python-omgeo Documentation

Release 1.6.0

Azavea Inc.

April 16, 2015

Contents

CHAPTER 1

Release Notes

1.1 v1.0, 2012-03-13

Initial Release

1.2 v1.1, 2012-03-22

Add ability to use ESRI premium tasks.

1.3 v1.2, 2012-04-04

Add Citizen Atlas (Washington DC) as a supported geocoder.

1.4 v1.3, 2012-04-20

Add ESRI SOAP locators as supported geocoders. Added suds dependency. Remove default rejected entities from BING setting to a postprocessor. Refactor constructors to fix bugs caused by mutable default values.

1.5 v1.3.1, 2012-04-23

Refactor test runner. Support multiple fields in GroupBy processor.

1.6 v1.3.2, 2012-04-25

Fix bug in ESRI geocoders in which defaults were appended rather than overwritten.

1.7 v1.3.3, 2012-04-27

- Add SnapPoints postprocessor.

- Fix bug in tests (Bing was not being used even if the environment variable BING_MAPS_API_KEY was set.
- Simplify ESRI ZIP processing
- Improve logging

1.8 v1.3.4, 2012-05-11

- Cast Nominatim coordinates returned in JSON from string to float. This was causing an error in the SnapPoints postprocessor added in v1.3.3, as the processor was attempting to evaluate mathematical equations using strings instead of numbers. A test was added to check that the types of the coordinates are successfully converted to floats.
- Add test for SnapPoints postprocessor.
- Change __unicode__() method for places.Candidate to display info indicating null or empty values, instead of just displaying "None".

1.9 v1.3.5, 2012-05-21

- Geocoder().geocode() method can take one-line string OR PlaceQuery instance.
- Improve speed by avoiding postprocessing of empty list
- Add support for MapQuest licensed geocoding API using NAVTEQ data

1.10 v1.3.6, 2012-05-22

- Add SSH support for MapQuest
- Add CancelIfRegexInAttr preprocessor to avoid geocoding attempts if a PlaceQuery instance attribute matches the given regex (such as a PO Box)
- Add timeout option that can be included in the GeocodeService settings parameter. There is now a default timeout of 10 seconds.

1.11 v1.3.7, 2012-05-23

- Add CancelIfPOBox preprocessor and tests
- Add __unicode__() / __str__() method to PlaceQuery for string representation

1.12 v1.4.0, 2012-06-13

- IMPORTANT: the Geocoder.geocode() method now returns a dictionary instead of a list of candidates. To just get a list of candidates, use Geocoder.get_candidates(). This functions the same the Geocoder.geocode() method in version 1.3.7.
- New dictionary return type includes a list of candidates as well as a list of UpstreamResponseInfo objects, which include information about the upstream API call, including response time and errors.

- Simplify error handling

1.13 v1.4.1, 2012-06-20

- Wrap entire JSON result processing from geocoder in try/catch
- Add separate logger for stats from geocoder

1.14 v1.4.3, 2012-08-09

- Expand information in UpstreamResponseInfo object
- Enhance logging

1.15 v1.4.4, 2012-08-13

- Add original PlaceQuery to nested dict method response

1.16 v1.5.0, 2012-09-19

- Add support for ESRI World Geocoder service
- Add documentation built in Sphinx (available at python-omgeo.readthedocs.org)
- Add shell script to rebuild, set API keys, and run tests
- Move pre- and post-processor modules to package base
- Add validation on Viewbox initialization
- Add repr() methods for place objects, including graphical Viewbox representation
- Modify ReplaceRangeWithNumber preprocessor to be friendly to ZIP+4 postal codes.

