django-calaccess Documentation

California Civic Data Coalition

Aug 16, 2018

Contents

Technical documentation for a collection of applications that make it easier to work with CAL-ACCESS, the jumbled, dirty and difficult government database that tracks campaign-finance and lobbying-activity in California politics. Built using the Django web framework.

This is a work in progress. It is maintained by the California Civic Data Coalition, an open-source team of journalists and computer programmers from news organizations across America. To learn more and get involved, read our backstory in *the FAQ* and reach out to our leadership team.

CHAPTER 1

Table of contents

1.1 Installation Guide

This guide will walk you through the process of installing the latest official release of *django-calaccess-processed-data* so that you can incorporate CAL-ACCESS data into your own Django project.

If, instead, you want to install the raw source code or contribute as a developer please refer to the "How to contribute" tutorial.

Warning: This library is intended to be plugged into a project created with the Django web framework. Before you can begin, you'll need to have one up and running. If you don't know how, check out the official Django documentation.

1.1.1 Installing the Django apps

The latest version of the application can be installed from the Python Package Index using `pip`.

```
$ pip install django-calaccess-processed-data
```

Like most Django applications, the app then needs to be added to the `INSTALLED_APPS` in your `settings.py` configuration file. You also need to include other Django apps it depends on:

```
INSTALLED_APPS = (
    # ... other apps up here ...
    'calaccess_raw',
    'calaccess_scraped',
    'calaccess_processed',
    'calaccess_processed_filings',
    'calaccess_processed_elections',
```

(continues on next page)

(continued from previous page)

```
    'calaccess_processed_flatfiles',
    'calaccess_processed_campaignfinance',
    'opencivicdata.core.apps.BaseConfig',
    'opencivicdata.elections.apps.BaseConfig',
)
```

A little more about these dependencies:

calaccess_raw This app downloads and extracts the raw data files exported each night from the CAL-ACCESS database. The app then loads these files into your Django project's database with minimal transformations. For more details, see the *django-calaccess-raw-data* section.

calaccess_scraped This app scrapes the CAL-ACCESS website and loads additional data not included in the nightly exports. For more details, see the *django-calaccess-scraped-data* section.

opencivicdata.core This app includes Django models and admin panels for the core data types of the Open Civic Data specification, including `Person`, `Organization`, `Post` and `Membership`.

opencivicdata.elections This app includes Django models and admins panels for election-related data types that have been provisionally included in the Open Civic Data specification.

1.1.2 Connecting to a local database

Also in the `settings.py` file, you will need to configure Django so it can connect to your database.

Note: Unlike a typical Django project, this application only supports PostgreSQL database backends. This is because we enlist specialized tools to load the immense amount of source data more quickly than Django typically allows. We haven't developed those routines for SQLite and the other Django backends yet, but we might someday.

Before you begin, make sure you have a PostgreSQL server installed. If you don't, now is the time to hit Google and figure out how. The official PostgreSQL documentation is another good place to start.

Once that's handled, add a database connection string like this to your `settings.py`.

```
DATABASES = {
    'default': {
        'NAME': 'calaccess_processed',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'your-username-here',
        'PASSWORD': 'your-password-here',
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

Return to the command line. This will create a PostgreSQL database to store the data.

```
$ createdb calaccess_processed
```

Note: If you'd prefer to load the CAL-ACCESS outside your default database, check out our guide to working with Django's system for multiple databases.

1.1.3 Loading the data

Now you're ready to create the database tables with Django using its `manage.py` utility belt.

```
$ python manage.py migrate
```

Once everything is set up, the updatecalaccessrawdata command will download the latest bulk data release from the Secretary of State's website and load it into your location database.

```
$ python manage.py updatecalaccessrawdata
```

Warning: This will take an hour or more. Go grab some coffee.

Because the nightly raw export is incomplete, we have to scrape additional data from the CAL-ACCESS website. Use the scrapecalaccess command to kick off this process, either after `updatecalaccessrawdata` finishes or in a separate terminal window:

```
$ python manage.py scrapecalaccess
```

Once the raw CAL-ACCESS data is loaded and the scrape has finished, you can transform all this messy data and load into a more simplified structure with the processcalaccessdata command:

```
$ python manage.py processcalaccessdata
```

1.2 Custom Project Settings

The settings listed below allow you to customize the behavior our apps to suit your needs. They should be declared in your Django project's `settings.py` file.

Read more about Django settings here.

1.2.1 CALACCESS_DATA_DIR

The local directory where the `calaccess_raw` and `calaccess_processed` management commands will download, extract and write files. By default, this is will be `{BASE_DIR}/data/`, where `BASE_DIR` is a setting pre-populated in `settings.py` when you set up a new Django project.

You can change this location — say to the `tmp/` directory at your file system's root — by adding a line to `settings.py`:

```
CALACCESS_DATA_DIR = '/tmp/'
```

1.2.2 CALACCESS_STORE_ARCHIVE

Enable archiving of all `.ZIP`, `.TSV` and `.CSV` files in order to preserve each snapshot of the raw and processed CAL-ACCESS data.

Be default, archiving is *disabled*. You can enable it by adding this line to `settings.py`:

```
CALACCESS_DATA_DIR = True
```

If you enable archiving, files will be saved in your Django project's default storage system, which you can also customize. For example, we use django-storages to upload our archived files to an AWS Simple Service Storage (S3) bucket.

If you enable archiving without configuring `DEFAULT_FILE_STORAGE`, files will be stored in the directory specified in your Django project's `MEDIA_ROOT`.

You can read more about how Django manages file storage here.

1.3 The Django apps

The full set of features available in each of our Django applications.

1.3.1 django-calaccess-raw-data

A Django app to download, extract and load campaign-finance and lobbying-activity data from the California Secretary of State's CAL-ACCESS database.

Management commands

The raw-data app includes the following commands for processing and verifying the raw data released in the CAL-ACCESS **'nightly exports'_**.

As with any Django app management command, these can be invoked on the command line or called within your Python code.

---

updatecalaccessrawdata

This is the master command. It brings together all of the other management commands listed below to download, unzip, clean and load the latest snapshot of the CAL-ACCESS database.

Examples

Running the entire routine is as simple as this.

```
$ python manage.py updatecalaccessrawdata
```

This command will either:

- Update your copy of the CAL-ACCESS data to the latest snapshot on the California Secretary of State's website
- Or complete your previously interrputed update, if possible.

You can skip the download's confirmation prompt using Django's standard `--noinput` option.

```
$ python manage.py updatecalaccessrawdata --noinput
```

The source files downloaded as part of the process will be deleted unless the `--keep-files` option is provided.

```
$ python manage.py updatecalaccessrawdata --keep-files
```

The other options are below.

Options

```
usage: manage.py updatecalaccessrawdata [-h] [--version] [-v {0,1,2,3}]
                                        [--settings SETTINGS]
                                        [--pythonpath PYTHONPATH]
                                        [--traceback] [--no-color]
                                        [--keep-files] [--noinput]
                                        [-a APP_NAME]

Download, unzip, clean and load the latest CAL-ACCESS database ZIP

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --keep-files          Keep zip, unzipped, TSV and CSV files
  --noinput             Update or resume previous update without asking
                        permission
  -a APP_NAME, --app-name APP_NAME
                        Name of Django app with models into which data will be
                        imported (if not calaccess_raw)
```

Note: The `updatecalaccessrawdata` command overwrites the previously downloaded, extracted and cleaned files in the application's download directory.

---

cleancalaccessrawfile

Clean a source CAL-ACCESS TSV file and reformat it as a CSV. A component of the master `updatecalaccessrawdata` command.

Examples

Provide the name of the TSV file you would like to process. The command will attempt to find it in the application's download directory.

```
$ python manage.py cleancalaccessrawfile RCPT_CD.TSV
```

The original TSV file will be deleted in favor of the new CSV unless the `--keep-file` option is provided.

```
$ python manage.py cleancalaccessrawfile RCPT_CD.TSV --keep-file
```

Options

```
usage: manage.py cleancalaccessrawfile [-h] [--version] [-v {0,1,2,3}]
                                       [--settings SETTINGS]
                                       [--pythonpath PYTHONPATH] [--traceback]
                                       [--no-color] [--keep-file]
                                       file_name

Clean a source CAL-ACCESS TSV file and reformat it as a CSV

positional arguments:
  file_name             Name of the TSV file to be cleaned and discarded for a
                        CSV

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --keep-file          Keep original TSV file
```

Note: The `cleancalaccessrawfile` command overwrites the CSV files previously processed from the original TSV files.

---

downloadcalaccessrawdata

Download the latest CAL-ACCESS database ZIP. A component of the master `updatecalaccessrawdata` command.

Examples

Here is how to run the command.

```
$ python manage.py downloadcalaccessrawdata
```

You will then see a prompt with the release date and size of the latest zip of raw CAL-ACCESS data files available to download from the California Secretary of State.

If your previous download did not complete *and* the same snapshot is still available to download, you will be prompted to resume your previous download.

You can skip the download's confirmation prompt using Django's standard `--noinput` option.

```
$ python manage.py downloadcalaccessrawdata --noinput
```

The other options are below.

The server hosting the ZIP doesn't always provide the most up-to-date resource (as we have documented). As such, a `CommandError` will be raised under either of the following conditions:

- If the actual size of the ZIP does not match the value of the `Content-Length` in the HEAD response.
- If the `Last-modified` of HEAD and GET are more than five minutes apart.

Options

```
usage: manage.py downloadcalaccessrawdata [-h] [--version] [-v {0,1,2,3}]
                                          [--settings SETTINGS]
                                          [--pythonpath PYTHONPATH]
                                          [--traceback] [--no-color]
                                          [--noinput] [--force-restart]

Download the latest CAL-ACCESS database ZIP

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --noinput             Download the ZIP archive without asking permission
  --force-restart, --restart
                        Force re-start (overrides auto-resume).
```

Note: The `downloadcalaccessrawdata` command overwrites the previously downloaded zip file.

extractcalaccessrawfiles

Extract the CAL-ACCESS raw data files from downloaded ZIP. A component of the master `updatecalaccessrawdata` command.

Examples

Here is how to run the command.

