pygeocodio Documentation

Release 0.5.0

Ben Lopatin

February 23, 2017

Contents

1 Features **3**
- 1.1 Installation . . . 3
- 1.2 Geocoding . . . 3
- 1.3 Address component parsing . . . 5
- 1.4 Reverse geocoding . . . 5
- 1.5 Data types . . . 6
- 1.6 Exceptions . . . 7

Python wrapper for Geocod.io geocoding API.

CHAPTER 1

Features

- Geocode an individual address
- Batch geocode up to 10,000 addresses at a time
- Parse an address into its identifiable components

Read the complete Geocod.io documentation for service documentation.

Installation

pygeocodio requires *requests* 1.0.0 or greater and will ensure requests is installed:

```
pip install pygeocodio
```

Geocoding

Single address geocoding

Import the API client and ensure you have a valid API key:

```
>>> from geocodio import GeocodioClient
>>> client = GeocodioClient(MY_KEY)
>>> geocoded_location = client.geocode("42370 Bob Hope Drive, Rancho Mirage CA")
```

The result from the Geocod.io service is a dictionary including your query, its components, and a list of matching results:

```
>>> geocoded_location
{
  "input": {
    "address_components": {
      "number": "42370",
      "street": "Bob Hope",
      "suffix": "Dr",
      "city": "Rancho Mirage",
      "state": "CA"
    },
    "formatted_address": "42370 Bob Hope Dr, Rancho Mirage CA"
  },
```

```
  "results": [
    {
      "address_components": {
        "number": "42370",
        "street": "Bob Hope",
        "suffix": "Dr",
        "city": "Rancho Mirage",
        "state": "CA",
        "zip": "92270"
      },
      "formatted_address": "42370 Bob Hope Dr, Rancho Mirage CA, 92270",
      "location": {
        "lat": 33.738987255507,
        "lng": -116.40833849559
      },
      "accuracy": 1
    },
    {
      "address_components": {
        "number": "42370",
        "street": "Bob Hope",
        "suffix": "Dr",
        "city": "Rancho Mirage",
        "state": "CA",
        "zip": "92270"
      },
      "formatted_address": "42370 Bob Hope Dr, Rancho Mirage CA, 92270",
      "location": {
        "lat": 33.738980796909,
        "lng": -116.40833917329
      },
      "accuracy": 0.8
    }
  ]
}
```


This returned several geolocation results in descending order of accuracy, so the first and most accurate geocoded location latitude and longitude can be accessed using the *coords* attribute:

```
>>> geocoded_location.coords
(-116.40833849559, 33.738987255507)
```

Note: To make working with other geographic data formats easier the *coords* method returns a tuple in (longitude, latitude) format.

Batch geocoding

You can also geocode a list of addresses:

```
>>> geocoded_addresses = geocodio.geocode(['1600 Pennsylvania Ave, Washington, DC',
        '3101 Patterson Ave, Richmond, VA, 23221'])
```

Return just the coordinates for the list of geocoded addresses:

```
>>> geocoded_addresses.coords
[(-116.40833849559, 33.738987255507), (-116.40833849559, 33.738987255507)]
```

Lookup an address by formatted address:

```
>>> geocoded_addresses.get('1600 Pennsylvania Ave, Washington, DC').coords
(-116.40833849559, 33.738987255507)
```

Note that to perform the key based lookup you must use the *get* method. This preserves the list's index based lookup.

Note: If one address cannot be parsed or geocoded the Geocod.io service will still respond, but the *response* value for that address will be an error message. E.g. if a query was an emptry string, the value for that particular query would look like this:

```
{
    "query": "",
    "response": {
        "error": "Could not parse address"
    }
}
```

In this case the a lookup for *""* would yield *None*. The *None* value is not removed from the list in the *LocationCollection* because then the indices in the response addresses would no longer match the indices in the request addresses.

Address component parsing

And if you just want to parse an individual address into its components:

```
>>> client.parse('1600 Pennsylvania Ave, Washington DC')
{
    "address_components": {
        "number": "1600",
        "street": "Pennsylvania",
        "suffix": "Ave",
        "city": "Washington",
        "state": "DC"
    },
    "formatted_address": "1600 Pennsylvania Ave, Washington DC"
}
```

The return value is simple enough to us as the returned dictionary.

Reverse geocoding

Reverse geocoding takes a point and returns a matching address.

Single point reverse-geocoding

Reverse geocoding:

```
>>> client.reverse((33.738987, -116.4083))
```

Batch reverse-geocoding

You can also reverse geocode in batch:

```
>>> client.reverse([(33.738987, -116.4083), (34.288, -112.12)])
```

As with geocoding, there is a limit of 10,000 points you can geocode at a time.