1.17 v1.5.1, 2012-09-20

- Fix ordering of default postprocessors for EsriWGS geocoder
- Improve handling of ZIP+4-only searching using EsriWGS
- Add repr() method to UpstreamResponseInfo class
- Minor documentation updates

1.18 v1.5.2, 2012-09-21

- Fix bug in AttrMigrator.__repr__()

1.19 v1.5.3, 2012-09-25

- Fix bug using wrong keys for ESRI findAddressCandidates endpoint in EsriWGS class

1.20 v1.5.4, 2012-10-09

- Fix bug using wrong key for MapQuest postalCode

1.21 v1.5.5, 2013-01-18

- Remove support for DC CitizenAtlas API
- Add try...except to stats logging command in geocode method
- Add option to raise exception on stats logging failure or add exception to general log.
- Add more documentation for Geocoder methods

1.22 v1.5.6, 2013-01-18

- Add test shell script template (test.dummy.sh) to source distribution.
- Remove unneeded requirements.txt from source distribution.

1.23 v1.6.0, 2013-10-31

- Add subregion and neighborhood parameters to queries
- Allow HTTP request headers to be specified in geocoder settings
- Documentation updates

1.24 v1.7.0, 2014-05-12

- Add Census geocoder

1.25 v1.7.1, 2014-08-25

- Return address components from US Census, EsriWGS geocoders.

1.26 v1.7.2, 2015-04-14

- Support EsriWGS magicKey.

CHAPTER 2

Tests

class omgeo.tests.tests.**GeocoderProcessorTest**(*methodName='runTest'*)

Tests using various pre- and post-processors.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_postpro_GroupBy()
Test GroupBy postprocessor.

test_pro_CancelIfPOBox()
Test CancelIfPOBox preprocessor.

test_pro_CancelIfRegexInAttr()
Test CancelIfRegexInAttr preprocessor.

test_pro_CancelIfRegexInAttr_case_sensitive()
Test CancelIfRegexInAttr preprocessor using case-sensitive option.

test_pro_SnapPoints()
Take two candidates within 50 metres and eliminate one.

test_pro_country_CountryPreProcessor()
Test CountryPreProcessor

test_pro_country_RequireCountry()
Test RequireCountry preprocessor.

test_pro_filter_AttrExclude_exact()
Test AttrExclude with `exact_match=True`.

test_pro_filter_AttrExclude_inexact()
Test AttrExclude with `exact_match=False`.

test_pro_filter_AttrFilter_exact()
Test AttrFilter postprocessor.

test_pro_filter_AttrFilter_inexact()
Test AttrFilter postprocessor with `exact_match=False`.

test_pro_parsing_ParseSingleLine()
Test ParseSingleLine preprocessor using single-line UK address.

test_pro_rename_AttrRename_exact()
Test AttrRename postprocessor using exact search string.

test_pro_rename_AttrRename_inexact()
Test AttrRename postprocessor using partial search string.

test_pro_scoring_ScoreSorter()
Test ScoreSorter postprocessor.

test_pro_scoring_ScoreSorter_asc()
Test ScoreSorter postprocessor with `reverse=False`.

test_pro_scoring_UseHighScoreIfAtLeast()
Test UseHighScoreIfAtLeast postprocessor.

test_pro_sort_AttrReverseSorter()
Test AttrReverseSorter postprocessor.

test_pro_sort_AttrSorter()
Test AttrSorter postprocessor.

test_pro_streetnumber_ReplaceRangeWithNumber()
Test ReplaceRangeWithNumber preprocessor.

class omgeo.tests.tests.**GeocoderTest**(*methodName='runTest'*)
Tests using various geocoding APIs. Requires internet connection.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_address_components()
Make sure EsriWGS returns address components

test_bounded_no_viewbox()
Should return a nice error saying that PlaceQuery can't be bounded without Viewbox.

test_esri_geocoder_eu_default_override()
Test for default argument bug in 3.1 – EsriNA and EsriEU append processors rather than replace them

test_esri_geocoder_na_default_override()
Test for default argument bug in 3.1 – EsriNA and EsriEU append processors rather than replace them

test_geocode_bing(**args*, ***kwargs*)
Test Azavea's address using Bing geocoder

test_geocode_census()
Test Azavea's address using US Census geocoder.

