Geocoder Documentation

Release 1.38.1

Denis Carriere

Oct 12, 2018

Contents

1 Testimonials **3**

2 API Documentation **5**
- 2.1 API Overview . . . 5
  - 2.1.1 Installation . . . 5
  - 2.1.2 Examples . . . 5
  - 2.1.3 Error Handling . . . 7
- 2.2 Access to geocoder results . . . 7
  - 2.2.1 [WIP] Multiple results . . . 7
  - 2.2.2 BBox & Bounds . . . 10
- 2.3 Refactoring guide to support multiple results . . . 11
  - 2.3.1 Changes for *OneResult* . . . 12
  - 2.3.2 Key changes for the Query manager . . . 13
  - 2.3.3 More examples . . . 14
- 2.4 QGIS Field Calculator . . . 15
  - 2.4.1 Output Field . . . 15
  - 2.4.2 Function Editor . . . 15
  - 2.4.3 Expression . . . 15

3 Providers **17**
- 3.1 ArcGIS . . . 17
  - 3.1.1 Geocoding . . . 17
- 3.2 Baidu . . . 18
  - 3.2.1 Geocoding . . . 18
  - 3.2.2 Reverse Geocoding . . . 18
- 3.3 Bing . . . 19
  - 3.3.1 Geocoding . . . 19
  - 3.3.2 Reverse Geocoding . . . 20
- 3.4 CanadaPost . . . 21
  - 3.4.1 Geocoding (Postal Code) . . . 21
- 3.5 FreeGeoIP.net . . . 22
  - 3.5.1 Geocoding (IP Address) . . . 22
- 3.6 Gaode . . . 22
  - 3.6.1 Geocoding . . . 23
- 3.7 GeocodeFarm . . . 23
  - 3.7.1 Geocoding . . . 23
  - 3.7.2 Reverse Geocoding . . . 24

3.8 Geocoder.ca . . . 24
3.8.1 Geocoding . . . 25
3.9 GeoNames . . . 25
3.9.1 Geocoding . . . 25
3.9.2 Details (inc. timezone, bbox) . . . 27
3.9.3 Children and Hierarchy . . . 28
3.10 GeoOttawa . . . 29
3.10.1 Geocoding . . . 29
3.11 Google . . . 30
3.11.1 Geocoding . . . 30
3.11.2 Reverse Geocoding . . . 30
3.11.3 Timezone . . . 30
3.11.4 Component Filtering . . . 30
3.11.5 Places . . . 30
3.11.6 Elevation . . . 31
3.12 HERE . . . 32
3.12.1 Geocoding . . . 32
3.13 IP Info.io . . . 33
3.13.1 Geocoding (IP Address) . . . 33
3.13.2 Geocode your own IP . . . 34
3.14 LocationIQ . . . 34
3.14.1 Geocoding . . . 35
3.14.2 Reverse Geocoding . . . 35
3.15 Mapbox . . . 36
3.15.1 Geocoding . . . 36
3.15.2 Reverse Geocoding . . . 36
3.16 MapQuest . . . 37
3.16.1 Geocoding . . . 37
3.16.2 Reverse Geocoding . . . 38
3.17 MaxMind . . . 39
3.17.1 Geocoding (IP Address) . . . 39
3.17.2 Geocode your own IP . . . 39
3.18 Opencage . . . 40
3.18.1 Geocoding . . . 40
3.18.2 Reverse Geocoding . . . 40
3.19 OpenStreetMap . . . 41
3.19.1 Geocoding . . . 41
3.19.2 Nominatim Server . . . 41
3.19.3 OSM Addresses . . . 41
3.20 Tamu . . . 42
3.20.1 Geocoding . . . 42
3.21 TomTom . . . 44
3.21.1 Geocoding . . . 44
3.22 What3Words . . . 45
3.22.1 Geocoding (3 Words) . . . 45
3.22.2 Reverse Geocoding . . . 45
3.23 Yahoo . . . 46
3.23.1 Geocoding . . . 46
3.24 Yandex . . . 47
3.24.1 Geocoding . . . 47
3.24.2 Reverse Geocoding . . . 47
3.25 TGOS . . . 48
3.25.1 Geocoding . . . 48

4 Contributor Guide **49**
4.1 Authors 49
4.1.1 Lead Developer 49
4.1.2 Contributors 49

Release v1.38.1. (*Installation*)

Simple and consistent geocoding library written in Python.

Many online providers such as Google & Bing have geocoding services, these providers do not include Python libraries and have different JSON responses between each other.

It can be very difficult sometimes to parse a particular geocoding provider since each one of them have their own JSON schema.

Here is a typical example of retrieving a Lat & Lng from Google using Python, things shouldn't be this hard.

```
>>> import requests
>>> url = 'https://maps.googleapis.com/maps/api/geocode/json'
>>> params = {'sensor': 'false', 'address': 'Mountain View, CA'}
>>> r = requests.get(url, params=params)
>>> results = r.json()['results']
>>> location = results[0]['geometry']['location']
>>> location['lat'], location['lng']
(37.3860517, -122.0838511)
```

Now lets use Geocoder to do the same task.

```
>>> import geocoder
>>> g = geocoder.google('Mountain View, CA')
>>> g.latlng
(37.3860517, -122.0838511)
```

CHAPTER 1

Testimonials

Tobias Siebenlist Geocoder: great geocoding library by @DenisCarriere.

mcbetz Very good companion for Geocoder. Glad to see Python getting more geo libraries for Non-GIS users.

CHAPTER 2

API Documentation

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Overview

2.1.1 Installation

To install Geocoder, simply:

```
$ pip install geocoder
```

Or on any of the supported Linux distros:

```
$ sudo snap install geocoder
```

GitHub Install

Installing the latest version from GitHub:

```
$ git clone https://github.com/DenisCarriere/geocoder
$ cd geocoder
$ python setup.py install
```

Or on any of the supported Linux distros:

```
$ sudo snap install geocoder --edge
```

2.1.2 Examples

Many properties are available once the geocoder object is created.

Forward Geocoding

```
>>> import geocoder
>>> g = geocoder.google('Mountain View, CA')
>>> g.geojson
>>> g.json
>>> g.wkt
>>> g.osm
...
```

Reverse Geocoding

```
>>> g = geocoder.google([45.15, -75.14], method='reverse')
>>> g.city
>>> g.state
>>> g.state_long
>>> g.country
>>> g.country_long
...
```

House Addresses

```
>>> g = geocoder.google("453 Booth Street, Ottawa ON")
>>> g.housenumber
>>> g.postal
>>> g.street
>>> g.street_long
...
```

IP Addresses

```
>>> import geocoder
>>> g = geocoder.ip('199.7.157.0')
>>> g = geocoder.ip('me')
>>> g.latlng
>>> g.city
```

Command Line Interface

Basic usesage with CLI

```
$ geocode "Ottawa, ON" --provider bing
```

Saving results into a file

```
$ geocode "Ottawa, ON"  >> ottawa.geojson
```

Reverse geocoding with CLI

```
$ geocode "45.15, -75.14" --provider google --method reverse
```