```
$ python manage.py extractcalaccessrawfiles
```

The downloaded zip file will be deleted unless the `--keep-files` option is provided.

```
$ python manage.py extractcalaccessrawfiles --keep-files
```

Options

```
usage: manage.py extractcalaccessrawfiles [-h] [--version] [-v {0,1,2,3}]
                                          [--settings SETTINGS]
                                          [--pythonpath PYTHONPATH]
                                          [--traceback] [--no-color]
                                          [--keep-files]

Extract the CAL-ACCESS raw data files from the database export ZIP

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --keep-files          Keep downloaded zipped files
```

Note: The `extractcalaccessrawfiles` command overwrites the previously extracted TSV files.

---

loadcalaccessrawfile

Load clean CAL-ACCESS CSV file into a database model. A component of the master `updatecalaccessrawdata` command.

Examples

The command expects the name of the Django database model where the file will be loaded.

```
$ python manage.py loadcalaccessrawfile RcptCd
```

The model will attempt to load its default CSV file unless one is provided with the `--csv` argument.

```
$ python manage.py loadcalaccessrawfile RcptCd --csv=/home/jerry/Data/MyFile.csv
```

Options

```
usage: manage.py loadcalaccessrawfile [-h] [--version] [-v {0,1,2,3}]
                                      [--settings SETTINGS]
                                      [--pythonpath PYTHONPATH] [--traceback]
                                      [--no-color] [--c CSV] [--keep-file]
                                      [-a APP_NAME]
                                      model_name

Load clean CAL-ACCESS CSV file into a database model

positional arguments:
  model_name            Name of the model into which data will be loaded

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --c CSV, --csv CSV    Path to comma-delimited file to be loaded. Defaults to
                        one associated with model.
  --keep-file          Keep clean CSV file after loading
  -a APP_NAME, --app-name APP_NAME
                        Name of Django app with models into which data will be
                        imported (if other not calaccess_raw)
```

Note: The `loadcalaccessrawfile` command deletes any data previously loaded into the calaccess_raw models before loading in the current data.

Models for tracking updates

The raw-data app also keeps track of each snapshot of the CAL-ACCESS database released by the California Secretary of State, including its release date and byte size, as well as the activity of the management commands that process this data.

This tracking information is stored in the data tables outlined below.

Note: By default, the raw-data app does *not* archive previous versions of the CAL-ACCESS database. Rather, with each call to the management commands, the data files they process are overwritten.

You can configure the raw-data app to keep each copy of the zip file downloaded from the California Secretary of State as well as the indivdual raw .csv files and cleaned .tsv files by flipping the `CALACCESS_STORE_ARCHIVE` to `True` in `settings.py`:

```
# in settings.py
CALACCESS_STORE_ARCHIVE = True
```

By default, the older copies of these files will be saved to the path specified by your Django project's `MEDIA_ROOT` setting (more on that here). However, if you've implemented a custom storage system or installed a third-party app (such as django-storages), that should work too.

RawDataVersion

Versions of CAL-ACCESS raw source data, typically released every day.

Fields

Instance methods and properties

Query set methods

`.complete()`

Filters down QuerySet to return only version that have a complete update.

