API Docs

Authentication

HTTP BASIC

all non GET requests have to be authenticated

End Points

Search

- Search
  - GET /place/search.json

- GET Params
  q: search string (see examples in wiki [[SearchTerms]])
  bbox: bounding box string if "false", will search for places without geometry, if empty string / not specified, will search for places with and without geos, not constrained by bbox
  per_page: results per page int, default=100
  page: page no (starting with 1) int, default=1
  start_date: YYYY or YYYY-MM-DD
  end_date:
  sort: optional - ['relevance', 'feature_code', 'start', 'end', 'name', 'uris', 'distance'] defaults to relevance if not given. Distance is distance from centre of bbox (and so only works if bbox is also present)
  order: asc, desc
  simplify: optional. if true will simplify large polygons (over 200 coordinates). If bbox is given, simplifcation tolerance is sub pixel relative to the bbox, otherwise a level of 0.01 is used. default=false
  format: optional. "csv" for csv download of search results. See BatchImport for more details of this format. default=geojson

- Returns
  GeoJSON feed of search results
  Extra properties of feed:
  total: total number of results
  page: page number
  pages: total pages
  example:

{"features": [{"geometry": {"type": "Polygon", "coordinates": [[[-73.85529167874355, 40.86177933823652], [-73.85535626390501, 40.861780177444416], [-73.85535516258005, 40.86182902078363], [-73.85529057737267, 40.86182818088898], [-73.85529167874355, 40.86177933823652]]]}, "type": "Feature", "properties": {"importance": null, "feature_code": "BLDG", "id": "5fea12e0f332b42c", "population": null, "is_composite": null, "name": "2425 GARAGE YATES AVENUE", "area": null, "admin": [], "is_primary": true, "alternate": [], "feature_code_name": "building(s)", "timeframe": {}, "uris": ["http://www.nyc.gov/html/dcp/html/bytes/applbyte.shtml.230012"]}}....], "max_score": 1.0, "page": 1, "per_page": 100, "total": 1983684, "type": "FeatureCollection", "pages": 19837}

Feature Codes

- Get / Search Feature Codes
  - GET /feature_codes.json
    Used by autocomplete code
    q: optional query to search codes
    page_limit:
    page:

- example URL
  http://dev.nypl.gazetteer.in/1.0/place/feature_codes.json?q=&page_limit=10&page=1

- returns
  JSON
  {"has_next": false, "items": [{"description": "a place of historical importance", "typ": "HSTS", "id": "HSTS", "name": "historical site", "cls": "S"}]}

Place

- Get Place
  - GET /place/{id}.json
    returns geojson
    example:

{"geometry": {"type": "Polygon", "coordinates": [[[-73.85529167874355, 40.86177933823652], [-73.85535626390501, 40.861780177444416], [-73.85535516258005, 40.86182902078363], [-73.85529057737267, 40.86182818088898], [-73.85529167874355, 40.86177933823652]]]}, "type": "Feature", "properties": {"importance": null, "feature_code": "BLDG", "id": "5fea12e0f332b42c", "population": null, "is_composite": null, "name": "2425 GARAGE YATES AVENUE", "area": null, "admin": [], "is_primary": true, "alternate": [], "feature_code_name": "building(s)", "timeframe": {}, "uris": ["http://www.nyc.gov/html/dcp/html/bytes/applbyte.shtml.230012"]}}

- Update Place
  - PUT /place/{id}.json
    JSON in body
    example json - notice the extra "comment" metadata value - this gets saved with the revision

{"type":"Feature","geometry":{"type":"Polygon","coordinates":[[[-73.85529167874355,40.86177933823652],[-73.85535626390501,40.861780177444416],[-73.85535516258005,40.86182902078363],[-73.85529057737267,40.86182818088898],[-73.85529167874355,40.86177933823652]]]},"properties":{"importance":null,"feature_code":"BLDG","id":"5fea12e0f332b42c","population":null,"is_composite":null,"name":"2425 GARAGE YATES AVENUE.","area":null,"admin":[],"is_primary":true,"alternate":[],"feature_code_name":"building(s)","timeframe":{},"uris":["http://www.nyc.gov/html/dcp/html/bytes/applbyte.shtml.230012"]},"comment":"sample commit message"}

- Create Place
  - POST /place.json
    JSON in body
    Note that the first URI value has to be unique as this is what the gazetteer uses to generate it's internal id

- Delete place
  - DELETE /place/{id}.json
    Not implemented yet

Place relations

- Get relations
  - GET /place/{id}/relations.json
    returns geojson feature collection{"type": "FeatureCollection", "features": [{ ...

- Add relations
  - PUT /place/{id_1}/{relation}/{id_2}.json
    and JSON Body for "comment"
    where id_1 and id_2 are the ids of places
    relation one of:

    'conflates'
    'contains'
    'replaces'
    'supplants'
    'comprises'
    'conflated_by'

'contained_by'
'replaced_by'
'supplanted_by'
'comprised_by'

a put request to between place A conflates place B will behind the scenes automatically assign place B "conflated_by" place A.
Both places therefore are connected.
Only one relation is possible per couple of places.
If you assign a new relation between places, it removes any existing relation between the 2 places.

- Delete Relation
  - DELETE /place/{id_1}/{relation}/{id_2}.json
    see above

Similar Places

- Get Similar Places
  - GET /place/{id}/similar.json
    Returns GeoJSON feed of similar places.
    total: total number of similar places

Place history

Returns array of revision objects for Place

- Get Place edit history
  - GET /place/{id}/history.json

- response

{"version": 1, "place": "6c314db93b4f1727", "revisions": [{"user_created": "ETL", "created_at": 1367408533.974922, "digest": "13feeb73c5fa342311167259eaa8a0a1ee871081"}]}

Place revision

- Get revision
  - GET /place/{id}/{revision}.json
    gets the place at the specified revision (digest)

- Rollback revision
  - PUT /place/{id}/{revision}.json
    Reverts / rollbacks the place to the specified revision
    JSON in body for "comment" value

LCGazetteer System Components & Tasks

Database

Data model design

- Geographic entities
- Feature types
- Time frames
- Alternate and multilingual names
- Entity (i.e. internal) relationships
- Concordance (i.e. external) relationships
- User and permissions model
- Entity revision history
- Entity revision proposal queue(?)
- Geographic hierarchy (probably continent, admin0, admin1)
- Data model design documentation
- Data model implementation in PostGIS
- Initial feature type import
- External data import and feature type mapping

- Geonames
- OpenStreetMap
- (possibly) Y! GeoPlanet
- LC Authority Records
- other data sources as directed by LC

- iteration to correct generic errors
- Initial entity conflation and deduplication
- Initial entity relationship construction
- Initial geographic hierarchy import
- Initial assignment of entities to geographic hierarchy

- Research into combined bounding box/fuzzy name index optimization
- Index optimization and testing
- Bounding box index
- Time index
- Name index (fuzzy and exact searches)
- Hierarchy index
- Concordance index

Application Programming Interface (API)

API method design and specification

Entity search

- **Name**
- **Bounding box**
- **Hierarchy**
- **Concordance**
- **Time frame, feature type, language filters**
- **Entity retrieval**
- **Entity revision history retrieval**
- **Entity revision**
- **Entity revision rollback**
- **Geographic hierarchy query**
- **Incremental conflation methods**

API service implementation

- Service methods
- Authentication and authorization (OAuth2)

API service testing

- Automated testing
- Monitoring support

Database export

- Full database dump
- Incremental database dump

Application development support

Administrative Interface

Overall user experience design

Entity search and view

- List view
- Map view

Entity creation and revision

Entity geography creation and revision

Entity relationship creation and revision

Region editor

Entity revision history view

Entity revision rollback

Entity revision proposal review(?)

Search result correction(?)

User management

Bulk import interface

Export interface(?)

User experience testing and review

Revision based on UX review

Browsing Interface

Graphic design and consultation

Overall user experience design

Map-based search

- Bounding box

- Name
- Time frame
- Feature type

Search result list

Search result correction interface (admin only)

Detailed entity view

Entity revision history view

Entity revision proposal form(?)

Machine-readable export

User experience testing and review

Revision based on UX review

Additional Tasks

Consultation on OpenStreetMap rendering

TBD

Servers

Location of app
ES index name
PG database name
(any git specifics)

No passwords or secrets here.

http://gazetteer.in

Server: topomancy.com
Location: /srv/gazetteer.in
Apache Access Log: /var/log/apache2/gazetteer.in.log
Apache Error Log: /var/log/apache2/gazetteer.in_errors.log
PG database name: gazetteer_prod
ES Index: search.topomancy.com:gazetteer
Deploy: fab gazetteer_in deploy

http://dev.gazetteer.in

Server: topomancy.com
Location: /srv/dev.gazetteer.in
Apache Access Log: /var/log/apache2/dev.gazetteer.in.log
Apache Error Log: /var/log/apache2/dev.gazetteer.in_errors.log
PG database name: gazetteer_dev
ES Index: search.topomancy.com:dev-gazetteer
Deploy: fab dev_gazetteer_in deploy

http://nypl.gazetteer.in

Server: topomancy.com
Location: /srv/nypl.gazetteer.in
Apache Access Log: /var/log/apache2/nypl.gazetteer.in.log
Apache Error Log: /var/log/apache2/nypl.gazetteer.in_errors.log
PG database name: gazetteer_nypl
ES Index: search.topomancy.com:nyc-dev-gazetteer
Deploy: fab nypl_gazetteer_in deploy

http://dev.nypl.gazetteer.in

Server: topomancy.com
Location: /srv/dev.nypl.gazetteer.in
Apache Access Log: /var/log/apache2/dev.nypl.gazetteer.in.log
Apache Error Log: /var/log/apache2/dev.nypl.gazetteer.in_errors.log
PG database name: gazetteer_dev_nypl
ES Index: search.topomancy.com:nyc-dev-gazetteer
Deploy: fab dev_nypl_gazetteer_in deploy

ElasticSearch

Version used

0.19.11

Is also expected to work with versions 0.20.x and 90.x

Plugins

Install the knapsack plugin https://github.com/jprante/elasticsearch-knapsack

Configuration changes

max_content_length

increase max_content_length from 100mb to 800mb

http.max_content_length 800mb

increase heap size

Increase the Java heap size for the server- on search.topomancy.com is 10gig

Linux file handles

"Make sure to increase the number of open files descriptors on the machine (or for the user running elasticsearch). Setting it to 32k or even 64k is recommended."

see: http://www.elasticsearch.org/guide/reference/setup/installation.html

Import Scripts

Gives information about the scripts needed to import into an ES installation. Assumes you have a number of gzipped json files with places.

