django-cities-light Documentation

Release 2.4.3

James Pic

March 03, 2016

Contents

This add-on provides models and commands to import country, region/state, and city data in your database.

The data is pulled from GeoNames and contains cities, regions/states and countries.

Spatial query support is not required by this application.

This application is very simple and is useful if you want to make a simple address book for example. If you intend to build a fully featured spatial database, you should use django-cities.

Requirements:

- Python 2.7 or 3.3,
- Django 1.4 or 1.5 or 1.6,
- PostgreSQL or SQLite.
- django-south is optionnal, but recommended.

MySQL support was dropped on 2.x.x because it stopped working. It is supported again in 3.x.x, which uses django.db.transaction.atomic, which requires Django >= 1.6.

CHAPTER 1

Upgrade

See CHANGELOG.

CHAPTER 2

Installation

Install django-cities-light:

```
pip install django-cities-light
```

Or the development version:

```
pip install -e git+git@github.com:yourlabs/django-cities-light.git#egg=cities_light
```

Add *cities_light* to your *INSTALLED_APPS*.

Now, run syncdb, it will only create tables for models that are not disabled:

```
./manage.py syncdb
```

Note that this project supports django-south. It is recommended that you use south too else you're on your own for migrations/upgrades.

Danger: Since version 2.4.0, django-cities-light uses django migrations by default. This means that django-south users should add to settings:

```
SOUTH_MIGRATION_MODULES = {
    'cities_light': 'cities_light.south_migrations',
}
```

CHAPTER 3

Data update

Finally, populate your database with command:

```
./manage.py cities_light
```

This command is well documented, consult the help with:

```
./manage.py help cities_light
```

CHAPTER 4

Resources

You could subscribe to the mailing list ask questions or just be informed of package updates.

- Mailing list graciously hosted by Google
- Git graciously hosted by GitHub,
- Documentation graciously hosted by RTFD,
- Package graciously hosted by PyPi,
- Continuous integration graciously hosted by Travis-ci

Contents:

4.1 Populating the database

4.1.1 Data install or update

Populate your database with command:

```
./manage.py cities_light
```

By default, this command attempts to do the least work possible, update what is necessary only. If you want to disable all these optimisations/skips, use –force-all.

This command is well documented, consult the help with:

```
./manage.py help cities_light
```

4.1.2 Signals

Signals for this application.

cities_light.signals.**city_items_pre_import**
Emited by city_import() in the cities_light command for each row parsed in the data file. If a signal reciever raises InvalidItems then it will be skipped.

An example is worth 1000 words: if you want to import only cities from France, USA and Belgium you could do as such:

```
    import cities_light

    def filter_city_import(sender, items, **kwargs):
        if items[8] not in ('FR', 'US', 'BE'):
            raise cities_light.InvalidItems()

    cities_light.signals.city_items_pre_import.connect(filter_city_import)
```

Note: this signal gets a list rather than a City instance for performance reasons.

cities_light.signals.**region_items_pre_import**
Same as *city_items_pre_import*, for example:

```
    def filter_region_import(sender, items, **kwargs):
        if items[0].split('.')[0] not in ('FR', 'US', 'BE'):
            raise cities_light.InvalidItems()
    cities_light.signals.region_items_pre_import.connect(
        filter_region_import)
```

cities_light.signals.**country_items_pre_import**
Same as *region_items_pre_import* and *cities_light.signals.city_items_pre_import*, for example:

```
    def filter_country_import(sender, items, **args):
        if items[0].split('.')[0] not in ('FR', 'US', 'BE'):
            raise cities_light.InvalidItems()

    cities_light.signals.country_items_pre_import.connect(
        filter_country_import)
```

cities_light.signals.**filter_non_cities**(*sender*, *items*, ***kwargs*)
Reports non populated places as invalid.

By default, this reciever is connected to city_items_pre_import, it raises InvalidItems if the row doesn't have PPL in its features (it's not a populated place).

exception cities_light.exceptions.**InvalidItems**
The cities_light command will skip item if a city_items_pre_import signal reciever raises this exception.