test_geocode_dupepicker()
Check that '340 12th St returns results'

test_geocode_esri_eu_soap()
Test ESRI Europe SOAP geocoder

test_geocode_esri_na_nz()
Test ESRI North America REST geocoder using a city in New Zealand.

test_geocode_esri_na_us()
Test ESRI North America REST geocoder

test_geocode_esri_na_us_soap()
Test ESRI North America SOAP geocoder

test_geocode_esri_wgs_340_12th_bounded()
Trying to geocode `340 12th St, Philadelphia PA` would normally return results for both `340 N 12th St` and `340 S 12th St`. Using a bounding box around Callowhill, we should only get the former.

test_geocode_esri_wgs_multipart()
Check that geocoding multipart address returns one result.

test_geocode_esri_wgs_senado_mx()
Attempt to geocode `Paseo de la Reforma 135, Tabacalera, Cuauhtémoc, Distrito Federal, 06030`.

test_geocode_esri_wgs_zip_plus_4()
Check that geocoding 19127-1112 returns one result.

test_geocode_karori(**args*, ***kwargs*)
Check that '102 Karori Road Karori Wellington' returns an address with the correct house number and postcode.

test_geocode_mapquest(**args*, ***kwargs*)
Test Azavea's address using MapQuest geocoder.

test_geocode_mapquest_ssl(**args*, ***kwargs*)
Test Azavea's address using secure MapQuest geocoder.

test_geocode_nom()
Test 1200 Callowhill Street using Nominatim geocoder. Also check to make sure coordinate values are floats and not some other data type.

test_geocode_snap_points_1()
Geocoder expected to return the same place twice – one with city as Flemington, and one with city as Readington Twp. This test checks that only one is picked.

test_geocode_snap_points_2(**args*, ***kwargs*)
Bing geocoder expected to return the same place twice – one with city as Alpha, and one with city as Phillipsburg. This test checks that only one is picked.

test_geocode_structured_esri_wgs_senado_mx()
Attempt to geocode `Paseo de la Reforma 135, Tabacalera, Cuauhtémoc, Distrito Federal, 06030` using a structured query to EsriWGS.

CHAPTER 3

Introduction

OMGeo - Python Edition

`python-omgeo` is a geocoding abstraction layer written in python. Currently supported geocoders:

- US Census Geocoder
- Bing Maps REST Locations API
- ESRI European Address Locator (REST & SOAP)
- ESRI North American Locator (TA_Address_NA_10) (REST & SOAP)
- ESRI World Geocoding Service
- MapQuest Licensed Data API
- MapQuest-hosted Nominatim Open Data API

Note: Check out this project's page on GitHub.

Installation:

```
sudo pip install python-omgeo
```

Documentation

Docs are available in HTML or PDF format.

Usage Example

Make a new geocoder and geocode and address:

```
>>> from omgeo import Geocoder
>>> g = Geocoder()
>>> result = g.geocode('340 12th St, Philadelphia PA')
```

Take a look at the result:

```
>>> result
{'candidates': [
  <340 S 12th St, Philadelphia, PA, 19107 (-75.161461, 39.94532) EsriWGS>,
  <340 N 12th St, Philadelphia, PA, 19107 (-75.158434, 39.958728) EsriWGS>
 ],
 'upstream_response_info': [<EsriWGS 1054ms>]}
```

Take a closer look at the information in our address Candidate objects:

```
>>> [c.__dict__ for c in result["candidates"]]
    [{'geoservice': 'EsriWGS',
      'locator': u'USA.AddressPoint',
      'locator_type': u'PointAddress',
      'match_addr': u'340 S 12th St, Philadelphia, PA, 19107',
      'score': 90.87,
      'wkid': 4326,
      'x': -75.161461,
      'y': 39.94532},
     {'geoservice': 'EsriWGS',
      'locator': 'interpolation',
      'locator_type': u'StreetAddress',
      'match_addr': u'340 N 12th St, Philadelphia, PA, 19107',
      'score': 90.87,
      'wkid': 4326,
      'x': -75.158434,
      'y': 39.958728}]
```