Using JQ to query out a specific attribute

```
$ geocode "453 Booth Street" -p canadapost --output json | jq .postal
```

Using a Session

In case you have several addresses to encode, to use persistent HTTP connection as recommended by the request-library http://docs.python-requests.org/en/master/user/advanced/#session-objects you might use the following:

```
>>> with requests.Session() as session:
>>>     berlin = geocoder.google("Ritterstr. 12, 10969 Berlin", session=session)
>>>     ottawa = geocoder.google("453 Booth Street, Ottawa ON", session=session)
```

2.1.3 Error Handling

If there is an error in the connection to the server, the exception raised by the *requests* library will be propagated up to the caller. This will be an instance of *requests.exceptions.RequestException*.

```
>>> import geocoder
>>> g = geocoder.osm("Tower Bridge, London", url="http://nonexistent.example.com")
Traceback (most recent call last):

...

requests.exceptions.ConnectionError: HTTPConnectionPool(host='nonexistent.example.com
↪', port=80): Max retries exceeded with url: /?limit=1&format=jsonv2&
↪addressdetails=1&q=foo (Caused by NewConnectionError('<requests.packages.urllib3.
↪connection.HTTPConnection object at 0x7f6b004d9390>: Failed to establish a new␣
↪connection: [Errno -2] Name or service not known',))
```

If geocoder was able to contact the server, but no result could be found for the given search terms, the *ok* attribute on the returned object will be *False*.

```
>>> import geocoder
>>> g = geocoder.osm("Mount Doom, Mordor")
>>> g.ok
False
>>> g.json
{'status': 'ERROR - No results found', 'location': 'Mount Doom, Mordor', 'provider':
↪'osm', 'status_code': 200, 'ok': False, 'encoding': 'utf-8'}
```

2.2 Access to geocoder results

2.2.1 [WIP] Multiple results

As for now, Geocoder always returns one result: the best match according to the queried provider.

```
>>> import geocoder
>>> g = geocoder.geonames('Mountain View, CA')
>>> g.latlng
['37.38605', '-122.08385']
```

(continues on next page)

(continued from previous page)

```
>>> g.country
'United States'
>>> g.json
...
```

A **Work** is **In Progress** to support multiple results (you will find which providers support this feature on the README file).

If you would like to contribute and extend your favorite provider, please refer to the *Refactoring guide*

Simply add *maxRows* in your query:

```
>>> import geocoder
>>> g = geocoder.geonames('Mountain View, CA', maxRows=5)
>>> for result in g:
...   print(result.address, result.latlng)
...
Mountain View ['37.38605', '-122.08385']
Mountain View Elementary School ['34.0271', '-117.59116']
Best Western Plus Mountainview Inn and Suites ['51.79516', '-114.62793']
Best Western Mountainview Inn ['49.3338', '-123.1446']
Mountain View Post Office ['37.393', '-122.07774']
```

Extending without breaking

The objective is to allow access to multiple results, without breaking the lib, neither adding to much complexity.

- all existing API calls shoul continue to Work
- access to all results should be easy

```
>>> import geocoder
>>> g = geocoder.geonames('Mountain View, CA', maxRows=5)

# API calls still work on best match
>>> g.latlng
['37.38605', '-122.08385']
>>> g.country
'United States'
>>> g.json
...

# the returned object support __len__, __getitem__, __iter__
>>> print(len(g))
5
>>> g[3].latlng
['49.3338', '-123.1446']
>>> for result in g:
...   print(result.address, result.latlng)
...
```

Note that the API calls are done on the best match from the provider, but you can change this behaviour by explicitely setting the default result to your desired one with the method *set_default_result*:

```
>>> g.set_default_result(3)
>>> g.latlng
```

(continues on next page)

(continued from previous page)

```
['49.3338', '-123.1446']
>>> g.address
'Best Western Plus Mountainview Inn and Suites'
```

"Breaking" change

The *geojson* property called on *g* will not apply to the best match anymore, as it would for all others providers and properties

e.g. provider not supporting multiple results:

```
>>> import geocoder
>>> g = geocoder.google('Mountain View, CA')
>>> g.geojson
{
'type':'Feature',
'properties':{
    'address':'Mountain View, CA, USA',
    ...
},
'bbox':[...],
'geometry':{...}
}
```

Instead, the *geojson* property will apply to **all** results, therefore returning a *FeatureCollection* of all *Features*:

```
>>> import geocoder
>>> g = geocoder.geonames('Mountain View, CA', maxRows=2)
>>> g.geojson
{
'type':'FeatureCollection',
'features':[
    {
        'type':'Feature',
        'properties':{
            'address':'Mountain View',
            ...
        },
        'geometry':{...}
    },
    {
        'type':'Feature',
        'properties':{
            'address':'Mountain View Elementary School',
            ...
        },
        'geometry':{...}
    }
]
}
```

More ?

The returned object *g* is a MutableSequence (python >= 3.3) because you might be interested in the actual order of the results given back by the provider, e.g. when querying the its hierarchy:

```
>>> import geocoder
>>> main = geocoder.geonames('Mountain View, CA')
>>> g = geocoder.geonames(main.geonames_id, method='hierarchy')
>>> for result in g:
...   print(result.address, result.latlng)
...
Earth ['0', '0']
North America ['46.07323', '-100.54688']
United States ['39.76', '-98.5']
California ['37.25022', '-119.75126']
Santa Clara County ['37.23249', '-121.69627']
Mountain View ['37.38605', '-122.08385']
```

2.2.2 BBox & Bounds

Overview

Some Geocoder results will contain a BBox/Bounds of the geographical extent of the result. There are two different widely adopted formats:

- ***Bounds*:** An Object defined which was first implemented by **Google Maps API** and adopted by many other providers such as Leaflet.

```
{
    northeast: [north, east],
    southwest: [south, west]
}
```

- ***BBox*:** An Array of 4 numbers which follows the **GeoJSON** BBox specification.

```
[west, south, east, north]
```

The major difference between both is the coordinates are flipped (LatLng => LngLat).

How to use

BBox or Bounds can be used in geocoding queries to limit the search to the given area. The two formats are accepted.

Let's look at a basic search for 'Paris'

```
>>> import geocoder
>>> g = geocoder.geonames('Paris', maxRows=3, key='<USERNAME>')
>>> print([(r.address, r.country, r.latlng) for r in g])
[ ('Paris', 'France', ['48.85341', '2.3488']),
  ('Paris', 'United States', ['33.66094', '-95.55551']),
  ('Paris', 'Denmark', ['56.51417', '8.48996'])]
```

Now, if you are **not** interested in any of those matches, you might have an hard time to find yours. That's where proximity comes into play.