Gazetteer Places

/etl/scripts/gazetteer_import.sh

```
sh gazetteer_import.sh dump_dir elasticsearch_host:port index_name
```

- dump_dir - directory to look in to get the place json.gz files (defaults to "dump")
- elasticsearch_host:port - ES server and port (defaults to localhost:9200)
- index_name - index name that you want to import into (defaults to gaztest2)

History Generation

/etl/scripts/history_generate.sh

```
sh history_generate.sh dump_dir index_name
```

- dump_dir - directory to look in to get the place json.gz files (defaults to "dump")
- index_name - index name that you want to import into (defaults to gaztest2)

This generates a number of other gzipped json files in the historydump directory

History Import

/etl/scripts/history_import.sh

```
sh history_import.sh history_dump_dir elasticsearch_host:port history_index_name
```

- history_dump_dir - directory to look in to get the history json.gz files created by history_generate script (defaults to "historydump")
- elasticsearch_host:port - ES server and port (defaults to localhost:9200)
- history_index_name - history index name that you want to import into (defaults to "gaztest2-history")

Examples

sh gazetteer_import.sh /home/tim/gazetteer-release-2012-admin search.topomancy.com:9200 dev-gazetteer

sh gazetteer_import.sh /home/tim/administrative_boundaries_es_dumps search.topomancy.com:9200 dev-gazetteer

sh history_import.sh historydump search.topomancy.com:9200 dev-gazetteer

AdminImport

Steps

1. Parse and Import administrative boundaries.

- Parse Natural Earth Admin 0
- Parse Natural Earth Admin 1
- Parse Tiger/Line States (Admin 1)
- Parse Tiger/Line Counties (Admin 2)
- (parse then into the gz json files)
- Gazetteer history generate
- Gazetteer import these 4 datasets
- Gazetteer history import

2. Conflate NE States with Tiger/Line
So a search for Texas will not show the natural earth entry, but will show the tiger line entry

- python admin_states_conflation.py
- (Note US virgin islands, guam and northern marina wont be conflated)

3. Import into PostGIS / Django from ElasticSearch with buffer for natural earth data

- python admin_extract.py
- (extract the boundaries and puts them into django postgis model, and buffers natural earth data)
  the conflated records wont be put into django / database

4. Assign admin to places json dumps

For each standard place do a point in polygon query to assign the administrative record to the place
This will take a while. On laptop, takes 2 1/2 mins for 5000 geonames records.

- admin_place_assign.py
- python python admin_place_assign.py ../etl/parser/geonames/gazetteer.00200.json.gz ../etl/parser/newgeoname
- adds the admin document to the place

batch mode:
in etl/scripts:

sh admin_assign.sh ../directory/to/existing/gz/files new_directory

5. Generate history from these new records

6. Import records

7. Import history

Conflation

Conflation is the process of identifying two or more feature records as representing the same *physical* place, and updating those features to reflect the identity. Features suitable for conflation may come either from the same source, or, more typically, different sources. Usually, when two or more places are conflated, one is marked as *primary* and the rest subsidiary.

Conflation may be performed automatically or manually. The collection of conflated relations between different datasets represents a *concordance* across those datasets.

Let us define the question of conflation as follows: For any given feature, which we'll call our *subject*, there may be one or more other features in the database that represent the same place. We query the database for the most likely candidates, establish for each candidate a confidence that it can be conflated with the subject, and then choose the candidates which pass a certain threshold.

Distance metrics

There are four attributes to which we can apply *distance metrics* to determine whether two features are indeed the same place:

1. Name
2. Centroid
3. Feature class
4. Feature type

The distance metrics between any two places over these four attributes can be combined to express a *confidence* that the two places are the same. Maximum confidence would obtain when all four attributes are the same, and therefore all distance metrics are by any measure zero. Similarly, minimum confidence would obtain when all attributes are completely different. Let us define the range of confidence as a *normalized* value between 0.0 and 1.0.

Name distance

For names, we can define distance measure as string edit distance, e.g. Levenshtein, which thankfully can be normalized to the lengths of the strings. Since many features have multiple names, the ideal measure to use is the minimum edit distance over the cross-product of the set of names for the subject and a given candidate. Subtracting this value from 1 yields a normalized value for the maximum possible similarity of the name attributes of the two places.

Empirically, we have found that a scaled Levenshtein distance under 0.25 (or even 0.35) is too low for conflation with high confidence. This still allows us to consider alternate versions of very short names, such as "Ham" versus "City of Ham".

Class and type distance

The other attributes are more problematic. Feature class and type values are discrete, and therefore the only possible metric is identity, which is binary. Geographic distance between centroids is effectively unbounded for our purposes. Moreover, we can note that geographic distance has a different meaning when applied to different feature types. Individual buildings occur more densely than cities, while cities occur more densely than provinces. Moreover, densities over all features are much higher in urban areas than in rural or more remote areas.

It stands to reason that a meaningful measure of geographic distance should be scaled to the density of the feature types in question in the vicinity of the subject. *Median absolute deviation* is one example of a robust measure of dispersal in a quantitative sample. For both the feature class and type of the subject, we can scale geographic distances by the MAD of the distances between all pairs of features of that particular class/type within a certain radius of the subject. As a heuristic, we have found in practice that places rarely conflate beyond 25km, so this is the bound we will use. We can then normalize the MAD-scaled distances from the subject feature, per class and type, between zero and maximum, across the entire bounded sample. Subtracting these values from 1 yields a normalized value for the maximum geographic similarity between the subject and any other place.

The similarity measure for either feature class or type becomes less meaningful when the values are not the same for the subject and a candidate. Generally, we can say that feature types (of which there are 600+) may differ between two features that are really the same place, due to inevitable inconsistencies either in the source datasets or in the mapping from the source taxonomy to our own. However, we can adopt a heuristic that worthy candidates must at least share the same feature class (of which only 9). Even when the feature types don't match, the type-scaled distance still reflects geographic distance from the subject within its local context for its *own* type. The incorporation of type distance into our confidence measure can be justified in this case by the assumption that, if the two places prove otherwise conflateable, then the candidate's feature type must be essentially erroneous.

Overall confidence

In principle, we now have derived three similarity values normalized to the range 0.0 - 1.0 for the subject and any candidate. Taking the product of these values yields a similarly normalized value which can be compared between any two pairs of features, and we will call this our *confidence score*.

As a check, we note if all four attributes are identical, the confidence measured this way will be 1.0. If all four attributes are different, the edit distance will be zero, and the product will be zero, regardless of geographic distance. This does, however, preclude detecting the circumstance where two features that occupy an identical location with the same feature type, but have completely different names (e.g. different languages) and no matching alternate names.

The final stage is selecting which candidates to conflate with the subject, for which we need a threshold, for which in turn we have no theoretical basis. We will have to determine the ideal threshold empirically.

Implementation

Revisiting our *similar places* query endpoint in the API, we will need to confirm that, for a given subject, it returns all features which:

1. share the same feature class,
2. lie within our maximum range (25km), and
3. lie within our maximum edit distance (at least 0.25), and
4. are marked primary (i.e. not already conflated).

For conflation, we will also want to ensure that the timeframe for all candidates is exactly the same as the subject. We may wish to allow *similar places* to return features with different time frames for the sake of identifying other kinds of relationships.

The search for conflation candidates can then be carried out with a script that uses the API as follows:

> *For each feature marked primary in the gazetteer: 1. Fetch all similar places.*
> *2. For each candidate, compute the geographic distance.*
> *3. Compute the MAD of the geo distance from the subject to all features.*
> *4. Compute the MAD of the geo distance from the subject to all features of the same type.*
> *5. For each candidate:*
> - *Scale the geo distance by the class MAD to get the scaled class distance.*
> - *Scale the geo distance by the type MAD to get the scaled type distance.*
>
> *6. Sum the scaled class distances. Do same with the scaled type distances.*
> *7. For each candidate:*
> - *Normalize the scaled class distance and subtract it from 1 to get the class similarity.*
> - *Normalize the scaled type distance and subtract it from 1 to get the type similarity.*
> - *Compute the normalized edit distance and subtract it from 1 to get the name similarity.*
> - *Take the product of the three similarities as the confidence.*
> - *If the confidence is above a certain threshold, accept the candidate.*

This process will need to be run a few times and the results eyeballed to determine the optimal threshold empirically.

Once the threshold is identified, another API script can be written that iterates over the list of candidates generated by the previous script, and creates the *conflates* relation from each subject to its candidates, choosing the primary based on its source, with the following priority:

1. TIGER/Line
2. Geonames
3. OSM
4. other datasets

In order to keep from stepping on its own toes, the script will have to check to be sure that each subject has not already been marked subsidiary by a previous conflation step.

BatchImport

documentation about the batch import process - how an admin user can upload a CSV and import records into the system

For the more advanced dataset import parser route, see [[WritingParsers]] documentation. That enables things like specifying multiple languages for alternate names, addresses etc.

Data Format

A text CSV file. Tab separated.

First row of the CSV file should have field names

Field names are in upper case

Field names are as follows:

ID
URIS
NAME
FEATURE_CODE
ALTERNATE
START
END
START_RANGE
END_RANGE
WKT

ALTERNATE and URIS allow multiple entries. Separate these using the pipe | character.