Some geocoders (EsriWGS and US Census) can return address components in addition to the full address:

```
>>> [{'geoservice': 'EsriWGS',
    'locator': 'interpolation',
    'locator_type': u'StreetAddress',
    'match_addr': u'340 N 12th St, Phila, Pennsylvania, 19107',
    'match_city': u'Phila',
    'match_country': u'USA',
    'match_postal': u'19107',
    'match_region': u'Pennsylvania',
    'match_streetaddr': u'340 N 12th St',
    'match_subregion': u'',
    'score': 90.1,
    'wkid': 4326,
    'x': -75.158384,
    'y': 39.958774}]
```

These are optional; their existence may change depending on the response from the geocoder.

Testing

There is a shell script in the root of the repository called *test.dummy.sh*. Copy it using `cp test.dummy.sh test.sh`. Edit *test.sh* to include the API keys that you obtained from the given geocoding service providers. Then, run the tests using `./test.sh`.

CHAPTER 4

Geocoder

class omgeo.**Geocoder**(*sources=None*, *preprocessors=None*, *postprocessors=None*, *waterfall=False*)

Class for building a custom geocoder using external APIs.

Parameters

- **sources** (*list*) – an array of GeocodeServiceConfig() parameters, keyed by module name for the GeocodeService to use, e.g.:

```
[['esri_wgs', {}],
 ['bing', {'settings': {'request_headers': {'User-Agent': 'Custom User Agent'} },
          'preprocessors': [],
          'postprocessors': []}],
  ...]
```

- **preprocessors** (*list*) – list of universal preprocessors to use
- **postprocessors** (*list*) – list of universal postprocessors to use
- **waterfall** (*bool*) – sets default for waterfall on geocode() method (default `False`)

add_source(*source*)

Add a geocoding service to this instance.

geocode(*pq*, *waterfall=None*, *force_stats_logging=False*)

Parameters

- **pq** (*PlaceQuery*) – PlaceQuery object (required).
- **waterfall** (*bool*) – Boolean set to True if all geocoders listed should be used to find results, instead of stopping after the first geocoding service with valid candidates (defaults to self.waterfall).
- **force_stats_logging** (*bool*) – Raise exception if stats logging fails (default False).

Returns Returns a dictionary including: * candidates - list of Candidate objects * upstream_response_info - list of UpstreamResponseInfo objects

get_candidates(*pq*, *waterfall=None*)

Geocode and return just the list of Candidate objects.

remove_source(*source*)

Remove a geocoding service from this instance.

set_sources(*sources*)

Creates GeocodeServiceConfigs from each str source

CHAPTER 5

places

class omgeo.places.**Candidate**(*locator=''*, *score=0*, *match_addr=''*, *x=None*, *y=None*, *wkid=4326*, ***kwargs*)

Class representing a candidate address returned from geocoders. Accepts arguments defined below, plus informal keyword arguments.

Usage Example:

```
c = Candidate('US_RoofTop', 91.5, '340 N 12th St, Philadelphia, PA, 19107',
    '-75.16', '39.95', some_key_foo='bar')
```

Parameters

- **locator** – Locator used for geocoding (default `''`)

  We try to standardize this to

  - `rooftop`,
  - `interpolated`,
  - `postal_specific`, and
  - `postal`.
- **score** – Standardized score (default `0`)
- **match_addr** (*str*) – Address returned by geocoder (default `''`)
- **x** – X-coordinate (longitude for lat-lon SRS) (default `None`)
- **y** – Y-coordinate (latitude for lat-lon SRS) (default `None`)
- **wkid** – Well-known ID for spatial reference system (default `4326`)

Keyword arguments can be added in order to be able to use postprocessors with API output fields are not well-fitted for one of the definitions above.

