elex

Release 1.0.0

January 26, 2016

Contents

CHAPTER 1

ELEX

Elex was developed by The New York Times and NPR and not in concert with the Associated Press. Though we plan on using Elex for the 2016 cycle, there is no guarantee that this software will work for you. If you're thinking about using Elex, check out the license and contact the authors.

Get database-ready election results from the Associated Press Election API v2.0.

Elex is designed to be fast, friendly, and largely agnostic to stack/language/database choice. Basic usage is as simple as:

```
elex results 2015-11-21 > results.csv
```

Important links:

- Documentation: http://elex.readthedocs.org/
- Repository: https://github.com/newsdev/elex/
- Issues: https://github.com/newsdev/elex/issues
- Roadmap: https://github.com/newsdev/elex/milestones

1.1 Elex projects and implementations

NPR

- NPR loader: A simple reference data loader for PostgreSQL.

New York Times

- New York Times loader: A more sophisticated data loader for PostgreSQL.
- New York Times Deja Vu: A webservice to replay JSON captured during an election.
- New York Times Elex Admin: An admin interface for Elex data loaded with the New York Times loader written in Flask.

Experimental

- node-elex-admin: Incomplete node-based admin interface.

1.2 News

- Introducing Elex, A Tool To Make Election Coverage Better For Everyone, Jeremy Bowers and David Eads, Source
- NPR and The New York Times teamed up to make election reporting faster, Benjamin Mullin, Poynter

1.3 Using the FTP system?

Use the Los Angeles Times' python-elections library.

1.4 Features

- Uses v2.0 of the Associated Press Election API **NOTE**: Requires a (paid) account with the AP.
- Intuitive command line interface: Get data as CSV or JSON and pipe to the data tool of your choice.
- Friendly Python API for use as a library.
- Simple election recording (to MongoDB).
- Comprehensive tests.
- Extensive documentation.
- Fast (performance is a work in progress; contributions are welcome).

1.5 Table of contents

1.5.1 Installation

Install the library

Install the Python library:

```
pip install elex
```

Set your AP API key:

```
export AP_API_KEY=<MY_AP_API_KEY>
```

Optional requirements

- MongoDB (for recording raw results during tests and elections)

1.5.2 Tutorial

Command Line Interface

This tool is primarily designed for use on the command line using standard *NIX operations like pipes and output redirection.

To write a stream of races in CSV format to your terminal, run:

```
elex races '11-03-2015'
```

To write this data to a file:

```
elex races '11-03-2015' > races.csv
```

To pipe it into PostgreSQL:

```
elex races 11-03-2015 | psql elections -c "COPY races FROM stdin DELIMITER ',' CSV HEADER;"```
```

To get JSON output:

```
elex races 11-03-2015 -o json
```

Output can be piped to tools like sed, awk, jq, or csvkit for further processing.

Python Modules

Perhaps you'd like to use Python objects in your application. This is how you would call the Elex modules directly without using the command line tool.

```
from elex.parser import api

# Setup and call the AP API.
e = api.Election(electiondate='2015-11-03', datafile=None, testresults=False, liveresults=True, is_te
raw_races = e.get_raw_races()
race_objs = e.get_race_objects(raw_races)

# Get lists of Python objects for each of the core models.
ballot_measures = e.ballot_measures
candidate_reporting_units = e.candidate_reporting_units
candidates = e.candidates
races = e.races
reporting_units = e.reporting_units
results = e.results
```

1.5.3 Command line interface

```
commands:

  ballot-measures
    Get ballot positions (also known as ballot issues)

  candidate-reporting-units
    Get candidate reporting units (without results)

  candidates
    Get candidates
```

```
  delegates
    Get all delegate reports

  elections
    Get list of available elections

  next-election
    Get the next election (if date is specified, will be relative to that date, otherwise will use to

  races
    Get races

  reporting-units
    Get reporting units

  results
    Get results

positional arguments:
  date                  Election date (e.g. "2015-11-03"; most common date
                        formats accepted).

optional arguments:
  -h, --help            show this help message and exit
  --debug               toggle debug output
  --quiet               suppress all output
  -o {json,csv}         output format (default: csv)
  -t, --test            Use testing API calls
  -n, --not-live        Do not use live data API calls
  -d DATA_FILE, --data-file DATA_FILE
                        Specify data file instead of making HTTP request when
                        using election commands like `elex results` and `elex
                        races`.
  --delegate-sum-file DELEGATE_SUM_FILE
                        Specify delegate sum report file instead of making
                        HTTP request when using `elex delegates`
  --delegate-super-file DELEGATE_SUPER_FILE
                        Specify delegate super report file instead of making
                        HTTP request when using `elex delegates`
  --format-json         Pretty print JSON when using `-o json`.
  -v, --version         show program's version number and exit
```

1.5.4 Python API

Elex provides a Python API that encapsulates Associated Press Election API results as Python objects.