Data types

The client returns values represented by lightly-extended Python dictionaries and lists. They have been extended to provide easier access to frequently requested or primary data elements, and to make accessing data simple.

By adding only a few methods and data lookup elements the source data is largely left as-is for developers to use as they see fit.

For example, if the data for a geocoded address returned by Geocodio includes *accuracy_type*, then you access that by referencing the key, *'accuracy_type'*.:

```
>>> geocoded_location = client.geocode("42370 Bob Hope Drive, Rancho Mirage CA")
>>> geocoded_location.accuracy
1
>>> geocoded_location.accuracy_type
Traceback (most recent call last)
    File "<stdin>", line 1, in <module>
AttributeError: 'Location' object has no attribute 'accuracy_type'.
>>> geocoded_location['accuracy_type']
"rooftop"
```

Address

An *Address* object is just a dictionary object that provides two access methods for returning the accuracy value of the geocoded address as an attribute and the coordinates of the address as an attribute.

`Address.__init__`(*results_list*, *order='lat'*)
results_list is the raw data

order allows you to change the default order of the coordinate points. Setting order to 'lng' or any value other than 'lat' will return the points in *(longitude, latitude)* order.

`Address.coords`()
A property method that returns the coordinates of the address or *None* if they are not available (in the case of a parsed address).

`Address.accuracy`()
A property method that returns the accuracy rating of the geocoded address.

When parsing an address, the result is returned as an *Address* for consistency, but the result's usefullness will be limited to the dictionary structure.

Location

A *Location* object is a dictionary object that provides the same access methods as an *Address* object. Because a geocoded Location may have more than one address returned, the methods refer to the coordinates and accuracy respectively of the most most accurate geocoded address.

Location.**__init__**(*results_dict*, *order='lat'*)
results_dict is the raw data

order allows you to change the default order of the coordinate points. Setting order to 'lng' or any value other than 'lat' will return the points in *(longitude, latitude)* order.

Location.**coords**()
A property method that returns the coordinates of the best matched address or *None* if they are not available.

Location.**accuracy**()
A property method that returns the accuracy rating of the best matched address.

LocationCollection

A *LocationCollection* object is a list of *Location* objects. It maintains an internal dictionary of each geocoding or reverse geocoding query with reference to the list index of the result. This allows the order of the list to be preserved but for simple lookup of the values without needing to iterate over the entire list.

Note: This demands an OrderedDict! Unfortunately OrderedDict was only introduced in Python 2.7, and Python 2.6 is a targeted supported version of Python for this project. So, boo.

LocationCollection.**get**(*key*)
A method that returns a *Location* object from the list of locations by checking against the query input.

The key can be the queried address as a string (geocoding) or the queried point (reverse geocoding). The point can be provided as a tuple of floats:

```
(33.12, -78.123)
```

a tuple of float-coerceable values (e.g. a float as a string):

```
("33.12", "-78.123")
```

or a string representing the query:

```
"33.12, -78.123"
```

This method is provided instead of overriding the *__getitem__* method as the latter allows index based access to the list.

LocationCollection.**coords**()
A property method that returns a list of all of the coordinates

Exceptions

The Geocod.io service may respond with errors, so *pygeocodio* provides some names exceptions that can be used to identify specific error types. The Geocod.io documentation lists the following expected responses:

```
200 OK Hopefully you will see this most of the time. Note that this status code will also be returned
403 Forbidden Invalid API key or other reason why access is forbidden
422 Unprocessable Entity A client error prevented the request from executing succesfully (e.g. invali
500 Server Error Hopefully you will never see this...it means that something went wrong in our end. W
```

To handle these:

- An HTTP 403 error raises a *GeocodioAuthError*

- An HTTP 422 error raises a *GeocodioDataError* and the error message will be reported through the exception
- An HTTP 5xx error raises a *GeocodioServerError*
- An unmatched non-200 response will simply raise *GeocodioError*

Symbols

__init__() (geocodio.data.Address method), 6
__init__() (geocodio.data.Location method), 6

A

accuracy() (geocodio.data.Address method), 6
accuracy() (geocodio.data.Location method), 7
Address, 6

C

coords() (geocodio.data.Address method), 6
coords() (geocodio.data.Location method), 7
coords() (geocodio.data.LocationCollection method), 7

D

data, 6

G

get() (geocodio.data.LocationCollection method), 7

L

Location, 6
LocationCollection, 6

T

types, 6