If possible, it is suggested for geocoders to additionally return the following address components:

•match_streetaddr (the street address, e.g. '340 N 12th Street')

•match_city

•match_subregion (county)

•match_region (state / province)

•match_postal

- match_country

However, these are not required. Currently the EsriWGS and US Census geocoders return these values.

class omgeo.places.**PlaceQuery**(*query=''*, *address=''*, *neighborhood=''*, *city=''*, *subregion=''*, *state=''*, *postal=''*, *country=''*, *viewbox=None*, *bounded=False*, ***kwargs*)

Class representing an address or place that will be passed to geocoders.

Parameters

- **query** (*str*) – A string containing the query to parse and match to a coordinate on the map. *ex: "340 N 12th St Philadelphia PA 19107" or "Wolf Building, Philadelphia"*
- **address** (*str*) – A string for the street line of an address. *ex: "340 N 12th St"*
- **neighborhood** (*str*) – A string for the subdivision of a city. Not used in US addresses, but used in Mexico and other places.
- **city** (*str*) – A string specifying the populated place for the address. This commonly refers to a city, but may refer to a suburb or neighborhood in certain countries.
- **subregion** (*str*) – A string for a region between the city and state level. Not used in US addresses.
- **state** (*str*) – A string for the state, province, territory, etc.
- **postal** (*str*) – A string for the postal / ZIP Code
- **country** (*str*) – A string for the country or region. Because the geocoder uses the country to determine which geocoding service to use, this is strongly recommended for efficency. ISO alpha-2 is preferred, and is required by some geocoder services.
- **viewbox** (*Viewbox*) – A Viewbox object indicating the preferred area to find search results (default `None`)
- **bounded** (*bool*) – Boolean indicating whether or not to only return candidates within the given Viewbox (default `False`)

Key float user_lat A float representing the Latitude of the end-user.

Key float user_lon A float representing the Longitude of the end-user.

Key str user_ip A string representing the IP address of the end-user.

Key str culture Culture code to be used for the request (used by Bing). For example, if set to 'de', the country for a U.S. address would be returned as "Vereinigte Staaten Von Amerika" instead of "United States".

class omgeo.places.**Viewbox**(*left=-180*, *top=90*, *right=180*, *bottom=-90*, *wkid=4326*)

Class representing a bounding box. Defaults to maximum bounds for WKID 4326.

Parameters

- **left** – Minimum X value (default `-180`)
- **top** – Maximum Y value (default `90`)
- **right** – Maximum X value (default `180`)
- **bottom** – Minimum Y value (default `-90`)
- **wkid** – Well-known ID for spatial reference system (default `4326`)

convert_srs(*new_wkid*)

Return a new Viewbox object with the specified SRS.

to_bing_str()
Convert Viewbox object to a string that can be used by Bing as a query parameter.

to_esri_wgs_json()
Convert Viewbox object to a JSON string that can be used by the ESRI World Geocoding Service as a parameter.

to_mapquest_str()
Convert Viewbox object to a string that can be used by MapQuest as a query parameter.

CHAPTER 6

services

class omgeo.services.**Bing**(*preprocessors=None*, *postprocessors=None*, *settings=None*)

Class to geocode using Bing services:

- Find a Location by Query
- Find a Location by Address

Settings used by the Bing GeocodeService object may include:

- api_key – The API key used to access Bing services.

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**EsriNA**(*preprocessors=None*, *postprocessors=None*, *settings=None*)

Esri REST Geocoder for North America

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**EsriNASoap**(*preprocessors=None*, *postprocessors=None*, *settings=None*)

Use the SOAP version of the ArcGIS-10-style Geocoder for North America

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**EsriWGS**(*preprocessors=None*, *postprocessors=None*, *settings=None*)
Class to geocode using the ESRI World Geocoding service.

This uses two endpoints – one for single-line addresses, and one for multi-part addresses.

An optional (key) parameter can be passed to the PlaceQuery which will be passed as a magicKey to the find endpoint if using a single line address/text search. This allows omgeo to be used with the Esri suggest endpoint.

