time2relax Documentation

Release 0.3.0

Raymond Wanyoike

Nov 16, 2017

Contents

Contents:

CHAPTER 1

time2relax - Python CouchDB driver

A CouchDB driver for Python.

- Documentation: https://time2relax.readthedocs.org
- Free software: MIT license

1.1 Features

- Runs python-requests under the hood.
- Only a minimum level of abstraction between you and CouchDB.
- Transparent URL and parameter encoding.
- Exceptions are modelled from CouchDB errors.
- Python 2.7 & 3.3–3.5 support.
- Tested with CouchDB 1.6.x (2.0?).

1.2 Quickstart

Install the latest time2relax if you haven't yet:

```
$ pip install -U time2relax
```

To use time2relax in a project:

```
>>> from time2relax import CouchDB
>>> db = CouchDB('http://localhost:5984/dbname')
```

Then:

```
>>> db.insert({'title': 'Ziggy Stardust'})
<Response [201]>
```

CHAPTER 2

Installation

2.1 Stable release

To install time2relax, run this command in your terminal:

```
$ pip install time2relax
```

This is the preferred method to install time2relax, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for time2relax can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/rwanyoike/time2relax
```

Or download the tarball:

```
$ curl -OL https://github.com/rwanyoike/time2relax/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use time2relax in a project:

```
>>> from time2relax import CouchDB
>>> db = CouchDB('http://localhost:5984/dbname')
```

Most of the API is exposed as:

```
>>> db.function(*args, **kwargs)
```

Where `**kwargs` are optional arguments that `requests.Request` takes.

3.1 Create a database

Initially the `CouchDB` object will check if the database exists, and try to create it if it does not. You can use `skip_setup=True` to skip this setup:

```
>>> db = CouchDB('http://localhost:5984/dbname', skip_setup=True)
```

3.2 Delete a database

Delete the database:

```
>>> db.destroy()
<Response [200]>
```

Further requests with the `CouchDB` object will raise a `time2relax.CouchDBError`:

```
>>> db.info()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
  File "time2relax/__init__.py", line 243, in info
    return self.request('GET', **kwargs)
  File "time2relax/__init__.py", line 371, in request
    raise CouchDBError('Database is destroyed')
time2relax.CouchDBError: Database is destroyed
```

3.3 Create/update a document

Note: There are some restrictions on valid property names of the documents.

Create a new document:

```
>>> db.insert({'_id': 'docid', 'title': 'Heros'})
<Response [201]>
```

To create a new document and let CouchDB auto-generate an `_id` for it:

```
>>> db.insert({'title': 'Ziggy Stardust'})
<Response [201]>
```

If the document already exists, you must specify its revision `_rev`, otherwise a conflict will occur. You can update an existing document using `_rev`:

```
>>> r = db.get('docid')
>>> result = r.json()
>>> db.insert({'_id': result['_id'], '_rev': result['_rev'], 'title': 'Dance'})
<Response [201]>
```

3.4 Fetch a document

Retrieve a document:

```
>>> db.get('docid')
<Response [200]>
```

3.5 Delete a document

You must supply the `_rev` of the existing document.

Delete a document:

```
>>> r = db.get('docid')
>>> result = r.json()
>>> db.remove(result['_id'], result['_rev'])
<Response [200]>
```

You can also delete a document by using `time2relax.CouchDB.insert()` with `{'_deleted': True}`:

```
>>> r = db.get('docid')
>>> result = r.json()
>>> result['_deleted'] = True
```

```
>>> db.insert(result)
<Response [200]>
```

3.6 Create/update a batch of documents

Create multiple documents:

```
>>> db.bulk_docs([
...     {'_id': 'doc1', 'title': 'Lisa Says'},
...     {'_id': 'doc2', 'title': 'Space Oddity'},
... ])
<Response [201]>
```

If you omit an `_id` parameter on a given document, the database will create a new document and assign the ID for you:

```
>>> db.bulk_docs([
...     {'title': 'Lisa Says'},
...     {'title': 'Space Oddity'},
... ])
<Response [201]>
```

To update a document, you must include both an `_id` parameter and a `_rev` parameter, which should match the ID and revision of the document on which to base your updates:

```
>>> db.bulk_docs([
...     {
...         '_id': 'doc1',
...         '_rev': '1-84abc2a942007bee7cf55007cba56198',
...         'title': 'Lisa Says',
...         'artist': 'Velvet Underground',
...     },
...     {
...         '_id': 'doc2',
...         '_rev': '1-7b80fc50b6af7a905f368670429a757e',
...         'title': 'Space Oddity',
...         'artist': 'David Bowie',
...     },
... ])
<Response [201]>
```