```
$ python manage.py shell
>>> from calaccess_raw.models.tracking import RawDataVersion
>>> RawDataVersion.objects.completed()
<QuerySet [<RawDataVersion: 2016-08-15 11:20:29+00:00>, <RawDataVersion: 2016-08-11␣
↪11:20:24+00:00>, <RawDataVersion: 2016-08-09 11:20:49+00:00>, <RawDataVersion: 2016-
↪08-05 11:20:27+00:00>, <RawDataVersion: 2016-08-04 11:20:28+00:00>,
↪<RawDataVersion: 2016-07-31 11:20:29+00:00>, <RawDataVersion: 2016-07-30␣
↪11:20:42+00:00>, <RawDataVersion: 2016-07-29 11:20:30+00:00>, <RawDataVersion: 2016-
↪07-28 11:20:30+00:00>, <RawDataVersion: 2016-07-26 11:20:28+00:00>,
↪<RawDataVersion: 2016-07-22 11:20:30+00:00>, <RawDataVersion: 2016-07-05␣
↪11:20:30+00:00>, <RawDataVersion: 2016-07-04 11:20:30+00:00>, <RawDataVersion: 2016-
↪06-28 11:20:28+00:00>, <RawDataVersion: 2016-06-14 11:20:49+00:00>,
↪<RawDataVersion: 2016-06-10 11:20:26+00:00>, <RawDataVersion: 2016-06-08␣
↪11:20:29+00:00>, <RawDataVersion: 2016-05-27 11:20:28+00:00>, <RawDataVersion: 2016-
↪05-21 15:35:11+00:00>, <RawDataVersion: 2016-05-20 13:59:57+00:00>, '...(remaining␣
↪elements truncated)...']>
```

RawDataFile

Data files included in the given version of the CAL-ACCESS raw source data.

Fields

Instance methods and properties

Changelog

2.0.0 (August 2018)

- Dropped MySQL support
- Refactored clean and loading commands for readability
- Added Django 2.1 support
- Added Python 3.7 support

1.6.2 (October 2017)

- Renamed raw data zipfile as *raw.zip* to match the language on our revamped downloads page

1.6.1 (September 2017)

- Upgrade to v2.0.0 of django-postgres-copy and refactor export commands accordingly
- Require Django 1.10 or above
- Correct documentation for a number of fields

1.6.0 (July 2017)

- Deprecate `--test, --use-test-data` options from `updatecalaccessrawdata` command.
- Renamed custom Django project setting `CALACCESS_DOWNLOAD_DIR` to `CALACCESS_DATA_DIR`.
- Removed `CALACCESS_TEST_DOWNLOAD_DIR` project setting.
- Extract and track any .TSV file regardless of location in download .ZIP directory tree.

1.5.2 (April 2017)

- Fix duplicate updates. Only create a new `RawDataVersion` if:
  - `Content-length` from `HEAD` differs from `expected_size` on previous version, or,
  - `Last-modified` is at least five minutes more recent than `release_datetime` on previous version.

1.5.1 (April 2017)

- Proceed with download of ZIP file as long as last-modified datetimes in `HEAD` and `GET` requests are within five minutes of each other.

1.5.0 (April 2017)

- Django 1.11 compatibility.
- Fix check for existing clean zipped file when resuming.
- Fix message on response status code log.
- Skip dropping/re-adding of database table constraints and indexes when loading into MySQL (transactional DDL statements are not supported).

1.4.9 (March 2017)

- Reset auto-increment fields after truncating database tables in postgres.
- Add prefixes on tracking model admins.
- When making requests to sos.ca.gov, log HTTP status code and reason and raise HTTP error if bad status.

1.4.8 (January 2017)

- Upgrade to `csvkit` version 1.0.

1.4.7 (December 2016)

- Fixed search field on admins for models with `ForeignKey` fields.

1.4.6 (November 2016)

- Upgraded to latest version of django-postgres-copy
- Small improvements to CAL-ACCESS field documentation
- Small expansion of unittests
- Clean up of migrations

1.4.5 (September 2016)

- Copyediting of CAL-ACCESS form documentation

1.4.2 (late-August 2016)

- Docstring edits

1.4.1 (late-August 2016)

- Increase max character length on `ReceivedFilingsCd` fields.
- Prevent unnecessary download of zip when resuming `updatecalaccessrawdata`.
- Include release datetimes in log when `downloadcalaccessrawdata` and `updatecalaccessrawdata` versions are incompatible.

1.4.0 (mid-August 2016)

- Added zipping up and archiving of cleaned CSVs and error logs.
  - Added `RawDataVersion.clean_zip_archive` FileField.
  - Renamed `RawDataVersion.zip_file_archive` to `RawDataVersion.download_zip_archive`.
- Smaller clean data files (removed unnecessary quote characters).
- Improvements to tracking models
  - Replaced `RawDataCommand` model with datetime fields and related properties
    - Added to `RawDataVersion` instances
      - `.update_start_datetime` and `.update_finish_datetime` to store version's most recent update start and finish datetimes.
      - `.update_completed` returns `True` if most recent update to version started and finished.
      - `.update_stalled` returns `True` if most recent update to version started but did not finish.
      - `.download_start_datetime` and `.download_finish_datetime` to store version's most recent download start and finish datetimes.
      - `.download_completed` returns `True` if most recent download of version started and finished.
      - `.download_stalled` returns `True` if most recent download version started but did not finish.
      - `.completed()` QuerySet method to `RawDataVersion` to get all versions where the update completed.
    - Added to `RawDataFile` instances
      - `.clean_start_datetime` and `.clean_finish_datetime` to store raw file's most recent clean start and finish datetimes.
      - `.load_start_datetime` and `.load_finish_datetime` to store raw file's most recent load start and finish datetimes.
  - Expanded file size tracking
    - Renamed `.size` to `.expected_size` on `RawDataVersion` instances.
    - Added `.download_zip_size` to `RawDataVersion` instances.
    - Added `.clean_zip_size` to `RawDataVersion` instances.
    - Added methods to get a pretty version (e.g., `723M`) of each file size field
      - Added to `RawDataVersion` instances
      - `.pretty_expected_size()`

            - .pretty_download_size()
            - .pretty_clean_size()
            - Added RawDataFile instances
            - .pretty_download_file_size()
            - .pretty_clean_file_size()
        * Raise CommandError if completed download file size is not the same as expected size.
        * Added RawDataVersion properties to calculate file and record counts:
            - .download_file_count
            - .download_record_count
            - .clean_file_count
            - .clean_record_count
            - .error_file_count
            - .error_count
- Added extractcalaccessrawfiles management command for unzipping and extracting raw data files from downloaded CAL-ACCESS database export.
    - Start and finish times stored in .start_extract_datetime and .finish_extract_datetime on RawDataVersion instances.
- Bug fixes.
    - In‘‘downloadcalaccessrawdata‘‘, skip download if the size of the local zip file is equal to or bigger than the expected zip file size.
    - Because the server hosting the ZIP doesn't always provide the most up-to-date resource (as we have documented), a CommandError will be raised under any of the following conditions:
        * If downloadcalaccessrawdata is not called from the command-line (presumably, then, it was called by updatecalaccessrawdata), and the RawDataVersion instance of the download command doesn't match the most recently started update.
        * If the ETag in the initial HEAD request made by downloadcalaccessrawdata does not match the ETag in the subsequent GET request.
        * If the actual size of the ZIP does not match the value of the Content-Length in the HEAD response.
    - If downloadcalaccessrawdata raises any of the above errors, updatecalaccessrawdata will wait five minutes and try again.
    - When archiving zips and files, open in binary ('rb') mode.
    - In cleancalaccessrawfile, fixed skipping of empty lines for Python 3.5.
- Support for Django 1.10.

1.3.0 (July 2016)

- Now distributing on wheels.
- Added error_count to output reportcalaccessrawdata and excluded any unspecified fields.
- Added model property to RawDataFile that returns the CalAccess model object.

1.2.0 (July 2016)

- Enhancements to tracking models
  - Zero pad datetime parts of archive dir (for better sorting)
  - Calculate and store load_columns_count and load_records_count in `loadcalaccessrawfile`
  - Added error_count and error_log_archive fields to RawDataFile in order to track bad line parses during `cleancalaccessrawfile`.
  - Added download_file_size and clean_file_size to RawDataFile.
- Enhancements to CalAccess models
  - Added inactive models group for CAL-ACCESS tables that are empty or apparently no longer in use.
  - Added a CalAccessMetaClass to automatically configure meta attributes common to all models.
  - Added a custom admin for every model.
  - Model verbose names are pre-fixed with model groups
  - Edits to model doc strings.
- Enhancements to management commands
  - Added standard logging.
  - Added a logger.info to the end of the `updatecalaccessrawdata` command to allow sending of emails when finished
  - Edits to command doc strings.
- More tests
  - Test to confirm that any field included in a model's `UNIQUE_KEY` attribute actually exists on the model.
  - Test to confirm that every model has a custom admin.
- Bug fixes
  - Fixed numbers in clean_records_count for RawDataFile.
  - Fixed line numbers logged in errors.csv files.
  - Write output of `reportcalaccessrawdata` to data directory instead of `REPO_DIR`, which may not be in settings.

1.1.0 (late June 2016)

- When *–noinput* is invoked for `updatecalaccessrawdata`, exit if previously updated to the currently available version.
- Enforce lowercase UNIQUE_KEY settings on models.
- Removed unnecessary pretty amount model methods as part of driving common.py models file test coverage up to 100%.

1.0.2 (early June 2016)

- Include migrations in official package.
- Fix `verbose_name` for `RawDataFile.clean_file_archive`.

1.0.0 (May 2016)

- Enhanced resume behavior
  - Allow previously interrupted updates to resume at any stage of the process: downloading, cleaning or loading.
  - Users will be prompted to resume (if possible). User may decline and re-start the entire update.
  - Removed `--resume-download` option from `updatecalaccessrawdata` and `downloadcalaccessrawdata` in favor of prompting the user to resume.
  - Removed `--database` option from all commands. Multi-database users are encouraged to use Django's database routers.
- Raw data file archiving
  - Added `CALACCESS_STORE_ARCHIVE` setting. When enabled, management commands will save each version of the downloaded .zip file, the extracted .tsv files and cleaned .csv files to the Django project's `MEDIA_ROOT`.
  - Added FileFields to RawDataVersion and RawDataFile in order to link the database records with the archived files they reference.
- Completed documentation of all 80 raw data models and 1,467 fields
  - Defined hundreds of choices for 182 look-up fields.
  - Published expanded Django project documentation. Added re-directs from old app-specific documentation.
  - Integrated references to official documents and filing forms into data models. PDFs on DocumentCloud.
- Expanded unit testing of data model documentation
  - Wider scope of choice field testing.
  - Verify that each model has a `UNIQUE_KEY` attribute set.
  - Verify that each model has a document reference.
  - Verify that each choice field has a document reference.
  - Verify that each model with a form_type or form_id field (with a few exceptions) is linked to filing forms.
  - Introduced `reportcalaccessrawdata` command, which generates a report outlining the number / proportion of files / records cleaned and loaded.
- Model Re-modeling:
  - Moved `BallotMeasuresCd` from `other.py` to `campaign.py`. Same with admin.
  - Moved remaining models in `other.py` to `common.py`. Removed `other.py`. Same with admins.
  - Re-ordered models into related groups.
- Bug fixes
  - Truncate time portions of raw datetime values (see #1457).
  - Strip newlines when loading into MySQL.

0.2.0 (January 2016)

- Support for Python 3.5
- Support for Django 1.9
- Simplified downloadcalaccessrawdata. Now only downloads, unzips and preps
- Introduced updatecalaccessrawdata, which downloads, cleans and loads data
- Added –resume-download option in case download is interrupted
- Added –csv option to loadcalaccessrawfile so that users can load from a file other than the one specified for the given calaccess_raw model
- Added –keep-files option. Unless the option is invoked downloadcalaccessrawdata, cleancalaccessrawfile, loadcalaccessrawfile and updatecalaccessrawdata now clear out original and intermediate files
- Support for multiple databases configured in Django DATABASE settings. Users can now load into a specified database using –database option
- Fixed verifycalaccessrawfile
- Updated management command options to most recent Django style, using argparse instead of optparse
- Hundreds of unique keys, field defs and choices patched by Code Rushers
- Automatically generated table documentation page
- Expanded documentation

0.1.2 (February 2015)

- Substituted clint for progressbar
- Improved choices for form type fields

0.1.1 (January 2015)

- Datetime support for MySQL fields
- Fixed bug that didn't allow null values in PostgreSQL datetime fields

0.1.0 (November 2014)

- Support for PostgreSQL database backends
- Upgraded to Django 1.7
- Prettified management command output and logging
- Improved docs, admins and configuration for some campaign-finance models
- Numerous small bug fixes and documentation corrections

0.0.7 (August 2014)

- Complete set of models that cover 100% of source CSV files
- Management commands that prep and load the data for MySQL backends
- Administration panels for previewing the data

Open-source resources

- Code: github.com/california-civic-data-coalition/django-calaccess-raw-data
- Issues: github.com/california-civic-data-coalition/django-calaccess-raw-data/issues
- Packaging: pypi.python.org/pypi/django-calaccess-raw-data
- Testing: travis-ci.org/california-civic-data-coalition/django-calaccess-raw-data
- Coverage: coveralls.io/r/california-civic-data-coalition/django-calaccess-raw-data

1.3.2 django-calaccess-scraped-data

A Django app to scrape campaign-finance data from the California Secretary of State's CAL-ACCESS website.

Management commands

The scraped-data app includes the following commands for scraping campaign finance data from the CAL-ACCESS website.

As with any Django app management command, these can be invoked on the command line or called within your Python code.

Raw content downloaded from CAL-ACCESS is stored in `.scraper_cache/`, found in the directory specified by `BASE_DIR` in your Django project's settings.

---

scrapecalaccess

This command runs the following management commands, in order:

1. `scrapecalaccesspropositions`
2. `scrapecalaccesscandidates`
3. `scrapecalaccessincumbents`

These commands are defined in more detail below.

Examples

The default behavior of the scraper commands is to avoid excessive downloads. As such, a CAL-ACCESS web page's content will only be downloaded if:

1. The page's content isn't cached; or

2. The byte size of the cached content differs from the size of the content on the server (as specified in `Content-Length` header).

You can override this default behavior by invoking the `force-download` option:

```
$ python manage.py scrapecalaccess --force-download
```

Alternatively, you can avoid making *any* network requests by invoking the `--cache-only` option so as to parse and store data only from previously cached content:

```
$ python manage.py scrapecalaccess --cache-only
```

By default, data saved to your database from previous scrapes is preserved, or you can invoke the `--flush` option to start over with empty data tables:

```
$ python manage.py scrapecalaccess --flush
```

Options

```
usage: manage.py scrapecalaccess [-h] [--version] [-v {0,1,2,3}]
                                 [--settings SETTINGS]
                                 [--pythonpath PYTHONPATH] [--traceback]
                                 [--no-color] [--flush] [--force-download]
                                 [--cache-only]

Run all scraper commands

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush database tables
  --force-download      Force the scraper to download URLs even if they are cached
  --cache-only          Skip the scraper's update checks. Use only cached
                        files.
```

---

scrapecalaccesscandidates

Scrape certified candidates for each election on the CAL-ACCESS site. A component of the `scrapecalaccess` command.

This command requests and parses content from the "certified" view of the `Campaign/Candidates/list.aspx` page (e.g., the 2016 General certified candidates). Data parsed from these pages are saved in the `CandidateElection` and `Candidate` models.

Examples

Here is how to run the command.

```
$ python manage.py scrapecalaccesscandidates
```

Options

```
usage: manage.py scrapecalaccesscandidates [-h] [--version] [-v {0,1,2,3}]
                                           [--settings SETTINGS]
                                           [--pythonpath PYTHONPATH]
                                           [--traceback] [--no-color]
                                           [--flush] [--force-download]
                                           [--cache-only]

Scrape certified candidates for each election on the CAL-ACCESS site.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush database tables
  --force-download      Force the scraper to download URLs even if they are
                        cached
  --cache-only          Skip the scraper's update checks. Use only cached
                        files.
```

---

scrapecalaccesscandidatecommittees

Scrape each candidate's committees from the CAL-ACCESS site.

This command requests and parses content from the "general" view of the `Campaign/Candidates/Detail.aspx` page for candidate's most recent "session" (e.g., Edward T. Gaines general information leading up to the 2016 General election). Data parsed from these pages are saved in the `CandidateCommittee` model.

Note: The `scrapecalaccesscandidatecommittees` command is not currently included in `scrapecalaccess` because of the number of CAL-ACCESS web pages it scrapes. This may change in the future.

Examples

Here is how to run the command.

