Package ‘nominatimlite’

May 11, 2023

Type Package

Title Interface with 'Nominatim' API Service

Version 0.2.0

Description Lite interface for getting data from 'OSM' service

‘Nominatim’ <https://nominatim.org/release-docs/latest/>. Extract coordinates from addresses, find places near a set of coordinates, search for amenities and return spatial objects on 'sf' format.

License MIT + file LICENSE


BugReports https://github.com/dieghernan/nominatimlite/issues

Depends R (>= 3.6.0)

Imports dplyr (>= 1.0.0), jsonlite (>= 1.7.0), sf (>= 0.9.0), utils

Suggests ggplot2 (>= 3.0.0), knitr, osmdata, rmarkdown, testthat (>= 3.0.0), tidygeocoder

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Copyright Data © OpenStreetMap contributors, ODbL 1.0. <https://www.openstreetmap.org/copyright>

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

X-schema.org-applicationCategory cartography

X-schema.org-keywords r, geocoding, openstreetmap, address, nominatim, reverse-geocoding, rstats, shapefile, r-package, spatial, cran, api-wrapper

NeedsCompilation no
bbox_to_poly

Create a bounding box sf object

Description

Create a sf polygon object from the coordinates of a bounding box

Usage

bbox_to_poly(bbox = NA, xmin = NA, ymin = NA, xmax = NA, ymax = NA, crs = 4326)

Arguments

bbox  numeric vector of 4 elements representing the coordinates of the bounding box. Values should be c(xmin, ymin, xmax, ymax)

xmin, ymin, xmax, ymax alternatively, you can use these named parameters instead of bbox

crs  coordinate reference system, something suitable as input to st_crs

Details

Bounding boxes can be located using different online tools, as Bounding Box Tool.
**Value**

A sfc object of class POLYGON.

**See Also**

`sfc::st_as_sfc()`

Get spatial ('sf') objects: `geo_address_lookup_sf()`, `geo_amenity_sf()`, `geo lite sf()`, `reverse geo lite sf()`

Search amenities: `geo_amenity_sf()`, `geo_amenity()`, `osm amenities`

**Examples**

```r
# bounding box of Germany
bbox_GER <- c(5.86631529, 47.27011137, 15.04193189, 55.09916098)

bbox_GER_sf <- bbox_to_poly(bbox_GER)

library(ggplot2)

ggplot(bbox_GER_sf) +
  geom_sf()

# Extract the bounding box of a sf object
sfobj <- geo lite sf("seychelles", points_only = FALSE)

dsfc

bbox <- sf::st_bbox(sfobj)

bbox

bbox_sfobj <- bbox_to_poly(bbox)

ggplot(bbox_sfobj) +
  geom_sf(fill = "lightblue", alpha = 0.5) +
  geom_sf(data = sfobj, fill = "wheat")
```

---

**Description**

The lookup API allows to query the address and other details of one or multiple OSM objects like node, way or relation. This function returns the tibble associated with the query, see `geo_address_lookup_sf()` for retrieving the data as a spatial object (sf format).
Usage

geo_address_lookup(
osm_ids,
type = c("N", "W", "R"),
lst = "lat",
long = "lon",
full_results = FALSE,
return_addresses = TRUE,
verbose = FALSE,
custom_query = list()
)

Arguments

osm_ids vector of OSM identifiers as **numeric** (c(00000, 11111, 22222)).
type vector character of the type of the OSM type associated to each osm_ids. Possible values are node ("N"), way ("W") or relation ("R"). If a single value is provided it would be recycled.
lst latitude column name in the output data (default "lat").
long longitude column name in the output data (default "long").
full_results returns all available data from the API service. If FALSE (default) only latitude, longitude and address columns are returned. See also return_addresses.
return_addresses return input addresses with results if TRUE.
verbose if TRUE then detailed logs are output to the console.
custom_query A named list with API-specific parameters to be used (i.e. list(countrycodes = "US")). See Details.

Details

See https://nominatim.org/release-docs/develop/api/Lookup/ for additional parameters to be passed to custom_query.

Value

A tibble with the results found by the query.