Let's assume for the sake of this example that you are seeking 'Paris' nearby [43.2, -80.3]. You just need to define your bbox, or your bounds, and use the 'proximity' parameter...

```
>>> bounds = {
        'southwest': [43.0, -80.5],
        'northeast': [43.5, -80.0]
    }
>>> g = geocoder.geonames('Paris', proximity=bounds, key='<USERNAME>')
>>> print([g.address, g.country, g.latlng])
['Paris', 'Canada', ['43.2001', '-80.38297']]

# let's do the same with bounds
>>> bbox = [-80.5, 43.0, -80.0, 43.5]
>>> g = geocoder.geonames('Paris', proximity=bbox, key='<USERNAME>')
>>> print([g.address, g.country, g.latlng])
['Paris', 'Canada', ['43.2001', '-80.38297']]
```

Actually, you can even just use a couple of (lat, lng) and the box will be created with a tolerance of 0.5 degrees in the four directions (west, south, east, north)

```
>>> latlng = [43.0, -80.0]
>>> g = geocoder.geonames('Paris', proximity=latlng, key='<USERNAME>')
>>> print([g.address, g.country, g.latlng])
['Paris', 'Canada', ['43.2001', '-80.38297']]
```

Compliant providers

- *Google Places*
- *Geonames*
- *Mapbox*

2.3 Refactoring guide to support multiple results

It basically comes to split the existing classes into two new ones

- *OneResult* handles all the properties of one given result from the provider
- *MultipleResultsQuery* handles the query to the provider, and provides the interface to iterate through the results:

```
# from geocoder.base import Base
#
# class Baidu(Base):
#     provider = 'mapquest'
#     method = 'geocode'
#
#     def __init__(self, location, **kwargs):
#         ...
#
#     @property
#     def lat(self):
#         return self.parse['location'].get('lat')
#
# becomes

from geocoder.base import OneResult, MultipleResultsQuery
```

(continues on next page)

(continued from previous page)

```
class BaiduResult(OneResult):

    @property
    def lat(self):
        return self.raw['location'].get('lat')


class BaiduQuery(MultipleResultsQuery):
    provider = 'mapquest'
    method = 'geocode'

    ...
```

2.3.1 Changes for *OneResult*

It is important to keep in mind that *OneResult* handles all the properties for **one** given result from the provider. The modifications you will need to do here are the ones that impact **one** result only, not the overall process of the query.

- The JSON returned from the provider moved from *self.parse* to *self.raw*:

```
class MapquestResult(OneResult):

    @property
    def lat(self):
        return self.raw['latLng'].get('lat')

    @property
    def lng(self):
        return self.raw['latLng'].get('lng')

...
```

- The constructor *__init__* takes the *json_content* corresponding to this result (and only this one). You might change it before storing it in self.raw:

```
class GoogleResult(OneResult):

    def __init__(self, json_content):
        # do some transformation
        ...
        # proceed with super.__init__
        super(GoogleResult, self).__init__(json_content)
```

or, with Mapbox:

```
class MapboxResult(OneResult):

    def __init__(self, json_content):
        super(MapboxResult, self).__init__(json_content)

        for item in json_content.get('context', []):
            if '.' in item['id']:
                # attribute=country & text=Canada
                attribute = item['id'].split('.')[0]
                self.raw[attribute] = item['text']
```

- the *ok* property should be overriden on the result level when necessary (which is usually the case when you want to reverse):

```
class GoogleReverseResult(GoogleResult):

    @property
    def ok(self):
        return bool(self.address)
```

2.3.2 Key changes for the Query manager

Here, it is important to keep in mind that *MultipleResultsQuery* handle the overall process of the query, and stores **all** the results. The responsabilities here are to

1. setup the context correctly
2. make the query with appropriate headers, params
3. parse the response from the provider and create results appropriatly

Let's detail those three steps

- **(Setup) The first modification will be to provide the *metadata* needed for the query, namely:**
  - what class to use for the result
  - what url to query
  - which key to use

```
class MapquestQuery(MultipleResultsQuery):

    _URL = 'http://www.mapquestapi.com/geocoding/v1/address'
    _RESULT_CLASS = MapquestResult
    _KEY = mapquest_key
    _KEY_MANDATORY = True

Because the default implementation expects an API Key, you will need to set _KEY_
↪MANDATORY to False if no API Key is required
```

- (**Query**) In order to make the query: the initialization of the params & headers is not done neither in the constructor anymore but in the appropriated hooks. As you can see, *location* and *provider_key* are passed through:

```
def _build_headers(self, provider_key, **kwargs):
    return {
        'referer': 'http://www.mapquestapi.com/geocoding/',
        'host': 'www.mapquestapi.com',
    }

def _build_params(self, location, provider_key, **kwargs):
    return {
        'key': provider_key,
        'location': location,
        'maxResults': kwargs.get("maxRows", 1),
        'outFormat': 'json',
    }
```

- (**Query**) In some cases (e.g reversing), you need more preparation before you can build your headers / params. You would use *_before_initialize* for this, which is called before connecting to the provider. A typical use-case is where *_URL* is dynamic and needs to be extended at runtime:

```
class MapboxReverse(MapboxQuery):

    def _before_initialize(self, location, **kwargs):
        self.location = str(Location(location))
        lat, lng = Location(location).latlng
        self.url = self.url.format(lng=lng, lat=lat)
```

- (**Parsing**) The treatment of the *json_response*, which probably contains multiple results, is not done in the constructor anymore, it is done through the following hooks:

```
def _catch_errors(self, json_response):
    if b'The AppKey submitted with this request is invalid' in json_response:
        raise ValueError('MapQuest API Key invalid')

def _adapt_results(self, json_response):
    results = json_response.get('results', [])
    if results:
        return results[0]['locations']
    return []
```

- (**Parsing**) In the cases where you are interested in some fields in the *json_response*, additionnaly to the results, you might want to override *_parse_results*. In which case you should also declare the new attribute in your child class. There is one example with GooglePlaces, where the *next_page_token* interests us:

```
class PlacesQuery(MultipleResultsQuery):

    def __init__(self, location, **kwargs):
        super(PlacesQuery, self).__init__(location, **kwargs)
        self.next_page_token = None

    ...

    def _parse_results(self, json_response):
        super(PlacesQuery, self)._parse_results(json_response)

        # store page token if any
        self.next_page_token = json_response.get('next_page_token')
```

2.3.3 More examples

Please have a look on the providers already "migrated", like

- geonames
- bing
- mapbox
- mapquest

The full list is available on the README

2.4 QGIS Field Calculator

Using the QGIS Field Calculator this will output WKT format.

2.4.1 Output Field

- **Name:** wkt
- **Type:** Text, unlimited length (text)

2.4.2 Function Editor

```
import geocoder

@qgsfunction(group='Geocoder')
def geocode(location, feature, parent):
    g = geocoder.google(location)
    return g.wkt
```

2.4.3 Expression

Find the **geocode** expression in the **Geocoder** function list, the final result will look something like this:

```
geocode("address")
```

Once the wkt field is added, you can then save your document as a CSV format and in the **Layer Options** define the **GEOMETRY = AS_WKT**.

CHAPTER 3

Providers

Detailed information about each individual provider that are within Geocoder.

3.1 ArcGIS

The World Geocoding Service finds addresses and places in all supported countries from a single endpoint. The service can find point locations of addresses, business names, and so on. The output points can be visualized on a map, inserted as stops for a route, or loaded as input for a spatial analysis. an address, retrieving imagery metadata, or creating a route.