```
$ python manage.py scrapecalaccesscandidatecommittees
```

Options

```
usage: manage.py scrapecalaccesscandidatecommittees [-h] [--version]
                                                    [-v {0,1,2,3}]
                                                    [--settings SETTINGS]
                                                    [--pythonpath PYTHONPATH]
                                                    [--traceback] [--no-color]
                                                    [--flush]
                                                    [--force-download]
                                                    [--cache-only]

Scrape each candidate's committees from the CAL-ACCESS site.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush database tables
  --force-download      Force the scraper to download URLs even if they are
                        cached
  --cache-only          Skip the scraper's update checks. Use only cached
                        files.
```

scrapecalaccessincumbents

Scrape list of incumbent state officials for each election on CAL-ACCESS site. A component of the `scrapecalaccess` command.

This command requests and parses content from the "incumbent" view of the `Campaign/Candidates/list.aspx` page (e.g., the 2017-2018 General incumbents). Data parsed from these pages are saved in the `IncumbentElection` and `Incumbent` models.

Examples

Here is how to run the command.

```
$ python manage.py scrapecalaccessincumbents
```

Options

```
usage: manage.py scrapecalaccessincumbents [-h] [--version] [-v {0,1,2,3}]
                                           [--settings SETTINGS]
                                           [--pythonpath PYTHONPATH]
                                           [--traceback] [--no-color]
                                           [--flush] [--force-download]
                                           [--cache-only]

Scrape list of incumbent state officials for each election on CAL-ACCESS site.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush database tables
  --force-download      Force the scraper to download URLs even if they are
                        cached
  --cache-only          Skip the scraper's update checks. Use only cached
                        files.
```

---

scrapecalaccesspropositions

Scrape links between filers and propositions from the official CAL-ACCESS site. A component of the `scrapecalaccess` command.

This command requests and parses content from the `Campaign/Measures/list.aspx` page (e.g., the 2015-2016 propositions and ballot measures) and "general" view of each propositions `Campaign/Measures/Detail.aspx` page (e.g., Prop 60's general information). Data parsed from these pages are saved in the `PropositionElection`, `Proposition` and `PropositionCommittee` models.

Examples

```
$ python manage.py scrapecalaccesspropositions
```

Options

```
usage: manage.py scrapecalaccesspropositions [-h] [--version] [-v {0,1,2,3}]
                                             [--settings SETTINGS]
                                             [--pythonpath PYTHONPATH]
                                             [--traceback] [--no-color]
                                             [--flush] [--force-download]
                                             [--cache-only]

Scrape links between filers and propositions from the official CAL-ACCESS
site.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush database tables
  --force-download      Force the scraper to download URLs even if they are
                        cached
  --cache-only          Skip the scraper's update checks. Use only cached
                        files.
```

Changelog

0.1.2 (November 2017)

- Added five second timeout to requests made to CAL-ACCESS

0.1.1 (October 2017)

- Updated dependencies
- Added fixtures to distribution

0.1.0 (July 2017)

- Initial release of management commands and models for scraping:
  - Ballot propositions (see scrapecalaccesspropositions)
  - "Certified" candidates (see scrapecalaccesscandidates)
  - Committees linked to certified candidates in each candidate's most recent election cycle (see scrapecalaccesscandidatecommittees)
  - Incumbent officeholders (see scrapecalaccessincumbents)
  - The entire routine (see scrapecalaccess)
- Includes admin panels for previewing any scraped data

Open-source resources

- Code: github.com/california-civic-data-coalition/django-calaccess-raw-data
- Issues: github.com/california-civic-data-coalition/django-calaccess-scraped-data/issues
- Packaging: pypi.python.org/pypi/django-calaccess-scraped-data
- Testing: travis-ci.org/california-civic-data-coalition/django-calaccess-scraped-data
- Coverage: coveralls.io/r/california-civic-data-coalition/django-calaccess-scraped-data

1.3.3 django-calaccess-processed-data

A Django app to clean, transform and refine campaign-finance and lobbying-activity data from the California Secretary of State's CAL-ACCESS database.

Management commands

The processed-data app includes the following commands for refining data extracted and scraped from CAL-ACCESS. Specifically, the raw data is loaded into the following types of models:

- `Filing` models that surface the most recent version of data included on a campaign-finance filing form, schedule or line item (e.g., a Form 460, its Schedule A or Line 1 on that schedule).
- `FilingVersion` models that surface every version of a campaign-finance filing form, schedule or line item.
- Models that implement the core data types of the Open Civic Data specification (e.g., `Person`, `Organization`, `Post` and `Membership`).
- Models that implement election-related data types that have been provisionally included in the Open Civic Data specification (e.g., `Election`, `CandidateContest` and `Candidacy`).

As with any Django app management command, these can be invoked on the command line or called within your Python code.

Note: Before using any of the commands below, make sure you need to download and extract the raw CAL-ACCESS data:

```
$ python manage.py updatecalaccessrawdata
```

And scrape supplementary data from the CAL-ACCESS website:

```
$ python manage.py scrapecalaccess
```

processcalaccessdata

This is the master command. It brings together all of the other management commands listed below to load data into processed CAL-ACCESS models.

If your Django project is configured for archiving (details here), this command also will export a csv file for each loaded model.

Examples

Running the entire routine is as simple as this.

```
$ python manage.py processcalaccessdata
```

If a previous processing job stalled for any reason, `processcalaccessdata` will pick up wherever you left off. You can override this behavior by invoking the `force-restart` option.

```
$ python manage.py processcalaccessdata --force-restart
```

Options

```
usage: manage.py processcalaccessdata [-h] [--version] [-v {0,1,2,3}]
                                      [--settings SETTINGS]
                                      [--pythonpath PYTHONPATH] [--traceback]
                                      [--no-color] [--force-restart]
                                      [--no-scrape]

Load data into processed CAL-ACCESS models, archive processed files and ZIP.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --force-restart, --restart
                        Force re-start (overrides auto-resume).
```

archivecalaccessprocessedfile

Export and archive a .csv file for a given model.

Examples

You must provide the `model_name` as the first and only positional argument. For example, here's how you archive the `Form460Filing` model:

```
$ python manage.py archivecalaccessprocessedfile Form460Filing
```

Or the `Candidacy` model, which is one Open Civic Data's election-related data types:

```
$ python manage.py archivecalaccessprocessedfile Candidacy
```

Options

```
usage: manage.py archivecalaccessprocessedfile [-h] [--version] [-v {0,1,2,3}]
                                               [--settings SETTINGS]
                                               [--pythonpath PYTHONPATH]
                                               [--traceback] [--no-color]
                                               model_name

Export and archive a .csv file for a given model.

positional arguments:
  model_name            Name of the model to archive

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

---

loadcalaccessfilings

Load the CAL-ACCESS `Filing` and `FilingVersion` models. A component of the master `processcalaccessdata` command.

If your Django project is configured for archiving (details here), this command also will export a csv file for each loaded model.

Examples

Here is how to run the command.

```
$ python manage.py loadcalaccessfilings
```

This command will skip any `Filing` or `FilingVersion` models already loaded with raw data from the current CAL-ACCESS snapshot. You can override this behavior by invoking the `force-restart` option.

```
$ python manage.py processcalaccessdata --force-restart
```

Options

```
usage: manage.py loadcalaccessfilings [-h] [--version] [-v {0,1,2,3}]
                                      [--settings SETTINGS]
                                      [--pythonpath PYTHONPATH] [--traceback]
                                      [--no-color] [--force-restart]

Load and archive the CAL-ACCESS Filing and FilingVersion models.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --force-restart, --restart
                        Force re-start (overrides auto-resume).
```

---

loadocdelections

Load OCD elections models with data extracted and scraped from CAL-ACCESS. A component of the master `processcalaccessdata` command.

This command runs the following management commands, in order:

1. `loadocdparties`
2. `loadocdballotmeasureelections`
3. `loadocdballotmeasurecontests`
4. `loadocdretentioncontests`
5. `loadocdcandidateelections`

6. `loadocdcandidatecontests`
7. `mergeocdpersonsbyfilerid`
8. `loadocdcandidaciesfrom501s`
9. `mergeocdpersonsbycontestandname`
10. `loadocdincumbentofficeholders`

If your Django project is configured for archiving (details here), this command also will export a csv file for each loaded model.

Examples

Here is how to run the command.

```
$ python manage.py loadocdelections
```

Options

```
usage: manage.py loadocdelections [-h] [--version] [-v {0,1,2,3}]
                                  [--settings SETTINGS]
                                  [--pythonpath PYTHONPATH] [--traceback]
                                  [--no-color]

Load OCD elections models with data extracted and scraped from CAL-ACCESS.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

---

loadocdballotmeasureelections

Load the OCD `Election` model from the scraped `PropositionElection` model. A component of the `loadocdelections` command.

Examples

Here is how to run the command.

```
$ python manage.py loadocdballotmeasureelections
```

Options

```
usage: manage.py loadocdballotmeasureelections [-h] [--version] [-v {0,1,2,3}]
                                               [--settings SETTINGS]
                                               [--pythonpath PYTHONPATH]
                                               [--traceback] [--no-color]

Load the OCD Election model from the scraped PropositionElection model

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

---

loadocdballotmeasurecontests

Load OCD `BallotMeasureContest` and related models with scraped CAL-ACCESS data. A component of the `loadocdelections` command.