See Also

*geo_address_lookup_sf()*

Address Lookup API: *geo_address_lookup_sf()*

Geocoding strings: *geo_address_lookup_sf(), geo_amenity_sf(), geo_amenity(), geo_lite_sf(), geo_lite()*
Examples

```r
ts <- geo_address_lookup(osm_ids = c(46240148, 34633854), type = "W")
ts

several <- geo_address_lookup(c(146656, 240109189), type = c("R", "N"))
several
```

---

**geo_address_lookup_sf**  
*Address Lookup API for OSM objects in Spatial Format*

---

**Description**

The lookup API allows to query the address and other details of one or multiple OSM objects like node, way or relation. This function returns the sf spatial object associated with the query, see `geo_address_lookup()` for retrieving the data in tibble format.

**Usage**

```r
go_address_lookup_sf(
  osm_ids, 
  type = c("N", "W", "R"), 
  full_results = FALSE, 
  return_addresses = TRUE, 
  verbose = FALSE, 
  custom_query = list(), 
  points_only = TRUE
)
```

**Arguments**

- `osm_ids` vector of OSM identifiers as numeric (`c(00000, 11111, 22222)`).
- `type` vector character of the type of the OSM type associated to each osm_ids. Possible values are node ("N"), way ("W") or relation ("R"). If a single value is provided it would be recycled.
- `full_results` returns all available data from the API service. If FALSE (default) only address columns are returned. See also return_addresses.
- `return_addresses` return input addresses with results if TRUE.
- `verbose` if TRUE then detailed logs are output to the console.
- `custom_query` A named list with API-specific parameters to be used (i.e. `list(countrycodes = "US")`). See Details.
**points_only** Logical TRUE/FALSE. Whether to return only spatial points (TRUE, which is the default) or potentially other shapes as provided by the Nominatim API (FALSE). See **About Geometry Types**.

**Details**

See https://nominatim.org/release-docs/latest/api/Lookup/ for additional parameters to be passed to custom_query.

**Value**

A sf object with the results.

**About Geometry Types**

The parameter points_only specifies whether the function results will be points (all Nominatim results are guaranteed to have at least point geometry) or possibly other spatial objects.

Note that the type of geometry returned in case of points_only = FALSE will depend on the object being geocoded:

- administrative areas, major buildings and the like will be returned as polygons
- rivers, roads and their like as lines
- amenities may be points even in case of a points_only = FALSE call

The function is vectorized, allowing for multiple addresses to be geocoded; in case of points_only = FALSE multiple geometry types may be returned.

**See Also**

geo_address_lookup()

Address Lookup API: geo_address_lookup()

Geocoding strings: geo_address_lookup(), geo_amenity_sf(), geo_amenity(), geo_lite_sf().

geo_lite()

Get spatial (`sf`) objects: bbox_to_poly(), geo_amenity_sf(), geo_lite_sf(), reverse_geo_lite_sf()

**Examples**

```
# Notre Dame Cathedral, Paris
NotreDame <- geo_address_lookup_sf(osm_ids = 201611261, type = "W")
library(ggplot2)

ggplot(NotreDame) +
  geom_sf()

NotreDame_poly <- geo_address_lookup_sf(201611261,
```

geo_amenity

```r

type = "W",
points_only = FALSE
)

ggplot(NotreDame_poly) +
  geom_sf()

# It is vectorized

several <- geo_address_lookup_sf(c(146656, 240109189), type = c("R", "N"))
several
```

---

### Description

This function search amenities as defined by OpenStreetMap on a restricted area defined by a bounding box in the form of \((\text{<min_latitude>}, \text{<min_longitude>}, \text{<max_latitude>}, \text{<max_longitude>})\). This function returns the `tibble` associated with the query, see `geo_amenity_sf()` for retrieving the data as a spatial object (sf format).