To use the election loader manually from within your project:

```python
from elex.api import Election

election = Election(electiondate='2015-11-03', testresults=False, liveresults=True, is_test=False)
races = election.races
```

Now you can process or load `races`.

Models:

elex.api.Election

class `elex.api.`**`Election`**(***kwargs*)

Canonical representation of an election on a single date.

`ballot_measures`

Return list of ballot measure objects with results.

`candidate_reporting_units`

Return list of candidate reporting unit objects.

`candidates`

Return list of candidate objects with results.

`get`(*path*, ***params*)

Farms out request to api_request. Could possibly handle choosing which parser backend to use – API-only right now. Also the entry point for recording, which is set via environment variable.

Parameters

- **`path`** – API url path.
- **`**params`** – A dict of optional parameters to be included in API request.

`get_race_objects`(*parsed_json*)

Get parsed race objects.

Parameters **`parsed_json`** – Dict of parsed JSON.

`get_raw_races`(***params*)

Convenience method for fetching races by election date. Accepts an AP formatting date string, e.g., YYYY-MM-DD. Accepts any number of URL params as kwargs.

If datafile passed to constructor, the file will be used instead of making an HTTP request.

Parameters **`**params`** – A dict of additional parameters to pass to API. Ignored if *datafile* was passed to the constructor.

`get_uniques`(*candidate_reporting_units*)

Parses out unique candidates and ballot measures from a list of CandidateReportingUnit objects.

`get_units`(*race_objs*)

Parses out races, reporting_units, and candidate_reporting_units in a single loop over the race objects.

Parameters **`race_objs`** – A list of top-level Race objects.

`races`

Return list of race objects.

`reporting_units`

Return list of reporting unit objects.

`results`

Return list of candidate reporting unit objects with results.

`serialize`()

Implements `APElection.serialize()`.

`set_id_field`()

Set id to <*electiondate*>.

elex.api.ReportingUnit

class `elex.api.`**`ReportingUnit`**`(`***kwargs*`)`

Canonical representation of a single level of reporting.

`pad_fipscode``()`

`serialize``()`

Implements `APElection.serialize()`.

`set_candidate_votepct``()`

Set vote percentage for each candidate.

`set_id_field``()`

Set id to <*reportingunitid*>.

`set_level``()`

New England states report at the township level. Every other state reports at the county level. So, change the level from 'subunit' to the actual level name, either 'state' or 'township'.

`set_votecount``()`

Set vote count.

elex.api.Race

class `elex.api.`**`Race`**`(`***kwargs*`)`

Canonical representation of a single race, which is a seat in a political geography within a certain election.

`serialize``()`

Implements `APElection.serialize()`.

`set_id_field``()`

Set id to <*raceid*>.

`set_new_england_counties``()`

Create new CandidateReportingUnits for each New England county that rolls up vote counts and precinct counts / pcts from each township under that county.

elex.api.Candidate

class `elex.api.`**`Candidate`**`(`***kwargs*`)`

Canonical representation of a candidate. Should be globally unique for this election, across races.

`serialize``()`

Implements `APElection.serialize()`.

`set_id_field``()`

Set id to <*unique_id*>.

`set_unique_id``()`

Generate and set unique id.

Candidate IDs are not globally unique. AP National Politian IDs (NPIDs or polid) are unique, but only national-level candidates have them; everyone else gets '0'. The unique key, then, is the NAME of the ID we're using and then the ID itself. Verified this is globally unique with Tracy.

elex.api.BallotMeasure

class `elex.api.`**`BallotMeasure`**`(`*`**kwargs`*`)`

Canonical representation of a ballot measure.

Ballot measures are similar to :class:‘Candidate‘s, but represent a position on a ballot such as “In favor of” or “Against” for ballot measures such as a referendum.

`serialize``()`

Implements `APElection.serialize()`.

`set_id_field``()`

Set id to <*unique_id*>.

`set_unique_id``()`

Generate and set unique id.

Candidate IDs are not globally unique. AP National Politian IDs (NPIDs or polid) are unique, but only national-level candidates have them; everyone else gets ‘0’. The unique key, then, is the NAME of the ID we’re using and then the ID itself. Verified this is globally unique with Tracy.

elex.api.CandidateReportingUnit

class `elex.api.`**`CandidateReportingUnit`**`(`*`**kwargs`*`)`

Canonical reporesentation of an AP candidate. Note: A candidate can be a person OR a ballot measure.

`serialize``()`

Implements `APElection.serialize()`.