Note: Use `loadocdballotmeasureelections` before using `loadocdballotmeasurecontests`.

Examples

Here is how to run the command.

```
$ python manage.py loadocdballotmeasurecontests
```

Options

```
usage: manage.py loadocdballotmeasurecontests [-h] [--version] [-v {0,1,2,3}]
                                              [--settings SETTINGS]
                                              [--pythonpath PYTHONPATH]
                                              [--traceback] [--no-color]
```

(continues on next page)

(continued from previous page)

```
                                                [--flush]

Load OCD BallotMeasureContest and related models with scraped CAL-ACCESS data

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush the database tables filled by this command.
```

loadocdcandidateelections

Load the OCD `Election` model with data from the scraped `CandidateElection` model. A component of the `loadocdelections` command.

Examples

Here is how to run the command.

```
$ python manage.py loadocdcandidateelections
```

Options

```
usage: manage.py loadocdcandidateelections [-h] [--version] [-v {0,1,2,3}]
                                           [--settings SETTINGS]
                                           [--pythonpath PYTHONPATH]
                                           [--traceback] [--no-color]
                                           [--flush]

Load the OCD Election model with data from the scraped CandidateElection model.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
```

(continues on next page)

(continued from previous page)

```
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush the database tables filled by this command.
```

loadocdcandidatecontests

Load the OCD `CandidateContest` and related models with scraped CAL-ACCESS data. A component of the `loadocdelections` command.

This command loads data from the `IncumbentElection` and `CandidateElection` models in `calaccess_scraped`.

Note: Use `loadocdcandidateelections` and `loadocdparties` before using `loadocdcandidatecontests`.

Examples

Here is how to run the command.

```
$ python manage.py loadocdcandidatecontests
```

Options

```
usage: manage.py loadocdcandidatecontests [-h] [--version] [-v {0,1,2,3}]
                                          [--settings SETTINGS]
                                          [--pythonpath PYTHONPATH]
                                          [--traceback] [--no-color] [--flush]

Load the OCD CandidateContest and related models with scraped CAL-ACCESS data

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
```

(continues on next page)

(continued from previous page)

```
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush the database tables filled by this command.
```

loadocdcandidaciesfrom501s

Load the OCD `Candidacy` model with data extracted from the `Form501Filing` model. A component of the `loadocdelections` command.

This command fills in `Candidacy` records with data missing on the CAL-ACCESS website (e.g., the candidate's party in each contest). It also adds additional `Candidacy` records.

Examples

Here is how to run the command.

```
$ python manage.py loadocdcandidaciesfrom501s
```

Options

```
usage: manage.py loadocdcandidaciesfrom501s [-h] [--version] [-v {0,1,2,3}]
                                            [--settings SETTINGS]
                                            [--pythonpath PYTHONPATH]
                                            [--traceback] [--no-color]

Load the OCD Candidacy model with data extracted from the Form501Filing model.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

loadocdincumbentofficeholders

Load the OCD `Membership` model with data from the scraped Incumbent model. A component of the `loadocdelections` command.

Note: Use `loadocdcandidateelections` before using `loadocdincumbentofficeholders`.

Examples

Here is how to run the command.

```
$ python manage.py loadocdincumbentofficeholders
```

Options

```
usage: manage.py loadocdincumbentofficeholders [-h] [--version] [-v {0,1,2,3}]
                                               [--settings SETTINGS]
                                               [--pythonpath PYTHONPATH]
                                               [--traceback] [--no-color]

Load the OCD Membership model with data from the scraped Incumbent model

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

loadocdretentioncontests

Load OCD `RetentionContest` and related models with data scraped from CAL-ACCESS. A component of the `loadocdelections` command.

Note: Use `loadballotmeasureelections` before using `loadocdretentioncontests`.

Examples

Here is how to run the command.

```
$ python manage.py loadocdretentioncontests
```

Options

```
usage: manage.py loadocdretentioncontests [-h] [--version] [-v {0,1,2,3}]
                                          [--settings SETTINGS]
                                          [--pythonpath PYTHONPATH]
                                          [--traceback] [--no-color] [--flush]

Load OCD RetentionContest and related models with data scraped from CAL-ACCESS

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush the database tables filled by this command.
```

---

loadocdparties

Load OCD `Organization` model with parties extracted from raw CAL-ACCESS data. A component of the `loadocdelections` command.

Examples

Here is how to run the command.

```
$ python manage.py loadocdparties
```

Options

```
usage: manage.py loadocdparties [-h] [--version] [-v {0,1,2,3}]
                                [--settings SETTINGS]
                                [--pythonpath PYTHONPATH] [--traceback]
```

(continues on next page)

(continued from previous page)

```
                                [--no-color] [--flush]

Load OCD Organization model with parties extracted from raw CAL-ACCESS data.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --flush               Flush the database tables filled by this command.
```

mergeocdpersonsbycontestandname

Find and merge OCD `Person` records that share a name and `CandidateContest`. A component of the `loadocdelections` command.

Examples

Here is how to run the command.

```
$ python manage.py mergeocdpersonsbycontestandname
```

Options

```
usage: manage.py mergeocdpersonsbycontestandname [-h] [--version]
                                                 [-v {0,1,2,3}]
                                                 [--settings SETTINGS]
                                                 [--pythonpath PYTHONPATH]
                                                 [--traceback] [--no-color]

Find and merge OCD Person records that share a name and CandidateContest

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
```

(continues on next page)

(continued from previous page)

```
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

mergeocdpersonsbyfilerid

Find and merge OCD `Person` records that share the same CAL-ACCESS filer_id. A component of the `loadocdelections` command.

Examples

Here is how to run the command.

```
$ python manage.py mergeocdpersonsbyfilerid
```

Options

```
usage: manage.py mergeocdpersonsbyfilerid [-h] [--version] [-v {0,1,2,3}]
                                          [--settings SETTINGS]
                                          [--pythonpath PYTHONPATH]
                                          [--traceback] [--no-color]

Find and merge OCD Person records that share the same CAL-ACCESS filer_id

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

Models for tracking updates

The processed-data app also keeps track of each snapshot of CAL-ACCESS database it processes. This tracking information is stored in the data tables outlined below.

Note: By default, the processed-data app does *not* archive previous versions of the CAL-ACCESS database. Rather, with each call to the management commands, the data files they process are overwritten.

You can configure the raw-data app to keep each copy of the zip file downloaded from the California Secretary of State as well as the indivdual raw .csv files and cleaned .tsv files by flipping the `CALACCESS_STORE_ARCHIVE` to `True` in `settings.py`:

```
# in settings.py
CALACCESS_STORE_ARCHIVE = True
```

By default, the older copies of these files will be saved to the path specified by your Django project's `MEDIA_ROOT` setting (more on that here). However, if you've implemented a custom storage system or installed a third-party app (such as django-storages), that should work too.

ProcessDataVersion

Versions of CAL-ACCESS raw source data, typically released every day.

Fields

Instance methods and properties

ProcessedDataFile

A data file included in a processed version of CAL-ACCESS.

Fields

Instance methods and properties

Changelog

0.1.0 (July 2017)

- Initial release of management commands that load CAL-ACCESS filing and Open Civic Data models.

Open-source resources

- Code: github.com/california-civic-data-coalition/django-calaccess-processed-data
- Issues: github.com/california-civic-data-coalition/django-calaccess-processed-data/issues
- Packaging: pypi.python.org/pypi/django-calaccess-processed-data
- Testing: travis-ci.org/california-civic-data-coalition/django-calaccess-processed-data
- Coverage: coveralls.io/r/california-civic-data-coalition/django-calaccess-processed-data

1.3.4 django-calaccess-downloads-website

An open-source archive of campaign-finance and lobbying-activity data from the California Secretary of State's CAL-ACCESS database.

Management commands

The downloads-website app includes the following commands for updating and publishing the website's content.

Our website is one of those trendy static content sites that you've probably heard a lot about lately. This just means that, instead of generating HTML on-the-fly with each request from a user's browser, we create and save all the web pages ahead of time by executing Python code against the database backend once a day.

This process is often called "baking", and there's a really handy app that we rely on to make all this work.

As with any Django app management command, these can be invoked on the command line or called within your Python code.

---

updatedownloadswebsite

Update to the latest CAL-ACCESS snapshot and bake static website pages.

This is the master command that performs the entire daily routine of downloading, processing and archiving the latest raw data, then re-building the downloads website's content.

```
$ python manage.py updatedownloadswebsite
```

In order to publish this content to the S3 bucket where it's served, you can invoke the `--publish` option:

```
$ python manage.py updatedownloadswebsite --publish
```

Also, this command is a sub-class of the raw-data app's `updatecalaccessrawdata` command, so it inherits all the options of the parent command. For example, if you want to keep copies of the latest raw data files on the app's server, you can:

```
$ python manage.py updatedownloadswebsite --keep-files
```

The other options are below.

Options

```
usage: manage.py updatedownloadswebsite [-h] [--version] [-v {0,1,2,3}]
                                        [--settings SETTINGS]
                                        [--pythonpath PYTHONPATH]
                                        [--traceback] [--no-color]
                                        [--keep-files] [--noinput] [--test]
                                        [-a APP_NAME] [--publish]

Update to the latest CAL-ACCESS snapshot and bake static website pages

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
```

(continues on next page)

(continued from previous page)

```
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
  --keep-files          Keep zip, unzipped, TSV and CSV files
  --noinput             Download the ZIP archive without asking permission
  --test, --use-test-data
                        Use sampled test data (skips download, clean a load)
  -a APP_NAME, --app-name APP_NAME
                        Name of Django app with models into which data will be
                        imported (if other not calaccess_raw)
  --publish             Publish baked content
```

createlatestlinks

Save copies of data files from the most recently completed update in a latest/ directory in the Django project's default file storage.

```
$ python manage.py createlatestlinks
```

This command will also clear out any objects currently saved under latest/ before saving new ones.