### Usage

```r
geo_amenity(
  bbox,
  amenity,
  lat = "lat",
  long = "lon",
  limit = 1,
  full_results = FALSE,
  return_addresses = TRUE,
  verbose = FALSE,
  custom_query = list(),
  strict = FALSE
)
```

### Arguments

- **bbox**: A numeric vector of latitude and longitude \((\text{<min_latitude>}, \text{<min_longitude>}, \text{<max_latitude>}, \text{<max_longitude>})\) that restrict the search area. See **Details**.
- **amenity**: A character of a vector of character with the amenities to be geolocated (i.e. `c("pub", "restaurant")`). See **Details** and `osm_amenities`.
- **lat**: latitude column name in the output data (default "lat").
- **long**: longitude column name in the output data (default "long").
limit  maximum number of results to return per input address. Note that each query returns a maximum of 50 results.
full_results  returns all available data from the API service. If FALSE (default) only latitude, longitude and address columns are returned. See also return_addresses.
return_addresses  return input addresses with results if TRUE.
verbose  if TRUE then detailed logs are output to the console.
custom_query  API-specific parameters to be used. See geo_lite().
strict  Logical TRUE/FALSE. Force the results to be included inside the bbox. Note that Nominatim default behavior may return results located outside the provided bounding box.

Details
Bounding boxes can be located using different online tools, as Bounding Box Tool.
For a full list of valid amenities see https://wiki.openstreetmap.org/wiki/Key:amenity.

Value
A tibble with the results.

See Also
geo_amenity_sf()

Search amenities: bbox_to_poly(), geo_amenity_sf(), osm_amenities
Geocoding strings: geo_address_lookup_sf(), geo_address_lookup(), geo_amenity_sf(),
geo_lite_sf(), geo_lite()

Examples

# Times Square, NY, USA
bbox <- c(-73.9894467311, 40.75573629, -73.9830630737, 40.75789245)
geo_amenity(bbox = bbox, amenity = "restaurant")

# Several amenities
geo_amenity(bbox = bbox, amenity = c("restaurant", "pub"))

# Increase limit and use with strict
geo_amenity(
  bbox = bbox, amenity = c("restaurant", "pub"), limit = 10,
  strict = TRUE
)
Description

This function searches amenities as defined by OpenStreetMap on a restricted area defined by a bounding box in the form of \((\text{<min_latitude>, <min_longitude>, <max_latitude>, <max_longitude>})\). This function returns the \texttt{sf} spatial object associated with the query, see \texttt{geo_amenity()} for retrieving the data in tibble format.

Usage

\begin{verbatim}
geo_amenity_sf(
    bbox, amenity,
    limit = 1,
    full_results = FALSE,
    return_addresses = TRUE,
    verbose = FALSE,
    custom_query = list(),
    points_only = TRUE,
    strict = FALSE
)
\end{verbatim}

Arguments

- \texttt{bbox} A numeric vector of latitude and longitude \((\text{<min_latitude>, <min_longitude>, <max_latitude>, <max_longitude>})\) that restrict the search area. See Details.
- \texttt{amenity} A character of a vector of character with the amenities to be geolocated (i.e. \texttt{c("pub", "restaurant")}). See Details and \texttt{osm_amenities}.
- \texttt{limit} maximum number of results to return per input address. Note that each query returns a maximum of 50 results.
- \texttt{full_results} returns all available data from the API service. If FALSE (default) only address columns are returned. See also \texttt{return_addresses}.
- \texttt{return_addresses} return input addresses with results if TRUE.
- \texttt{verbose} if TRUE then detailed logs are output to the console.
- \texttt{custom_query} A named list with API-specific parameters to be used (i.e. \texttt{list(countrycodes = "US")}). See Details.
- \texttt{points_only} Logical TRUE/FALSE. Whether to return only spatial points (TRUE, which is the default) or potentially other shapes as provided by the Nominatim API (FALSE). See About Geometry Types.
- \texttt{strict} Logical TRUE/FALSE. Force the results to be included inside the bbox. Note that Nominatim default behavior may return results located outside the provided bounding box.
Details

Bounding boxes can be located using different online tools, as Bounding Box Tool.
For a full list of valid amenities see https://wiki.openstreetmap.org/wiki/Key:amenity.

Value

A sf object with the results.

About Geometry Types

The parameter points_only specifies whether the function results will be points (all Nominatim results are guaranteed to have at least point geometry) or possibly other spatial objects.

Note that the type of geometry returned in case of points_only = FALSE will depend on the object being geocoded:

• administrative areas, major buildings and the like will be returned as polygons
• rivers, roads and their like as lines
• amenities may be points even in case of a points_only = FALSE call

The function is vectorized, allowing for multiple addresses to be geocoded; in case of points_only = FALSE multiple geometry types may be returned.