Whilst some fields are optional, when updating, missing fields or empty values will clear the equivalent property of the place.

any additional fields or columns are safe to be in the CSV file but will be ignored when importing. Whilst some values for the columns are optional, the CSV file still needs the headers for them.
The exception to this are Address fields. These are: "NUMBER", "STREET", "CITY", "STATE", "POSTCODE". If these are in the csv file, and a record has data in there, then the place will get an "address" property. Note that the address property is not being actively used at time of writing.

ID

The 16 character ID of a place, if it has one, to update an existing place. **Optional**
This must be left blank to add a new place (although see the note in URIs for potential update behaviour)

URIS

A string describing the URIs of the place. **Required**.
The **first URI has to be unique** - the ID of the place is based on this

examples "http://osm.org/node/1234" "http://osm.org/node/1234|our_route123"

NOTE: If the first URI belongs to an already existing place in the database, even if the row has no ID, then the batch loading process will update/overwrite that place with the details in the row.

NAME

The name of the place. **Required**, does not have to be unique. If a row does not have a name, that row will be skipped.

FEATURE_CODE

The geonames feature code to use. **Required**

example "BLDG"

ALTERNATE

Alternate names. **Optional**. Separate mutliple names using the pipe character

examples: "leeds" "leeds|leodis"

Alternate names will be assumed to be of language "en"

START

Start date. Optional.

In format YYYY-MM-DD

example: "2012-05-20"

END

End date. Optional.

In format YYYY-MM-DD

example: "2012-05-20"

START_RANGE

Start range. optional

integer - represents number of days

example : "14"

END_RANGE

End range. optional

integer - represents number of days

example : "14"

WKT

Well Known Text geometry representation. **Required**

example "POINT (30 10)"

Process

1. Create and upload

- Log in as a Django admin.
- Add new batch import

admin/gazetteer/batchimport/add/

- Enter in a name for the import.
- Assign the user for use in the metadata when importing the records.
- Upload the CSV File.
- Click Save

2. Validations

The CSV file will have some simple validations run on it. These are:

- Check to see if the CSV has all the fields
- checks to see if each row has a feature_code
- checks to see if each row has a URI
- checks to see if there are duplicate first URIS within that file

3. Run import

Once that has been saved and created, the record needs to be edited.

- edit record
- Check the "Start Import" checkbox
- click save button

The records will be imported.

The batch import record will be updated with the number of records imported and the time when it was imported.

Considerations

The following are not currently possible using this batch upload solution:

- Alternate names in other languages
- Custom administrative entries
- Custom centroids
- population
- Area
- Importance
- Miscellaneous storage of additional fields (source)

As such it is recommended that a custom parser is written. See [[WritingParsers]] for more information.

Validations are currently basic and will not cover the entire spectrum. URIs are not checked for existence with the data in current database, for example.

Finding the records you have just entered should be possible using the updated field query string param

"* updated:[2013-01-01 TO 2013-12-01]"

Data Examples

Here, there are two columns which are ignored(materials, layer_year) and two which are empty (start_range, end_range)
Tab separated, so the formatting looks off!

ID URIS NAME FEATURE_CODE ALTERNATE START END START_RANGE END_RANGE materials WKT layer_year
buildings.114153 S. Eighth St.1 BLDG pizza house 2012-01-10 2013-01-01 0 20 brick POLYGON ((-73.966043813826 40.709680137063, -73.966081364752 40.709472752459, -73.966025038363 40.709466652902, -73.965990169646 40.709678103883, -73.966043813826 40.709680137063)) 1886

Within the code base see the CSV files in the data directory: data/batch_import1.csv

Files

build_tab_csv2.csv	2.27 KB	05/22/2013	Tim Waters

Import and Export - ElasticSearch and Postgres

Notes on exporting and importing directly from elasticsearch using the pgdump and the Knapsack ElasticSearch plugin

You can export (and reimport) an entire index at once.

Please note the following from the knapsack readme:

> *Knapsack is very simple and works without locking or index snapshots.*
> *So it is up to you to organize the safe export and import. If the index*
> *changes while Knapsack is exporting, you may lose data in the export.*
> *Do not run Knapsack in parallel on the same export.*

Install Knapsack on the export server

https://github.com/jprante/elasticsearch-knapsack

export the index (not just the type)

Use the target for export - specify the tar.gz

```
target=/big/space/gazetteer-index.tar.gz
```

make sure the filename is the same name as the index

```
curl -XPOST localhost:9200/dev-gazetteer/_export?target=/var/lib/elasticsearch/dev-gazetteer.tar.gz
curl -XPOST localhost:9200/dev-gazetteer-history/_export?target=/var/lib/elasticsearch/dev-gazetter-history.tar.gz
```

The response for these will be a bit of json:

```
{"ok":true}
```

However, the export will take a few minutes (main LC index took ~10 minutes on Topomancy server) and so you should tail the elasticsearch logs to see any output:

```
tail -f /var/log/elasticsearch/topomancy.log
```

```
[2013-06-29 19:09:49,757][INFO ][rest.action               ] [Saint Sebastian] starting export to
/var/lib/elasticsearch/dev-gazetteer.tar.gz
[2013-06-29 19:19:10,560][INFO ][rest.action               ] [Saint Sebastian] export to /var/lib/elasticsearch/dev-gazetteer.tar.gz
completed
```

export admin boundaries

```
pg_dump -t 'gazetteer_adminboundary'  gazfromdb > admin_boundary.sql
```

import indexes

```
curl -XPOST localhost:9200/new-gazetteer/_import?target=/var/lib/elasticsearch/dev-gazetteer
curl -XPOST localhost:9200/new-gazetteer-history/_import?target=/var/lib/elasticsearch/dev-gazetteer-history
```

use target, but leave out the tar.gz

import admin boundaries

psql -d gaztodb -f admin_boundary.sql

Writing Parsers

Describing a parser, the import format and structure and the helper libraries

What is a parser?

A parser reads in a dataset and parses it into a format suitable for importing into the elasticsearch database using the import scripts. It is different from the batch import feature in that it allows for parsers to have their own logic when reading the datasets, allows addresses, advanced alternate names and more detail to be stored.

Examples of parsers

Look in the code base in the etl/parsers directory. There are parsers for a wide range of datasets, from shapefiles to csv files and more. For more specific examples, look at the OSM parser, Geonames parser and LC Auth parsers.

The JSON dump format.

The result of a parser is a JSON file similar to the following:

```
{"index": {"_id": "41dab90514cfc28e"}}
{"relationships": [], "admin": [], "updated": "2006-01-15T01:00:00+01:00", "name": "Vonasek Dam",
 "geometry": {"type": "Point", "coordinates": [-98.20869, 42.67167]},
"is_primary": true, "uris": ["geonames.org/5081200"], "feature_code": "DAM",
"centroid": [-98.20869, 42.67167], "timeframe": {}}
```

Where the first line has the UUID for the place (based on the first uris value, which must be unique), and the second line has the properties for the place.

The place JSON mapping

The mapping for the place is similar to what follows.

```
"name":"a name",
"centroid":[-98.20869, 42.67167],
"feature_code": "HSTS",
"geometry":{"type": "Point", "coordinates": [-98.20869, 42.67167]},
"is_primary": True,
"source": {},
"alternate": [],
"updated": "2006-01-15T01:00:00+01:00",
"uris":["geonames.org/5081200],
"relationships": [],
"timeframe":{},
"admin":[],
"address": {}
```

You will see that there are some empty values. These are:

- source
  - An optional javascript hash where you can store other information in there, for example, other fields in the dataset which are not parsed out.

- alternate
  - optional alternate names array
  - Example
    "alternate": [{"type": "link", "lang":"en", "name": "http://en.wikipedia.org/wiki/Mount_Chocorua"}, {"lang":"fr", "Mt Chocoura"}]

- relationships
  - Keep this as an empty Array

- timeframe
  - Optional timeframe hash where start and end are in YYY-MM-DD and start_range and end_range are in number of days
  - Example
    "timeframe": {"end_range": 12,"start": "1800-01-01","end": "1900-01-01","start_range": 0 }

- Admin
  - Is almost always kept as an empty array.
  - Exception: You could give the place an administrative boundary which does not exist in the gazetteer - in which case leave the id value blank.
  - Example : [{"name" : "unknown boundary", "feature_code" : "ADM4" , "alternate_names": []}]

- Address
  - Optional hash with the following structure

number:string
street:string
city:string
postcode:string

  - Example:

{"city": "New York City", "number": "", "state": "NY", "street": "East 42nd Street"}

The core library.

The parsers use the core library module to help parse the datasets. These gz the json files, calculate the ids and help open tab formatted CSV files.

Found in /etl/parsers/core.py

Considerations when writing

- Make sure that the URI is unique to the dataset, and each place has a unique URI itself.
- UTF-8 Encoded please

Source And Origin

describing how to add a new origin / source to the django interface or source fixtures file and what that means.

Source / Origin

Enables the application to display and search for places based on their first URI - the origin of the data.
When adding data from a source, the first URI should both be unique to the place, but also unique to the dataset.
Internally, these are used and translated into a query string.

Properties

Name
Display name - used in the UI

Description

not displayed - this is for management of the data within the django interface

Example

Not displayed - for use when crafting the code field

Code

The code to be used in the ElasticSearch query. This can accept wildcards, and should be unique. The code is used in the equivalent query : feature_code:{code} e.g. a query string could be:"battlefield feature_code:hmdb"

Fixtures

Initial fixtures for NYPL and LoC projects are in the fixtures folder. In JSON format.
A developer can edit this file and import them.