3.1.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.arcgis('Redlands, CA')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

Command Line Interface

```
$ geocode 'Redlands, CA' --provider arcgis
```

Parameters

- *location*: Your search location you want geocoded.
- *maxRows*: (default=1) Max number of results to fetch

- *limit*: *Deprecated*, same as maxRows
- *method*: (default=geocode) Use the following:
  - geocode

References

- ArcGIS Geocode API

3.2 Baidu

Baidu Maps Geocoding API is a free open the API, the default quota one million times / day.

3.2.1 Geocoding

```
>>> import geocoder # pip install geocoder
>>> g = geocoder.baidu('', key='<API KEY>')
>>> g.json
...
```

3.2.2 Reverse Geocoding

```
>>> import geocoder
>>> latlng = [45.3, -105.1]
>>> g = geocoder.baidu(latlng, method='reverse', key='<API KEY>')
>>> g.json
...
```

Command Line Interface

```
$ geocode '' --provider baidu
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export BAIDU_API_KEY=<Secret API Key>
$ export BAIDU_SECURITY_KEY=<Secret API Security Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: Baidu API key.
- *sk*: Baidu API security key. Use with key. For Baidu developers.

- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- API Reference
- Get Baidu key
- Signature algorithm

3.3 Bing

The Bing™ Maps REST Services Application Programming Interface (API) provides a Representational State Transfer (REST) interface to perform tasks such as creating a static map with pushpins, geocoding an address, retrieving imagery metadata, or creating a route. Using Geocoder you can retrieve Bing's geocoded data from Bing Maps REST Services.

3.3.1 Geocoding

```
>>> import geocoder # pip install geocoder
>>> g = geocoder.bing('Mountain View, CA', key='<API KEY>')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'. If you want to search using a structured address, use the detailed method.

```
>>> import geocoder # pip install geocoder
>>> g = geocoder.bing(None, locality='Ottawa', adminDistrict='Ontario', method=
↪'details', key='<API KEY>')
>>> g.json
...
```

This provider gives access to batch geocoding services that allow you to geocode multiple addresses at the same time. The amount of addresses you can geocode at once depends on the kind of key you have. It is described on Bing Geocode Limits.

```
>>> import geocoder
>>> g = geocoder.bing(['Mountain View, CA', 'Boulder, Co'], method='batch')
>>> for result in g:
...   print(result.latlng)
...
[37.39008, -122.08139]
[40.015831, -105.27927]
...
```

3.3.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.bing([45.15, -75.14], method='reverse')
>>> g.json
...
```

Batch reverse geocoding is also available through the *batch_reverse* method:

```
>>> import geocoder
>>> g = geocoder.bing([[40.7943, -73.970859], [48.845580, 2.321807]], method='batch_
↪reverse')
>>> for result in g:
...   print(result.address, result.city, result.postal, result.state, result.country)
...
('208 W 96th St, New York, NY 10025', 'New York', '10025', 'NY', 'United States')
('114B Rue de Vaugirard, 75006 Paris', 'Paris', '75006', 'Ile-de-France', 'France')
...
```

Command Line Interface

```
$ geocode 'Mountain View, CA' --provider bing
$ geocode '45.15, -75.14' --provider bing --method reverse
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export BING_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *addressLine*: (method=details) Official street line, uses *location* if not provided.
- *postalCode*: (method=details) The post code, postal code, or ZIP.
- *locality*: (method=details) The locality, such as the city or neighborhood.
- *adminDistrict*: (method=details) The subdivision name in the country of region for an address.
- *countryRegion*: (method=details) The ISO country code for the country.
- *key*: use your own API Key from Bing.
- *maxRows*: (default=1) Max number of results to fetch
- *method*: (default=geocode) Use the following:
  - geocode
  - details
  - reverse

- batch
- batch_reverse

References

- Bing Maps REST Services
- Bing Geocode Limits

3.4 CanadaPost

The next generation of address finders, AddressComplete uses intelligent, fast searching to improve data accuracy and relevancy. Simply start typing a business name, address or Postal Code and AddressComplete will suggest results as you go. Using Geocoder you can retrieve CanadaPost's geocoded data from Addres Complete API.

3.4.1 Geocoding (Postal Code)

```
>>> import geocoder
>>> g = geocoder.canadapost('453 Booth Street, Ottawa', key='<API KEY>')
>>> g.postal
'K1R 7K9'
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

Command Line Interface

```
$ geocode '453 Booth Street, Ottawa' --provider canadapost | jq .postal
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export CANADAPOST_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: (optional) API Key from CanadaPost Address Complete.
- *language*: (default=en) Output language preference.
- *country*: (default=ca) Geofenced query by country.
- *maxRows*: (default=1) Max number of results to fetch

- *method*: (default=geocode) Use the following:
  - geocode

References

- Addres Complete API

3.5 FreeGeoIP.net

freegeoip.net provides a public HTTP API for software developers to search the geolocation of IP addresses. It uses a database of IP addresses that are associated to cities along with other relevant information like time zone, latitude and longitude.

You're allowed up to 10,000 queries per hour by default. Once this limit is reached, all of your requests will result in HTTP 403, forbidden, until your quota is cleared.

3.5.1 Geocoding (IP Address)

```
>>> import geocoder
>>> g = geocoder.freegeoip('99.240.181.199')
>>> g.json
...
```

Command Line Interface

```
$ geocode '99.240.181.199' --provider freegeoip
```

Parameters

- *location*: Your search location you want geocoded.
- *method*: (default=geocode) Use the following:
  - geocode

References

- API Reference

3.6 Gaode

Gaode(AMap) Maps Geocoding API is a free open the API, the default quota one 2000 times / day.

This API only support china.

3.6.1 Geocoding

```
>>> import geocoder # pip install geocoder
>>> g = geocoder.gaode('A', key='<API KEY>')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'A' --provider gaode
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export GAODE_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: Gaode API key.
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- API Reference
- Get Gaode key

3.7 GeocodeFarm

Geocode.Farm is one of the few providers that provide this highly specialized service for free. We also have affordable paid plans, of course, but our free services are of the same quality and provide the same results. The major difference between our affordable paid plans and our free API service is the limitations. On one of our affordable paid plans your limit is set based on the plan you signed up for, starting at 25,000 query requests per day (API calls). On our free API offering, you are limited to 250 query requests per day (API calls).

3.7.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.geocodefarm('Mountain View, CA')
>>> g.json
...
```

This provider may return multiple results. You can access those results as described in the page '*Access to geocoder results*'.

3.7.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.geocodefarm([45.15, -75.14], method='reverse')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'Mountain View, CA' --provider geocodefarm
$ geocode '45.15, -75.14' --provider geocodefarm --method reverse
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export GEOCODEFARM_API_KEY=<Secret API Key>
```

Parameters

- *location*: The string to search for. Usually a street address. If reverse then should be a latitude/longitude.
- *key*: (optional) API Key. Only Required for Paid Users.
- *lang*: (optional) 2 digit lanuage code to return results in. Currently only "en"(English) or "de"(German) supported.
- *country*: (optional) The country to return results in. Used for biasing purposes and may not fully filter results to this specific country.
- *maxRows*: (default=1) Max number of results to fetch
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- GeocodeFarm API Documentation

3.8 Geocoder.ca

Geocoder.ca - A Canadian and US location geocoder. Using Geocoder you can retrieve Geolytica's geocoded data from Geocoder.ca.

3.8.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.geolytica('Ottawa, ON')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'Ottawa, ON' --provider geolytica
```

Parameters

- *location*: Your search location you want geocoded.
- *auth*: The authentication code for unthrottled service.
- *strictmode*: Optionally you can prevent geocoder from making guesses on your input.
- *strict*: Optional Parameter for enabling strict parsing of free form location input.
- *method*: (default=geocode) Use the following:
  - geocode
- *auth*: (optional) The authentication code for unthrottled service (premium API)

References

- API Reference

3.9 GeoNames

GeoNames is mainly using REST APIs. It offers 40 different webservices, listed with examples in its overview

Geocoder supports the following ones:

- (geocoding) retrieve GeoNames's geocoded data from a query string, and various filters
- (details) retrieve all geonames data for a given *geonames_id*
- (children) retrieve the hierarchy of a given *geonames_id*
- (hierarchy) retrieve all children for a given *geonames_id*

3.9.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.geonames('New York', key='<USERNAME>')
>>> g.address
"New York City"
>>> g.geonames_id
5128581
```

(continues on next page)

(continued from previous page)

```
>>> g.description
"city, village,..."
>>> g.population
8175133
```

Geonames web service support a lot of filters that you can use to refine your query. Here follows a list from the official documentation

- 'name', 'name_equals', 'name_startsWith',
- 'maxRows', 'startRow',
- 'country', 'countryBias', 'continentCode',
- 'adminCode1', 'adminCode2', 'adminCode3',
- 'featureClass', 'featureCode',
- 'lang', 'type', 'style', 'fuzzy',
- 'isNameRequired', 'tag', 'operator', 'charset'
- 'east', 'west', 'north', 'south'

They are all supported

```
>>> import geocoder
>>> g = geocoder.geonames('New York', key='<USERNAME>', featureClass='A')
>>> g.address
"New York"
>>> g.geonames_id
5128638
>>> g.description
"first-order administrative division"
>>> g.population
19274244
```

Multiple filters for some parameters

As pointed out in Geonames specs, the search (geocoding) service accepts multiple values for some parameters, namingly:

- country
- featureClass
- featureCode

This is also supported by Geocoder, which will expect an array instead of the normal string.

```
>>> g = geocoder.geonames('Paris', maxRows=5, country=['FR', 'US'], key='<USERNAME>')
>>> print([(r.address, r.country) for r in g])
[('Paris', 'France'), ('Paris', 'United States'), ('Paris', 'France'), ('Paris',
↪'United States'), ('Paris', 'United States')]
```

Proximity

As mentioned above, Geomanes allows the extra parameters 'east', 'west', 'north', 'south' to restrict the query to the therefore defined box.

```
>>> g = geocoder.geonames('Paris', key='<USERNAME>', east=-96.0, west=-95.0, north=33.
↪5, south=33.0)
>>> g.address
'Kosciusko'
>>> g.country
'United States'
```

For consistency purpose, geocoder also accepts a 'proximity' parameter, which can be a bbox, bounds or a dictionnary with all directions. Please refer to *this section* for more details.

3.9.2 Details (inc. timezone, bbox)

This method requires a valid *geonames_id*, which you can get with the geocode method. It will fetchs all available information from geonames, including timezone and bbox.

```
g = geocoder.geonames(6094817, method='details', key='<USERNAME>')

>>> g.lat
"45.41117"
>>> g.lng
"-75.69812"
>>> g.geonames_id
6094817
>>> g.address
"Ottawa"
>>> g.feature_class
"P"
>>> g.class_description
"city, village,..."
>>> g.code
"PPLC"
>>> g.description
"capital of a political entity"
>>> g.continent
"NA"
>>> g.country_geonames_id
"6251999"
>>> g.country_code
"CA"
>>> g.country
"Canada"
>>> g.state
"Ontario"
>>> g.state_code
"08"
>>> g.state_geonames_id
"6093943"
>>> g.admin2
""
>>> g.admin3
""
>>> g.admin4
""
>>> g.admin5
""
>>> g.population
```

(continues on next page)

(continued from previous page)

```
812129
>>> g.srtm3
71
>>> g.wikipedia
"en.wikipedia.org/wiki/Ottawa"
>>> g.timeZoneId
"America/Toronto"
>>> g.timeZoneName
"America/Toronto"
>>> g.rawOffset
-5
>>> g.dstOffset
-4
>>> g.bbox
{'northeast': [45.58753415000007, -75.07957784899992], 'southwest': [44.
↪962202955000066, -76.35400795899994]}
```

3.9.3 Children and Hierarchy

These two web services expect a *geonames_id*, which means you first need to make geocode your location. They will return multiple results most of the time, which you can access as described in the page '*Access to geocoder results*'.

```
>>> import geocoder
>>> g = geocoder.geonames('New York', key='<USERNAME>', method='children')
>>> c = geocoder.geonames(g.geoname_id, key='<USERNAME>', method='children')
>>> c.geojson
...
>>> h = geocoder.geonames(g.geoname_id, key='<USERNAME>', method='hierarchy')
>>> h.geojson
...
```

Command Line Interface

```
$ geocode 'New York City' --provider geonames
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export GEONAMES_USERNAME=<Secret Username>
```

Parameters

- *location*: Your search location you want **geocoded**.
- *geonameid*: The place you want **details / children / hierarchy** for.
- *key*: (required) geonames *username* needs to be passed with each request.
- *maxRows*: (default=1) Max number of results to fetch

- *proximity*: Search within given area (bbox, bounds, or around latlng)
- *method*: (default=geocode) Use the following:
  - geocode
  - details (mainly for administrive data and timezone)
  - timezone (alias of details)
  - children
  - hierarchy

References

- GeoNames REST Web Services

3.10 GeoOttawa

This data was collected in the field using GPS software on handheld computers. Not all information has been verified for accuracy and therefore should only be used in an advisory capacity. Forestry Services reserves the right to revise the data pursuant to further inspection/review. If you find any errors or omissions, please report them to 3-1-1.

3.10.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.ottawa('453 Booth Street')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

Command Line Interface

```
$ geocode '453 Booth Street' --provider ottawa
```

Parameters

- *location*: Your search location you want geocoded.
- *maxRows*: (default=1) Max number of results to fetch
- *method*: (default=geocode) Use the following:
  - geocode

References

- GeoOttawa Map

3.11 Google

Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which you can use to place markers or position the map. Using Geocoder you can retrieve google's geocoded data from Google Geocoding API.

3.11.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.google('Mountain View, CA')
>>> g.json
...
```

3.11.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.google([45.15, -75.14], method='reverse')
>>> g.json
...
```

3.11.3 Timezone

```
>>> import geocoder
>>> g = geocoder.google([45.15, -75.14], method='timezone')
>>> g.timeZoneName
'Eastern Daylight Time'
>>> g.timeZoneId
'America/Toronto'
>>> g.dstOffset
3600
>>> g.rawOffset
-18000
```

3.11.4 Component Filtering