See Also

geo_amenity()
Search amenities: bbox_to_poly(), geo_amenity(), osm_amenities
Geocoding strings: geo_address_lookup_sf(), geo_address_lookup(), geo_amenity(), geo_lite_sf(), geo_lite()
Get spatial ('sf') objects: bbox_to_poly(), geo_address_lookup_sf(), geo_lite_sf(), reverse_geo_lite_sf()

Examples

# Madrid, Spain
library(ggplot2)
bbox <- c(-3.888954, 40.311977, -3.517916, 40.643729)
# Restaurants and pubs
rest_pub <- geo_amenity_sf(bbox, c("restaurant", "pub"), limit = 50)
ggplot(rest_pub) +
  geom_sf()

# Hospital as polygon
hosp <- geo_amenity_sf(bbox, "hospital", points_only = FALSE)

ggplot(hosp) +
geom_sf()

---

**Description**

Geocodes addresses given as character values. This function returns the tibble associated with the query, see `geo_lite_sf()` for retrieving the data as a spatial object (sf format).

**Usage**

```r
geo_lite(
  address,
  lat = "lat",
  long = "lon",
  limit = 1,
  full_results = FALSE,
  return_addresses = TRUE,
  verbose = FALSE,
  custom_query = list()
)
```

**Arguments**

- `address` character with single line address ("1600 Pennsylvania Ave NW, Washington") or a vector of addresses (c("Madrid", "Barcelona").
- `lat` latitude column name in the output data (default "lat").
- `long` longitude column name in the output data (default "long").
- `limit` maximum number of results to return per input address. Note that each query returns a maximum of 50 results.
- `full_results` returns all available data from the API service. If FALSE (default) only latitude, longitude and address columns are returned. See also `return_addresses`.
- `return_addresses` return input addresses with results if TRUE.
- `verbose` if TRUE then detailed logs are output to the console.
- `custom_query` A named list with API-specific parameters to be used (i.e. list(countrycodes = "US")). See Details.
geo_lite_sf

Description

This function allows you to geocode addresses and return the corresponding spatial object. This function returns the sf spatial object associated with the query, see geo_lite_sf() for retrieving the data in tibble format.

Usage

geo_lite_sf(
  address,
  limit = 1,
  return_addresses = TRUE,
  full_results = FALSE,
  verbose = FALSE,
)
geo_lite_sf

    custom_query = list(),
    points_only = TRUE
)

Arguments

address character with single line address ("1600 Pennsylvania Ave NW, Washington")
or a vector of addresses (c("Madrid", "Barcelona").

limit maximum number of results to return per input address. Note that each query
returns a maximum of 50 results.

return_addresses return input addresses with results if TRUE.

full_results returns all available data from the API service. If FALSE (default) only address
columns are returned. See also return_addresses.

verbose if TRUE then detailed logs are output to the console.

custom_query A named list with API-specific parameters to be used (i.e. list(countrycodes
    = "US")). See Details.

points_only Logical TRUE/FALSE. Whether to return only spatial points (TRUE, which is the
default) or potentially other shapes as provided by the Nominatim API (FALSE).
See About Geometry Types.

Details

See https://nominatim.org/release-docs/latest/api/Search/ for additional parameters to
be passed to custom_query.

Value

A sf object with the results.

About Geometry Types

The parameter points_only specifies whether the function results will be points (all Nominatim
results are guaranteed to have at least point geometry) or possibly other spatial objects.
Note that the type of geometry returned in case of points_only = FALSE will depend on the object
being geocoded:

• administrative areas, major buildings and the like will be returned as polygons
• rivers, roads and their like as lines
• amenities may be points even in case of a points_only = FALSE call

The function is vectorized, allowing for multiple addresses to be geocoded; in case of points_only
= FALSE multiple geometry types may be returned.