Options

```
usage: manage.py createlatestlinks [-h] [--version] [-v {0,1,2,3}]
                                   [--settings SETTINGS]
                                   [--pythonpath PYTHONPATH] [--traceback]
                                   [--no-color]

Save copies of data files from the most recently completed update in a latest
directory in the default file storage of the Django project.

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v {0,1,2,3}, --verbosity {0,1,2,3}
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Raise on CommandError exceptions
  --no-color            Don't colorize the command output.
```

Fab tasks index

We deploy and manage the downloads website infrastructure using Fabric, which makes processes like deploying the entire downloads website as simple as invoking a few commands from the command-line.

Below is the complete list of available Fabric tasks.

Note: Fabric allows you to run one task after another in a single fab command-line call like so:

```
$ fab task1:pos_arg1 task2:opt_arg=some_value
```

This can be useful for chaining tasks together for ad-hoc administrative processes. Read more here.

Amazon

Tasks for managing Amazon Web Service (AWS) resources.

createec2

Spin up a new Ubuntu 14.04 server on Amazon EC2. Returns the id and public address.

```
$ fab createec2
```

The address for your new EC2 instance will also be added to your current environment's configuration (stored in `.env`). If you already have an EC2 host set in your current env, its address will be replaced.

Optional arguments:

- `instance_name` (default is `calaccess_website`)
- `block_gb_size` (default is `100`)
- `instance_type` (default is `c3.large`)
- `ami` (default is `ami-978dd9a7`)

createkey

Creates an EC2 key pair and saves it to a .pem file.

The `name` for the key pair is the only positional argument:

```
$ fab createkey:ccdc-key
```

You'll be stopped if you try to re-use an existing key pair name.

A new key pair will then be stored in `~/.ec2/<your-key-name>.pem`, and the key pair name will be added to your current environment's configuration (stored in `.env`). If you already have a key name set in your current env, it will be replaced.

createrds

Spin up a new database backend with Amazon RDS.

The `instance_name` is the only positional argument:

```
$ fab createrds:downloads-website
```

This may take several minutes.

The address for your new RDS instance will be added to your current environment's configurations (stored in `.env`). If you already have an RDS host set in your current env, its address will be replaced.

Optional arguments:

- `database_port` (default is `5432`)
- `block_gb_size` (default is `100`)
- `instance_type` (default is `db.t2.large`)

copydb

Copy the most recent snapshot on the source AWS RDS instance to the destination RDS instance.

The positional arguments are:

- `src_db_instance_id`, which identifies the source instance from which to create a copy
- `dest_db_instance_id`, which identifies the destination instance for the copy.

Warning: The current database on the destination instance will be deleted.

You might execute this task if, for example, you want to replicate the production database to a dev instance.

```
$ fab copydb:prod-db,dev-db
```

The process may take several minutes to complete.

If you would like to create a new snapshot of the source db instance before making a copy, you can pass in `make_snapshot=True`.

copys3

Copy objects in the source AWS S3 bucket to the destination S3 bucket.

Ignores source bucket objects with the same name as objects already in the destination bucket.

The positional arguments are:

- `src_bucket`, which identifies the bucket *from* which objects will be copied.
- `dest_bucket`, which identifies the bucket *to* which objects will be copied.

You might execute this task if, for example, you want to replicate the production archived data bucket to a dev instance.

```
$ fab copys3:prod-archived-data,dev-archived-data
```

The process may take several minutes to complete.

App

Tasks for deploying and managing the Django app.

collectstatic

Roll out the Django app's latest static files.

```
$ fab collectstatic
```

deploy

Run a full deployment of code to the remote server.

```
$ fab deploy
```

More specifically, this task executes the following sub-tasks in order:

1. `pull`
2. `rmpyc`
3. `pipinstall`
4. `migrate`
5. `collectstatic`

manage

Run a manage.py command inside the Django virtualenv.

The only positional argument is `cmd`. For example, if you wanted to kickstart the CAL-ACCESS raw data update process:

```
$ fab manage:updatecalaccessrawdata
```

migrate

Migrate the database using Django's built-in `migrate` command.

```
$ fab migrate
```

pipinstall

Install the Python requirements inside the virtualenv:

```
$ fab pipinstall
```

pull

Pull the latest changes from the GitHub repo:

```
$ fab pull
```

rmpyc

Erase .pyc files from the app's code directory.

```
$ fab rmpyc
```

Chef

Tasks related to installing and executing Chef, the Ruby framework we use to set up the Ubuntu server that hosts the downloads website code.

bootstrap

Install Chef and use it to install the app on an EC2 instance.

```
$ fab bootstrap
```

More specifically, this task executes the following sub-tasks in order:

1. `rendernodejson`
2. `installchef`
3. `cook`
4. `copyconfig`
5. `migrate`
6. `collectstatic`

This task also sets the environment in which the website will run on the server based on your current local `CALACCESS_WEBSITE_ENV` environment variable (defaults to `DEV` if not set).

cook

In order to do its thing, Chef requires a cookbook that contains recipes (basically, short Ruby scripts) that outline the configuration scenario on the remote server. You can see our cookbook for this project here.

This task updates the Chef cookbook on the server and executes it.

```
$ fab cook
```

installchef

Install all the dependencies to run a Chef cookbook.

```
$ fab installchef
```

More specifically, this task:

1. Updates apt-get
2. Installs git
3. Installs Ruby
4. Installs Chef

rendernodejson

Render chef's node.json file from a template.

```
$ fab rendernodejson
```

In addition to the cookbook, some of the settings Chef requires are stored in a local `node.json` file, which is rendered from a template.

This template file is where you can, for example, change the run times for the crontab job that updates the download website with the latest CAL-ACCESS data export.

In order for any changes you make to node.json.template to take effect on the server, you need to execute both the `rendernodejson` and `cook` tasks.

Configure

Tasks for configuring the downloads website Django environment.

createconfig

Prompt users for settings to be stored in `.env` file.

```
$ fab createconfig
```

You will prompted to provide:

- An AWS Access Key ID and Secret Access Key (read more here).

- An AWS region (defaults to `us-west-2`).
- An SSH key-pair file name (defaults to `my-key-pair`). This assumes you have a key pair stored in `~/.ec2/my-key-pair.pem` (if you don't, you should create one).
- The name of the PostgreSQL database that will serve as the backend for the downloads website (defaults to `calaccess_website`).
- The name of the database user the Django app will use to connect to the database (defaults to `ccdc`).
- The password for the database user.
- The name of the S3 bucket where the data files will be archived (defaults to `django-calaccess-dev-data-archive`).
- The name of the S3 bucket where the "baked" content files will stored (defaults to `django-calaccess-dev-baked-content`).
- The host email address and password (press ENTER to skip).
- Addresses for the RDS and EC2 instances, in case these servers are already up and running. If not, press ENTER to skip for now, and spin them up later.

These configurations will be stored in a `.env` file (ignored by git) along with settings for other envs you have configured, each denoted by a section header such as `[DEV]` and `[PROD]`.

copyconfig

Copy current configuration in local `.env` file to the EC2 instance.

```
$ fab copyconfig
```

printconfig

Print the configuration settings for the local environment.

```
$ fab printconfig
```

printenv

Print the Fabric env settings.

```
$ fab printenv
```

setconfig

Add or edit a key-value pair in the `.env` configuration file.

```
$ fab setconfig:key=<new-variable-name>,value=<some-value>
```

Note that these changes will only take effect locally. In order to copy your new configuration to the EC2 instance, execute the `copyconfig` task.

Dev

Tasks for connecting to and running the downloads website server.

rs

Start up the Django runserver.

```
$ fab rs
```

The only optional argument is `port`, which defaults to `8000`.

ssh

Log into the EC2 instance using SSH.

```
$ fab ssh
```

By default, you will connect to the instance specified in `ec2_host` under your current environmnet in the `.env` file. If you want to connect to another EC2 instance you have up and running, pass in the address like so:

```
$ fab ssh:<ec2_instance_address>
```

Env

Tasks for temporarily switching environments before running subsequent tasks.

For example, if your OS `CALACCESS_WEBSITE_ENV` environment variable is set to `DEV`, but you want to quickly deploy some recent changes to the production server, you can:

```
$ fab prod deploy
```

dev

Operate on the development environment.

```
$ fab dev <task1> <task2>
```

prod

Operate on the production environment.

```
$ fab prod <task1> <task2>
```

Open-source resources

- Code: github.com/california-civic-data-coalition/django-calaccess-downloads-website
- Issues: github.com/california-civic-data-coalition/django-calaccess-downloads-website/issues
- Documentation: django-calaccess.californiacivicdata.org
- Testing: travis-ci.org/california-civic-data-coalition/django-calaccess-downloads-website
- Coverage: coveralls.io/github/california-civic-data-coalition/django-calaccess-downloads-website

1.4 How to contribute

This is an open-source project that welcomes contributions from anyone who has the time and energy to help us untangle the CAL-ACCESS database. Here's how to get started.

1.4.1 What developers can do

Contribute to django-calaccess-raw-data

This walkthrough is for developers who want to contribute to *django-calaccess-raw-data*, a Django app to download, extract and load campaign-finance and lobbying-activity data from the California Secretary of State's CAL-ACCESS database.

It will show you how to install the source code of this application to fix bugs and develop new features.

Preparing a development environment

It is not required, but it is recommended that development of the library be done from within a contained virtual environment.

One way to accomplish that is with Python's `virtualenv` tool and its helpful companion `virtualenvwrapper`. If you have that installed, a new project can be started with the following:

```
$ mkproject django-calaccess-raw-data
```

That will jump into a new folder in your code directory, where you can clone our code repository from GitHub after you make a fork of your own. Don't know what that means? Read this.

```
$ git clone https://github.com/<YOUR-USERNAME>/django-calaccess-raw-data.git .
```

Next install the other Python libraries our code depends on.

```
$ pip install -r requirements.txt
```

Connecting to a local database

Unlike a typical Django project, this application only supports the PostgreSQL database backend. This is because we enlist specialized tools to load the immense amount of source data more quickly than Django typically allows.