Finally, to delete a document, include a `_deleted` parameter with the value `True`:

```
>>> db.bulk_docs([
...     {
...         '_id': 'doc1',
...         '_rev': '1-84abc2a942007bee7cf55007cba56198',
...         'title': 'Lisa Says',
...         '_deleted': True,
...     },
...     {
...         '_id': 'doc2',
...         '_rev': '1-7b80fc50b6af7a905f368670429a757e',
...         'title': 'Space Oddity',
...         '_deleted': True,
```

```
...     },
... ])
<Response [201]>
```

3.7 Fetch a batch of documents

Fetch multiple documents:

```
>>> params = {'include_docs': True, 'attachments': True}
>>> db.all_docs(params=params)
<Response [200]>
```

You can use `startkey`/`endkey` to find all docs in a range:

```
>>> params = {'include_docs': True, 'attachments': True, 'startkey': 'bar', 'endkey':
↪'quux'}
>>> db.all_docs(params=params)
<Response [200]>
```

You can also do a prefix search – i.e. "give me all the documents whose `_id` start with `'foo'`" – by using the special high Unicode character `'\ufffd'`:

```
>>> params = {'include_docs': True, 'attachments': True, 'startkey': 'foo', 'endkey':
↪'foo\ufffd'}
>>> db.all_docs(params=params)
<Response [200]>
```

3.8 Replicate a database

Replicate the database to a target:

```
>>> db.replicate_to('http://localhost:5984/otherdb')
<Response [200]>
```

The target has to exist, add `json={'create_target': True}` to create it prior to replication.

3.9 Save an attachment

This method will update an existing document to add the attachment, so it requires a `_rev` if the document already exists. If the document doesn't already exist, then this method will create an empty document containing the attachment.

Attach a binary object:

```
>>> with open('/tmp/att.txt') as att:
...     db.insert_att('docid', None, 'att.txt', att, 'text/plain')
...
<Response [201]>
```

3.10 Get an attachment

Get attachment data:

```
>>> db.get_att('docid', 'att.txt')
<Response [200]>
```

3.11 Delete an attachment

You must supply the `_rev` of the existing document.

Delete an attachment:

```
>>> r = db.get('docid')
>>> result = r.json()
>>> db.remove_att(result['_id'], result['_rev'], 'att.txt')
<Response [200]>
```

3.12 Get database information

Get information about the database:

```
>>> db.info()
<Response [200]>
```

3.13 Compact the database

This reduces the database's size by removing unused and old data, namely non-leaf revisions and attachments that are no longer referenced by those revisions.

Trigger a compaction operation:

```
>>> db.compact()
<Response [202]>
```

3.14 Run a list function

Make sure you understand how list functions work in CouchDB. A good start is the CouchDB guide entry on lists:

```
>>> db.insert({
...     '_id': '_design/testid',
...     'views': {
...         'viewid': {
...             'map': "function (doc) {"
...                    "    emit(doc._id, 'value');"
...                    "}",
...         },
...     },
```

```
...     'lists': {
...         'listid': "function (head, req) {"
...                   "    return 'Hello World!';"
...                   "}",
...     },
... })
<Response [201]>
>>> db.ddoc_list('testid', 'listid', 'viewid')
<Response [200]>
```

3.15 Run a show function

Make sure you understand how show functions work in CouchDB. A good start is the CouchDB guide entry on shows:

```
>>> db.insert({
...     '_id': '_design/testid',
...     'shows': {
...         'showid': "function (doc, req) {"
...                   "    return {body: 'relax!'}"
...                   "}",
...     },
... })
<Response [201]>
>>> db.ddoc_show('testid', 'showid')
<Response [200]>
```

3.16 Run a view function

Make sure you understand how view functions work in CouchDB. A good start is the CouchDB guide entry on views:

```
>>> db.insert({
...     '_id': '_design/testid',
...     'views': {
...         'viewid': {
...             'map': "function (doc) {"
...                    "    emit(doc.key);"
...                    "}",
...         },
...     },
... })
<Response [201]>
>>> params = {'reduce': False, 'key': 'key2'}
>>> db.ddoc_view('testid', 'viewid', params=params)
<Response [200]>
```

CHAPTER 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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/rwanyoike/time2relax/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.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

time2relax could always use more documentation, whether as part of the official time2relax docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/rwanyoike/time2relax/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 :)

4.2 Get Started!

Ready to contribute? Here's how to set up *time2relax* for local development.

1. Fork the *time2relax* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/time2relax.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 time2relax
$ cd time2relax/
$ 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 time2relax tests
$ python setup.py test or 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.

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 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/rwanyoike/time2relax/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_time2relax
```

CHAPTER 5

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

5.1 Development Lead

- Raymond Wanyoike <raymond.wanyoike@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

- Python 2.6 is no longer supported by the Python core team.

6.1 0.3.0 (2017-03-03)

- Thin python-requests wrapper.
- New driver API.
- Add design document functions; list, show, view.

6.2 0.2.0 (2016-12-10)

- Python 3.3, 3.4, 3.5 support.

6.3 0.1.0 (2016-12-10)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search