See Also

Geocoding strings: geo_address_lookup_sf(), geo_address_lookup(), geo_amenity_sf(),
geo_amenity(), geo_lite()

Get spatial ("sf") objects: bbox_to_poly(), geo_address_lookup_sf(), geo_amenity_sf(), reverse_geo_lite_sf()
Examples

# Map - Points
library(ggplot2)

string <- "Statue of Liberty, NY, USA"
sol <- geo_lite_sf(string)

ggplot(sol) +
  geom_sf()

sol_poly <- geo_lite_sf(string, points_only = FALSE)

ggplot(sol_poly) +
  geom_sf() +
  geom_sf(data = sol, color = "red")

# Several results
Madrid <- geo-lite_sf("Madrid",
  limit = 2,
  points_only = FALSE, full_results = TRUE)

ggplot(Madrid) +
  geom_sf(fill = NA)

Starbucks <- geo-lite_sf("Starbucks, New York",
  limit = 20, full_results = TRUE)

ggplot(Starbucks) +
  geom_sf()

---

osm_amenities  

OpenStreetMap amenity database

Description

Database with the list of amenities available on OpenStreetMap.

Format

A tibble with with 100 rows and fields:
**osm_amenities**

**category**  The category of the amenity  
**amenity**  The name of the amenity  

**Details**

<table>
<thead>
<tr>
<th>category</th>
<th>amenity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sustenance</td>
<td>bar</td>
</tr>
<tr>
<td>Sustenance</td>
<td>biergarten</td>
</tr>
<tr>
<td>Sustenance</td>
<td>cafe</td>
</tr>
<tr>
<td>Sustenance</td>
<td>fast_food</td>
</tr>
<tr>
<td>Sustenance</td>
<td>food_court</td>
</tr>
<tr>
<td>Sustenance</td>
<td>ice_cream</td>
</tr>
<tr>
<td>Sustenance</td>
<td>pub</td>
</tr>
<tr>
<td>Sustenance</td>
<td>restaurant</td>
</tr>
<tr>
<td>Education</td>
<td>college</td>
</tr>
<tr>
<td>Education</td>
<td>driving_school</td>
</tr>
<tr>
<td>Education</td>
<td>kindergarten</td>
</tr>
<tr>
<td>Education</td>
<td>language_school</td>
</tr>
<tr>
<td>Education</td>
<td>library</td>
</tr>
<tr>
<td>Education</td>
<td>toy_library</td>
</tr>
<tr>
<td>Education</td>
<td>music_school</td>
</tr>
<tr>
<td>Education</td>
<td>school</td>
</tr>
<tr>
<td>Education</td>
<td>university</td>
</tr>
<tr>
<td>Transportation</td>
<td>bicycle_parking</td>
</tr>
<tr>
<td>Transportation</td>
<td>bicycle_repair_station</td>
</tr>
<tr>
<td>Transportation</td>
<td>bicycle_rental</td>
</tr>
<tr>
<td>Transportation</td>
<td>boat_rental</td>
</tr>
<tr>
<td>Transportation</td>
<td>boat_sharing</td>
</tr>
<tr>
<td>Transportation</td>
<td>bus_station</td>
</tr>
<tr>
<td>Transportation</td>
<td>car_rental</td>
</tr>
<tr>
<td>Transportation</td>
<td>car_sharing</td>
</tr>
<tr>
<td>Transportation</td>
<td>car_wash</td>
</tr>
<tr>
<td>Transportation</td>
<td>vehicle_inspection</td>
</tr>
<tr>
<td>Transportation</td>
<td>charging_station</td>
</tr>
<tr>
<td>Transportation</td>
<td>ferry_terminal</td>
</tr>
<tr>
<td>Transportation</td>
<td>fuel</td>
</tr>
<tr>
<td>Transportation</td>
<td>grit_bin</td>
</tr>
<tr>
<td>Transportation</td>
<td>motorcycle_parking</td>
</tr>
<tr>
<td>Transportation</td>
<td>parking</td>
</tr>
<tr>
<td>Transportation</td>
<td>parking_entrance</td>
</tr>
<tr>
<td>Transportation</td>
<td>parking_space</td>
</tr>
<tr>
<td>Transportation</td>
<td>taxi</td>
</tr>
<tr>
<td>Financial</td>
<td>atm</td>
</tr>
<tr>
