_transaction()* later manually:

```
collection.begin_transaction()
try:
    ... # Do things.
except:
    collection.abort_transaction()
    raise
else:
    collection.commit_transaction()
```

Or as a context manager:

```
with collection.begin_transaction():
    ... # Do things.
```

In the latter usage, *abort_transaction()* will be called automatically when the block exits with an exception; if the block exits normally, *commit_transaction()* will be called.

commit_transaction()
Commit a transaction.

count(**queries*, *hints={}*)
Get the number of documents in this collection.

Parameters **hints** – A mapping of possible hints to the selection.

create_array_index(*path*)

create_index(*path*, *index_type*)

create_istring_index(*path*)

create_number_index(*path*)

create_string_index(*path*)

delete_many(**queries*, *hints={}*)
Delete documents in the collection.

This is an optimized shortcut for `find({..., '$dropall': True})`. Use the formal syntax if you want to get the content of deleted documents.

Parameters **hints** – A mapping of possible hints to the selection.

Returns Count of documents deleted.

delete_one(**queries*, *hints={}*)
Delete a single document in the collection.

This is an optimized shortcut for `find_one({..., '$dropall': True})`. Use the formal syntax if you want to get the deleted document's content.

Parameters **hints** – A mapping of possible hints to the selection.

Returns A boolean specifying whether a document is deleted.

drop()

find(**queries*, *hints={}*)
Find documents in the collection.

Parameters **hints** – A mapping of possible hints to the selection.

Returns A `Cursor` instance corresponding to this query.

find_one(**queries*, *hints={}*)
Find a single document in the collection.

Parameters **hints** – A mapping of possible hints to the selection.

Returns A mapping for the document found, or `None` if no matching document exists.

insert_many(*documents*)
Insert a list of documents.

Returns A list of OIDs of the inserted documents.

insert_one(*document*)
Insert a single document.

Returns OID of the inserted document.

is_in_transaction()

optimize_array_index(*path*)

optimize_index(*path*, *index_type*)

optimize_istring_index(*path*)

optimize_number_index(*path*)

optimize_string_index(*path*)

rebuild_array_index(*path*)

rebuild_index(*path*, *index_type*)

rebuild_istring_index(*path*)

rebuild_number_index(*path*)

rebuild_string_index(*path*)

remove(*oid*)
Remove the document matching the given OID from the collection.

This method is provided for compatibility with `ejdb-python`.

remove_array_index(*path*)

remove_index(*path*, *index_type=None*)
Remove index(es) on `path` from the collection.

The index of specified type on `path`, if given by `index_type`, will be removed. If `index_type` is `None`, all indexes on `path` will be removed.

remove_istring_index(*path*)

remove_number_index(*path*)

remove_string_index(*path*)

save(**documents*, *merge=False*)
Persist one or more documents in the collection.

If a saved document doesn't have a `_id` key, an automatically generated unused OID will be used. Otherwise the OID is set to the given document's `_id` field, possibly overwriting an existing document in the collection.

This method is provided for compatibility with `ejdb-python`.

Parameters **merge** – If evalutes to `True`, content of existing document with matching `_id` will be merged with the provided document's content.

database
The *Database* instance this collection belongs to.

name
Name of this collection.

class ejdb.**Database**(*path=''*, *options=READ*)
Representation of an EJDB.

A *Database* instance can be created like this:

```
db = ejdb.Database(
    path='path_to_db',
    options=(ejdb.WRITE | ejdb.TRUNCATE),
)
```

The database is opened immediately, unless the `path` argument evalutes to `False`. In such cases the user needs to set the path and manually call *open()* later.

close()
Close this EJDB.

create_collection(*name*, *exist_ok=False*, ***options*)
Create a collection in this database with given options.

The newly-created collection is returned. If `exist_ok` is `True`, existing collection with the same name will be returned, otherwise an error will be raised.

Options only apply to newly-created collection. Existing collections will not be affected. Possible options include:

Parameters

- **large** – If `True`, the collection can be larger than 2 GB. Default is `False`.
- **compressed** – If `True`, the collection will be compressed with DEFLATE compression. Default is `False`.
- **records** – Expected records number in the collection. Default is `128000`.
- **cachedrecords** – Maximum number of records cached in memory. Default is `0`.

drop_collection(*name*, *unlink=True*)
Drop a collection in this database.

Does nothing if a database with matching name does not exist.

Parameters

- **name** – Name of collection to drop.
- **unlink** – If `True`, removes all related index and collection files. Default is `True`.

find(*collection_name*, **queries*, *hints={}*)
Shortcut to query a collection in the database.

The following usage:

```
db.find('people', {'name': 'TP'})
```

is semantically identical to:

```
collection = db.create_collection('people', exist_ok=True)
collection.find({'name': 'TP'})
```

find_one(*collection_name*, **queries*, *hints={}*)
Shortcut to query a collection in the database.