```
>>> g = geocoder.google("Santa Cruz", components="country:ES")
```

Read me at Google's Geocoding API

https://developers.google.com/maps/documentation/geocoding/intro#ComponentFiltering

3.11.5 Places

```
>>> import geocoder
>>> g = geocoder.google('white house', method='places')
>>> g.latlng
[38.8976763, -77.0365298]
>>> g.address
```

(continues on next page)

(continued from previous page)

```
u'1600 Pennsylvania Ave NW, Washington, DC 20500, United States'
>>> g.json
...
```

3.11.6 Elevation

```
>>> import geocoder
>>> g = geocoder.google([45.15, -75.14], method='elevation')
>>> g.meters
71.0
>>> g.feet
232.9
>>> g.resolution
38.17580795288086
```

Command Line Interface

```
$ geocode 'Mountain View, CA' --provider google
$ geocode 'white house' --provider google --method places
$ geocode '45.15, -75.14' --provider google --method reverse
$ geocode '45.15, -75.14' --provider google --method timezone
$ geocode '45.15, -75.14' --provider google --method elevation
```

Environment Variables

To make sure your API key is stored safely on your computer, you can define that API key using your system's environment variables.

```
$ export GOOGLE_API_KEY=<Secret API Key>
$ export GOOGLE_CLIENT=<Secret Client>
$ export GOOGLE_CLIENT_SECRET=<Secret Client Secret>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: Your Google developers free key.
- *language*: 2-letter code of preferred language of returned address elements.
- *client*: Google for Work client ID. Use with client_secret. Cannot use with key parameter
- *client_secret*: Google for Work client secret. Use with client.
- *proximity*: Search within given area (bbox, bounds, or around latlng)
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse
  - timezone

- elevation
- places

References

- Google Geocoding API
- Google Geocoding Best Practices
- Google Places API

3.12 HERE

Send a request to the geocode endpoint to find an address using a combination of country, state, county, city, postal code, district, street and house number. Using Geocoder you can retrieve geocoded data from the HERE Geocoder REST API.

3.12.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.here('Espoo, Finland')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

A bounding box can be supplied as an array of the form [minX, minY, maxX, maxY] to restrict results.

```
>>> import geocoder
>>> bbox = [-118.604794, 34.172684, -118.500938, 34.236144]
>>> g = geocoder.here("Winnetka", bbox=bbox)
>>> g.address
"Winnetka, CA, United States"
>>> g = geocoder.here("Winnetka")
>>> g.address
"Winnetka, IL, United States"
...
```

Please refer to *this section* for more details. Reverse Geocoding ~~~~~~~~~~~~~~~~~~

```
>>> import geocoder
>>> g = geocoder.google([45.15, -75.14], method='reverse')
>>> g.json
...
```

Using API Key

If you want to use your own *app_id* & *app_code*, you must register an app at the HERE Developer.

```
>>> g = geocoder.here('Espoo, Finland',
                      app_id='<YOUR APP ID>',
                      app_code='<YOUR APP CODE>')
```

Command Line Interface

```
$ geocode 'Espoo, Finland' --provider here
$ geocode '45.15, -75.14' --provider here --method reverse
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export HERE_APP_ID=<Secret APP ID>
$ export HERE_APP_CODE=<Secret APP Code>
```

Parameters

- *location*: Your search location you want geocoded.
- *app_code*: (optional) use your own Application Code from HERE.
- *app_id*: (optional) use your own Application ID from HERE.
- *bbox*: Search within a bounding box [minX, minY, maxX, maxY]. Pass as an array.
- *maxRows*: (default=1) Max number of results to fetch
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- HERE Geocoder REST API

3.13 IP Info.io

Use the IPInfo.io IP lookup API to quickly and simply integrate IP geolocation into your script or website. Save yourself the hassle of setting up local GeoIP libraries and having to remember to regularly update the data.

3.13.1 Geocoding (IP Address)

```
>>> import geocoder
>>> g = geocoder.ipinfo('199.7.157.0')
>>> g.latlng
[45.413140, -75.656703]
>>> g.city
'Toronto'
>>> g.json
...
```

3.13.2 Geocode your own IP

To retrieve your own IP address, simply have *''* or *'me'* as the input.

```
>>> import geocoder
>>> g = geocoder.ipinfo('me')
>>> g.latlng
[45.413140, -75.656703]
>>> g.ip
'199.7.157.0'
>>> g.json
...
```

Command Line Interface

```
$ geocode '199.7.157.0' --provider ipinfo | jq .
```

Parameters

- *location*: Your search location you want geocoded.
- *location*: (optional) *''* will return your current IP address's location.
- *method*: (default=geocode) Use the following:
  - geocode

References

- IpinfoIo

3.14 LocationIQ

LocationIQ provides geocoding based on OpenStreetMap's Nominatim. It is fully compatible with OSM except for the requirement of an API Key. Please refer to the *OpenStreetMap* docs for more details. You can signup for a free API Key *here <http://locationiq.org/#register>*.

3.14.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.locationiq('New York city', key='...')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

3.14.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.locationiq([45.15, -75.14], key='...', method='reverse')
>>> g.json
...
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export LOCATIONIQ_API_KEY=<Secret API Key>
```

Command Line Interface

```
$ geocode 'New York city' --provider locationiq --key <key> --output geojson | jq .
$ geocode 'New York City' --provider locationiq --key <key> --output osm
```

Parameters

- *location*: Your search location you want geocoded.
- *url*: Custom OSM Server (ex: localhost)
- *maxRows*: (default=1) Max number of results to fetch
- *limit*: *Deprecated*, same as maxRows
- *method*: (default=geocode) Use the following:
  - geocode

References

- LocationIQ
- Nominatim

3.15 Mapbox

The Mapbox Geocoding API lets you convert location text into geographic coordinates (1600 Pennsylvania Ave NW → -77.0366,38.8971).

3.15.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.mapbox('San Francisco, CA', key='<TOKEN>')
>>> g.json
...
```

This provider may return multiple results. You can access those results as described in the page '*Access to geocoder results*'.

Request feature data that best matches input and is biased to the given {latitude} and {longitude} coordinates. In the above example, a query of "200 Queen Street" returns a subset of all relevant addresses in the world. By adding the proximity option, this subset can be biased towards a given area, returning a more relevant set of results. In addition, a bounding box can be supplied to restrict results.

```
>>> import geocoder
>>> latlng = [45.3, -66.1]
>>> g = geocoder.mapbox("200 Queen Street", proximity=latlng)
>>> g.address
"200 Queen St W, Saint John, New Brunswick E2M 2C8, Canada"
>>> g = geocoder.mapbox("200 Queen Street")
>>> g.address
"200 Queen Street, Colac, Victoria 3250, Australia"
>>> bbox = [-118.604794, 34.172684, -118.500938, 34.236144]
>>> g = geocoder.mapbox("Winnetka", bbox=bbox)
>>> g.address
"Winnetka, Winnetka, California 91306, United States"
>>> g = geocoder.mapbox("Winnetka")
>>> g.address
"Winnetka Heights, Dallas, Texas 75211, United States"
...
```

Please refer to *this section* for more details.