Once initially imported an admin user should use the django interface to maintain and edit these

Loading Fixtures

```
python manage.py loaddata origins_nypl.json
```

Examples

HMDB (small, simple)

```
"name": "HMDB",
"description": "Historical Marker Database",
"example": "http://www.hmdb.org/marker.asp?marker=37624#32902",
"code": "hmdb"
```

Perris Layer (longer, with slashes)

```
"name": "Perris Map New York, 1854",
"description": "Maps of the city of New York / surveyed under directions of insurance companies of said city. ",
"example": "http://maps.nypl.org/warper/layers/861.buildings.76989",
"code": "maps.nypl.org/warper/layers/861"
```

New York Block (with Wildcard)

"name": "NYC Tax Lots",
"description": "NYC Tax Lots",
"example": "https://data.cityofnewyork.us/Property/Department-of-Finance-Digital-Tax-Map/smk3-tmxj#lot/3041110053",
"code": "cityofnewyork*lot"

OpenStreetMap

This set of commands builds and dumps a set of osm2pgsql tables for the entire planet.osm database using the gazetteer place.style.

The code is meant to run on a EC2 m2.4xlarge instance with 64 GB of RAM and a sizeable instance store on /mnt.

Install PostgreSQL/PostGIS and the prereqs for building osm2pgsql

```
sudo apt-get update
sudo apt-get --yes install git-core libgeos-dev libxml2-dev libpq-dev libbz2-dev \
   protobuf-c-compiler libprotobuf-c0-dev build-essential devscripts debhelper \
   libgeos++-dev postgresql-9.1-postgis automake autoconf libtool libproj-dev \
   postgresql-contrib
```

Create a PostgreSQL user for `ubuntu`

```
sudo su -c 'createuser -s ubuntu' postgres
```

Fetch schuyler's branch of osm2pgsql and build it

This actually builds two packages: osm2pgsql and openstreetmap-postgis-db-setup, which creates a PostGIS database.

```
cd /mnt
git clone https://github.com/schuyler/osm2pgsql.git
cd osm2pgsql/
debuild
```

Non-interactively install the built .debs

Install the package for osm2pgsql and the `gis` database used by osm2pgsql.

The DEBIAN_FRONTEND shenanigans is to get the database setup package to install without asking stupid questions, and the double 'dpkg --install' lets us apt-get install missing dependencies.

```
sudo su -c 'DEBIAN_FRONTEND=noninteractive dpkg --install ../*.deb'
sudo apt-get --yes --fix-broken install # this step should be unnecessary
```

Tune PostgreSQL and the Linux kernel to use lots of RAM

The postgresql.conf and sysctl.conf files live in the gazetteer repo under etl/osm2pgsql/.

```
sudo bash -c 'cat postgresql.conf >>/etc/postgresql/9.1/main/postgresql.conf'
sudo bash -c 'cat sysctl.conf >>/etc/sysctl.d/30-postgresql-shm.conf'
sudo sysctl -w kernel.shmmax=9000000000
sudo sysctl -w kernel.shmall=9000000000
```

Stop PostgreSQL, move the database into the instance store, and restart it

```
sudo /etc/init.d/postgresql stop
sudo cp -a /var/lib/postgresql /mnt
sudo rm -r /var/lib/postgresql
sudo ln -s /mnt/postgresql /var/lib
sudo /etc/init.d/postgresql start
```

Download planet-latest.osm.pbf

```
cd /mnt
sudo chmod 777 .
wget -c http://planet.openstreetmap.org/pbf/planet-latest.osm.pbf
```

Run the database import

This import relies on a custom osm2pgsql "style" which lives in etl/osm2pgsql/place.style and which determines which OSM objects get loaded into PostgreSQL, and with which columns.

The --hstore-column parameter lets load in all of the names, regardless of language.

```
time osm2pgsql --latlong --multi-geometry --cache 32000 \
  --input-reader pbf --create --unlogged \
  --hstore-column "name:" --extra-attributes \
  --style ~/gazetteer/etl/osm2pgsql/place.style \
  planet-latest.osm.pbf
```

This process takes about 2.5 hours as described above.

Run the OSM parser to generate gazetteer records from PostgreSQL

The osm.py parser takes two arguments: The name of the PostgreSQL database (gis per osm2pgsql's default) and the name of the directory in which to create the gazetteer record dump.

```
python gazetteer/etl/parser/osm.py gis dump
```

The records are then loaded into the gazetteer in the usual way. The parser uses the feature type mappings from `osm_type.py`, which were compiled by hand using the 100 or so most commonly used OSM tags.

Geonames

The Geonames.org dataset can be converted to gazetteer records using the parser in etl/parser/geonames.py.

The parser uses the allCountries.txt and alternateNames.txt files, which must be pre-sorted:

```
sort -n allCountries.txt > allCountries.sorted.txt
sort -k2 -n alternateNames.txt > alternateNames.sorted.txt
```

The parser takes two arguments: the path to the directory containing the Geonames files, and the desired path for writing the record output.

```
python etl/parser/geonames.py path/to/geonames/data/ path/to/dump/files/
```

The record dumps can then be loaded into the gazetteer in the usual way.

Data Sources for Gazetteer/Concordance Service

LC Name and Subject Authority Records
http://id.loc.gov/download/

MARC Geographic Records
(for name variants of geographic entities linked in authority records).
http://id.loc.gov/vocabulary/geographicAreas.html
http://www.loc.gov/marc/geoareas/gacs_name.html

LC/ECS Region Authority Records
(To be provided by Sandra, LC/ECS staff)

Issues with Authority Records

1. There are both "name" and "subject" authority records. Geographic entities appear in both of them, but not always in the same format.

2. The easiest way to start is the "name" authority records, with a "type" of geographic. Those are the clearest examples of geographic locations.

Those are the ones that seem to have lat long assigned to them. The Library of Congress worked with another organization, OCLC, to have the lat long assigned to these records -- I believe they were matched with Geonames. So, you should be able to match them with Geonames yourself when it's a clean match.

Here's an example of a simple one for a placename in the US.
http://id.loc.gov/authorities/names/n82014081.html

3. If you look around the subject headings (you can search more easily in authorities.loc.gov), you will see that geographic names are used in subject headings often as "regions" or as part of other topics. They sometimes (but not always) appear in formatted sub-headings, separated by "--".

http://id.loc.gov/authorities/subjects/sh85054467.html
http://id.loc.gov/authorities/subjects/sh97001885.html
http://id.loc.gov/authorities/subjects/sh2010109473.html
http://id.loc.gov/authorities/subjects/sh2010006642.html

You may find the "type" code to be helpful in the analysis.

4. The use case requirement is that the Gazetteer should be able to map any geographic term that appears in all of the name and subject authority records when presented to the Gazetteer. I think that will require some analysis and programming work to make that happen. We welcome your recommendations on how to proceed.

5. The MARC List of Geographic Area Codes (another entity under id.loc.gov) identifies separate countries, first order political divisions of some countries, regions, geographic features, areas in outer space, and celestial bodies. The list's codes are one-to-seven lowercase alphabetic strings that serve as identifiers. These words are used in the geographic entity entries in the "name" and "subject" authority records (basically a controlled vocabulary).

I'm guessing you'll want to include these somehow. You might find this helpful (or you might want to run screaming from the room):
http://www.loc.gov/marc/uma/

Recommendations on Treating Authority Records

The options that seem the most obvious to start with (and they are not mutually exclusive):

Option 1

Conflation of LC authority records with Gazetteer records, as feasible and practical.

Option 1.1

Conflate records from the Name Authority file and the Subject Authority file with type = "geographic" with Gazetteer records, to the extent possible based on automated matching. Document this in the conflated record. I assume the "name" from the authority record would be an alternate name in the conflated record. If it is not feasible to match exactly with a one-for-one Gazetteer record, develop recommendations for how to deal with the unmatched records with type = "geographic".

Option 1.2
Topomancy looks at all the other types of authority records, and determines if any others can be conflated. Prepare recommendation to LC on how to proceed.

Option 1.3
Add "region" authority records as new Gazetteer records (i.e., are not already in the Gazetteer). Please work with Sandra Hoyer on this; ECS has the information on the "region" records that LC staff provided to us (for example, what states are included in "Southern States").

Option 2.

Parse the geographic terms in all LC authority records and determine what will happen if each string is entered as a query to the Gazetteer (in the API or the website). For examples,

http://id.loc.gov/authorities/subjects/sh85050898.html
Fort Sanders, Battle of, Knoxville, Tenn., 1863
If this string text was submitted to the Gazetteer API, what would the Gazetteer return?

http://id.loc.gov/authorities/subjects/sh87002049.html
Railroads--Tennessee
If this string text was submitted to the Gazetteer API, what would the Gazetteer return?

http://id.loc.gov/authorities/subjects/sh2009118668.html
Central business districts--Michigan--Detroit--Maps
If this string text was submitted to the Gazetteer API, what would the Gazetteer return?

http://id.loc.gov/authorities/subjects/sh94000545.html
Robert Gould Shaw Memorial (Boston, Mass.)
If this string text was submitted to the Gazetteer API, what would the Gazetteer return?

http://id.loc.gov/authorities/names/no2010029436.html
Elks (Fraternal order). Lodge No. 2 (Philadelphia, Penn.)
If this string text was submitted to the Gazetteer API, what would the Gazetteer return?

Option 3:

Propose recommendations for updates to the Gazetteer if updates are done to the authority records.

HistoricalDataSources

See:http://support.topomancy.com/issues/160

Notes about NHGIS, NPS, and HMDB Imports

*National Historical GIS (NHGIS)
*http://nhgis.org

These datasets are the only ones we identified which contain time frame or periodicity. We selected U.S. state, territory, and county boundaries for every decade from 1790 to 2000, which are included as polygons in our Gazetteer database. For the sake of simplicity, we pulled together all of the records, simplified them to a tolerance of 0.01º (approx. 1km), and merged the records over the decades for which a geometry did not change (viz. boundaries remained stable). Our database thus includes decade ranges and geometries for both states and counties.

*NPS Historic Places Register
*http://nrhp.focus.nps.gov/natreg/docs/Download.html

The National Park Service (NPS) Register of Historic Places, suggested by the Library for inclusion, contains several rich datasets, in different open (kml, gpx, shp) and proprietary formats (mdb) and containing points, lines and polygons. For our Gazetteer we used their set of KML files containing around 76,293 points for historic places. Unfortunately, the dataset provides no structured feature type information. Further analysis processing of the name fields could yield a provisional feature typology for the various monuments, sites, and historic places.