The following usage:

```
db.find_one('people', {'name': 'TP'})
```

is semantically identical to:

```
collection = db.create_collection('people', exist_ok=True)
collection.find_one({'name': 'TP'})
```

get_collection(*name*)
Get the collection with name `name` inside this EJDB.

has_collection(*name*)
Check whether this EJDB contains a collection named `name`.

is_open()
Check whether this EJDB is currently open.

open()
Open this EJDB.

This can be used directly, with the user calling *close()* later manually:

```
db.open()
try:
    ... # Do things.
except:
    ... # Handle exceptions.
finally:
    db.close()
```

Or as a context manager:

```
with db.open():
    ... # Do things.
```

In the latter usage, *close()* will be called automatically when the block exits.

save(*collection_name*, **documents*, *merge=False*)
Shortcut to save to a collection in the database.

The following usage:

```
db.save({'people', {'name': 'TP'})
```

is semantically identical to:

```
collection = db.create_collection('people', exist_ok=True)
collection.save({'name': 'TP'})
```

collection_names

collections

options
Options for the EJDB.

This can be modified if the database instance is not opened.

path
Path to the EJDB.

This can be modified if the database instance is not opened.

writable

ejdb.**get_ejdb_version**(**args*, ***kwargs*)
Get version of the underlying EJDB C library.

ejdb.**init**(*ejdb_path=None*)

ejdb.**is_valid_oid**(**args*, ***kwargs*)
Check whether the given string can be used as an OID in EJDB.

The current OID format (as of 1.2.x) is a 24-character-long hex string.

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/uranusjr/ctypes-ejdb/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

ctypes-ejdb could always use more documentation, whether as part of the official ctypes-ejdb docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/uranusjr/ctypes-ejdb/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.4.2 Get Started!

Ready to contribute? Here's how to set up `ctypes-ejdb` for local development.

1. Fork the `ctypes-ejdb` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ctypes-ejdb.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ctypes-ejdb
$ cd ctypes-ejdb/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ctypes-ejdb tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/uranusjr/ctypes-ejdb/pull_requests and make sure that the tests pass for all supported Python versions.

1.4.4 Tips

To run a subset of tests:

```
$ py.test tests/test_api.py::test_get_ejdb_version
```

1.5 Credits

1.5.1 Contributors

- Tzu-ping Chung <uranusjr@gmail.com>
- Gary Lee <garywlee@gmail.com>

1.6 History

1.6.1 0.4.7 (2016-07-20)

- Fix crash when querying with invalid parameter names. This now raises an `CommandError`.
- Fix memory leak when calling `Collection.count`.
- Add API to query for a list of collection names in a database without needing to construct the collections themselves.
- Add API to check whether a database is writable.
- Add flag to disable coloring in CLI, and disable it on Windows by default.
- `ejdb.cli` now has a `--version` option.

1.6.2 0.4.6 (2015-10-06)

- Fix Python 2 compatibility regarding `ejdb.cfg` usage.
- Fix segmentation fault when trying to reuse collection instances retrieved from iterating through a database.
- `ejdb.cli` now creates a non-existent database if the path given does not exist.

- Add a more meaningful error message when the EJDB binary path is not configured properly.
- Fix documentation on `Collection.delete_one()` and `Collection.delete_many()`.

1.6.3 0.4.5 (2015-09-07)

- Fix `Collection.delete_one` and `Collection.delete_many`.

1.6.4 0.4.4 (2015-07-30)

- Fix query flag passing.

1.6.5 0.4.3 (2015-07-29)

- Move `exit()` fix in CLI.

1.6.6 0.4.2 (2015-07-29)

- Fix `exit()` call in CLI.

1.6.7 0.4.1 (2015-07-27)

- Fix missing `NOBLOCK` constant.

1.6.8 0.4 (2015-07-25)

- Move command line interface dependencies to extras. New installations now needs to run `pip install ctypes-ejdb[cli]` to install it. This is better for those who want only the core library.

1.6.9 0.3.3 (2015-07-24)

- Fix Python 2 compatibility.

1.6.10 0.3.2 (2015-07-07)

- Fix attribute lookup in `DatabaseError` construction.
- Add options to config EJDB path by environ or `.cfg` file.
- Make document repr look like a dict so it prints better.

1.6.11 0.3.1 (2015-07-03)

- Fixed context manager usage opening a `Database`.
- Fixed attribute error in `Collection.count`.
- Fixed document iterator slicing.
- Experimental CLI utility `ejdb.cli` based on Click and ptpython.

1.6.12 0.3 (2015-07-01)

- Make EJDB path configurable with `ejdb.init(path)`.

1.6.13 0.2.1 (2015-07-01)

- Add save shortcut on database.

1.6.14 0.2 (2015-07-01)

- Fix segmentation fault when converting BSON OID to string.
- Fix error message retrieval in `Database.close`.
- Tests now run on Windows.

1.6.15 0.1.1 (2015-06-30)

- Fix encoding error in `pip install`.

1.6.16 0.1.0 (2015-06-28)

- First release on PyPI.

e

S

T

W