Note: Based on tests using the magicKey parameter, it is recommended that a viewbox not be used with in conjuction with the magicKey. Additionally, address/search text passed via the query may be ignored when using a magicKey.

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**EsriWGSSSL**(*preprocessors=None*, *postprocessors=None*, *settings=None*)
Class to geocode using the *ESRI World Geocoding service over SSL <https://geocode.arcgis.com/arcgis/geocoding.html>_*

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**MapQuest**(*preprocessors=None*, *postprocessors=None*, *settings=None*)

Class to geocode using MapQuest licensed services.

Overwrite _preprocessors, _postprocessors, and _settings if they are set.

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

class omgeo.services.**Nominatim**(*preprocessors=None*, *postprocessors=None*, *settings=None*)

Class to geocode using Nominatim services hosted by MapQuest.

geocode(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Return type tuple

Returns

post-processed list of Candidate objects and and UpstreamResponseInfo object if an API call was made.

Examples:

Preprocessor throws out request:

```
([], None)
```

Postprocessor throws out some candidates:

```
([<Candidate obj>, <Candidate obj>, ...], <UpstreamResponseInfo obj>)
```

Postprocessor throws out all candidates:

```
([], <UpstreamResponseInfo obj>)
```

An exception occurs while making the API call:

```
([], <UpstreamResponseInfo obj>)
```

CHAPTER 7

Preprocessors for cleaning user input

class omgeo.preprocessors.**CancelIfRegexInAttr**(*regex*, *attrs*, *ignorecase=True*)

Return False if given regex is found in ANY of the given PlaceQuery attributes, otherwise return original PlaceQuery instance. In the event that a given attribute does not exist in the given PlaceQuery, no exception will be raised.

Parameters

- **regex** (*str*) – a regex string to match (represents what you do *not* want)
- **attrs** – a list or tuple of strings of attribute names to look through
- **ignorecase** (*bool*) – set to `False` for a case-sensitive match (default `True`)

class omgeo.preprocessors.**CountryPreProcessor**(*acceptable_countries=None*, *country_map=None*)

Used to filter acceptable countries and standardize country names or codes.

Parameters

- **acceptable_countries** (*list*) – A list of acceptable countries. None is used to indicate that all countries are acceptable. (default `[]`)

  An empty string is also an acceptable country. To require a country, use the *RequireCountry* preprocessor.

- **country_map** (*dict*) – A map of the input PlaceQuery.country property to the country value accepted by the geocoding service.

  For example, suppose that the geocoding service recognizes 'GB', but not 'UK' – and 'US', but not 'USA':

  ```
  country_map = {'UK':'GB', 'USA':'US'}
  ```

process(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Returns modified PlaceQuery, or `False` if country is not acceptable.

class omgeo.preprocessors.**ParseSingleLine**(***kwargs*)

Adapted from Cicero Live

In a subclass, arguments may be formally defined to avoid the use of keywords (and to throw errors when bogus keyword arguments are passed):

```
def __init__(self, arg1='foo', arg2='bar')
```

process(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Returns PlaceQuery instance with `query` converted to individual elements

class `omgeo.preprocessors.`**ReplaceRangeWithNumber**(***kwargs*)

Class to take only the first part of an address range or hyphenated house number to use for geocoding.

This affects the query and address PlaceQuery attributes.

Input	Output
`4109-4113 Main St`	`4109 Main St`
`4109-13 Main St`	`4109 Main St`
`322-1/2 Water Street`	`322 Water Street`
`123-2 Maple Lane`	`123 Maple Lane`
`272-B Greenough St, 19127`	`272 Greenough St, 19127`
`272 Greenough St 19127-1112`	`272 Greenough St 19127-1112`
`19127-1112`	`19127-1112` (not affected)
`76-20 34th Ave, Queens NY`	`76 34th Ave, Queens NY` (see warning)

Warning: This may cause problems with addresses presented in the hyphenated Queens-style format, where the part before the hyphen indicates the cross street, and the part after indicates the house number.

In a subclass, arguments may be formally defined to avoid the use of keywords (and to throw errors when bogus keyword arguments are passed):

```
def __init__(self, arg1='foo', arg2='bar')
```

process(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Returns PlaceQuery instance with truncated address range / number

class `omgeo.preprocessors.`**RequireCountry**(*default_country=''*)

Return False if no default country is set in first parameter. Otherwise, return the default country if country is empty.

Parameters **default_country** (*str*) – default country to use if there is

no country set in the PlaceQuery instance sent to this processor. If this argument is not set or empty and PlaceQuery instance does not have a country (pq.country == ''), the processor will return False and the PlaceQuery will be rejected during geocoding. (default `''`)

process(*pq*)

Parameters **pq** (*PlaceQuery*) – PlaceQuery instance

Returns One of the three following values: * unmodified PlaceQuery instance if pq.country is not empty * PlaceQuery instance with pq.country changed to default country. * `False` if pq.country is empty and self.default_country == ''.

CHAPTER 8

Postprocessors for filtering and sorting results

class omgeo.postprocessors.**AttrExclude**(*bad_values=[]*, *attr='locator'*, *exact_match=True*)
PostProcessor used to ditch results with unwanted attribute values.

Parameters

- **bad_values** (*list*) – A list of values whose candidates we will not accept results from (default [])
- **attr** (*string*) – The attribute type on which to filter
- **exact_match** (*bool*) – True if attribute must match a bad value exactly. False if the bad value can be a substring of the attribute value. In other words, if our Candidate attribute is 'Postcode3' and one of the bad values is 'Postcode' because we want something more precise, like 'Address', we will not keep this candidate.

class omgeo.postprocessors.**AttrFilter**(*good_values=[]*, *attr='locator'*, *exact_match=True*)
PostProcessor used to ditch results with unwanted attribute values.

Parameters

- **good_values** (*list*) – A list of values whose candidates we will accept results from (default [])
- **attr** (*string*) – The attribute type on which to filter
- **exact_match** (*bool*) – True if attribute must match a good value exactly. False if the attribute can be a substring in a good value. In other words, if our Candidate attribute is 'US_Rooftop' and one of the good_values is 'Rooftop', we will keep this candidate.

class omgeo.postprocessors.**AttrMigrator**(*attr_from*, *attr_to*, *attr_map=None*, *exact_match=False*, *case_sensitive=False*)
PostProcessor used to migrate the given attribute to another attribute.

class omgeo.postprocessors.**AttrRename**(*attr*, *attr_map=None*, *exact_match=False*, *case_sensitive=False*)
PostProcessor used to rename the given attribute, with unspecified attributes appearing at the end of the list.

Parameters

- **attr** (*str*) – Name of the attribute
- **attr_map** (*dict*) – Map of old names : new names.
- **exact_match** (*bool*) –
- **case_sensitive** (*bool*) –

process(*candidates*)

Parameters **candidates** (*list*) – list of Candidate instances

Returns list of Candidate instances with modified values for the given attribute

class omgeo.postprocessors.**AttrReverseSorter**(*ordered_values=None*, *attr='locator'*)
PostProcessor used to sort by the given attributes in reverse order, with unspecified attributes appearing at the end of the list.

This is good to use when a list has already been defined in a script and you are too lazy to use the reverse() function, or don't want to in order to maintain more readable code.

Parameters

- **ordered_values** (*list*) – A list of values placed in the reverse of the desired order.
- **attribute** (*str*) – The attribute on which to sort

class omgeo.postprocessors.**AttrSorter**(*ordered_values=None*, *attr='locator'*)
PostProcessor used to sort by a the given attribute, with unspecified attributes appearing at the end of the list.

Parameters

- **ordered_values** (*list*) – A list of values placed in the desired order.
- **attr** (*str*) – The attribute on which to sort.

class omgeo.postprocessors.**DupePicker**(*attr_dupes*, *attr_sort*, *ordered_list*, *return_clean=False*)
PostProcessor used to choose the best candidate(s) where there are duplicates (or more than one result that is very similar*) among high-scoring candidates, such as an address.

•When comparing attribute values, case and commas do not count.

Usage Example:

match_addr	score	locator
123 N Wood St	90	roof
123 S Wood St	90	address
123 N WOOD ST	77	address
123, S Wood ST	85	roof

Above, the first two results have the highest scores. We could just use those, because one of the two likely has the correct address. However, the second result does not have the most precise location for 123 S. Wood Street. While the fourth result does not score as high as the first two, its locator method is more desirable. Since the addresses are the same, we can assume that the fourth result will provide better data than the second.

We can get a narrowed list as described above by running the process() method in the DupePicker() PostProcessor class as follows, assuming that the "candidates" is our list of candidates:

```
dp = DupePicker(
    attr_dupes='match_addr',
    attr_sort='locator',
    ordered_list=['rooftop', 'address_point', 'address_range'])

return dp.process(candidates)
```

Output:

match_addr	score	locator
123 N Wood St	90	roof
123, S Wood ST	85	roof

Output with return_clean=True:

match_addr	score	locator
123 N WOOD ST	90	roof
123 S WOOD ST	85	roof

Parameters

- **attr_dupes** (*str*) – Property on which to look for duplicates.
- **attr_sort** (*str*) – Property on which to sort using ordered_list
- **ordered_list** (*list*) – A list of property values, from most desirable to least desirable.
- **return_clean** (*bool*) – Boolean indicating whether or not to homogenize string values into uppercase without commas.

class omgeo.postprocessors.**GroupBy**(*attr='match_addr'*)
Groups results by a certain attribute by choosing the first occurrence in the list (this means you will want to sort ahead of time).

attr – The attribute on which to combine results or a list or tuple of attributes where all attributes must match between candidates.

class omgeo.postprocessors.**GroupByMultiple**(*attrs*)
Groups results by a certain attribute by choosing the first occurrence in the list of candidates (this means you will want to sort ahead of time).

attrs – A list or tuple of attributes on which to combine results

class omgeo.postprocessors.**LocatorFilter**(*good_locators*)
PostProcessor used to ditch results with lousy locators.

Parameters

- **good_locators** (*list*) – A list of locators to accept results from (default [])
- **good_locators** – A list of locators to accept results from (default None)

process(*candidates*)

Parameters **candidates** (*list*) – list of Candidate instances

class omgeo.postprocessors.**LocatorSorter**(*ordered_locators*)
PostProcessor used to sort by locators

Parameters **ordered_locators** (*list*) – a list of `Candidate.locator` values

placed in the desired order, such as `rooftop`, `interpolation`, or `postal`.

process(*unordered_candidates*)

Parameters **candidates** (*list*) – list of Candidate instances

class omgeo.postprocessors.**ScoreSorter**(*reverse=True*)
PostProcessor class to sort `Candidate` scores.

Parameters **reverse** (*bool*) – indicates if the scores should be sorted in descending

order (e.g. 100, 90, 80, ...) (default `True`)

process(*candidates*)

Parameters **candidates** (*list*) – list of Candidates

Returns score-sorted list of Candidates

class omgeo.postprocessors.**SnapPoints**(*distance=50*)
Chooses the first of two or more points where they are within the given sphere-based great circle distance.

Parameters **distance** – maximum distance (in metres) between two points in which

the the first will be kept and the second thrown out (default 50).

class omgeo.postprocessors.**UseHighScoreIfAtLeast**(*min_score*)
Limit results to results with at least the given score, if and only if one or more results has, at least, the given score. If no results have at least this score, all of the original results are returned intact.

process(*candidates*)

Parameters **candidates** (*list*) – list of Candidates

Returns list of Candidates where score is at least min_score, if and only if one or more Candidates have at least min_score. Otherwise, returns original list of Candidates.

CHAPTER 9

Indices and tables

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

O

U

V