*Historical Place Marker Database (HMDB)
*http://hpdb.org

Also suggested for inclusion by the Library, HPDB provided 41,067 points in GPX file format, both derived from public sources such as NPS and also user-contributed via their website. The copyright specifies no restrictions on non-commercial use, though the reliability of the data and evenness of coverage requires further analysis for inclusion in a full implementation.

NHGIS (county and state historic boundaries 1790 -2000)

Shapefiles were merged in postgis to merge the shapes where the geometry was different and extend the year range.

THese shapefiles FYI are in /home/sderle/nhgis/merged (nhgis/reprojected for the source ones).
Parser was updated for these shapefiles and new dumps created. Dumps are in /home/tim/nhgis/merged_dumps

Getting Data

Select data from NHGIS:

This is for all of the boundaries:

> county OR nation OR state
> Years
> 1790 OR 1800 OR 1810 OR 1820 OR 1830 OR
> 1840 OR 1850 OR 1860 OR 1870 OR 1880 OR
> 1890 OR 1900 OR 1910 OR 1920 OR 1930 OR
> 1940 OR 1950 OR 1960 OR 1970 OR 1980 OR
> 1990 OR 2000 OR 2010

https://data2.nhgis.org/downloads/revise/29182 "all boundaries"

However, this was unable to be downloaded - possibly too big for their servers. Thus, we split these:

https://data2.nhgis.org/downloads/revise/29183 To 1810
https://data2.nhgis.org/downloads/revise/29186 To 1850
https://data2.nhgis.org/downloads/revise/29188 To 1910
https://data2.nhgis.org/downloads/revise/29189 To 1970
https://data2.nhgis.org/downloads/revise/29191 To 2010

These should be able to be accessed by anyone with an nhgis account

Processing shapefiles.

From the above downloads you get 149 NHGIS Shapefiles totalling 6 Gig (compressed it's 4.4 Gig) in size.

Extracting to 1 directory

```
while [ "`find . -type f -name '*.zip' | wc -l`" -gt 0 ]; do find -type f -name "*.zip" -exec unzip -- '{}' \; -exec rm -- '{}' \;; done
```

Several of the 149 files are "conflated" versions. These I think are versions that have the borders changed so that they can be compared with other decades. Thus, we will remove the conflated files.

```
rm *conflated*
```

Skip the 2010 files as they are already identical to the areas and places in the census (tiger/line) dataset

```
rm  *_2010*
```

Projections:

Most NHGIS shapefiles use one of ArcGIS's predefined projected coordinate systems for North America; either the USA Contiguous Albers Equal Area Conic, the Alaska Albers Equal Area Conic or the Hawaii Albers Equal Area Conic, as appropriate.
Shapefiles for Puerto Rico use an Albers Equal Area Conic with central meridian, standard parallels and latitude of origin set to match the Puerto Rico State Plane Coordinate System's.

The gazetteer uses EPSG:4326 and so all 149 shapefiles need to be converted to EPSG:4326

So, need to convert to EPSG 4326 for each.

reproject.sh

```
#!/bin/bash/
for i in $(ls *.shp); do
  ogr2ogr -t_srs EPSG:4326  reprojected/$i $i
done
```

Merging shapefiles.

Each shapefile represents a decade, 10 years. However many counties and states have not changed from decade to decade, and so we can merge these shapes together. When we merge the shapes together, we merge the date ranges.

For example Orange County has only changed shape once, from 1800 to 1900 and from 1900 to 2010 - then it will be individually present in each shapefile.
We can create one shapefile where only two Orange County shapefiles exist - merging the date ranges together.
One shape will be 1700-1900 and the other shape will be 1900-2010.

Thus, instead of 20 or so shapes in the database each with a decade for the time span, we have 2 shapes each with a different century as the time span.

See /etl/parser/nhsgis/README.txt for more details.

We use PostGIS spatial database to do the heavy lifting. We import the reprojected shapefiles into the database, and union or group the all the shapes together..

```
# new database
createdb -Ttemplate_postgis nhgis

# import the NHGIS shapefiles
unzip /home/tim/nhgis_shapefiles/nhgis_reprojected.zip
cd reprojected/
for i in *.shp; do shp2pgsql -s4326 $i ${i%%.shp}; done | psql nhgis
```

```
# create the all_tables view
echo '\d' | psql nhgis >view.sql
vi view.sql # a whole bunch of munging by hand with regexes in vim
psql nhgis <view.sql

# group and extract the objects
psql nhgis <extract.sql
cd ..
mkdir merged
pgsql2shp -f merged/nhgis_merged.shp nhgis merged
```

These merged shapefiles were then processed using the etl/parser/nhgis.py parser

Examples

Example: Richmond City, VA

Showing all the county boundaries for Richmond City thoughout the years
http://dev.gazetteer.in/?q=Richmond%20City&feature_type=ADM2H&page=1&lat=37.51667020955629&lon=-77.46599609375&zoom=12

Showing just the boundary for 1910-1915
http://dev.gazetteer.in/?q=Richmond%20City&feature_type=ADM2H&page=1&start_date=1910&end_date=1915&lat=37.51667020955629&lon=-77.464599609375&zoom=12

Showing all historical counties nearby for that same time frame:
http://dev.gazetteer.in/?q=*&feature_type=ADM2H&page=1&start_date=1910&end_date=1915&lat=37.00858404683155&lon=-76.58843994140625&zoom=10

Showing all the "Washington Counties" on the NE over time.
http://dev.gazetteer.in/?q=washington%20county&feature_type=ADM2H&page=1&lat=38.34596449365382&lon=-7.25561523437499&zoom=7

States:
States from 1790 - 1801
http://dev.gazetteer.in//?q=*&feature_type=ADM1H&page=1&start_date=1790&end_date=1801&lat=38.2597480039479&lon=-76.607666015625&zoom=7

HRHP National Register of Historical Places (64,306 points of historical site information)

Downloaded from http://nrhp.focus.nps.gov/natreg/docs/Download.html

Download the spatial.mdb database http://nrhp.focus.nps.gov/natreg/docs/spatial.mdb

Using a Windows 7 Machine and the freeware MDB Viewer Plus software - opened and exported the "points" table into a CSV file

This csv table was able to be read by the /etl/parser/nrhp.py parser

Created from the spatial.mdb using the points and main table and reprojecting the data.

original place dumps at tim/nrhp/nrhp_dump ,

admin assigned using gazetteer dev (search.topomancy.com - dev-gazetteer) /home/tim/nrhp/nrhp_dump_with_admin

history dumps /home/tim/nrhp/nrhp_admin_history_dump

all added into dev.gazetteer.in

examples

See: http://dev.gazetteer.in/?q=uris%3A*nrhp*&page=1&lat=34.56085936708384&lon=-95.09765625&zoom=5

Historic Sites, House in CA:

http://dev.gazetteer.in/?q=House%20CA%20&feature_type=HSTS&page=1&lat=34.633207911379 59&lon=-116.80640625&zoom=5

HMDB (56K historic place markers)

From hmdb.org downloaded as .GPX

```
wget http://www.hmdb.org/gpx/gpx.asp
```

using quantum GIS converted this to shapefile

Using etl/parser/hmdb.py parser.

hmdb orig dumps: /home/tim/hmdb/orig

hmdb with admin assigns (dev.gaz) /home/tim/hmdb/hmdb_with_admin

hmdb history dumps: /home/tim/hmdb/history_dumps

example searches

Example searches:

Source is hmdb, Type is Historical Site. and Montana
http://dev.gazetteer.in/?q=uris%3A*hmdb*%20Montana&feature_type=HSTS&page=1&lat=47.953144950155 94&lo=-114.76318359375&zoom=6

Search for Cowboy in USA with source is hmdb

http://dev.gazetteer.in/?q=uris%3A*hmdb*%20cowboy&page=1&lat=40.01078714046552&lon=-99.140625&zoom=4

LC Gazetteer Internship

Workflow

The interns will be:
1. Creating new Gazetteer records
2. Modifying Gazetteer records.
3. Testing the Gazetteer to see if they get the results expected.

They will be using the list of "unmatched" entries that ECS found when trying to match values in LC records with Geonames entries.

We already had the interns set up spreadsheets with the proposed actions, based on their analysis of "unmatched" entries. Here are examples.

Examples

Case no.1 "Phila"

1. Geographic value in LC record: "Phila"
2. Current result: no match
3. Desired result: Synonym for "Philadelphia, PA, US".
4. Administrator (intern) enters data into Gazetteer to set this up. (I assume this would be a "alternative" name in the record for Philadelphia?).
5. Intern uses user interface to test this (search "Phila"..get "Philadelphia").

Solutions for no.1

a. Philia can be added as an "alternate name" for Philadelphia
b. "Philadelphia, PA, US" utilise administrative boundaries (PA, US) where each place has been assigned the county, state and country which it is in.
Example:
- Went to: http://dev.gazetteer.in/feature/b300a43109a8b754 (Philadelphia from OSM)
- clicked on alternate names, clicked edit
- Added "philia" as an alternate name
- back to search. Search for "philia"

http://dev.gazetteer.in/?q=philia&feature_type=PPL&page=1&lat=39.986590631428534&lon=-74.97344970703124&zoom=9

Development status

Resolved.

Case no.2 "US South"

1. Geographic value in LC record: "US South"
2. Current result: no match
3. Desired result: variant of region "Southern States", which is an LC authority record.
4. Administrator (intern) enters data into Gazetteer to set this up. (I assume this would be a "alternative" name in a new record set up for "Southern States"?)
5. How can a bounding box be set up for the coordinates for this record?
6. Intern uses user interface to test this.

Solutions for no.2

a. This is an example of the "Composite Place" feature

Development status

In development see tickets #177 "Region / composite place Editor (US Southern States)"
Database and backend development completed.

TODO: Front end UI and API methods.