<td>Financial</td>
<td>bank</td>
</tr>
<tr>
<td>Financial</td>
<td>bureau_de_change</td>
</tr>
<tr>
<td>Healthcare</td>
<td>baby_hatch</td>
</tr>
<tr>
<td>Healthcare</td>
<td>clinic</td>
</tr>
<tr>
<td>Healthcare</td>
<td>dentist</td>
</tr>
<tr>
<td>Category</td>
<td>Subcategory</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Healthcare</td>
<td>doctors</td>
</tr>
<tr>
<td>Healthcare</td>
<td>hospital</td>
</tr>
<tr>
<td>Healthcare</td>
<td>nursing_home</td>
</tr>
<tr>
<td>Healthcare</td>
<td>pharmacy</td>
</tr>
<tr>
<td>Healthcare</td>
<td>social_facility</td>
</tr>
<tr>
<td>Healthcare</td>
<td>veterinary</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>arts_centre</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>brothel</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>casino</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>cinema</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>community_centre</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>conference_centre</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>events_venue</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>fountain</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>gambling</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>love_hotel</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>nightclub</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>planetarium</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>public_bookcase</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>social_centre</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>stripclub</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>studio</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>swingerclub</td>
</tr>
<tr>
<td>Entertainment-Arts-Culture</td>
<td>theatre</td>
</tr>
<tr>
<td>Public Service</td>
<td>courthouse</td>
</tr>
<tr>
<td>Public Service</td>
<td>embassy</td>
</tr>
<tr>
<td>Public Service</td>
<td>fire_station</td>
</tr>
<tr>
<td>Public Service</td>
<td>police</td>
</tr>
<tr>
<td>Public Service</td>
<td>post_box</td>
</tr>
<tr>
<td>Public Service</td>
<td>post_depot</td>
</tr>
<tr>
<td>Public Service</td>
<td>post_office</td>
</tr>
<tr>
<td>Public Service</td>
<td>prison</td>
</tr>
<tr>
<td>Public Service</td>
<td>ranger_station</td>
</tr>
<tr>
<td>Public Service</td>
<td>townhall</td>
</tr>
<tr>
<td>Facilities</td>
<td>bbq</td>
</tr>
<tr>
<td>Facilities</td>
<td>bench</td>
</tr>
<tr>
<td>Facilities</td>
<td>dog_toilet</td>
</tr>
<tr>
<td>Facilities</td>
<td>drinking_water</td>
</tr>
<tr>
<td>Facilities</td>
<td>give_box</td>
</tr>
<tr>
<td>Facilities</td>
<td>shelter</td>
</tr>
<tr>
<td>Facilities</td>
<td>shower</td>
</tr>
<tr>
<td>Facilities</td>
<td>telephone</td>
</tr>
<tr>
<td>Facilities</td>
<td>toilets</td>
</tr>
<tr>
<td>Facilities</td>
<td>water_point</td>
</tr>
<tr>
<td>Facilities</td>
<td>watering_place</td>
</tr>
<tr>
<td>Waste Management</td>
<td>sanitary_dump_station</td>
</tr>
<tr>
<td>Waste Management</td>
<td>recycling</td>
</tr>
<tr>
<td>Waste Management</td>
<td>waste_basket</td>
</tr>
</tbody>
</table>
### reverse_geo_lite

