pneumatic Documentation

Release 0.1.7

Anthony DeBarros

November 25, 2016

Contents

CHAPTER 1

A Bulk-Upload Library for DocumentCloud

pneumatic is a Python 3 library that adds some luxury and safeguards to the bulk-uploading of hundreds, thousands or hundreds of thousands of files to DocumentCloud. It is meant to do one thing – upload – and serve as an adjunct to, but not a replacement for, the excellent python-documentcloud API wrapper.

pneumatic's name is inspired by the pneumatic dispatch systems in newsrooms of yore, which featured a series of pneumatic tubes for sending copy from the newsrooms to other departments such as the composing room.

CHAPTER 2

Features

- Catalogs the API response for each upload in a SQLite database along with the file's canonical URL.
- Dumps the SQLite data to a CSV if you wish.
- Multiprocessing (under Mac/Linux) for faster submission of files to DocumentCloud's API.
- Prevents inadvertent submission of file types DocumentCloud doesn't handle, such as audio.

CHAPTER 3

Links

- Documentation: http://pneumatic.readthedocs.io/en/latest/
- Repository: https://github.com/anthonydb/pneumatic
- Issues: https://github.com/anthonydb/pneumatic/issues

CHAPTER 4

Basic Usage

You will need an active DocumentCloud account and Python 3.4+. First, install via pip:

```
pip install pneumatic
```

Example use: To upload all files in a directory (and all sub-directories below it), assign them to an existing project, set the files to public access, and tag each with metadata, run the following code:

```
from pneumatic import DocumentCloudUploader

uploader = DocumentCloudUploader('person@example.com', 'your-password')
uploader.upload(
    file_directory='/govfiles',
    project='17477-loudoun-county-government',
    access='public',
    data={'type': 'government', 'action': 'lawsuit'})
```

4.1 Table of contents

4.1.1 Installation

To get started with pneumatic, install using pip:

```
pip install pneumatic
```

Supported Python versions

pneumatic supports Python 3.4+. Python 2.x is currently not supported.

4.1.2 Uploading

pneumatic is designed for one thing: uploading documents. It supports all parameters found in the DocumentCloud API upload endpoint.

Create a client

To create an upload client, import the `DocumentCloudUploader` class in your Python script and create a client object. You must have a valid DocumentCloud account. Supply your email and password as arguments:

```
from pneumatic import DocumentCloudUploader

uploader = DocumentCloudUploader('person@example.com', 'your-password')
```

Start an upload

Use the `upload` method to upload files, passing in parameters as desired. For example, after you import pneumatic and creat a client as shown above, you can initiate the upload with code such as this:

```
uploader.upload(
    file_directory='/govfiles',
    project='17477-loudoun-county-government',
    access='public',
    data={'type': 'government', 'action': 'lawsuit'})
```

In this example, pneumatic will attempt to upload all files in the `/govfiles` directory and all subdirectories below it. In addition, it will assign the files to an existing project (must have been created already), set each file to public access, and tag each file with custom metadata.

If no `file_directory` parameter is supplied, pneumatic uploads files in the directory from which you execute your Python script.

Upload parameters

pneumatic handles all of the parameters supported by DocumentCloud's upload endpoint. All are optional:

- `file_directory`: string; the path to your files. If no `file_directory` parameter is supplied, pneumatic uploads files in the directory from which you execute your Python script.
- `title`: string; the file title. Normally not used with pneumatic because if you specify this, it applies to all files you upload.
- `source`: string; the source for the files.
- `description`: string; the description of the files.
- `language`: string; the language of the document for OCR. Default is `eng`.
- `related_article`: string; URL of an article related to the files.
- `published_url`: string: URL of the page where the documents are published.
- `access`: string; use `access='public'` to make documents publicly viewable upon upload. Default is private.
- `project`: string; ID of a project.
- `data`: dictionary of keys and values to tag documents. Example: `data={'type': 'government', 'action': 'lawsuit'}`
- `secure`: boolean; set to `secure=True` if you want to prevent DocumentCloud from passing your document's text through the OpenCalais API.
- `force_ocr`: boolean; set to `force_ocr=True` to have DocumentCloud OCR your documents even if they contain a text layer.

For more information on these parameters, consult the DocumentCloud API documentation

File exclusions

DocumentCloud accepts a variety of file types for upload. In addition to PDFs, users can submit Word and Excel files, PowerPoint, and a variety of images. pneumatic will not submit files that exceed DocumentCloud's 400MB size limit, nor will it submit file types that the platform does not handle, such as audio.

Upload results

DocumentCloud's API returns data for each file uploaded, and pneumatic catalogs key pieces of this data in a SQLite database that is automatically created under a `pneumatic_db` directory in the directory specified for uploads. See the database features documentation for additional information.

Multiprocessing support

If you are using a Mac or Linux computer, pneumatic uses the Python multiprocessing library to more rapidly submit files to the DocumentCloud API. Multiprocessing is not currently supported under Windows. Note that this speed improvement does not impact actual processing time for files on DocumentCloud.