Likely workflow:

- "create new composite", edit the name and feature type, go to add components (which is just like add relationships), type into a search box
- from the search results you tick checkboxes for the ones to add, then click "add"
- the new boundaries of the composite object are computed and shown on the map
- composite places can also have alternate names, other relationships, just like any other kind of place
- the only thing they don't have is a geography independent of their component

Example:

- create a new place. Mark it as a composite place
- Add in the name "Southern States".
- Add alternate name "US South"
- Add component states to this new place as relations
- Save place
- Search for "US South" - shows new place with the geometry of this composed of the component states, and with a list / map of the component states shown.

Case no.3 "Fort Panmure"

1. Geographic value in LC record: "Fort Panmure, Mississippi".
2. Current result: no match. (It's a historical placename).
3. Desired result: intern proposes the closest Geonames feature to use.
4. Administrator (intern) enters data into Gazetteer to set this up. (This requires some kind of Gazetteer function or look-up table that maps the historical placename to the desired Geonames placename).
5. Intern uses user interface to test this.

Solutions for no.3

a. Fort Panmure as an alternate name for an existing place in the same location.
b. Fort Panmure as a new place with type Historical Site.
c. Fort Panmure as a historical site with a timeframe of when it was present.

Development Status

Backend and API completed and resolved.

For the user interface, in development: see #195 "Geometry Editing" and #196 "Create new places in UI"

Case no.4 "Pacific Coast"

1. Geographic value in LC record: Pacific Coast, South America.
2. Current result: no match
3. Desired result: Not clear. We would ask intern or Topomancy to suggest a good result.
4. Administrator (intern) enters data into Gazetteer to set this up.
5. Intern uses user interface to test this.

Solutions for no.4

This is an example of a composite place where the components of Pacific Coast are chosen and added to the place. See case no. 2

Development Status

in development (See notes in case no. 2 above regarding composite places)

Case no.5 "La Haye"

1. Geographic value in LC record: "La Haye"
2. Current result: no match
3. Desired result: Synonym for "The Hague"
4. Administrator (intern) enters data into Gazetteer to set this up. (I assume this would be a "alternative" name in the record for The Hague?

5. Intern uses user interface to test this.

Solutions for no.5

This is an example of adding an alternate name to an existing place.

example:
See http://dev.gazetteer.in/feature/356e889037309f2f
Click on alternate names. Observe mutliple different names for the place.

Development Status

Resolved

Case no.6

1. Geographic value in LC record: "Hudson River Valley- New Jersey"
2. Current result: no match
3. Desired result: Synonym for LC authority record "Hudson River Valley (NY and NJ)"
4. Administrator (intern) enters data into Gazetteer to set this up.
4.1 Need advice from Topomancy on how to set up a Gazetteer record and coordinates for "Hudson River Valley (NY and NJ)"
4.2 set up new Gazetteer record.
4.3 Set up alternate for "Hudson River Valley- New Jersey"
5. Intern uses user interface to test this.

Solutions for no.6

a. Creating a new place, possibly a composite place if made up of existing components, or...
b. Creating a new place and drawing in geometry manually

Development Status
Backend and API completed and resolved.

For the user interface, in development: see #195 "Geometry Editing" and #196 "Create new places in UI"

LC Gazetteer Priority Users Storyboards

From: https://docs.google.com/document/d/18Xhr61YhPNXdQXnmdkz31BuooQR9DDCWLboTRzTz-JE/edit#

Aim

This document extends the previous “LoC Gazetteer User Story Boards” document.
It’s intention is to draft a list of activities for a user to create a workplan of the classes of task that the user would like to take. For example, “Amanda wants to find, conflate and clean the record for Istanbul”.

The two main classes of user identified are the Admin Expert and the Developer Expert users.

Admin expert – Amanda

This is the highest priority user.

An employee whose job it is to maintain geographic records in the gaz.

Amanda is able to edit features, and uses the full application. She is a point of contact (email / telephone) on the application. She works with other editors to see the changes made and rollback any wrong ones. LC-staff members who are not editors can "suggest" edits by actually making changes and including a justification that then Amanda can accept, reject, or revise and accept.

Activities

Finding records based on a name i.e. “Istanbul”
Finding records based on a name and a time/era/epoch “Constantinople”, “8th Century”
Adding in additional records manually
Conflation of records - identification and merging of duplicate records.
Correcting erroneous conflations.
User Administration

Developer expert – Dave

A geospatial or librarian web developer who wants to know how to use and get started with the API and website.

Dave fully explores the site and access the data externally, using the API with other geospatial and/or information applications.

This would be the highest priority for use of the gazetteer. The aim is to use the gazetteer as an authoritative look-up for the geographic search facets and coordinates for our www.loc.gov/search and object display. It is also wanted to use the gazetteer to get coordinates and data for map displays being developed as future enhancements for this application.

Activities

Dave will use the following workflow:
Metadata records from LC data sources are pulled for ETL transformation
Specific locational values are passed to Gazetteer (as part of ETL transform process) as text strings.
The gazetteer API returns values from the specified fields/values.
Output values from Gazetteer become values for the geographic location
Final result is presented as a facet in Search.

LC Gazetteer User Story Boards

From:
https://docs.google.com/document/d/14yJuqyNQlrXI8dFSlWRbsIm_-D4_0sYUESbTKA6j2uM/edit

also see for more comments and amendments:
https://docs.google.com/a/topomancy.com/document/d/1VQQhBEJhPRvdUS-rs8w8n18Fyfu45q8zscPfWj41eJg/edit

Aim

To visualise at a high level the likely workflow and operations for the main users for the application. Things to help visualise the likely workflows of many different users.
Actors

- public casual/expert
- internal casual/expert
- admin expert
- developer casual/developer

Actors StoryBoard

Here, the above actors are given a brief high level description of the likely things they would do.

Public casual - George

Found the main website and the gazetteer and wants to know what's going on.

George has visited http://www.loc.gov/index.html and seen a link to the new gazatteer.
George sees an empty map and a search box and types in his home town.
The results from his search appears in the results and are shown on the map
George clicks on the top result, sees the map move and then clicks on the second result.
He sees a link to the about page and clicks this, scans over it and leaves.

Public expert - Annabel

A user who understands gazetteers and libraries and wants to explore.

Annabel understands gazetteers and wants to see how its been used in a library setting. She searches for a location, sees the results and clicks through to see more details. She notes that a feature can have different names and that some features have related features. She explores a bit more, bookmarks a page but doesn't look at the API documentation.

Internal casual - Mary

An employee, has heard about the app and wants to understand more. Possibly a librarian?

Mary understands metadata records, is a librarian, and wants to see how the gazetteer is being used with regards to her work. Of particular interest is the authority records. She gets to see how one authority record can have different names and similar features.

Internal expert - Bill

An employee who works with library data and wants to understand the application with a view to working with it.

Bill gets to explore all the corners of the application, is able to see how authority records reference to other features, how some places have different names, and how he could use the search engine to see similar records, for cross linking to other catalogues. He sees the API documentation and has heard of external applications using it.

Admin expert - Amanda

An employee whose job it is to maintain geographic records in the gaz.

Amanda is able to edit features, and uses the full application. She is the point of contact (email / telephone) on the application. She works with other editors to see the changes made and rollback any wrong ones. She receives emails from other members of the LoC telling her that some places are wrong, and then responds to those emails by correcting the items in the application.

Developer casual - Robert

A web developer who wants to understand what's available through the website and API for future projects

Robert explores the site briefly and uses the API documentation to build something during a hackday for a charity.

Developer expert - Dave

A geospatial or librarian web developer who wants to know how to use and get started with the API and website.

Dave fully explores the site and access the data externally, using the API with other geospatial and/or information applications.

SearchTerms

General

AND OR works for names. By default, AND is assumed.

Substring search

use the wildcard asterisk

"denv*" to get denver

Only works for words beginning with the substring

http://dev.gazetteer.in/?q=denv*&lat=39.620499321968104&lon=-104.78759765625&zoom=8&page=1

Search for everything in BBOX

add * in search box

http://dev.gazetteer.in/?q=uris%3A*census*&lat=40.73698221818716&lon=-73.91704559326172&zoom=12&page=1?

Search for everything with a particular source

we can use the wildcard * character

uris:*census* in search box

http://dev.gazetteer.in/?q=uris%3A*census*&lat=40.73698221818716&lon=-73.91704559326172&zoom=12&page=1

Search for particular feature type

feature_code:BLDG in search box

http://dev.gazetteer.in/?q=feature_code%3ABLDG%20bank&lat=40.73698221818716&lon=-73.91704559326172&zoom=12&page=1

Exclude a feature type

placename NOT feature:code:BDG

http://dev.gazetteer.in/?q=bridge%20NOT%20feature_code%3ABDG&lat=33.47727218776036&lon=-104.56787109374999&zoom=6&page=1

Date range search

start_date=YYYY end_date=YYYY

http://dev.gazetteer.in/?q=uris%3A*census*&lat=41.22721616850761&lon=-73.970947265625&zoom=9&page=1&start_date=1992&end_date=2000

in 1909, NYC

Search for churches, using name:

> http://dev.gazetteer.in/?q=church&lat=40.74035989044736&lon=-73.87630820274353&zoom=17&page=1&start_date=1909&end_date=1909

using feature code

feature_code:CH

> http://dev.gazetteer.in/?q=feature_code%3ACH&lat=40.73965673093549&lon=-73.87440919876099&zoom=16&page=1&start_date=1909&end_date=1909

multiple field search

> birmingham OR central feature_code:("LIBR" OR "CH")

searches for (Birmingham OR central) AND (feature_code:LIBR OR feature_code:CH)

Feature codes

Use the "code" value for use in an API query

for example, for the first, "code": "maps.nypl.org/warper/layers/867"