<table>
<thead>
<tr>
<th>Waste Management</th>
<th>waste_disposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waste Management</td>
<td>waste_transfer_station</td>
</tr>
<tr>
<td>Others</td>
<td>animal_boarding</td>
</tr>
<tr>
<td>Others</td>
<td>animal_breeding</td>
</tr>
<tr>
<td>Others</td>
<td>animal_shelter</td>
</tr>
<tr>
<td>Others</td>
<td>baking_oven</td>
</tr>
<tr>
<td>Others</td>
<td>childcare</td>
</tr>
<tr>
<td>Others</td>
<td>clock</td>
</tr>
<tr>
<td>Others</td>
<td>crematorium</td>
</tr>
<tr>
<td>Others</td>
<td>dive_centre</td>
</tr>
</tbody>
</table>

**Note**

Data extracted on **14 June 2021**.

**Source**

https://wiki.openstreetmap.org/wiki/Key:amenity

**See Also**

Search amenities: `bbox_to_poly()`, `geo_amenity_sf()`, `geo_amenity()`

**Examples**

```r
amenities <- nominatimlite::osm_amenities

amenities
```

---

**reverse_geo_lite**  
*Reverse Geocoding API for OSM objects*

**Description**

Generates an address from a latitude and longitude. Latitudes must be between \([-90, 90]\) and longitudes between \([-180, 180]\). This function returns the tibble associated with the query, see `reverse_geo_lite_sf()` for retrieving the data as a spatial object (sf format).
reverse_geolite

Usage

reverse_geolite(
  lat,
  long,
  address = "address",
  full_results = FALSE,
  return_coords = TRUE,
  verbose = FALSE,
  custom_query = list()
)

Arguments

lat        latitude values in numeric format. Must be in the range [-90, 90].
long       longitude values in numeric format. Must be in the range [-180, 180].
address    address column name in the output data (default "address").
full_results returns all available data from the API service. If FALSE (default) only latitude,
longitude and address columns are returned. See also return_addresses.
return_coords return input coordinates with results if TRUE.
verbose     if TRUE then detailed logs are output to the console.
custom_query API-specific parameters to be used, passed as a named list (ie. list(zoom = 3)).

Details

See https://nominatim.org/release-docs/develop/api/Reverse/ for additional parameters
to be passed to custom_query.

Value

A tibble with the results.

About Zooming

Use the option custom_query = list(zoom = 3) to adjust the output. Some equivalences on terms
of zoom:

<table>
<thead>
<tr>
<th>zoom</th>
<th>address_detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>country</td>
</tr>
<tr>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>8</td>
<td>county</td>
</tr>
<tr>
<td>10</td>
<td>city</td>
</tr>
<tr>
<td>14</td>
<td>suburb</td>
</tr>
<tr>
<td>16</td>
<td>major streets</td>
</tr>
<tr>
<td>17</td>
<td>major and minor streets</td>
</tr>
<tr>
<td>18</td>
<td>building</td>
</tr>
</tbody>
</table>
**reverse_geo_lite_sf**

**See Also**

reverse_geo_lite_sf(), tidygeocoder::reverse_geo()

Reverse geocoding coordinates: reverse_geo_lite_sf()

**Examples**

reverse_geo_lite(lat = 40.75728, long = -73.98586)

# Several coordinates
reverse_geo_lite(lat = c(40.75728, 55.95335), long = c(-73.98586, -3.188375))

# With options: zoom to country level
sev <- reverse_geo_lite(
  lat = c(40.75728, 55.95335), long = c(-73.98586, -3.188375),
  custom_query = list(zoom = 0, extratags = 1),
  verbose = TRUE, full_results = TRUE
)

dplyr::glimpse(sev)

---

**reverse_geo_lite_sf**  *Reverse Geocoding API for OSM objects in Spatial format*

**Description**

Generates an address from a latitude and longitude. Latitudes must be between \([-90, 90]\) and longitudes between \([-180, 180]\). This function returns the sf spatial object associated with the query, see reverse_geo_lite() for retrieving the data in tibble format.

**Usage**

reverse_geo_lite_sf(
  lat,
  long,
  address = "address",
  full_results = FALSE,
  return_coords = TRUE,
  verbose = FALSE,
  custom_query = list(),
  points_only = TRUE
)
Arguments

lat  latitude values in numeric format. Must be in the range \([-\theta, \theta]\).
long longitude values in numeric format. Must be in the range \([-180, 180]\).
address address column name in the output data (default "address").
full_results returns all available data from the API service. If FALSE (default) only latitude, longitude and address columns are returned. See also return_addresses.
return_coords return input coordinates with results if TRUE.
verbose if TRUE then detailed logs are output to the console.
custom_query API-specific parameters to be used, passed as a named list (ie. `list(zoom = 3)`). See Details.
points_only Logical TRUE/FALSE. Whether to return only spatial points (TRUE, which is the default) or potentially other shapes as provided by the Nominatim API (FALSE). See About Geometry Types.

Details

See https://nominatim.org/release-docs/develop/api/Reverse/ for additional parameters to be passed to custom_query.

Value

A sf object with the results.