3.15.2 Reverse Geocoding

```
>>> import geocoder
>>> latlng = [45.3, -105.1]
>>> g = geocoder.mapbox(latlng, method='reverse')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'San Francisco, CA' --provider mapbox --out geojson
$ geocode '45.15, -75.14' --provider mapbox --method reverse
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export MAPBOX_ACCESS_TOKEN=<Secret Access Token>
```

Parameters

- *location*: Your search location you want geocoded.
- *proximity*: Search nearby [lat, lng].
- *bbox*: Search within a bounding box [minX, minY, maxX, maxY]. Pass as an array.
- *key*: Use your own access token from Mapbox.
- *country*: Filtering by country code {cc} ISO 3166 alpha 2.
- *proximity*: Search within given area (bbox, bounds, or around latlng)
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- Mapbox Geocoding API
- Get Mapbox Access Token

3.16 MapQuest

The geocoding service enables you to take an address and get the associated latitude and longitude. You can also use any latitude and longitude pair and get the associated address. Three types of geocoding are offered: address, reverse, and batch. Using Geocoder you can retrieve MapQuest's geocoded data from Geocoding Service.

3.16.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.mapquest('San Francisco, CA', key='<API KEY>')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

A bounding box can be supplied as an array of the form [minX, minY, maxX, maxY] to bump results within a the bounding box to the top.

```
>>> import geocoder
>>> bbox = [-118.604794, 34.172684, -118.500938, 34.236144]
>>> g = geocoder.here("Winnetka", bbox=bbox)
>>> g.lng, g.lat
(-118.571098, 34.213299)
>>> g = geocoder.here("Winnetka")
>>> g.lng, g.lat
(-87.734719, 42.107106)
...
```

This provider gives access to batch geocoding services that allow you to geocode up to 100 addresses at the same time.

```
>>> import geocoder
>>> g = geocoder.mapquest(['Mountain View, CA', 'Boulder, Co'], method='batch')
>>> for result in g:
...   print(result.address, result.latlng)
...
('Mountain View', [37.39008, -122.08139])
('Boulder', [40.015831, -105.27927])
...
```

3.16.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.mapquest([45.15, -75.14], method='reverse', key='<API KEY>')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'San Francisco, CA' --provider mapquest --out geojson
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export MAPQUEST_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *maxRows*: (default=1) Max number of results to fetch
- *bbox*: Search within a bounding box [minX, minY, maxX, maxY]. Pass as an array.
- *method*: (default=geocode) Use the following:
  - geocode
  - batch

References

- Mapquest Geocoding Service
- Get Free API Key

3.17 MaxMind

MaxMind's GeoIP2 products enable you to identify the location, organization, connection speed, and user type of your Internet visitors. The GeoIP2 databases are among the most popular and accurate IP geolocation databases available. Using Geocoder you can retrieve Maxmind's geocoded data from MaxMind's GeoIP2.

3.17.1 Geocoding (IP Address)

```
>>> import geocoder
>>> g = geocoder.maxmind('199.7.157.0')
>>> g.latlng
[45.413140, -75.656703]
>>> g.city
'Toronto'
>>> g.json
...
```

3.17.2 Geocode your own IP

To retrieve your own IP address, simply have *''* or *'me'* as the input.

```
>>> import geocoder
>>> g = geocoder.maxmind('me')
>>> g.latlng
[45.413140, -75.656703]
>>> g.ip
'199.7.157.0'
>>> g.json
...
```

Command Line Interface

```
$ geocode '8.8.8.8' --provider maxmind | jq .
```

Parameters

- *location*: Your search IP Address you want geocoded.
- *location*: (optional) *'me'* will return your current IP address's location.
- *method*: (default=geocode) Use the following:
  - geocode

References

- MaxMind's GeoIP2

3.18 Opencage

OpenCage Geocoder simple, easy, and open geocoding for the entire world Our API combines multiple geocoding systems in the background. Each is optimized for different parts of the world and types of requests.We aggregate the best results from open data sources and algorithms so you don't have to. Each is optimized for different parts of the world and types of requests. Using Geocoder you can retrieve OpenCage's geocoded data from OpenCage Geocoding Services.

3.18.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.opencage('San Francisco, CA', key='<API Key>')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

3.18.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.opencage([45.15, -75.14], method='reverse')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'San Francisco, CA' --provider opencage --out geojson --key '<API Key>' |␣
↪jq .
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export OPENCAGE_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: (optional) use your own API Key from OpenCage.
- *maxRows*: (default=1) Max number of results to fetch

- *method*: (default=geocode) Use the following:
  - geocode

References

- OpenCage Geocoding Services

3.19 OpenStreetMap

Nominatim (from the Latin, 'by name') is a tool to search OSM data by name and address and to generate synthetic addresses of OSM points (reverse geocoding). Using Geocoder you can retrieve OSM's geocoded data from Nominatim.

3.19.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.osm('New York city')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

3.19.2 Nominatim Server

Setting up your own offline Nominatim server is possible, using Ubuntu 14.04 as your OS and following the Nominatim Install instructions. This enables you to request as much geocoding as your little heart desires!

```
>>> url = 'http://localhost/nominatim/'
>>> url = 'localhost'
>>> g = geocoder.osm("New York City", url=url)
>>> g.json
...
```

3.19.3 OSM Addresses

The addr tag is the prefix for several *addr:** keys to describe addresses.

This format is meant to be saved as a CSV and imported into JOSM.

```
>>> g = geocoder.osm('11 Wall Street, New York')
>>> g.osm
{
    "x": -74.010865,
    "y": 40.7071407,
    "addr:country": "United States of America",
    "addr:state": "New York",
    "addr:housenumber": "11",
    "addr:postal": "10005",
```

(continues on next page)

(continued from previous page)

```
    "addr:city": "NYC",
    "addr:street": "Wall Street"
}
```

Command Line Interface

```
$ geocode 'New York city' --provider osm --out geojson | jq .
$ geocode 'New York city' -p osm -o osm
$ geocode 'New York city' -p osm --url localhost
```

Parameters

- *location*: Your search location you want geocoded.
- *url*: Custom OSM Server (ex: localhost)
- *maxRows*: (default=1) Max number of results to fetch
- *limit*: *Deprecated*, same as maxRows
- *method*: (default=geocode) Use the following:
  - geocode

References

- Nominatim
- Nominatim Install
- addr tag

3.20 Tamu

The Texas A&M Geoservices Geocoding API provides output including Lat and Lon and numerous census data values.

An API key linked to an account with Texas A&M is required.

Tamu's API differs from the other geocoders in this package in that it requires the street address, city, state, and US zipcode to be passed in separately, rather than as a single string. Because of this requirement, the "location", "city", "state", and "zipcode" parameters are all required when using the Tamu provider. The "location" parameter should contain only the street address of the location.