4.1.3 Configure logging

This command is made to be compatible with background usage like from cron, to keep the database fresh. So it doesn't do direct output. To get output from this command, simply configure a handler and formatter for *cities_light* logger. For example:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console':{
            'level':'DEBUG',
            'class':'logging.StreamHandler',
            'formatter': 'simple'
```

```
        },
    },
    'loggers': {
        'cities_light': {
            'handlers':['console'],
            'propagate': True,
            'level':'DEBUG',
        },
        # also use this one to see SQL queries
        'django': {
            'handlers':['console'],
            'propagate': True,
            'level':'DEBUG',
        },
    }
}
```

4.2 Simple django app

4.2.1 Settings

Settings for this application. The most important is TRANSLATION_LANGUAGES because it's probably project specific.

TRANSLATION_LANGUAGES List of language codes. It is used to generate the alternate_names property of cities_light models. You want to keep it as small as possible. By default, it includes the most popular languages according to wikipedia, which use a rather ascii-compatible alphabet. It also contains 'abbr' which stands for 'abbreviation', you might want to include this one as well.

See:

- http://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers
- http://download.geonames.org/export/dump/iso-languagecodes.txt

COUNTRY_SOURCES A list of urls to download country info from. Default is countryInfo.txt from geonames download server. Overridable in settings.CITIES_LIGHT_COUNTRY_SOURCES.

REGION_SOURCES A list of urls to download region info from. Default is admin1CodesASCII.txt from geonames download server. Overridable in settings.CITIES_LIGHT_REGION_SOURCES

CITY_SOURCES A list of urls to download city info from. Default is cities15000.zip from geonames download server. Overridable in settings.CITIES_LIGHT_CITY_SOURCES

TRANSLATION_SOURCES A list of urls to download alternate names info from. Default is alternateNames.zip from geonames download server. Overridable in settings.CITIES_LIGHT_TRANSLATION_SOURCES

SOURCES A list with all sources.

DATA_DIR Absolute path to download and extract data into. Default is cities_light/data. Overridable in settings.CITIES_LIGHT_DATA_DIR

INDEX_SEARCH_NAMES If your database engine for cities_light supports indexing TextFields (ie. it is **not** MySQL), then this should be set to True. You might have to override this setting if using several databases for your project.

4.2.2 Models

See source for details.

4.2.3 Admin

See source for details.

4.3 cities_light.contrib

4.3.1 For django-ajax-selects

4.3.2 For djangorestframework

The contrib contains support for both v1 and v2 of django restframework.

Django REST framework 2

This contrib package defines list and detail endpoints for City, Region and Country. If rest_framework (v2) is installed, all you have to do is add this url include:

```
url(r'^cities_light/api/', include('cities_light.contrib.restframework2')),
```

This will configure six endpoints:

```
^cities/$ [name='cities-light-api-city-list']
^cities/(?P<pk>[^/]+)/$ [name='cities-light-api-city-detail']
^countries/$ [name='cities-light-api-country-list']
^countries/(?P<pk>[^/]+)/$ [name='cities-light-api-country-detail']
^regions/$ [name='cities-light-api-region-list']
^regions/(?P<pk>[^/]+)/$ [name='cities-light-api-region-detail']
```

All list endpoints support search with a query parameter q:: /cities/?q=london

For Region and Country endpoints, the search will be within name_ascii field while for City it will search in search_names field. HyperlinkedModelSerializer is used for these models and therefore every response object contains url to self field and urls for related models. You can configure pagination using the standard rest_framework pagination settings in your project settings.py.

4.3.3 For django-autocomplete-light

For autocomplete-light, we propose an autocomplete channel that attempts to behave like google map's autocomplete. We did some research and it turns out every user is apparently able to use it without problems.

Basic Channel

Remote channels

Check out the example usage. This is the API:

4.3.4 Ideas for contributions

- templatetag to render a city's map using some external service
- flag images, maybe with django-countryflags
- currencies
- generate po files when parsing alternate names

CHAPTER 5

FAQ

5.1 MySQL errors with special characters, how to fix it ?

The `cities_light` command is continuously tested on travis-ci on all supported databases: if it works there then it should work for you.

If you're new to development in general, you might not be familiar with the concept of encodings and collations. Unless you have a good reason, you **must** have utf-8 database tables. See MySQL documentation for details.

We're pointing to MySQL documentations because PostgreSQL users probably know what UTF-8 is and won't have any problem with that.

5.2 Some data fail to import, how to skip them ?

GeoNames is not perfect and there might be some edge cases from time to time. We want the `cities_light` management command to work for everybody so you should open an issue in GitHub if you get a crash from that command.

However, we don't want you to be blocked, so keep in mind that you can use *Signals* like *`cities_light.city_items_pre_import`*, *`cities_light.region_items_pre_import`*, *`cities_light.country_items_pre_import`*, to skip or fix items before they get inserted in the database by the normal process.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

C

C

F

I

R