`set_id_field``()`

Set id to <*raceid*>-<*uniqueid*>-<*reportingunitid*>.

`set_unique_id``()`

Generate and set unique id.

Candidate IDs are not globally unique. AP National Politian IDs (NPIDs or polid) are unique, but only national-level candidates have them; everyone else gets ‘0’. The unique key, then, is the NAME of the ID we’re using and then the ID itself. Verified this is globally unique with Tracy.

elex.api.APElection

class `elex.api.`**`APElection`**

Base class for most objects. Handy container for methods for first level transformation of data and AP connections.

`serialize``()`

Serialize the object. Should be implemented in all classes that inherit from `APElection`.

Should return an OrderedDict.

`set_candidates``()`

Set candidates.

If this thing (race, reportingunit) has candidates, serialize them into objects.

`set_polid``()`

Set politication id.

If *polid* is zero, set to *None*.

set_reportingunitids()
Set reporting unit ID.

Per Tracy / AP developers, if the level is "state", the reportingunitid is always 1.

set_reportingunits()
Set reporting units.

If this race has reportingunits, serialize them into objects.

set_state_fields_from_reportingunits()
Set state fields.

elex.api.Elections

class elex.api.**Elections**
Holds a collection of election objects

get_elections(*datafile=None*)
Get election data from API or cached file.

Parameters **datafile** – If datafile is specified, use instead of making an API call.

get_next_election(*datafile=None*, *electiondate=None*)
Get next election. By default, will be relative to the current date.

Parameters

- **datafile** – If datafile is specified, use instead of making an API call.
- **electiondate** – If electiondate is specified, gets the next election after the specified date.

Utilities:

elex.api.utils

Utility functions to record raw election results and handle low-level HTTP interaction with the Associated Press Election API.

class elex.api.utils.**UnicodeMixin**
Python 2 + 3 compatibility for __unicode__

elex.api.utils.**api_request**(*path*, ***params*)
Function wrapping Python-requests for making a request to the AP's elections API.

A properly formatted request: * Modifies the BASE_URL with a path. * Contains an API_KEY. * Returns a response object.

Parameters ****params** – Extra parameters to pass to *requests*.

elex.api.utils.**write_recording**(*payload*)
Record a timestamped version of an Associated Press Elections API data download.

Presumes JSON at the moment. Would have to refactor if using XML or FTP. FACTOR FOR USE; REFACTOR FOR REUSE.

Parameters **payload** – JSON payload from Associated Press Elections API.

elex.api.maps

Defines `FIPS_TO_STATE`, `STATE_ABBR`, `OFFICE_NAMES` and `PARTY_NAMES` look-up constants.

1.5.5 Recording results

Flat files

Will record timestamped and namespaced files to the `ELEX_RECORDING_DIR` before parsing.

```
export ELEX_RECORDING=flat
export ELEX_RECORDING_DIR=/tmp
```

MongoDB

Will record a timestamped record to MongoDB, connecting via `ELEX_RECORDING_MONGO_URL` and writing to the `ELEX_RECORDING_MONGO_DB` database.

```
export ELEX_RECORDING=mongodb
export ELEX_RECORDING_MONGO_URL=mongodb://localhost:27017/  # Or your own connection string.
export ELEX_RECORDING_MONGO_DB=ap_elections_loader
```

1.5.6 Recipes

Useful Elex patterns. Contribute your own.

Filter with jq and upload to S3

This recipe uses the jq json filtering tool to create a national results json data file with a limited set of data fields and the AWS cli tools to upload the filtered json to S3.

Requirements:

- Amazon web services account
- jq
- AWS cli tools

```
#!/bin/bash

# S3 url: MUST be set to your bucket and path.
ELEX_S3_URL='mybucket.tld/output/path.json'

# Get results and upload to S3
elex results 2012-11-06 -o json \
| jq -c '[
            .[] |
            select(.level == "state" ) |
            select(.officename == "President") |
            {
              officename: .officename,
              statepostal: .statepostal,
              first: .first,
                last: .last,
                party: .party,
                votecount: .votecount,
                votepct: .votepct,
                winner: .winner,
                level: .level
              }
          ]' \
| gzip -vc \
| aws s3 cp - s3://$ELEX_S3_URL \
    --acl public-read \
    --content-type=application/json \
    --content-encoding gzip

# Check response headers
curl -I $ELEX_S3_URL

# Get first entry of uploaded json
curl -s --compressed $ELEX_S3_URL | jq '[.[]][0]'
```

`ELEX_S3_URL` **must** be set to your s3 bucket and path.

Steps:

- Get election results in json format with `elex`
- Pipe results to `jq` for filtering
- Pipe filtered results to `gzip` to compress
- Pipe gzipped results to `aws s3 cp` to send to S3.