> http://dev.nypl.gazetteer.in/?q=uris:*maps.nypl.org/warper/layers/867*&lat=40.73698221818716&lon=-73.91704559326172&zoom=12&page=1


```
[
    {
        "model": "gazetteer.origin",
        "pk": 1,
        "fields": {
            "name": "Robinsons Atlas Brooklyn 1886",
            "description": "Robinson's atlas of the city of Brooklyn, New York : embracing all territory within its corporate limits; from official records ... / by and under the supervision of E. Robinson and R.H. Pidgeon, civil engineers. ",
            "example": "http://maps.nypl.org/warper/layers/867.3523",
            "code": "maps.nypl.org/warper/layers/867"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 2,
        "fields": {
            "name": "Perris Map New York, 1854",
            "description": "Maps of the city of New York / surveyed under directions of insurance companies of said city. ",
            "example": "http://maps.nypl.org/warper/layers/861.buildings.76989",
            "code": "maps.nypl.org/warper/layers/861"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 3,
        "fields": {
            "name": "Queens Atlas 1909",
            "description": "Atlas of the city of New York, borough of Queens, Long Island City, Newtown, Flushing, Jamaica, Far Rockaway, from actual surveys and official plans / by George W. and Walter S. Bromley.",
            "example": "http://maps.nypl.org/warper/layers/870.15055",
```

```
            "code": "maps.nypl.org/warper/layers/870"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 4,
        "fields": {
            "name": "NYC Buildings 2010",
            "description": "Buildings from nyc.gov bytes of the big apple",
            "example": "http://www.nyc.gov/html/dcp/html/bytes/applbyte.shtml.631695",
            "code": "applbyte"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 5,
        "fields": {
            "name": "NYC Historical Counties",
            "description": "NYC Historical Counties",
            "example": "http://publications.newberry.org/ahcbp/downloads/united_states.html.counties.nys_nca2.1",
            "code": "newberry.org/ahcbp/downloads/united_states.html.counties"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 6,
        "fields": {
            "name": "NYC Historical States",
            "description": "NYC Historical States",
            "example": "http://publications.newberry.org/ahcbp/downloads/united_states.html.states.de_state.1",
            "code": "newberry.org/ahcbp/downloads/united_states.html.states"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 7,
        "fields": {
            "name": "NYC Landmarks Districts",
            "description": "NYC Landmarks Districts",
            "example":
"https://data.cityofnewyork.us/Cultural-Affairs/Historic-district-maps/d4dh-78mx#LPChistoricDistricts/LP-1051",
            "code": "district-maps"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 8,
        "fields": {
            "name": "NYC Landmarks Points",
            "description": "NYC Landmarks Points",
            "example":
"http://www.nyc.gov/html/datamine/html/data/terms.html?dataSetJs=geo.js&theIndex=89.1011110001.4526",
            "code": "nyc.gov/html/datamine"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 10,
        "fields": {
            "name": "NRHP Historical Points",
            "description": "NRHP Historical Points",
            "example": "http://nrhp.focus.nps.gov/natreg/docs/Download.html.81000408",
            "code": "nrhp.focus.nps.gov/natreg"
        }
    },
    {
        "model": "gazetteer.origin",
```

```
        "pk": 11,
        "fields": {
            "name": "NYC Tax Blocks",
            "description": "NYC Tax Blocks",
            "example":
"https://data.cityofnewyork.us/Property/Department-of-Finance-Digital-Tax-Map/smk3-tmxj#block/48861/31072",
            "code": "cityofnewyork*block"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 12,
        "fields": {
            "name": "NYC Tax Lots",
            "description": "NYC Tax Lots",
            "example":
"https://data.cityofnewyork.us/Property/Department-of-Finance-Digital-Tax-Map/smk3-tmxj#lot/3041110053",
            "code": "cityofnewyork*lot"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 13,
        "fields": {
            "name": "NYC Townships 1990",
            "description": "NYC Townships 1990",
            "example": "http://www.census.gov/geo/www/cob/cs1990.html.36047005.1032",
            "code": "census.gov/geo/www/cob/cs1990"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 14,
        "fields": {
            "name": "NYC Townships 2000",
            "description": "NYC Townships 2000",
            "example": "http://www.census.gov/geo/www/cob/cs2000.html.60323.1031",
            "code": "census.gov/geo/www/cob/cs2000"
        }
    },
    {
        "model": "gazetteer.origin",
        "pk": 15,
        "fields": {
            "name": "NYCTownships2010",
            "description": "NYCTownships2010",
            "example": "http: //www2.census.gov/geo/tiger/GENZ2010/gz_2010_36_060_00_500k.0600000US3608160323.190",
            "code": "census.gov/geo/tiger/*gz_2010_36_060_00_500k"
        }
    }
]
```

Library of Congress Gazetteer Development RoadMap

1. Deliverables and Due Dates

- Period of Performance: 1 July 2012 to 30 June 2013
- Total Hours Sanctioned: 810 Hours (67.5 hours per month)
- Additional Hours Sanctioned: 103 Hours (in May and June 2013)

Milestones completed up to March 2013

Gazetteer Roadmap
- 2 months after award, Every 3 months, 50 Hours
- August 2012, November 2012, February 2013

UI for Adding/Updating Gazetteer Database
- 4 months after award, 120 Hours

UI for Querying Gazetteer Database
- 4 months after award, 80 Hours

Milestones completed up to April 2013

Design of Full-Production Gazetteer Database
- 9 months after award, 100 Hours

Batch Interface for Adding/Updating DB
- 9 months after award, 40 Hours
- Delivery delayed to June 2013

Implementation of Full-Production Gazetteer Database
- 9 months after award, 60 Hours

Prototype Web Services API
- 9 months after award, 40 Hours

Milestones due by July 2013

High Availability Scaleable Web Services API
- 12 months after award, 100 Hours

Maintenance and Improvements to User Interface
- 12 months after award, 80 Hours

Definition of Requirements for Confidence Measures
- 12 months after award, 20 Hours

Enhanced Contract Milestones due by July 2013

- API usage testing and analysis, 3 hours
- API performance optimization, 10 hours
- Data ingestion analysis and optimization, 15 hours
- User experience testing and analysis, 10 hours
- Optimization of user interface and experience, 15 hours
- Consultation and analysis on search result ranking, 5 hours
- Optimization of search result ranking based on feedback, 25 hours
- Consultation and analysis on presentation of historical places, 5 hours
- Optimization of historical place presentation based on feedback, 15 hours
- Analysis and resolution of remaining service tickets, 5 hours

Recurring Milestones

Monthly Gazetteer Releases
- Monthly, beginning 2 months after award, 120 Hours
- September 2012 to June 2013

2. Project Overview

Our proposed design for a fully-functional Geographic Gazetteer and Concordance service is based on the following four interactive components:

- A fully versioned spatio-temporal database and index of authority records,*
- representing geographic entities, including official name(s), identifier, feature type, locations, timeframes (i.e. the Gazetteer),
- and Linked Data references to any equivalent representations of each entity in, e.g., Wikipedia, Geonames, Yahoo! GeoPlanet, etc. (i.e. the Concordance);
- * a web services API, to*
- enable searching for authority records by name, feature type, timeframe, location, and region, and
- enable authenticated applications to extend and revise authority record entries;
- * a browsing interface, for*
- geographic and full text search and exploration of the Gazetteer,
- viewing results by name, geography, and timeframe, and
- (possibly) submitting new and revised authority record entries;
- and, lastly, an administrative interface, to provide a user interface for:*
- authoring and editing methods of the API for all classes of data in the Gazetteer,
- bulk import and maintenance of authority records,
- acceptance and rejection of submitted changes, and
- maintenance of authoring and review credentials.

The Gazetteer will be designed in such a way as to make it possible to extend future versions with additional features, including automated geographic remediation of MODS metadata records.

3. Design Principles

Our intention is that the work be made available as free and open source software (FOSS), and as services for the Library of Congress and similar institutions.

The system's data model, content, and interchange formats must also be based on freely available, publicly licensed data sources and standards.

We feel strongly that a modern digital gazetteer must implement, at a minimum, a REST-ful JSON interface to the database. XML is fine for interoperability but not actually necessary for a prototype.

In order to ensure the authority and integrity of the Gazetteer, the Library must develop the database and associated services internally, and be prepared to maintain both on a long-term and ongoing basis. We anticipate that the final implementation plan will eventually include the following tasks:

Design and implementation of a versioned, spatio-temporal database for storing the Gazetteer and Concordance. The technologies selected for this purpose will likely be based on "best of breed" free and open source GIS software, and be tailored to the specific IT infrastructure already present at the Library.

Aggregation and conflation of public domain gazetteer data sources, including existing Library authority records, within the gazetteer database, with specific attention to recording the provenance of all data sources used in a Concordance service.

Training of Library staff in the use of the Gazetteer administrative and editing tools. The training and development of these tools should proceed iteratively, such that development of the tools is informed by feedback acquired during the initial training.

Design and implementation of the following services to facilitate management of the gazetteer by the Library, and use of the gazetteer by the Library, as well as by other institutions and the general public:

User interfaces for maintaining and editing the gazetteer content for use by Library staff. The maintenance tools should implement suitable access controls to restrict write access to authorized Library staff.

User interfaces for public viewing and searching of gazetteer content, both geographically and by fuzzy name search.

User interfaces to permit users outside the Library to propose additional gazetter content for review by Library staff.

High-availability Web services, based on open library science and GIS standards, to allow applications developed both within and outside the Library to search and access all gazetteer content. The services must particularly support high-volume use from within the Library itself.

Design towards a capstone "geographic metadata augmentation service," which will accept bibliographic metadata records in standard formats (e.g. MODS-XML) and return them augmented with all relevant geographic information that can be identified from the information provided, using lessons learned from the metadata remediation pilot. Augmented records will feature confidence measures to allow users to determine the degree of precision demanded by their applications.