4.1.3 Database Features

Each time you start an upload, pneumatic creates a table called `uploads` in a SQLite database in a folder called `pneumatic_db` in the current directory. The `uploads` table captures the following local file data plus data from DocumentCloud's API response:

- `id`: The unique document id stored by DocumentCloud. Typically, it's in the form of a number followed by several words. e.g. `12345-some-document-title`.
- `title`: The document's title.
- `file_name`: Name of the file submitted for upload.
- `full_path`: The full directory path to the file, including the file name.
- `upload_time`: Timestamp indicating the time of upload.
- `pages`: The number of pages in the document.
- `file_hash`: A unique identifier calculated based on the document. Useful for finding identical documents.
- `result`: HTTP response code from DocumentCloud. `200` indicates success.
- `canonical_url`: The URL to the document displayed in the viewer on DocumentCloud.
- `pdf_url`: The URL to the PDF.
- `text_url`: The URL to the extracted text of the document.
- `exclude_flag`: Flag indicating whether pneumatic excluded the file from being uploaded. For information on why files might be excluded, see the documentation on File Exclusions.
- `exclude_reason`: The reason a file was excluded from upload.
- `error_msg`: Server error message if upload attempt failed.

Printing the database name and location

After you initiate a `DocumentCloudUploader` object and begin the upload, you can instruct pneumatic to print the path to and name of the database file:

```
uploader.db.print_db_name()
```

Viewing database contents

There are number of tools available for viewing and querying the SQLite database's contents. One example that's simple to use is the SQLite Manager browser plugin for Firefox.

pneumatic also provides a method to dump the database to a CSV file, which you can then load into Excel or other applications. If you choose to do so in the same script where you have initiated an upload client, use the following:

```
uploader.db.dump_to_csv()
```

If you're coming back to an old database, or you forgot to dump the CSV while the object was alive, you can still dump to a CSV by importing a `Database()` object:

```
from pneumatic import Database

db = Database()
db.dump_to_csv('path/to/file.db')
```

Updating database contents

The API response to a DocumentCloud upload does not include an actual value for `pages` or `file_hash`. These are calculated as part of DocumentCloud's processing of the document. Once your documents are finished processing, you can use pneumatic to go back and retrieve those values with the `update_processed_files` method.

You can use the method within a current session, which will update the database active in the session, or you can pass in the name of any other database file pneumatic created.

To use within the current session:

```
uploader.update_processed_files()
```

To come back to an older database and process it:

```
from pneumatic import DocumentCloudUploader

uploader = DocumentCloudUploader('person@example.com', 'your-password')
uploader.update_processed_files('path/to/file.db')
```

Currently, only `title`, `pages` and `file_hash` are updated.

4.1.4 Use With python-documentcloud

pneumatic happily coexists with python-documentcloud, a full-featured wrapper for the DocumentCloud API. python-documentcloud offers extensive methods for interacting with documents, projects, annotations, entities and other aspects of the platform.

For example, you can create a project with python-documentcloud, then use pneumatic to upload files and capture the results data:

```
# Import both libraries.
from pneumatic import DocumentCloudUploader
from documentcloud import DocumentCloud

# Create the respective clients.
uploader = DocumentCloudUploader('person@example.com', 'your-password')
dc_client = DocumentCloud('person@example.com', 'your-password')

# Using python-documentcloud, create a project under your account.
project = dc_client.projects.create('Loudoun Fire')

# Using pneumatic, upload your files. Add them to the project id obtained when
# the project was created just now.
uploader.upload(
    file_directory='/path-to/files',
    project=project.id,
    source='Loudoun County Fire and Rescue',
    data={'topic': 'fires'})

# pneumatic made a database of your upload results. Print its name and location.
uploader.db.print_db_name()

# Dump the pneumatic upload results data to a CSV.
uploader.db.dump_to_csv()
```

4.1.5 Changelog

0.1.7 - November 24, 2016

- Delete SQLite database if, upon program exit, it contains no records. (#14)
- Add some color, improve readability in console output. (#18)
- Add colorama dependency for printing ANSI escape code colors in Windows.

0.1.6 - April 28, 2016

- Provide `User-Agent` and `From` HTTP header fields.
- Correct issue where `force_ocr` and `secure` parameters were not being set default to `false` in the Ruby way. (#17)
- Handle 50X errors from the API (which do not return JSON). (#15, thank you, Tom Meagher!)

0.1.5 - March 9, 2016

- Add `update_processed_files` method to get page, file hash and other data that's not available upon upload.
- Add `pages`, `file_hash`, `id` and `title` to items tracked in database.
- Create database upon `DocumentCloudUploader` initialization.
- Get rid of the file extension in the document title. (#13)
- Bug fix: Properly test for presence of data and title keyword arguments.

0.1.4 - February 17, 2016

- Remove extra line space on csv dump in Windows.

0.1.3 - February 16, 2016

- Report number of files to be uploaded before starting.
- Better reporting of upload progress and results.
- More comprehensive filetype exclusion list.
- Record pdf and text URLs in database.
- Uploads that return status codes other than 200 are handled.

0.1.2 - February 4, 2016

- `dump_to_csv` outputs contents of SQLite database. (#2)
- Add `force_ocr` parameter to upload options.
- Removed multiprocessing support for Windows for now.
- Report when upload file directory does not exist.
- Better testing for prohibited file types.
- Only create database after file path verified.

0.1.1 - December 31, 2015

- Packaged for release to PyPi.

0.1 - December 16, 2015

- Add multiprocessing. (#1)
- Exclude files of 400MB or larger from upload. (#3)
- Add initial tests.
- Scaffolding for documentation.

0.0.1 - November 26, 2015

- Pre-alpha prototype

4.2 License

copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

4.3 Indices and tables

- genindex
- modindex
- search