Creating your database

Create the database the PostgreSQL way.

```
$ createdb calaccess_raw -U postgres
```

Create a file at `example/project/settings_local.py` to save your custom database credentials. That might look something like this.

```
DATABASES = {
    'default': {
        'NAME': 'calaccess_raw',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'username', # <-- Change this
        'PASSWORD': 'password', # <-- And this
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

Note: If you'd prefer to load the CAL-ACCESS outside your default database, check out our guide to working with Django's system for multiple databases.

Once the database is configured

Now create the tables and get to work.

```
$ python example/manage.py migrate
```

Once everything is set up, the updatecalaccessrawdata command will download the latest bulk data release from the Secretary of State's website load it into your local database.

```
$ python example/manage.py updatecalaccessrawdata
```

Warning: This will take a while. Go grab some coffee.

Welcome aboard!

Now that your development environment is set up, check out the GitHub issue tracker where plenty of work awaits.

As you submit your work, please pay attention to the results of our integration tests (more details *here*).

Contribute to django-calaccess-scraped-data

This walkthrough is for developers who want to contribute to *django-calaccess-scraped-data*, a Django app to scrape from the CAL-ACCESS website supplementary data not included in the California Secretary of State's nightly data dumps.

It will show you how to install the source code of this application to fix bugs and develop new features.

Preparing a development environment

It is not required, but it is recommended that development of the library be done from within a contained virtual environment.

One way to accomplish that is with Python's `virtualenv` tool and its helpful companion `virtualenvwrapper`. If you have that installed, a new project can be started with the following:

```
$ mkproject django-calaccess-scraped-data
```

That will jump into a new folder in your code directory, where you can clone our code repository from GitHub after you make a fork of your own. Don't know what that means? Read this.

```
$ git clone https://github.com/<YOUR-USERNAME>/django-calaccess-scraped-data.git .
```

Next install the other Python libraries our code depends on.

```
$ pip install -r requirements.txt
```

Connecting to a local database

The `calaccess_scraped` app doesn't have any specific database requirements. However, we recommend PostgreSQL 9.4 (or greater), which is a hard requirement of other apps in our tool chain.

Create the database the PostgreSQL way.

```
$ createdb calaccess_scraped -U postgres
```

Create a file at `example/project/settings_local.py` to save your custom database credentials. That might look something like this.

```
DATABASES = {
    'default': {
        'NAME': 'calaccess_scraped',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'username', # <-- Change this
        'PASSWORD': 'password', # <-- And this
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

Note: If you'd prefer to load the CAL-ACCESS outside your default database, check out our guide to working with Django's system for multiple databases.

Once the database is configured

Now create the tables and get to work.

```
$ python example/manage.py migrate
```

Now you're ready to scrape. The scrapecalaccess command will download, cache and parse content from the CAL-ACCESS website:

```
$ python example/manage.py scrapecalaccess
```

Welcome aboard!

Now that your development environment is set up, check out the GitHub issue tracker where plenty of work awaits.

As you submit your work, please pay attention to the results of our integration tests (more details *here*).

Contribute to django-calaccess-processed-data

This walkthrough is for developers who want to contribute to *django-calaccess-processed-data*, a Django app to transform and refine campaign-finance and lobbying-activity data from the California Secretary of State's CAL-ACCESS database.

It will show you how to install the source code of this application to fix bugs and develop new features.

Preparing a development environment

It is not required, but it is recommended that development of the library be done from within a contained virtual environment.

One way to accomplish that is with Python's `virtualenv` tool and its helpful companion `virtualenvwrapper`. If you have that installed, a new project can be started with the following:

```
$ mkproject django-calaccess-processed-data
```

That will jump into a new folder in your code directory, where you can clone our code repository from GitHub after you make a fork of your own. Don't know what that means? Read this.

```
$ git clone https://github.com/<YOUR-USERNAME>/django-calaccess-processed-data.git .
```

Next install the other Python libraries our code depends on.

```
$ pip install -r requirements.txt
```

Connecting to a local database

Unlike a typical Django project, this application only supports PostgreSQL version 9.6 and above as a database backend. This is because we enlist specialized tools to load the immense amount of source data. We haven't developed those routines for SQLite and the other Django backends yet, but we might someday.

Create the database the PostgreSQL way.

```
$ createdb calaccess_processed -U postgres
```

Create a file at `example/project/settings_local.py` to save your custom database credentials. That might look something like this.

```
DATABASES = {
    'default': {
        'NAME': 'calaccess_processed',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'username', # <-- Change this
        'PASSWORD': 'password', # <-- And this
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

Note: If you'd prefer to load the CAL-ACCESS outside your default database, check out our guide to working with Django's system for multiple databases.

Once the database is configured

Now create the tables and get to work.

```
$ python example/manage.py migrate
```

Once everything is set up, the updatecalaccessrawdata command will download the latest bulk data release from the Secretary of State's website and load it into your local database.

```
$ python example/manage.py updatecalaccessrawdata
```

Warning: This will take a while. Go grab some coffee.

Because the nightly raw export is incomplete, we have to scrape additional data from the CAL-ACCESS website. Use the scrapecalaccess command to kick off this process, either after `updatecalaccessrawdata` finishes or in a separate terminal window:

```
$ python example/manage.py scrapecalaccess
```

Once the raw CAL-ACCESS data is loaded and the scrape has finished, you can transform all this messy data you've collected into our easy-to-understand, well-documented models with the processcalaccessdata command:

```
$ python example/manage.py processcalaccessdata
```

Welcome aboard!

Now that your development environment is set up, check out the GitHub issue tracker where plenty of work awaits.

As you submit your work, please pay attention to the results of our integration tests (more details *here*).

Contribute to django-calaccess-downloads-website

This walkthrough is for developers who want to contribute to *django-calaccess-downloads-website*, a open-source archive of campaign-finance and lobbying-disclosure data from the California Secretary of State's CAL-ACCESS database.

It will show you how to install the source code of this application to fix bugs, develop new features and deploy an archive to the Internet using Amazon Web Services.

Preparing a development environment

In order to contribute you first need to set up a local development environment by installing the source code and configuring a few settings.

While not required, we recommend that development be done within a contained virtual environment.

One way to accomplish that is with a two related Python packages: `virtualenv` and `virtualenvwrapper`. If you have both of these installed, a new project can be easily created like so:

```
$ mkproject django-calaccess-downloads-website
```

That will jump into a new folder in your code directory, where you can fork our code repository from GitHub. Don't know what that means? Read this.

Once you've created a fork, you should clone it to your computer.

```
$ git clone https://github.com/<YOUR-USERNAME>/django-calaccess-downloads-website.git␣
↪.
```

Next, install the other Python libraries our code depends on, like the Django web framework.

```
$ pip install -r requirements.txt
```

Configuring settings

Many of the settings in this project can vary depending on where the code is being run. For instance, your local installation of the code will likely connect to a different database than the public website.

To keep these different environments straight and avoid including sensitive passwords in public repositories we have developed a system for storing many of the configuration options in a file named `.env` at the project's root.

The file is excluded from Git's version control system and needs to be created fresh each time the code is installed.

How .env works

The `.env` file is expected to contain a separate section for each environment, using the structure favored by Python's ConfigParser module. Here's a simple example:

```
[DEV]
database_name=calaccess
mysecretpassword=password

[PROD]
database_name=calaccess
mysecretpassword=hotpockets
```

By default, the source code will draw settings from a section name `DEV`. To configure it to use a different set of variables (like the''PROD'' section above), you must set the `CALACCESS_WEBSITE_ENV` environment variable.

```
$ export CALACCESS_WEBSITE_ENV=PROD
```

If you are using virtualenv and virtualenvwrapper, you could add the above line of code to `$VIRTUAL_ENV/bin/postactivate` so that whenever you start the project's virtual environment, the variable will be exported automatically.

Note: You could also add the following line to your `$VIRTUAL_ENV/bin/postdeactivate` script to clear the variable whenever you deactivate the virtual environment:

```
$ unset CALACCESS_WEBSITE_ENV
```

Connecting to a local database

Unlike a typical Django project, this application only supports PostgreSQL version 9.6 and above as a database backend. This is because we enlist specialized tools to load the immense amount of source data more quickly than Django typically allows.

Create the database the PostgreSQL way.

```
$ createdb calaccess_website -U postgres
```

Creating an archive on Amazon S3

Even a development project that will run only on your computer needs an account with Amazon Web Services to store archived files in its S3 file service.

If you don't already have an AWS account, make one now and request a key pair that lets you access its services via Python.

Then create a new S3 "bucket" to store files archived by this project.

Filling in .env for the first time

The development environment can be created in the `.env` file by running a Fabric task that will ask you to provide a value for all of this project's settings.

```
$ fab createconfig
```

You will prompted to provide the project's full list of settings, though some of them are only necessary when deploying the code and site with Amazon Web Services.

Setting	Required in development	Definition
db_name	Yes	Name of your database.
db_user	Yes	Database user.
db_password	Yes	Database password.
db_host	Yes	Database host location.
aws_access_key_id	Yes	Shorter secret key for accessing Amazon Web Services.
aws_secret_access_key	Yes	The longer secret key for accessing Amazon Web Services.
aws_region_name	Yes	Amazon Web Services region where you resources are located.
s3_archived_data_bucket	Yes	Amazon S3 bucket where archived CAL-ACCESS data will be stored.
s3_baked_content_bucket	No	Amazon S3 bucket where the public-facing website will be stored.
key_name	No	Name of the SSH `.pem` file associated with Amazon Web Services. Should be found in `~/.ec2`.
ec2_host	No	Public address of website's Amazon EC2 instance.
email_user	No	Gmail account for sending error emails.
email_password	No	Gmail password for sending error emails.

If necessary, you can overwrite a specific setting or append a new one:

```
$ fab setconfig:key=<new-variable-name>,value=<some-value>
```

You can also print your current app environment's configuration:

```
$ fab printconfig
```

Or everything in the Fabric environment:

```
$ fab printenv
```

Bootstrapping the project

Now that everything is configured, create the database tables.