4. System Components & Tasks

4.1. Database

- Data model design
- Geographic entities
- Feature types
- Time frames
- Alternate and multilingual names
- Entity (i.e. internal) relationships
- Concordance (i.e. external) relationships
- User and permissions model
- Entity revision history
- Entity revision proposal queue (?)
- Geographic hierarchy (probably continent, admin0, admin1)
- Data model design documentation
- Data model implementation in PostGIS
- Initial feature type import
- External data import and feature type mapping
- Geonames
- OpenStreetMap
- (possibly) Y! GeoPlanet
- LC Authority Records
- other data sources as directed by LC
- iteration to correct generic errors
- Initial entity conflation and deduplication
- Initial entity relationship construction
- Initial geographic hierarchy import
- Initial assignment of entities to geographic hierarchy
- Research into combined bounding box/fuzzy name index optimization
- Index optimization and testing
- Bounding box index
- Time index
- Name index (fuzzy and exact searches)
- Hierarchy index
- Concordance index

4.2. Application Programming Interface (API)

- API method design and specification
- Entity search
- Name
- Bounding box
- Hierarchy
- Concordance
- Time frame, feature type, language filters
- Entity retrieval
- Entity revision history retrieval
- Entity revision
- Entity revision rollback
- Geographic hierarchy query

- Incremental conflation methods
- API service implementation
- Service methods
- Authentication and authorization (OAuth2)
- API service testing
- Automated testing
- Monitoring support
- Database export
- Full database dump
- Incremental database dump
- Application development support

4.3. Administrative Interface

- Overall user experience design
- Entity search and view
- List view
- Map view
- Entity creation and revision
- Entity geography creation and revision
- Entity relationship creation and revision
- Region editor
- Entity revision history view
- Entity revision rollback
- Entity revision proposal review (?)
- Search result correction (?)
- User management
- Bulk import interface
- Export interface (?)
- User experience testing and review
- Revision based on UX review

4.4. Browsing Interface

- Graphic design and consultation
- Overall user experience design
- Map-based search
- Bounding box
- Name
- Time frame
- Feature type
- Search result list
- Search result correction interface (admin only)
- Detailed entity view
- Entity revision history view
- Entity revision proposal form (?)
- Machine-readable export
- User experience testing and review
- Revision based on UX review

4.5. Additional Tasks

- Consultation on OpenStreetMap rendering
- TBD

5. LC Use Cases & Requirements

The following requirements and use cases have been stated by the Library in an earlier phase of consultancy between April-August 2011:

- Include ability for automated handling of spatio-temporal boundaries in the gazetteer and associated services (this would assume input sources with temporal information about changes in geographic objects). Include specific use cases (examples) as to how this would work. For example, if input XML record has historical place name, how would the coordinates and a contemporary place name be generated?

- The API will permit searching for locations by identifier, by fuzzy text match, and spatio-temporal range. The browsing interface will expose this, as well. A first query could identify the authority record identifier for a known historic place; a second could, using that identifier, obtain the contemporary name and location of the same discrete "place".

- Include Web-based curation tools to permit Library staff to establish different versions of the same geographic entity for different periods of time in the past, in essence defining a "change history" for each place that can be plotted and searched against. Provide the ability to track changes to geographic entities over time, and to track separately changes by Library staff to the records themselves.

** The database will be able to establish multiple time frames for each discrete place. Each place may have a different set of attributes for each time frame. Time frames are valid from a start date either to a specific end date, or until the next start date for a given place. Additionally, the database will retain a complete journal of all edits, including provenance and timing.

- Include references or imports from records which originate in LC-internal and external authority files and reference files. Include tools for Library staff to select and/or prioritize records from these diverse files. Examples of sets of files: LC name authority records, LC subject authority records, Geonames.org, Wikipedia, Freebase, Geoplanet, internal LC-specific tables provided in an agreed-upon standard format.

** The database will allow the linked reference to and caching of records from any desired service. At the outset, we propose to develop an import API, and write scripts that use the API to import relevant data from (and links to) any LC-internal source, and the following external sources: Geonames, Y! Geoplanet, and Wikipedia. The import scripts will serve as examples for loading data and/or links for other data sources.

- Provide the Library a way to publish enhancements to authority records through a machine-readable, standards-compliant service.

** The Gazetteer will provide consistent, permanent, and RESTful URLs for information on the latest revision of every geographic authority record maintained by the Library. These URLs may then be linked to by applications and other data sets, both inside and outside the Library. These per-entity URLs can expose authority records in a variety of standard formats, including RDF, GeoJSON, GeoRSS, and MODS. RESTful URLs will also exist to describe historic places by geography and timeframe. Initial efforts will focus on supporting GeoJSON.

- For a specific geographic record identifier, provide the official name of that geographic entity, its complete geographic containment hierarchy, its geographic location or bounding box, other identifiers that uniquely reference the entity, temporal (meta)data, perhaps even a complete spatial geometry.

** This requirement is fulfilled by the database design described above. We will most likely make use of the feature type thesaurus in the the Alexandria Digital Library Project developed by UC Santa Barbara, or that provided by GNIS.

- Provide a gazetteer lookup/search API, designed to be robust and potentially to be maintained as a reliable public service over the long term. These API functions would allow users to look up places by any combination of exact or fuzzy name search, geographic bounding box, geographic containment, time frame, and feature type.

** These facets would comprise the main query methods and/or parameters to the API.

- A Web-based browsing service to allow users to explore the complete set of authority records geographically and by text search. Users would be able to submit proposed corrections to the database through this interface for review by Library staff.

** This requirement is fulfilled by the browsing interface described above.

- Assume that the total number of records in an LC-specific gazetteer could be up to 1 million. Assume that the number of simultaneous users (batch programs and web service users) will be up to 55.

** We intend to construct the Gazetteer and services using Free and Open Source Software (FOSS) components that have been demonstrated to support this load in comparable applications.

6. LC Team Feedback, July-August 2012

6.1. Jane Mandelbaum

We're most interested in building up the local database (through manual and batch updates) with the ability to edit attributes through an admin interface. Our primary goal is to make the local gazetteer data available to our SOLR search application to create consistent geographic names for search facets and for map displays. (this is the search available at www.loc.gov).

So we're most interested in a fully populated set of locally-authoritative hierarchical attributes for a place (city, county, state, country) and the coordinates for each of those places. We may want to do most of the work in an ETL process as we bring records into SOLR.

We are also interested in being able to take historical placenames and variant placenames and directing those to the equivalent locally-authoritative hierarchical attributes for a place (city, county, state, country implemented as level1, level2, level3, etc) and the coordinates for each of those. Possibly this would be implemented as a "synonym" service.

We also need to deal with named "regions" or "areas" (e.g., "Southern States"), in terms of both building a set of hierarchical attributes and coordinates in the form of bounding boxes. These are commonly-used terms in the subjects for many of our maps.

We also need to deal with the fact that many of our maps have geographic names from LC subject headings. Because we ARE LC, we may have to give those names higher weight in the conflation.

We aren't immediately offering any warping or public UI. We're interested in populating the records with temporal data, but that isn't our highest priority. We are also considering setting up an OpenStreetMaps instance locally.

6.2. Chris Thatcher

1) Setup a local instance of the gazetteer inside LC, lx16 is a good candidate machine, powerful with lots of space and not doing much yet. We can stay an iteration behind the Topomancy development release and begin using the current API in our development ETL processes to provide more targeted feedback for API features. A 1 hour peer-to-peer phone/screen sessions where the Topomancy developers demos API features and known limitations might be worth while early to make sure we are aware of the entire API.

2) Add the Gazateer's source data ETL process to our automation processes. This will help us ensure that we stay current with the source data and help keep the Gazetteer a lively process. This would need to include the ability to export a data diff so changes managed through the admin interface can be reapplied to a freshly built gazetteer database.

3) Add a simple caching layer to optimize repeated queries for a given location. This could

be done with well known solutions like the variety apache caching strategies, or could involve more novel or nuanced approaches if Topomancy has ideas.

4) Guide us through setting up a local openstreetmaps service instance so we can generate and deliver tiles from a local server. Pointing the Gazateer OpenStreetMaps at our local service can be a proof-of-concept goal. Generally this process involves getting the latest OpenStreetMaps data export, standing up a fresh PostgresGIS DB, importing the data and running a tile generating process against it. Knowledge of how to tweak the output tile vector styles would be a plus.

5) Add a canonical terms index which provides synonymous terms. This API would allow us to walk the canonical terms building a solr synonyms dictionary so that searches using non-canonical location names can be more effectively indexed and more relevant scoring applied.

6) Add optional world region parameter to query API. This would supplement the usual political boundaries provided with large non-political, potentially overlapping boundaries such as Africa, Antarctic Regions, Central and South Asia, East Asia, Europe, Latin America and the Caribbean, Middle East and North Africa, North Africa, Oceania and the Pacific, and South East Asia. These help map interfaces like the World Digital Library provide a manageable, top-level drill down of results at the level of the entire earth. LC could provide the definition of those areas, which are likely already described in naming authority records.

6.3. Trevor Owens

Viewshare Use Cases for in-house Geo Stack

Briefly, Viewshare is a free platform for generating and customizing views (interactive maps, timelines, facets, tag clouds) to digital cultural heritage collections. Viewshare is an example of the kind of consumer we would imagine for this work at the library. This is a tool that the Library makes available to anyone working for or with a cultural heritage organization. The best way to get a sense of what it does is to watch this two minute video.
http://viewshare.org/screencast/

Right now, working through our contractor, we pay Geonames to do place name reconciliation, and we piggyback on some free (and somewhat flaky) openstreetmap tile servers. In the long run, it would be ideal to be able to have Viewshare act as a consumer of our own stack of these technologies.

These would all require some work on Viewshare itself (which I would imagine would happen independently) but they illustrate use cases for APIs.

Generate Lat/lons (and vectors in the future) from place names: Viewshare users can pick any set of fields from an uploaded set and have the application lookup lat/lons in geonames. (See step three in this documantation page). We would ultimatly like the ability to run this off the gazatter instead.

Use an LC instance of Open Layers in Viewshare: Have all of the Viewshare maps run off an in house stack of technologies. Run placename lookups for places associated with dates: Do the same kind of place name look up in Viewshare but be able to associate a date with the lookup call.

Let users upload their own baselayer maps: Enabling viewshare users to upload georectified images to use as base layers for viewshare's maps. The user has a georectified image. They upload it and LC adds it to the collection