About Zooming

Use the option custom_query = `list(zoom = 3)` to adjust the output. Some equivalences on terms of zoom:

<table>
<thead>
<tr>
<th>zoom</th>
<th>address_detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>country</td>
</tr>
<tr>
<td>5</td>
<td>state</td>
</tr>
<tr>
<td>8</td>
<td>county</td>
</tr>
<tr>
<td>10</td>
<td>city</td>
</tr>
<tr>
<td>14</td>
<td>suburb</td>
</tr>
<tr>
<td>16</td>
<td>major streets</td>
</tr>
<tr>
<td>17</td>
<td>major and minor streets</td>
</tr>
<tr>
<td>18</td>
<td>building</td>
</tr>
</tbody>
</table>

About Geometry Types

The parameter points_only specifies whether the function results will be points (all Nominatim results are guaranteed to have at least point geometry) or possibly other spatial objects.

Note that the type of geometry returned in case of points_only = FALSE will depend on the object being geocoded:

- administrative areas, major buildings and the like will be returned as polygons
reverse_geo_lite_sf

- rivers, roads and their like as lines
- amenities may be points even in case of a points_only = FALSE call

The function is vectorized, allowing for multiple addresses to be geocoded; in case of points_only = FALSE multiple geometry types may be returned.

See Also

reverse_geo_lite()
Reverse geocoding coordinates: reverse_geo_lite()
Get spatial ('sf') objects: bbox_to_poly(), geo_address_lookup_sf(), geo_amenity_sf(), geo_lite_sf()

Examples

library(ggplot2)

Coliseum <- geo_lite("Coliseo, Rome, Italy")

# Coliseum
Col_sf <- reverse_geo_lite_sf(
  lat = Coliseum$lat,
  lon = Coliseum$lon,
  points_only = FALSE
)

ggplot(Col_sf) +
  geom_sf()

# City of Rome - Zoom 10
Rome_sf <- reverse_geo_lite_sf(
  lat = Coliseum$lat,
  lon = Coliseum$lon,
  custom_query = list(zoom = 10),
  points_only = FALSE
)

ggplot(Rome_sf) +
  geom_sf()

# County - Zoom 8
County_sf <- reverse_geo_lite_sf(
  lat = Coliseum$lat,
  lon = Coliseum$lon,
  custom_query = list(zoom = 8),
  points_only = FALSE
)
```r
ggplot(County_sf) +
  geom_sf()
```
Index

* amenity
  bbox_to_poly, 2
  geo_amenity, 7
  geo_amenity_sf, 9
  osm_amenities, 14
* datasets
  osm_amenities, 14
* geocoding
  geo_address_lookup, 3
  geo_address_lookup_sf, 5
  geo_amenity, 7
  geo_amenity_sf, 9
  geo_lite, 11
  geo_lite_sf, 12
* lookup
  geo_address_lookup, 3
  geo_address_lookup_sf, 5
* reverse
  reverse_geo_lite, 17
  reverse_geo_lite_sf, 19
* spatial
  bbox_to_poly, 2
  geo_address_lookup_sf, 5
  geo_amenity_sf, 9
  geo_lite_sf, 12
  reverse_geo_lite_sf, 19
bbox_to_poly, 2, 6, 8, 10, 13, 17, 21
geo_address_lookup, 3, 6, 8, 10, 12, 13
geo_address_lookup(), 5, 6
geo_address_lookup_sf, 3, 4, 5, 8, 10, 12, 13, 21
geo_address_lookup_sf(), 3, 4
geo_amenity, 3, 4, 6, 7, 10, 12, 13, 17
geo_amenity(), 9, 10
geo_amenity_sf, 3, 4, 6, 8, 9, 12, 13, 17, 21
geo_amenity_sf(), 7, 8
geo_lite, 4, 6, 8, 10, 11, 13
geo_lite(), 8
geo_lite_sf, 3, 4, 6, 8, 10, 12, 12, 21
geo_lite_sf(), 11, 12
osm_amenities, 3, 7–10, 14
reverse_geo_lite, 17, 21
reverse_geo_lite(), 19, 21
reverse_geo_lite_sf, 3, 6, 10, 13, 19, 19
reverse_geo_lite_sf(), 17, 19
sf::st_as_sfc(), 3
st_crs, 2
tidygeocoder::geo(), 12
tidygeocoder::reverse_geo(), 19