ggeocoder Documentation

Release 1.0.0

Aaron Madison

January 06, 2013

CONTENTS

Contents:

CHAPTER

ONE

OVERVIEW

1.1 Installing ggeocoder

install ggeocoder from pypi, you can check by doing:

```
pip install ggeocoder
```

It is a good idea to familiarize yourself with the Google Maps v3 API before you get started.

Next, we'll explore how to use the Geocoder API.

CHAPTER

TWO

USING THE GEOCODER API

2.1 Geocoding

To geocode an address, simply create a geocoder instance, and request a geocode.

```
>>> g = Geocoder()
>>> results = g.geocode('1600 Amphitheatre Pkwy')
```

If you want to provide any additional parameters that the Maps API accepts, send them after the address.

```
>>> g.geocode('Toledo', region='es', language='es', sensor='true')
```

See working_with_results for what to do next.

2.2 Reverse Geocoding

To do a reverse geocode, simply use the *reverse_geocode* method and provide lat/lng values

```
>>> g = Geocoder()
>>> results = g.reverse_geocode(37.4220827, -122.08289)
```

See working_with_results for what to do next.

2.3 Working with Results

If the geocode is successful, you will have an object that is has a list of geocoded results. You can check the number of results returned.

```
>>> len(results)
1
```

You can access each result by referencing its index in your result set.

```
>>> first_result = results[0]
```

Once you have geocoded an address, it is very easy to work with. There are a couple convenience methods for the GeoResult object.

The string representation will always be the formatted address. You can also access formatted address directly, coordinates, and a boolean check *is_valid_address* to see if the address is a valid street address. Finally, there is a raw attribute which gives you access to the *raw* data returned by the Google Maps API.

```
>>> str(first_result)
'1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA'

>>> first_result.formatted_address
u'1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA'

>>> first_result.coordinates
(37.4220827, -122.08289)

>>> first_result.is_valid_address
True
```

The 'address_components' that the Maps API provides are made easily accessible by simple attribute lookups.

```
>>> first_result.street_number
u'1600'
>>> first_result.postal_code
u'94043'
>>> first_result.country
u'United States'
```

By default, the GeoResult will give you the 'long_name' of the address component. If you want the short name, just add it to your attribute lookup.

```
>>> first_result.country__short_name
u'US'
```

CHAPTER

THREE

CUSTOMIZING THE RESULT CLASS RETURNED

3.1 Customizing Result Class

Sometimes you might want to interact with the results using your own custom class. One particularly handy use case is giving the 'administrative_area...' items a more friendly name for your context.

You can do this by subclassing the GeoResult class and providing an *attr_mapping* field.

We can create easy accessors to these pieces of data by doing something like the following:

```
from ggeocoder import Geocoder, GeoResult

class USGeoResult(GeoResult):
    attr_mapping = {
        'city': 'locality',
        'state': 'administrative_area_level_1',
        'county': 'administrative_area_level_2',
        'township': 'administrative_area_level_3',
        'ZIP': 'postal_code',
    }

>>> g = Geocoder()
>>> best_result = g.geocode('1600 Amphitheatre Pkwy', result_class=USGeoResult)[0]
>>> best_result.state
u'California'
>>> best_result.state__short_name
u'CA'
>>> best_result.county
u'Santa Clara'
>>> best_result.city
u'Mountain View'
>>> best_result.ZIP
u'94043'
```

CHAPTER

FOUR

MODULES PROVIDED

Python library for using Google Geocoding API V3.

class ggeocoder.**Geocoder**(*client_id=None*, *private_key=None*)
Interface for interacting with Google's Geocoding V3's API. http://code.google.com/apis/maps/documentation/geocoding/

If you have a Google Maps Premier account, you can supply your client_id and private_key and the Geocoder will make the request with a properly signed url

geocode(*address*, ***params*)

Params may be any valid parameter accepted by Google's API.
http://code.google.com/apis/maps/documentation/geocoding/#GeocodingRequests

reverse_geocode(*lat*, *lng*, ***params*)

Params may be any valid parameter accepted by Google's API.
http://code.google.com/apis/maps/documentation/geocoding/#GeocodingRequests

class ggeocoder.**GeoResult**(*data*)
Represents the data for a single result returned by the google maps api. Allows you to access the data from the response using attributes instead of diving deep into the returned dictionary structure.

You can create aliases for the default 'address_components' of the response by settings the *attr_mapping* field to a dict of your mapping.:

```
attr_mapping = {'state': 'administrative_area_level_1'}
```

coordinates
returns lat, lng coordinates as float types

formatted_address
returns fully formatted address result.

is_valid_address
returns True when geocode result is a street address, False if not

raw
Returns the raw dictionary object that the maps api returned for this single result.

exception ggeocoder.**GeocodeError**(*status*, *url=None*, *response=None*)
Base class for errors in the ggeocoder module.

Methods of the Geocoder raise this when the Google Maps API returns a status of anything other than 'OK'.

See http://code.google.com/apis/maps/documentation/geocoding/index.html#StatusCodes for status codes and their meanings.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX