

# Package ‘tidygeocoder’

October 14, 2022

**Type** Package

**Title** Geocoding Made Easy

**Version** 1.0.5

**Description** An intuitive interface for getting data from geocoding services.

**License** MIT + file LICENSE

**URL** <https://jessecambon.github.io/tidygeocoder/>,  
<https://github.com/jessecambon/tidygeocoder>

**BugReports** <https://github.com/jessecambon/tidygeocoder/issues>

**Depends** R (>= 3.3.0)

**Imports** tibble, dplyr, httr, jsonlite, lifecycle, progress

**Suggests** knitr, rmarkdown, ggplot2, ggrepel, maps, testthat (>= 3.0.2)

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** Jesse Cambon [aut, cre] (<<https://orcid.org/0000-0001-6854-1514>>),  
Diego Hernangómez [aut] (<<https://orcid.org/0000-0001-8457-4658>>),  
Christopher Belanger [aut] (<<https://orcid.org/0000-0003-2070-5721>>),  
Daniel Possenriede [aut] (<<https://orcid.org/0000-0002-6738-9845>>)

**Maintainer** Jesse Cambon <[jesse.cambon@gmail.com](mailto:jesse.cambon@gmail.com)>

**Repository** CRAN

**Date/Publication** 2021-11-02 22:20:05 UTC

## R topics documented:

|  |   |
|--|---|
| <code>api_info_reference</code> . . . . .      | 2 |
| <code>api_key_reference</code> . . . . .       | 3 |
| <code>api_parameter_reference</code> . . . . . | 3 |

|                                   |           |
|-----------------------------------|-----------|
| batch_limit_reference . . . . .   | 5         |
| extract_results . . . . .         | 5         |
| extract_reverse_results . . . . . | 6         |
| geo . . . . .                     | 7         |
| geocode . . . . .                 | 12        |
| geocode_combine . . . . .         | 14        |
| geo_census . . . . .              | 16        |
| geo_combine . . . . .             | 17        |
| get_api_query . . . . .           | 19        |
| louisville . . . . .              | 20        |
| min_time_reference . . . . .      | 20        |
| query_api . . . . .               | 21        |
| reverse_geo . . . . .             | 23        |
| reverse_geocode . . . . .         | 26        |
| sample_addresses . . . . .        | 28        |
| <b>Index</b>                      | <b>29</b> |

---

api\_info\_reference      *Geocoding service links and information*

---

## Description

This dataset is used for generating package documentation.

## Usage

api\_info\_reference

## Format

A tibble dataframe

**method** Geocoding service name

**method\_display\_name** Geocoding service display name

**site\_url** Link to the main site of the geocoding service

**api\_documentation\_url** Link to API documentation

**api\_usage\_policy\_url** Link to the usage policy

---

|                   |  |
|-------------------|--|
| api_key_reference | <i>API key environmental variables</i> |
|-------------------|--|

---

### Description

API keys are obtained from environmental variables. The [geo](#) and [reverse\\_geo](#) functions use this dataset to know which environmental variable to use for each geocoding service.

### Usage

```
api_key_reference
```

### Format

A tibble dataframe

**method** Geocoding service name

**env\_var** Environmental variable name

### See Also

[geo](#) [reverse\\_geo](#)

---

|                         |  |
|-------------------------|--|
| api_parameter_reference | <i>Geocoding service API parameter reference</i> |
|-------------------------|--|

---

### Description

This dataset contains the mapping that allows this package to use a universal syntax to specify parameters for different geocoding services. Note that latitude and longitude input parameters for reverse geocoding are not in this dataset and are instead handled directly by the [reverse\\_geo](#) function.

The `generic_name` column is a universal parameter name that is shared between services. The `api_name` column is the parameter name for the given geocoding service specified by the `method` column. When `generic_name` is missing this means the parameter is specific to that geocoding service.

While the "census" and "google" services do not have a `limit` argument in their APIs, tidygeocoder provides a passthrough so you can still use the `limit` argument in [geo](#) and [reverse\\_geo](#) to limit the number of results per input.

Note that some geocoding services only use the `limit` argument for forward geocoding. Refer to API documentation of each service for more information.

Reference the documentation for [geo](#) and [reverse\\_geo](#) for more information. Also reference `vignette("tidygeocoder")` for more details on constructing API queries.

**Usage**

`api_parameter_reference`

**Format**

A tibble dataframe

**method** Geocoding service name

**generic\_name** Universal parameter name

**api\_name** Name of the parameter for the specified geocoding service

**default\_value** Default value of the parameter

**required** Is the parameter required by the specified geocoding service?

**Details**

The API documentation for each service is linked to below:

- [Nominatim](#)
- [US Census](#)
- [ArcGIS](#)
- [Geocodio](#)
- [Location IQ](#)
- [Google](#)
- [OpenCage](#)
- [Mapbox](#)
- [HERE](#)
- [TomTom](#)
- [MapQuest](#)
- [Bing](#)
- [Geoapify](#)

**See Also**

[geo\\_reverse\\_geo](#) [get\\_api\\_query](#) [query\\_api](#) [min\\_time\\_reference](#) [batch\\_limit\\_reference](#)

---

batch\_limit\_reference *Geocoding batch size limits*

---

### Description

The [geo](#) and [reverse\\_geo](#) functions use this dataset to set the maximum batch query size for each service.

### Usage

```
batch_limit_reference
```

### Format

A tibble dataframe

**method** Geocoding service name

**batch\_limit** The maximum number of addresses or coordinates allowed per batch

### See Also

[geo reverse\\_geo](#)

---

extract\_results *Extract forward geocoding results*

---

### Description

Parses the output of the [query\\_api](#) function for single address geocoding (ie. not batch geocoding). Latitude and longitude are extracted into the first two columns of the returned dataframe. Refer to [query\\_api](#) for example usage.

### Usage

```
extract_results(  
  method,  
  response,  
  full_results = TRUE,  
  flatten = TRUE,  
  limit = 1  
)
```

**Arguments**

|              |   |
|--------------|---|
| method       | method name   |
| response     | content from the geocoding service (returned by the <a href="#">query_api</a> function) |
| full_results | if TRUE then the full results (not just latitude and longitude) will be returned.       |
| flatten      | if TRUE then flatten any nested dataframe content                                       |
| limit        | only used for "census" and "google" methods. Limits number of results per address.      |

**Value**

geocoding results in tibble format

**See Also**

[get\\_api\\_query](#) [query\\_api](#) [geo](#)

---

extract\_reverse\_results

*Extract reverse geocoding results*

---

**Description**

Parses the output of the [query\\_api](#) function for reverse geocoding. The address is extracted into the first column of the returned dataframe. This function is not used for batch geocoded results. Refer to [query\\_api](#) for example usage.

**Usage**

```
extract_reverse_results(
  method,
  response,
  full_results = TRUE,
  flatten = TRUE,
  limit = 1
)
```

**Arguments**

|              |   |
|--------------|---|
| method       | method name   |
| response     | content from the geocoding service (returned by the <a href="#">query_api</a> function) |
| full_results | if TRUE then the full results (not just an address column) will be returned.            |
| flatten      | if TRUE then flatten any nested dataframe content                                       |
| limit        | only used for the "google" method(s). Limits number of results per coordinate.          |

**Value**

geocoding results in tibble format

**See Also**

[get\\_api\\_query](#) [query\\_api](#) [reverse\\_geo](#)

---

geo *Geocode addresses*

---

**Description**

Geocodes addresses given as character values. The [geocode](#) function utilizes this function on addresses contained in dataframes. See example usage in `vignette("tidygeocoder")`.

Note that not all geocoding services support certain address component parameters. For example, the Census geocoder only covers the United States and does not have a "country" parameter.

Refer to [api\\_parameter\\_reference](#), [min\\_time\\_reference](#), and [batch\\_limit\\_reference](#) for more details on geocoding service parameters and usage.

This function uses the [get\\_api\\_query](#), [query\\_api](#), and [extract\\_results](#) functions to create, execute, and parse geocoder API queries.

**Usage**

```
geo(  
  address = NULL,  
  street = NULL,  
  city = NULL,  
  county = NULL,  
  state = NULL,  
  postalcode = NULL,  
  country = NULL,  
  method = "osm",  
  cascade_order = c("census", "osm"),  
  lat = "lat",  
  long = "long",  
  limit = 1,  
  full_results = FALSE,  
  mode = "",  
  unique_only = FALSE,  
  return_addresses = TRUE,  
  min_time = NULL,  
  progress_bar = show_progress_bar(),  
  quiet = getOption("tidygeocoder.quiet", FALSE),  
  api_url = NULL,  
  timeout = 20,  
  flatten = TRUE,
```

```

batch_limit = NULL,
batch_limit_error = TRUE,
verbose = getOption("tidygeocoder.verbose", FALSE),
no_query = FALSE,
custom_query = list(),
api_options = list(),
return_type = "locations",
iq_region = "us",
geocodio_v = 1.6,
param_error = TRUE,
mapbox_permanent = FALSE,
here_request_id = NULL,
mapquest_open = FALSE
)

```

### Arguments

|            |   |
|------------|---|
| address    | single line address (ie. '1600 Pennsylvania Ave NW, Washington, DC'). Do not combine with the address component arguments below (street, city, county, state, postalcode, country).   |
| street     | street address (ie. '1600 Pennsylvania Ave NW')   |
| city       | city (ie. 'Tokyo')  |
| county     | county (ie. 'Jefferson')  |
| state      | state (ie. 'Kentucky')  |
| postalcode | postalcode (ie. zip code if in the United States)   |
| country    | country (ie. 'Japan')   |
| method     | <p>the geocoding service to be used. API keys are loaded from environmental variables. Run <code>usethis::edit_r_environ()</code> to open your <code>.Renviron</code> file and add an API key as an environmental variable. For example, add the line <code>GEOCODIO_API_KEY="YourAPIKeyHere"</code></p> <ul style="list-style-type: none"> <li>• "osm": <b>Nominatim</b>.</li> <li>• "census": <b>US Census</b>. Geographic coverage is limited to the United States. Batch geocoding is supported.</li> <li>• "arcgis": <b>ArcGIS</b>.</li> <li>• "geocodio": <b>Geocodio</b>. Geographic coverage is limited to the United States and Canada. An API key must be stored in the environmental variable "GEOCODIO_API_KEY". Batch geocoding is supported.</li> <li>• "iq": <b>Location IQ</b>. An API key must be stored in the environmental variable "LOCATIONIQ_API_KEY".</li> <li>• "google": <b>Google</b>. An API key must be stored in the environmental variable "GOOGLEGEOCODE_API_KEY".</li> <li>• "opencage": <b>OpenCage</b>. An API key must be stored in the environmental variable "OPENCAGE_KEY".</li> <li>• "mapbox": <b>Mapbox</b>. An API key must be stored in the environmental variable "MAPBOX_API_KEY".</li> </ul> |



- "here": **HERE**. An API key must be stored in the environmental variable "HERE\_API\_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
- "tomtom": **TomTom**. An API key must be stored in the environmental variable "TOMTOM\_API\_KEY". Batch geocoding is supported.
- "mapquest": **MapQuest**. An API key must be stored in the environmental variable "MAPQUEST\_API\_KEY". Batch geocoding is supported.
- "bing": **Bing**. An API key must be stored in the environmental variable "BINGMAPS\_API\_KEY". Batch geocoding is supported, but must be explicitly called with mode = "batch".
- "geoapify": **Geoapify**. An API key must be stored in the environmental variable "GEOAPIFY\_KEY".
- "cascade" [**Deprecated**] use [geocode\\_combine](#) or [geo\\_combine](#) instead. The "cascade" method first uses one geocoding service and then uses a second geocoding service if the first service didn't return results. The services and order is specified by the cascade\_order argument. Note that this is not compatible with full\_results = TRUE as geocoding services have different columns that they return.

|                  |  |
|------------------|--|
| cascade_order    | <b>[Deprecated]</b> a vector with two character values for the method argument in the order in which the geocoding services will be attempted for method = "cascade" (ie. c("census", "geocodio"))   |
| lat              | latitude column name. Can be quoted or unquoted (ie. lat or "lat").  |
| long             | longitude column name. Can be quoted or unquoted (ie. long or "long").   |
| limit            | maximum number of results to return per input address. For many geocoding services the maximum value of the limit parameter is 100. Pass limit = NULL to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if return_addresses = TRUE. Refer to <a href="#">api_parameter_reference</a> for more details.                       |
| full_results     | returns all available data from the geocoding service if TRUE. If FALSE (default) then only latitude and longitude columns are returned from the geocoding service.  |
| mode             | set to 'batch' to force batch geocoding or 'single' to force single address geocoding (one address per query). If not specified then batch geocoding will be used if available (given method selected) when multiple addresses are provided; otherwise single address geocoding will be used. For the "here" and "bing" methods the batch mode should be explicitly specified with mode = 'batch'. |
| unique_only      | only return results for unique inputs if TRUE  |
| return_addresses | return input addresses with results if TRUE. Note that most services return the input addresses with full_results = TRUE and setting return_addresses to FALSE does not prevent this.  |
| min_time         | minimum amount of time for a query to take (in seconds). If NULL then min_time will be set to the default value specified in <a href="#">min_time_reference</a> .  |
| progress_bar     | if TRUE then a progress bar will be displayed for single input geocoding (1 input per query). By default the progress bar will not be shown for code executed when knitting R Markdown files or code within an RStudio notebook  |

|                   |   |
|-------------------|---|
|                   | chunk. Can be set permanently with <code>options(tidygeocoder.progress_bar = FALSE)</code> .  |
| quiet             | if TRUE then console messages that are displayed by default regarding queries will be suppressed. FALSE is default. Can be set permanently with <code>options(tidygeocoder.quiet = TRUE)</code> .   |
| api_url           | custom API URL. If specified, the default API URL will be overridden. This parameter can be used to specify a local Nominatim server, for instance.   |
| timeout           | query timeout (in minutes)  |
| flatten           | if TRUE (default) then any nested dataframes in results are flattened if possible. Note that in some cases results are flattened regardless such as for Geocodio batch geocoding.   |
| batch_limit       | limit to the number of addresses in a batch geocoding query. Defaults to the value in <a href="#">batch_limit_reference</a> if not specified.   |
| batch_limit_error | <b>[Deprecated]</b> if TRUE then an error is thrown if the number of addresses exceeds the batch limit. (if executing a batch query). This is reverted to FALSE when using the cascade method.  |
| verbose           | if TRUE then detailed logs are output to the console. FALSE is default. Can be set permanently with <code>options(tidygeocoder.verbose = TRUE)</code>   |
| no_query          | if TRUE then no queries are sent to the geocoding service and verbose is set to TRUE. Used for testing.   |
| custom_query      | API-specific parameters to be used, passed as a named list (ex. <code>list(extratags = 1)</code> ).   |
| api_options       | a named list of parameters specific to individual services. (ex. <code>list(geocodio_v = 1.6, geocodio_hipaa = TRUE)</code> ). Each parameter begins with the name of the method (service) it applies to. The possible parameters are shown below with their default values. <ul style="list-style-type: none"> <li>• <code>census_return_type</code> (default: "locations"): set to "geographies" to return additional geography columns. Make sure to use <code>full_results = TRUE</code> if using the "geographies" setting.</li> <li>• <code>iq_region</code> (default: "us"): set to "eu" to use the European Union API endpoint</li> <li>• <code>geocodio_v</code> (default: 1.6): the version number of the Geocodio API to be used</li> <li>• <code>geocodio_hipaa</code> (default: FALSE): set to TRUE to use the HIPAA compliant Geocodio API endpoint</li> <li>• <code>mapbox_permanent</code> (default: FALSE): set to TRUE to use the <code>mapbox.places-permanent</code> endpoint. Note that this option should be used only if you have applied for a permanent account. Unsuccessful requests made by an account that does not have access to the endpoint may be billable.</li> <li>• <code>mapbox_open</code> (default: FALSE): set to TRUE to use the Open Geocoding endpoint which relies solely on OpenStreetMap data</li> <li>• <code>here_request_id</code> (default: NULL): this parameter would return a previous HERE batch job, identified by its RequestID. The RequestID of a batch job</li> </ul> |

is displayed when verbose is TRUE. Note that this option would ignore the current address parameter on the request, so the return\_addresses or return\_coords parameters need to be FALSE.

|                  |  |
|------------------|--|
| return_type      | <b>[Deprecated]</b> use the api_options parameter instead  |
| iq_region        | <b>[Deprecated]</b> use the api_options parameter instead  |
| geocodio_v       | <b>[Deprecated]</b> use the api_options parameter instead  |
| param_error      | <b>[Deprecated]</b> if TRUE then an error will be thrown if any address parameters are used that are invalid for the selected service (method). If method = "cascade" then no errors will be thrown. |
| mapbox_permanent | <b>[Deprecated]</b> use the api_options parameter instead  |
| here_request_id  | <b>[Deprecated]</b> use the api_options parameter instead  |
| mapquest_open    | <b>[Deprecated]</b> use the api_options parameter instead  |

### Value

tibble (dataframe)

### See Also

[geocode](#) [api\\_parameter\\_reference](#) [min\\_time\\_reference](#) [batch\\_limit\\_reference](#)

### Examples

```
options(tidygeocoder.progress_bar = FALSE)

geo(street = "600 Peachtree Street NE", city = "Atlanta",
    state = "Georgia", method = "census")

geo(address = c("Tokyo, Japan", "Lima, Peru", "Nairobi, Kenya"),
    method = 'osm')

geo("100 Main St New York, NY", full_results = TRUE,
    method = "census", api_options = list(census_return_type = 'geographies'))

geo(county = 'Jefferson', state = "Kentucky", country = "US",
    method = 'osm')
```

geocode

*Geocode addresses in a dataframe***Description**

Takes a dataframe containing addresses as an input and returns the results from a specified geocoding service in a dataframe format using the `geo` function. See example usage in `vignette("tidygeocoder")`.

This function passes all additional parameters (...) to the `geo` function, so you can refer to its documentation for more details on possible arguments.

Note that the arguments used for specifying address columns (`address`, `street`, `city`, `county`, `state`, `postalcode`, and `country`) accept either quoted or unquoted column names (ie. `"address_col"` and `address_col` are both acceptable).

**Usage**

```
geocode(
  .tbl,
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  lat = "lat",
  long = "long",
  return_input = TRUE,
  limit = 1,
  return_addresses = NULL,
  unique_only = FALSE,
  ...
)
```

**Arguments**

|                         |   |
|-------------------------|---|
| <code>.tbl</code>       | dataframe containing addresses  |
| <code>address</code>    | single line street address column name. Do not combine with address component arguments ( <code>street</code> , <code>city</code> , <code>county</code> , <code>state</code> , <code>postalcode</code> , <code>country</code> ) |
| <code>street</code>     | street address column name  |
| <code>city</code>       | city column name  |
| <code>county</code>     | county column name  |
| <code>state</code>      | state column name   |
| <code>postalcode</code> | postalcode column name (zip code if in the United States)   |
| <code>country</code>    | country column name   |

|                               |   |
|-------------------------------|---|
| <code>lat</code>              | latitude column name. Can be quoted or unquoted (ie. <code>lat</code> or <code>'lat'</code> ).  |
| <code>long</code>             | longitude column name. Can be quoted or unquoted (ie. <code>long</code> or <code>'long'</code> ).   |
| <code>return_input</code>     | if TRUE then the input dataset will be combined with the geocoder query results and returned. If FALSE only the geocoder results will be returned.  |
| <code>limit</code>            | maximum number of results to return per input address. For many geocoding services the maximum value of the limit parameter is 100. Pass <code>limit = NULL</code> to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if <code>return_addresses = TRUE</code> . To use <code>limit &gt; 1</code> or <code>limit = NULL</code> set <code>return_input</code> to FALSE. Refer to <a href="#">api_parameter_reference</a> for more details. |
| <code>return_addresses</code> | if TRUE return input addresses. Defaults to TRUE if <code>return_input</code> is FALSE and FALSE if <code>return_input</code> is TRUE. This argument is passed to the <code>geo()</code> function.  |
| <code>unique_only</code>      | if TRUE then only unique results will be returned and <code>return_input</code> will be set to FALSE.   |
| <code>...</code>              | arguments passed to the <a href="#">geo</a> function  |

**Value**

tibble (dataframe)

**See Also**

[geo](#)

**Examples**

```
library(dplyr, warn.conflicts = FALSE)
sample_addresses %>% slice(1:2) %>%
  geocode(addr, method = 'arcgis')

louisville %>% head(2) %>%
  geocode(street = street, city = city, state = state,
          postalcode = zip, method = 'census', full_results = TRUE)

sample_addresses %>% slice(8:9) %>%
  geocode(addr, method = 'osm', limit = 2,
          return_input = FALSE, full_results = TRUE)

sample_addresses %>% slice(4:5) %>%
  geocode(addr, method = 'arcgis',
          lat = latitude, long = longitude,
          full_results = TRUE)
```

---

geocode\_combine      *Combine multiple geocoding queries*

---

### Description

Executes multiple geocoding queries on a dataframe input and combines the results. To use a character vector input instead, see the [geo\\_combine](#) function. Queries are executed by the [geocode](#) function. See example usage in `vignette("tidygeocoder")`.

Query results are by default labelled to show which query produced each result. Labels are either placed in a query column (if `return_list = FALSE`) or used as the names of the returned list (if `return_list = TRUE`). By default the method parameter value of each query is used as a query label. If the same method is used in multiple queries then a number is added according to the order of the queries (ie. `osm1`, `osm2`, ...). To provide your own custom query labels use the `query_names` parameter.

### Usage

```
geocode_combine(
  .tbl,
  queries,
  global_params = list(),
  return_list = FALSE,
  cascade = TRUE,
  query_names = NULL,
  lat = "lat",
  long = "long"
)
```

### Arguments

|                            |  |
|----------------------------|--|
| <code>.tbl</code>          | dataframe containing addresses   |
| <code>queries</code>       | a list of queries, each provided as a list of parameters. The queries are executed by the <a href="#">geocode</a> function in the order provided. (ex. <code>list(list(method = 'osm'), list(method = 'census'), ...)</code> ) |
| <code>global_params</code> | a list of parameters to be used for all queries (ex. <code>list(address = 'address', full_results = TRUE)</code> )   |
| <code>return_list</code>   | if TRUE then results from each service will be returned as separate dataframes. If FALSE (default) then all results will be combined into a single dataframe.  |
| <code>cascade</code>       | if TRUE (default) then only addresses that are not found by a geocoding service will be attempted by subsequent queries. If FALSE then all queries will attempt to geocode all addresses.                                      |
| <code>query_names</code>   | optional vector with one label for each query provided (ex. <code>c('geocodio batch', 'geocodio single')</code> ).   |
| <code>lat</code>           | latitude column name. Can be quoted or unquoted (ie. <code>lat</code> or <code>'lat'</code> ).   |
| <code>long</code>          | longitude column name. Can be quoted or unquoted (ie. <code>long</code> or <code>'long'</code> ).  |

**Value**

tibble (dataframe)

**See Also**

[geo\\_combine](#) [geo\\_geocode](#)

**Examples**

```
library(dplyr, warn.conflicts = FALSE)

sample_addresses %>%
  geocode_combine(
    queries = list(list(method = 'census'), list(method = 'osm')),
    global_params = list(address = 'addr'), cascade = TRUE)

more_addresses <- tibble::tribble(
  ~street_address, ~city, ~state, ~zip_cd,
  "624 W DAVIS ST #1D", "BURLINGTON", "NC", 27215,
  "201 E CENTER ST #268", "MEBANE", "NC", 27302,
  "100 Wall Street", "New York", "NY", 10005,
  "Bucharest", NA, NA, NA
)

more_addresses %>%
  geocode_combine(
    queries = list(
      list(method = 'census', mode = 'batch'),
      list(method = 'census', mode = 'single'),
      list(method = 'osm')
    ),
    global_params = list(street = 'street_address',
      city = 'city', state = 'state', postalcode = 'zip_cd'),
    query_names = c('census batch', 'census single', 'osm')
  )

more_addresses %>%
  geocode_combine(
    queries = list(
      list(method = 'census', mode = 'batch', street = 'street_address',
        city = 'city', state = 'state', postalcode = 'zip_cd'),
      list(method = 'arcgis', address = 'street_address')
    ),
    cascade = FALSE,
    return_list = TRUE
  )
```

---

`geo_census`*Convenience functions for calling geo()*

---

### Description

The method for `geo()` is specified in the function name.

#### **[Deprecated]**

Use the `geo` function directly instead.

### Usage

`geo_census(...)``geo_osm(...)``geo_geocodio(...)``geo_iq(...)``geo_google(...)``geo_opencage(...)``geo_mapbox(...)``geo_here(...)``geo_tomtom(...)``geo_mapquest(...)``geo_bing(...)``geo_arcgis(...)``geo_cascade(...)`

### Arguments

`...` arguments to be passed to the `geo` function



---

 geo\_combine

 Combine multiple geocoding queries
 

---

### Description

Passes address inputs in character vector form to the [geocode\\_combine](#) function for geocoding.

Note that address inputs must be specified for queries either with the `queries` parameter (for each query) or the `global_params` parameter (for all queries). For example `global_params = list(address = 'address')` passes addresses provided in the `address` parameter to all queries.

### Usage

```
geo_combine(
  queries,
  global_params = list(),
  address = NULL,
  street = NULL,
  city = NULL,
  county = NULL,
  state = NULL,
  postalcode = NULL,
  country = NULL,
  lat = lat,
  long = long,
  ...
)
```

### Arguments

|                            |  |
|----------------------------|--|
| <code>queries</code>       | a list of queries, each provided as a list of parameters. The queries are executed by the <a href="#">geocode</a> function in the order provided. (ex. <code>list(list(method = 'osm'), list(method = 'census'), ...)</code> )   |
| <code>global_params</code> | a list of parameters to be used for all queries (ex. <code>list(address = 'address', full_results = TRUE)</code> )   |
| <code>address</code>       | single line address (ie. '1600 Pennsylvania Ave NW, Washington, DC'). Do not combine with the address component arguments below ( <code>street</code> , <code>city</code> , <code>county</code> , <code>state</code> , <code>postalcode</code> , <code>country</code> ). |
| <code>street</code>        | street address (ie. '1600 Pennsylvania Ave NW')  |
| <code>city</code>          | city (ie. 'Tokyo')   |
| <code>county</code>        | county (ie. 'Jefferson')   |
| <code>state</code>         | state (ie. 'Kentucky')   |
| <code>postalcode</code>    | postalcode (ie. zip code if in the United States)  |
| <code>country</code>       | country (ie. 'Japan')  |
| <code>lat</code>           | latitude column name. Can be quoted or unquoted (ie. <code>lat</code> or <code>"lat"</code> ).   |

long longitude column name. Can be quoted or unquoted (ie. long or "long").  
 ... arguments passed to the [geocode\\_combine](#) function

**Value**

tibble (dataframe)

**See Also**

[geocode\\_combine](#) [geo](#) [geocode](#)

**Examples**

```
options(tidygeocoder.progress_bar = FALSE)
example_addresses <- c("100 Main St New York, NY", "Paris", "Not a Real Address")

geo_combine(
  queries = list(
    list(method = 'census'),
    list(method = 'osm')
  ),
  address = example_addresses,
  global_params = list(address = 'address')
)

geo_combine(
  queries = list(
    list(method = 'arcgis'),
    list(method = 'census', mode = 'single'),
    list(method = 'census', mode = 'batch')
  ),
  global_params = list(address = 'address'),
  address = example_addresses,
  cascade = FALSE,
  return_list = TRUE
)

geo_combine(
  queries = list(
    list(method = 'arcgis', address = 'city'),
    list(method = 'osm', city = 'city', country = 'country')
  ),
  city = c('Tokyo', 'New York'),
  country = c('Japan', 'United States'),
  cascade = FALSE
)
```

---

|               |                                       |
|---------------|---------------------------------------|
| get_api_query | <i>Construct a geocoder API query</i> |
|---------------|---------------------------------------|

---

### Description

The geocoder API query is created using universal "generic" parameters and optional api-specific "custom" parameters. Generic parameters are converted into api parameters using the [api\\_parameter\\_reference](#) dataset.

The [query\\_api](#) function executes the queries created by this function.

### Usage

```
get_api_query(method, generic_parameters = list(), custom_parameters = list())
```

### Arguments

|                    |                                |
|--------------------|--------------------------------|
| method             | method name (ie. 'census')     |
| generic_parameters | universal 'generic' parameters |
| custom_parameters  | custom api-specific parameters |

### Value

API parameters as a named list

### See Also

[query\\_api](#) [api\\_parameter\\_reference](#) [geo](#) [reverse\\_geo](#)

### Examples

```
get_api_query("osm", list(address = 'Hanoi, Vietnam'))

get_api_query("census", list(street = '11 Wall St', city = "NY", state = 'NY'),
  list(benchmark = "Public_AR_Census2010"))
```

---

|            |  |
|------------|--|
| louisville | <i>Louisville, Kentucky street addresses</i> |
|------------|--|

---

**Description**

Louisville, Kentucky street addresses

**Usage**

louisville

**Format**

A tibble dataframe with component street addresses

**street** Description of the address

**city** Single line address

**state** state

**zip** zip code

**Source**

Downloaded from [OpenAddresses.io](https://openaddresses.io) on June 1st 2020

---

|                    |  |
|--------------------|--|
| min_time_reference | <i>Minimum time required per query</i> |
|--------------------|--|

---

**Description**

The [geo](#) and [reverse\\_geo](#) functions use this dataset to set the maximum query rate for each geocoding service. This rate is based on the usage restriction policies for each geocoding service.

**Usage**

min\_time\_reference

**Format**

A tibble dataframe

**method** Geocoding service name

**min\_time** The minimum number of seconds required per query to comply with usage restrictions

**description** A description of the usage rate restriction

## Details

Links to the usage policies of each geocoding service are below:

- [Nominatim](#)
- [US Census](#)
- [ArcGIS](#)
- [Geocodio](#)
- [Location IQ](#)
- [Google](#)
- [OpenCage](#)
- [Mapbox](#)
- [HERE](#)
- [TomTom](#)
- [MapQuest](#)
- [Bing](#)
- [Geoapify](#)

## See Also

[geo reverse\\_geo](#)

---

query\_api

*Execute a geocoder API query*

---

## Description

The [get\\_api\\_query](#) function can create queries for this function to execute.

## Usage

```
query_api(  
  api_url,  
  query_parameters,  
  mode = "single",  
  batch_file = NULL,  
  input_list = NULL,  
  content_encoding = "UTF-8",  
  timeout = 20,  
  method = ""  
)
```

**Arguments**

|                  |  |
|------------------|--|
| api_url          | Base URL of the API. query parameters are appended to this   |
| query_parameters | api query parameters in the form of a named list   |
| mode             | determines the type of query to execute <ul style="list-style-type: none"> <li>- "single": geocode a single input (all methods)</li> <li>- "list": batch geocode a list of inputs (ex. geocodio)</li> <li>- "file": batch geocode a file of inputs (ex. census)</li> </ul> |
| batch_file       | a csv file of input data to upload (for mode = 'file')   |
| input_list       | a list of input data (for mode = 'list')   |
| content_encoding | Encoding to be used for parsing content  |
| timeout          | timeout in minutes   |
| method           | if 'mapquest' or 'arcgis' then the query status code is changed appropriately  |

**Value**

a named list containing the response content (content) and the HTTP request status (status)

**See Also**

[get\\_api\\_query](#) [extract\\_results](#) [extract\\_reverse\\_results](#) [geo](#) [reverse\\_geo](#)

**Examples**

```
raw1 <- query_api("http://nominatim.openstreetmap.org/search",
  get_api_query("osm", list(address = 'Hanoi, Vietnam')))

raw1$status

extract_results('osm', jsonlite::fromJSON(raw1$content))

raw2 <- query_api("http://nominatim.openstreetmap.org/reverse",
  get_api_query("osm", custom_parameters = list(lat = 38.895865, lon = -77.0307713)))

extract_reverse_results('osm', jsonlite::fromJSON(raw2$content))
```

---

`reverse_geo`*Reverse geocode coordinates*

---

## Description

Reverse geocodes geographic coordinates (latitude and longitude) given as numeric values. Latitude and longitude inputs are limited to possible values. Latitudes must be between -90 and 90 and longitudes must be between -180 and 180. Invalid values will not be sent to the geocoding service. The [reverse\\_geocode](#) function utilizes this function on coordinates contained in dataframes. See example usage in `vignette("tidygeocoder")`.

Refer to [api\\_parameter\\_reference](#), [min\\_time\\_reference](#), and [batch\\_limit\\_reference](#) for more details on geocoding service parameters and usage.

This function uses the [get\\_api\\_query](#), [query\\_api](#), and [extract\\_reverse\\_results](#) functions to create, execute, and parse geocoder API queries.

## Usage

```
reverse_geo(  
  lat,  
  long,  
  method = "osm",  
  address = "address",  
  limit = 1,  
  full_results = FALSE,  
  mode = "",  
  unique_only = FALSE,  
  return_coords = TRUE,  
  min_time = NULL,  
  progress_bar = show_progress_bar(),  
  quiet = getOption("tidygeocoder.quiet", FALSE),  
  api_url = NULL,  
  timeout = 20,  
  flatten = TRUE,  
  batch_limit = NULL,  
  verbose = getOption("tidygeocoder.verbose", FALSE),  
  no_query = FALSE,  
  custom_query = list(),  
  api_options = list(),  
  iq_region = "us",  
  geocodio_v = 1.6,  
  mapbox_permanent = FALSE,  
  here_request_id = NULL,  
  mapquest_open = FALSE  
)
```

**Arguments**

|              |   |
|--------------|---|
| lat          | latitude values (input data)  |
| long         | longitude values (input data)   |
| method       | <p>the geocoding service to be used. API keys are loaded from environmental variables. Run <code>usethis::edit_r_environ()</code> to open your <code>.Renviron</code> file and add an API key as an environmental variable. For example, add the line <code>GEOCODIO_API_KEY="YourAPIKeyHere"</code></p> <ul style="list-style-type: none"> <li>• "osm": <b>Nominatim</b>.</li> <li>• "arcgis": <b>ArcGIS</b>.</li> <li>• "geocodio": <b>Geocodio</b>. Geographic coverage is limited to the United States and Canada. An API key must be stored in the environmental variable "GEOCODIO_API_KEY". Batch geocoding is supported.</li> <li>• "iq": <b>Location IQ</b>. An API key must be stored in the environmental variable "LOCATIONIQ_API_KEY".</li> <li>• "google": <b>Google</b>. An API key must be stored in the environmental variable "GOOGLEGEOCODE_API_KEY".</li> <li>• "opencage": <b>OpenCage</b>. An API key must be stored in the environmental variable "OPENCAGE_KEY".</li> <li>• "mapbox": <b>Mapbox</b>. An API key must be stored in the environmental variable "MAPBOX_API_KEY".</li> <li>• "here": <b>HERE</b>. An API key must be stored in the environmental variable "HERE_API_KEY". Batch geocoding is supported, but must be explicitly called with <code>mode = "batch"</code>.</li> <li>• "tomtom": <b>TomTom</b>. An API key must be stored in the environmental variable "TOMTOM_API_KEY". Batch geocoding is supported.</li> <li>• "mapquest": <b>MapQuest</b>. An API key must be stored in the environmental variable "MAPQUEST_API_KEY". Batch geocoding is supported.</li> <li>• "bing": <b>Bing</b>. An API key must be stored in the environmental variable "BINGMAPS_API_KEY". Batch geocoding is supported, but must be explicitly called with <code>mode = "batch"</code>.</li> <li>• "geoapify": <b>Geoapify</b>. An API key must be stored in the environmental variable "GEOAPIFY_KEY".</li> </ul> |
| address      | name of the address column (in the output data)   |
| limit        | <p>maximum number of results to return per input coordinate. For many geocoding services the maximum value of the limit parameter is 100. Pass <code>limit = NULL</code> to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if <code>return_coords = TRUE</code>. Refer to <a href="#">api_parameter_reference</a> for more details.</p>   |
| full_results | returns all available data from the geocoding service if <code>TRUE</code> . If <code>FALSE</code> (default) then only a single address column is returned from the geocoding service.  |
| mode         | <p>set to 'batch' to force batch geocoding or 'single' to force single coordinate geocoding (one coordinate per query). If not specified then batch geocoding will be used if available (given method selected) when multiple coordinates are provided; otherwise single address geocoding will be used. For the "here" and "bing" methods the batch mode should be explicitly specified with <code>mode = 'batch'</code>.</p>  |



|               |  |
|---------------|--|
| unique_only   | only return results for unique inputs if TRUE  |
| return_coords | return input coordinates with results if TRUE. Note that most services return the input coordinates with <code>full_results = TRUE</code> and setting <code>return_coords</code> to FALSE does not prevent this.   |
| min_time      | minimum amount of time for a query to take (in seconds). If NULL then <code>min_time</code> will be set to the default value specified in <a href="#">min_time_reference</a> .   |
| progress_bar  | if TRUE then a progress bar will be displayed for single input geocoding (1 input per query). By default the progress bar will not be shown for code executed when knitting R Markdown files or code within an RStudio notebook chunk. Can be set permanently with <code>options(tidygeocoder.progress_bar = FALSE)</code> .   |
| quiet         | if TRUE then console messages that are displayed by default regarding queries will be suppressed. FALSE is default. Can be set permanently with <code>options(tidygeocoder.quiet = TRUE)</code> .  |
| api_url       | custom API URL. If specified, the default API URL will be overridden. This parameter can be used to specify a local Nominatim server, for instance.  |
| timeout       | query timeout (in minutes)   |
| flatten       | if TRUE (default) then any nested dataframes in results are flattened if possible. Note that in some cases results are flattened regardless such as for Geocodio batch geocoding.  |
| batch_limit   | limit to the number of coordinates in a batch geocoding query. Defaults to the value in <a href="#">batch_limit_reference</a> if not specified.  |
| verbose       | if TRUE then detailed logs are output to the console. FALSE is default. Can be set permanently with <code>options(tidygeocoder.verbose = TRUE)</code>  |
| no_query      | if TRUE then no queries are sent to the geocoding service and <code>verbose</code> is set to TRUE. Used for testing.   |
| custom_query  | API-specific parameters to be used, passed as a named list (ex. <code>list(extratags = 1)</code> ).  |
| api_options   | a named list of parameters specific to individual services. (ex. <code>list(geocodio_v = 1.6, geocodio_hipaa = TRUE)</code> ). Each parameter begins with the name of the method (service) it applies to. The possible parameters are shown below with their default values. <ul style="list-style-type: none"> <li>• <code>census_return_type</code> (default: "locations"): set to "geographies" to return additional geography columns. Make sure to use <code>full_results = TRUE</code> if using the "geographies" setting.</li> <li>• <code>iq_region</code> (default: "us"): set to "eu" to use the European Union API endpoint</li> <li>• <code>geocodio_v</code> (default: 1.6): the version number of the Geocodio API to be used</li> <li>• <code>geocodio_hipaa</code> (default: FALSE): set to TRUE to use the HIPAA compliant Geocodio API endpoint</li> <li>• <code>mapbox_permanent</code> (default: FALSE): set to TRUE to use the <code>mapbox.places-permanent</code> endpoint. Note that this option should be used only if you have applied for a permanent account. Unsuccessful requests made by an account that does not have access to the endpoint may be billable.</li> </ul> |

- `mapbox_open` (default: `FALSE`): set to `TRUE` to use the Open Geocoding endpoint which relies solely on OpenStreetMap data
- `here_request_id` (default: `NULL`): this parameter would return a previous HERE batch job, identified by its RequestID. The RequestID of a batch job is displayed when `verbose` is `TRUE`. Note that this option would ignore the current address parameter on the request, so the `return_addresses` or `return_coords` parameters need to be `FALSE`.

`iq_region`        **[Deprecated]** use the `api_options` parameter instead  
`geocodio_v`       **[Deprecated]** use the `api_options` parameter instead  
`mapbox_permanent` **[Deprecated]** use the `api_options` parameter instead  
`here_request_id` **[Deprecated]** use the `api_options` parameter instead  
`mapquest_open`   **[Deprecated]** use the `api_options` parameter instead

### Value

tibble (dataframe)

### See Also

[reverse\\_geocode](#) [api\\_parameter\\_reference](#) [min\\_time\\_reference](#) [batch\\_limit\\_reference](#)

### Examples

```
options(tidygeocoder.progress_bar = FALSE)

reverse_geo(lat = 38.895865, long = -77.0307713, method = 'osm')

reverse_geo(
  lat = c(38.895865, 43.6534817, 300),
  long = c(-77.0307713, -79.3839347, 600),
  method = 'osm', full_results = TRUE
)
```

---

reverse\_geocode

*Reverse geocode coordinates in a dataframe*

---

### Description

Takes a dataframe containing coordinates (latitude and longitude) and returns the reverse geocoding query results from a specified service by using the [reverse\\_geo](#) function. See example usage in `vignette("tidygeocoder")`.

This function passes all additional parameters (`...`) to the [reverse\\_geo](#) function, so you can refer to its documentation for more details on possible arguments.

**Usage**

```
reverse_geocode(
  .tbl,
  lat,
  long,
  address = "address",
  return_input = TRUE,
  limit = 1,
  return_coords = NULL,
  unique_only = FALSE,
  ...
)
```

**Arguments**

|                            |   |
|----------------------------|---|
| <code>.tbl</code>          | dataframe containing coordinates  |
| <code>lat</code>           | latitude column name (input data). Can be quoted or unquoted (ie. <code>lat</code> or <code>'lat'</code> ).   |
| <code>long</code>          | longitude column name (input data). Can be quoted or unquoted (ie. <code>long</code> or <code>'long'</code> ).  |
| <code>address</code>       | address column name (output data). Can be quoted or unquoted (ie. <code>addr</code> or <code>'addr'</code> ).   |
| <code>return_input</code>  | if TRUE then the input dataset will be combined with the geocoder query results and returned. If FALSE only the geocoder results will be returned.  |
| <code>limit</code>         | maximum number of results to return per input coordinate. For many geocoding services the maximum value of the limit parameter is 100. Pass <code>limit = NULL</code> to use the default limit value of the selected geocoding service. For batch geocoding, limit must be set to 1 (default) if <code>return_coords = TRUE</code> . To use <code>limit &gt; 1</code> or <code>limit = NULL</code> set <code>return_input</code> to FALSE. Refer to <a href="#">api_parameter_reference</a> for more details. |
| <code>return_coords</code> | if TRUE return input coordinates. Defaults to TRUE if <code>return_input</code> is FALSE and FALSE if <code>return_input</code> is TRUE. This argument is passed to the <code>reverse_geo()</code> function.  |
| <code>unique_only</code>   | if TRUE then only unique results will be returned and <code>return_input</code> will be set to FALSE.   |
| <code>...</code>           | arguments passed to the <a href="#">reverse_geo</a> function  |

**Value**

tibble (dataframe)

**See Also**

[reverse\\_geo](#)

## Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

tibble(
  latitude = c(38.895865, 43.6534817),
  longitude = c(-77.0307713, -79.3839347)
) %>%
reverse_geocode(
  lat = latitude,
  long = longitude,
  method = 'osm',
  full_results = TRUE
)

louisville %>% head(3) %>%
reverse_geocode(lat = latitude, long = longitude,
method = 'arcgis')

louisville %>% head(2) %>%
reverse_geocode(lat = latitude, long = longitude,
method = 'osm',
limit = 2, return_input = FALSE)
```

---

sample\_addresses

*Sample addresses for testing*

---

## Description

Sample addresses for testing

## Usage

```
sample_addresses
```

## Format

A tibble dataframe with single line addresses

**name** Description of the address

**addr** Single line address

# Index

## \* datasets

- api\_info\_reference, 2
- api\_key\_reference, 3
- api\_parameter\_reference, 3
- batch\_limit\_reference, 5
- louisville, 20
- min\_time\_reference, 20
- sample\_addresses, 28

- api\_info\_reference, 2
- api\_key\_reference, 3
- api\_parameter\_reference, 3, 7, 9, 11, 13, 19, 23, 24, 26, 27

- batch\_limit\_reference, 4, 5, 7, 10, 11, 23, 25, 26

- extract\_results, 5, 7, 22
- extract\_reverse\_results, 6, 22, 23

- geo, 3–6, 7, 12, 13, 15, 16, 18–22
- geo\_arcgis (geo\_census), 16
- geo\_bing (geo\_census), 16
- geo\_cascade (geo\_census), 16
- geo\_census, 16
- geo\_combine, 9, 14, 15, 17
- geo\_geocodio (geo\_census), 16
- geo\_google (geo\_census), 16
- geo\_here (geo\_census), 16
- geo\_iq (geo\_census), 16
- geo\_mapbox (geo\_census), 16
- geo\_mapquest (geo\_census), 16
- geo\_opencage (geo\_census), 16
- geo\_osm (geo\_census), 16
- geo\_tomtom (geo\_census), 16
- geocode, 7, 11, 12, 14, 15, 17, 18
- geocode\_combine, 9, 14, 17, 18
- get\_api\_query, 4, 6, 7, 19, 21–23

- louisville, 20

- min\_time\_reference, 4, 7, 9, 11, 20, 23, 25, 26

- query\_api, 4–7, 19, 21, 23

- reverse\_geo, 3–5, 7, 19–22, 23, 26, 27
- reverse\_geocode, 23, 26, 26

- sample\_addresses, 28