3.20.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.tamu(
            '595 Market St',
            city='San Francisco',
            state='California',
            zipcode='94105',
```

(continues on next page)

(continued from previous page)

```
            key='demo')
>>> g.json
...
```

Command Line Interface

```
$ geocode '595 Market St' --provider tamu --city San Francisco --state CA --zipcode␣
↪94105 --key <Secret API Key>
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export TAMU_API_KEY=<Secret API Key>
```

Parameters

- *location*: The street address of the location you want geocoded.
- *city*: The city of the location to geocode.
- *state*: The state of the location to geocode.
- *zipcode*: The zipcode of the location to geocode.
- *key*: use your own API Key from Tamu.
- *method*: (default=geocode) Use the following:
  - geocode

Census Output Fields

Note: "FIPS" stands for "Federal Information Processing System"

- *census_block*: Census Block value for location
- *census_tract*: Census Tract value for location
- *census_county_fips*: Census County FIPS value
- *census_cbsa_fips*: Census Core Base Statistical Area FIPS value
- *census_mcd_fips*: Census Minor Civil Division FIPS value
- *census_msa_fips*: Census Metropolitan Statistical Area FIPS value
- *census_place_fips*: Census Place FIPS value
- *census_state_fips*: Census State FIPS value
- *census_year*: Census Year from which these values originated

References

- Tamu Geocoding API

3.21 TomTom

The Geocoding API gives developers access to TomTom's first class geocoding service. Developers may call this service through either a single or batch geocoding request. This service supports global coverage, with house number level matching in over 50 countries, and address point matching where available. Using Geocoder you can retrieve TomTom's geocoded data from Geocoding API.

3.21.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.tomtom('San Francisco, CA', key='<API KEY>')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

Command Line Interface

```
$ geocode 'San Francisco, CA' --provider tomtom --out geojson
```

Environment Variables

To make sure your API key is store safely on your computer, you can define that API key using your system's environment variables.

```
$ export TOMTOM_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: Use your own API Key from TomTom.
- *maxRows*: (default=1) Max number of results to fetch
- *countrySet*: Comma separated string of country codes (e.g. FR,ES). This will limit the search to the specified countries
- *lon*: Longitude e.g. lon=-121.89 lat./lon. where results should be biased (supplying a lat./lon. without a radius will only bias the search results to that area)
- *lat*: Latitude e.g.lat=37.337 lat./lon. where results should be biased (supplying a lat./lon. without a radius will only bias the search results to that area)
- *radius*: If radius and position are set, the results will be constrained to the defined area. The radius parameter is specified in meters.

- *method*: (default=geocode) Use the following:
  - geocode

References

- TomTom Geocoding API

3.22 What3Words

What3Words is a global grid of 57 trillion 3mx3m squares. Each square has a unique 3 word address that people can find and communicate quickly, easily, and without ambiguity.

Addressing the world

Everyone and everywhere now has an address

3.22.1 Geocoding (3 Words)

```
>>> import geocoder
>>> g = geocoder.w3w('embedded.fizzled.trial')
>>> g.json
...
```

3.22.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.w3w([45.15, -75.14], method='reverse')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'embedded.fizzled.trial' --provider w3w
$ geocode '45.15, -75.14' --provider w3w --method reverse
```

Environment Variables

For safe storage of your API key on your computer, you can define that API key using your system's environment variables.

```
$ export W3W_API_KEY=<Secret API Key>
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: use your own API Key from What3Words.
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- API Reference
- Get W3W key

3.23 Yahoo

Yahoo PlaceFinder is a geocoding Web service that helps developers make their applications location-aware by converting street addresses or place names into geographic coordinates (and vice versa). Using Geocoder you can retrieve Yahoo's geocoded data from Yahoo BOSS Geo Services.

3.23.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.yahoo('San Francisco, CA')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'San Francisco, CA' --provider yahoo --out geojson
```

Parameters

- *location*: Your search location you want geocoded.
- *method*: (default=geocode) Use the following:
  - geocode

References

- Yahoo BOSS Geo Services

3.24 Yandex

Yandex (Russian:) is a Russian Internet company which operates the largest search engine in Russia with about 60% market share in that country.

The Yandex home page has been rated as the most popular website in Russia.

3.24.1 Geocoding

```
>>> import geocoder
>>> g = geocoder.yandex('Moscow Russia')
>>> g.json
...
```

This provider may return multiple results by setting the parameter *maxRows* to the desired number (1 by default). You can access those results as described in the page '*Access to geocoder results*'.

3.24.2 Reverse Geocoding

```
>>> import geocoder
>>> g = geocoder.yandex([55.95, 37.96], method='reverse')
>>> g.json
...
```

Command Line Interface

```
$ geocode 'Moscow Russia' --provider yandex --out geojson
$ geocode '55.95, 37.96' --provider yandex --method reverse
```

Parameters

- *location*: Your search location you want geocoded.
- *maxRows*: (default=1) Max number of results to fetch
- *lang*: Chose the following language:
  - **ru-RU** — Russian (by default)
  - **uk-UA** — Ukrainian
  - **be-BY** — Belarusian
  - **en-US** — American English
  - **en-BR** — British English
  - **tr-TR** — Turkish (only for maps of Turkey)
- *kind*: Type of toponym (only for reverse geocoding):
  - **house** — house or building
  - **street** — street
  - **metro** — subway station

  - **district** — city district
  - **locality** — locality (city, town, village, etc.)
- *method*: (default=geocode) Use the following:
  - geocode
  - reverse

References

- Yandex API Reference

3.25 TGOS

TGOS Map is official map service of Taiwan.

3.25.1 Geocoding

```
>>> import geocoder # pip install geocoder
>>> g = geocoder.tgos('735', key='<API KEY>')
>>> g.json
...
```

Command Line Interface

```
$ geocode '735' --provider tgos
```

Parameters

- *location*: Your search location you want geocoded.
- *key*: Use your own API Key from TGOS.
- *method*: (default=geocode) Use the following:
  - geocode
- *language*: (default=taiwan) Use the following:
  - taiwan
  - english
  - chinese

References

- TGOS Maps API

CHAPTER 4

Contributor Guide

If you want to contribute to the project, this part of the documentation is for you.

4.1 Authors

4.1.1 Lead Developer

- Denis Carriere - Creator of Python's Geocoder

4.1.2 Contributors

A big thanks to all the people that help contribute:

- Virus Warnning - Implemented TGOS provider
- Kevin Brolly - Implemented GeocodeFarm provider
- Michael R. Okun - Implemented Tamu provider
- Palo Dravecky - Added Google for Work.
- Dunice Vadimh - Added IPInfo provider.
- Yed Podtrzitko - Cleaned up code & Added Six
- Thomas Gratier - Wrote an article about Geocoder vs. Geopy
- Max Arnold - Submitted Github Issue
- Thanh Ha - Cleaned up code & Unit Testing
- Mahdi Yusuf - Promoted by Pycoders Weekly, Issue #155 Nimoy
- Alex Pilon - Cleaned up code
- Philip Hubertus - Provided HERE improvements & documentation

- Antonio Lima - Improved code quality and introduced Rate Limits
- Alexander Lukanin - Improved Python 3 compatibilty
- flebel - Submitted Github Issues
- patrickyan - Submitted Github Issues
- esy - Submitted Github Issues
- Sergei Grabalin ( ) - Fixed Python2 Unicode Issues
- Matthieu Rigal - Added session support