Inspect with an ORM using Flask and Peewee

This recipe uses the Flask web framework and the Peewee Python ORM to model, query and update data that `elex` provides.

Requirements:

- Elex loader, an NYT project that calls `elex` to load data into a Postgres database with CSV and the Postgres `COPY` command.
- Elex admin, an NYT project that is a simple, web-based admin for creating and editing data to override AP election results, including candidate names, race descriptions, and race calls.

Steps:

- Install `elex-loader` using **‘these instructions <>‘_**.
- Install `elex-admin` using **‘these instructions <>‘_**.

Extra steps:

- Use the `models.py` that come with `elex-admin` to query data.

1.5.7 Contributing

We welcome contributions of all sizes. You got this!

Find a task

1. Check out the issue tracker and pick out a task or create a new issue
2. Leave a comment on the ticket so that others know you're working on it.

Install Elex development environment

1. Fork the project on Github.
2. Install a development version of the code with:

```
mkvirtualenv elex-dev
pip install -e git+git@github.com:<YOUR_GITHUB_USER>/elex#egg=elex``
```

3. Install developer dependencies for tests and docs:

```
pip install Sphinx==1.3.1
pip install nose2==0.5.0
pip install tox==2.3.1
```

Now you can run the following commands when you want to activate your enviroment and cd to the source directory.

```
workon elex-dev
cd ${VIRTUAL_ENV}/src/elex
```

Running tests

Edit or write the code or docs, taking care to include well=crafted docstrings and generally following the format of the existing code.

Write tests for any new features you add. Add to the tests in the `tests` directory or follow the format of files like `tests/test_election.py`.

Make sure all tests are passing in your environment by running the nose2 tests.

```
nose2 tests
```

If you have Python 2.7, 3.5, and pypy installed, run can run `tox` to test in multiple environments.

Writing docs

Write documentation by adding to one of the files in `docs` or adding your own.

To build a local preview, run:

```
make -C docs html
```

The documentation is built in `docs/_build/html`. Use Python's simple HTTP server to view it.

```
cd docs/_build/html
python -m http.server
```

Python 2.7 users should use `SimpleHTTPServer` instead of `http.server`.

Submitting code

Submit a pull request on Github.

Testing performance

To get detailed information about performance, run the tests with the ==profile flag:

```
nose2 tests --profile
```

Testing API request limit

You can test the API request limit, but only by setting an environment variable. Use with extreme care.

```
AP_RUN_QUOTA_TEST=1 nose2 tests.test_ap_quota
```

Authors

elex is maintained by Jeremy Bowers <jeremy.bowers@nytimes.com> and David Eads <deads@npr.org>.

These individuals have contributed code, tests, documentation, and troubleshooting:

- Jeremy Bowers
- David Eads
- Livia Labate
- Wilson Andrews
- Eric Buth
- Juan Elosua

1.5.8 Changelog

1.0.0 - Jan. 25, 2016

The 1.0.x release is named for Martha Ellis Gellhorn, one of the greatest war correspondents of the 20th century.

- Delegate counts (#138, #194). Delegate counts can be accessed with `elex delegates`.
- Rename `elex.api.api` to `elex.api.models` and allow model objects to be imported with statements like `from elex.api import Election` (#146). Python modules directly calling Elex will need to update their import statements accordingly.
- Fix duplicate IDs (#176).
- Handle incorrect null/none values in some cases (#173, #174, #175).
- Expand contributing / developer guide (#151).
- Add recipe for filtering with jq and uploading to s3 in a single command (#131).

0.2.0 - Dec. 24, 2015

- Tag git versions (#170).
- Fix elections command (#167).
- Use correct state code for county level results (#164).
- Use tox to test multiple Python versions (#153).

- Allow API url to be specified in environment variable (#144).
- Don't sort results for performance and stability (#136).
- Capture and log full API request URL in command line debugging mode (#134).
- Python 3 compatibility (#99).

0.1.2 - Dec. 21, 2015

- Fix missing vote percent in results (#152).

0.1.1 - Dec. 10, 2015

- Add Travis CI support (#101).
- Fix packaging.

0.1.0 - Dec. 10, 2015

First major release.

- Decided on *elex* for name (#59).
- Initial tests (#70, #107).
- First draft of docs (#18).
- Set up http://elex.readthedocs.org/ (#60).
- Handle New England states (townships and counties) (#123).
- Remove date parsing (#115) and dynamic field setter (#117) to improve performance.

0.0.0 - 0.0.42

Initial Python API and concept created by Jeremy Bowers; initial command line interface created by David Eads.

Python Module Index

Index