```
$ python manage.py migrate
```

Once everything is set up, the `updatedownloadswebsite` command will download the latest bulk data release from the Secretary of State's website load it into your local database and archive the files on Amazon S3.

```
$ python manage.py updatedownloadswebsite
```

Warning: This will take a while. Go grab some coffee.

Exploring the site

Finally, start the development server and visit localhost:8000/admin/ in your browser to inspect the site.

```
$ python manage.py runserver
```

Preparing a production server

This section will walk you through deploy the downloads website on the Internet via Amazon Web Services. You will need to have completed the steps above.

Change your environment

As described above, the source code will draw settings from a section of the *.env* file named `DEV`.

To switch to configuring your project for a production environment, you should set the `CALACCESS_WEBSITE_ENV` environment variable to `PROD`.

```
$ export CALACCESS_WEBSITE_ENV=PROD
```

If you are using virtualenv and virtualenvwrapper, you could add the above line of code to `$VIRTUAL_ENV/bin/postactivate` so that whenever you start the project's virtual environment, this variable will be exported automatically whenever you use `workon` to begin work.

Creating an RDS database

You will need to create a hosted database to store the data and keep tabs on the archive over time. Our recommended method for doing this is using Amazon's Relational Database Service.

You can spin up a PostgreSQL server there using our prepackaged Fabric commands. You're only required to provide a name like `download-website`:

```
$ fab createrds:download-website
```

Then, wait several minutes while the server is provisioned.

By default, the new database server will have 100 GB of disk space allocated on a t2.large RDS class instance. If need be, you can override these settings:

```
$ fab createrds:download-website,block_gb_size=80,instance_type=db.m4.large
```

The address for the RDS host will automatically be added to the configuration for your current environment, which is stored in `.env`. If you already had an RDS host set for your current env, its address will be overwritten.

Create an EC2 Instance

Next you should create a new Ubuntu 14.04 server on Amazon's Elastic Compute Cloud to host the Django project.

```
$ fab createec2
```

By default, the server will have 100 GB of disk space allocated on a c3.large class instance. If need be, you can override these settings:

```
$ fab createec2:block_gb_size=80,instance_type=c3.xlarge
```

You can also override our default Amazon Machine Image (AMI):

```
$ fab createec2:ami=<some-other-ami-id>
```

As with creating an RDS instance, the address for your new EC2 instance will automatically be added to the configuration for your current environment, which is stored in `.env`. If you already had an EC2 host set, its address will be overwritten.

Filling in .env for the second time

Now you'll want to run our configuration command again, this time filling in the new details from your AWS account, database and server. You may want to create a new set of S3 buckets separate from your development buckets.

```
$ fab createconfig
```

Bootstrap the Django project

Finally, you're ready to bootstrap the Django project on the Ubuntu server.

```
$ fab bootstrap
```

After connecting to your current EC2 instance, a framework called Chef and its dependencies, including Ruby, will be installed on the server. Chef is used to configure the server and install the downloads website's code.

The `bootstrap` task also sets up a crontab job to execute run as command every six hours that will automate the collection, extraction and processing of the daily CAL-ACCESS database exports.

Wrapping up

And that's it! You know have a live CAL-ACCESS archive running in the cloud.

Testing

Our code is tested using Django's built-in unittesting system via the TravisCI continuous integration service.

In addition, prior to the Django unittests, code is evaluated using Python's pep8 and pyflakes style-guide enforcement tools.

When a commit or pull request is made with our repository, those tests are rerun with the latest code. We try not to be too uptight, but we generally expect the tests to be pass before we will merge a request.

You can also run these tests locally. Change your directory into your local copy of any of our repos, and then:

```
$ make test
```

1.4.2 What anyone can do

Contribute to our documentation

We're maintaining a single repository for all documents related to the Django CAL-ACCESS project. This section is for anyone who wants to contribute to these documents.

Do I need to know Python (or Django)?

No. But you should be familiar with the syntax of reStructuredText, since that's the format in which these documents are written.

Which files should I edit?

Generally, you should be editing the `.rst` files in `docs/` directory, rather than any of the `.html` files in the `_build/` directory. The `.html` files are compiled using Python's Sphinx documentation framework (for more on this, see *Viewing Local Changes*).

Viewing changes locally

After you make changes to your local versions of the `.rst` files in the `docs/` directory, you can view your edits as they will appear on ReadTheDocs:

```
$ make html
```

This will invoke the sphinx command to compile the `.html` and `.doctree` files as they will be when deployed to ReadTheDocs. These files are then saved in `docs/_build/html`, where you can open them with your favorite web browser and see how they look.

However, you might instead prefer to see your edits as you're making them:

```
$ make livehtml
```

This will start serving the docs on http://127.0.0.1:8000, where you can point your favorite web browser. While the server is running, Sphinx will also detect any change you make to the `.rst` files and, on save, automatically re-build the `.html` files.

1.4.3 What maintainers should know

Version release checklist

The steps to follow each time release a new version of a Django package.

- Update the changelog
- Update `setup.py` with new version and any new dependencies
  - Consider including RC (release candidate) in release name until we're sure we're uploading a package with all tiny details accounted for
- Update `requirements.txt` with any new dependencies
- Run `python setup.py sdist bdist_wheel`, make sure their aren't any errors
- Spotcheck new release package in dist/ to make sure all files made it in.
- Run `twine upload dist/* --skip-existing`
- Release on GitHub
  - `git commit` final change and run `git tag "v#.#.#"` with whatever the release number is
  - Run `git push origin master --tags`
  - Add list of changes to the page release on GitHub

1.5 Frequently asked questions

Questions and answers about the technical aspects of our work. A broader FAQ about the CAL-ACCESS database and our work with it can be found here.

1.5.1 How do the Django applications fit together?

The *django-calaccess-raw-data* application is intended as the base layer below more sophisticated apps, like *django-calaccess-processed-data*, that transform the source data and load it into simplified models to serve as a platform for investigative analysis.

1.5.2 Why does django-calaccess-raw-data use loading techniques not supported by Django?

Because the CAL-ACCESS database is huge. With more than 35 million records sprawled across 80 tables, it can take a long time to load into a database using the standard Django tools, which insert one record at a time. In our early testing, it took as long as 24 hours to load all of CAL-ACCESS into a database on a standard laptop computer.

To speed things up, our loading commands take advantage of the built-in bulk loading tools offered by PostgreSQL and MySQL, which are not currently included in Django's system. These tools (`COPY` in PostgreSQL and `LOAD DATA INFILE` in MySQL) insert CSV files from the file system directly into the database in a small fraction of the time it would take to load them row by row.

As part of developing these tools we released django-postgres-copy, a Django extension that makes it easier for us and other developers to work with these valuable tools.

1.5.3 Why does django-calaccess-raw-data only work with PostgreSQL and MySQL databases?

Because of the answer above. To run our loading routines in an acceptable amount of time, we need to take advantage of bulk file loading tools not currently supported by Django.

So far, we have only written custom loading routines for MySQL and PostgreSQL. We would welcome contributions that would expand our database support to other systems, like SQLite and Microsoft SQL Server. But we haven't got there yet.

1.5.4 Do I have to load the CAL-ACCESS data into my default database?

No, *django-calaccess-raw-data* supports the use of automatic database routing, which Django's own documentation describes as "the easiest way to use multiple databases".

If you fall into this category, first of all, be sure you've carefully read through Django's multiple databases topic guide.

Next, configure your additional databases in `settings.py`. Let's assume you want two PostgreSQL databases: One for all CAL-ACCESS data called `calaccess_raw`, and a default `my_project` database for everything else:

```
DATABASES = {
    'default': {
        'NAME': 'my_project',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'your-username-here',
        'PASSWORD': 'your-password-here',
        'HOST': 'localhost',
```

(continues on next page)

(continued from previous page)

```
        'PORT': '5432'
    },
    'calaccess_raw': {
        'NAME': 'calaccess',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'USER': 'your-username-here',
        'PASSWORD': 'your-password-here',
        'HOST': 'localhost',
        'PORT': '5432'
    },
}
```

Then, create a `routers.py` file in your Django project's directory (same place as `manage.py` and `settings.py`). Following from the above example, here's how you could implement a router to send calaccess-raw model data to their own database and everything else to `default`:

```
class ExampleRouter(object):
    """
    Send calaccess_raw models to their own db. Everything else to default.
    """

    def get_db(self, model=None, app_label=None):
        app_label = app_label or model._meta.app_label
        if app_label == 'calaccess_raw'
            db_label = 'calaccess_raw'
        else:
            db_label = 'default'
        return db_label

    def db_for_read(self, model, **hints):
        """
        Attempts to read calaccess_raw models go to calaccess_raw db.
        """
        return self.get_db(model=model)

    def db_for_write(self, model, **hints):
        """
        Attempts to write calaccess_raw models go to calaccess_raw db.
        """
        return self.get_db(model=model)

    def allow_relation(self, obj1, obj2, **hints):
        """
        Allow relations if a model in the calaccess_raw app is involved.
        """
        return self.get_db(model=obj1) == self.get_db(model=obj2)

    def allow_migrate(self, db, app_label, model=None, **hints):
        """
        Make sure the calaccess_raw app only appears in the calaccess_raw
        database.
        """
        intended_db = self.get_db(app_label=app_label)
        return (db == intended_db) or (db == 'default' and intended_db is None)
```

Finally, configure the router in `setting.py`:

```
DATABASE_ROUTERS = ['example.routers.ExampleRouter']
```

And everything should be ready.