Package: photon (via r-universe)

November 12, 2024

Type Package

Title High-Performance Geocoding using 'photon'

Version 0.3.1.9000

Description Features unstructured, structured and reverse geocoding
using the 'photon' geocoding API <https://photon.komoot.io/>.
Facilitates the setup of local 'photon' instances to enable
offline geocoding.

License Apache License (>= 2)

URL <https://github.com/jslth/photon/>, <https://jslth.github.io/photon/>

BugReports <https://github.com/jslth/photon/issues>

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Imports utils, cli, countrycode, httr2, R6, sf, processx

Suggests testthat (>= 3.0.0), tibble, knitr, rmarkdown, webfakes, ps

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libssl-dev
libproj-dev libsqlite3-dev libudunits2-dev

Repository https://jslth.r-universe.dev

RemoteUrl https://github.com/jslth/photon

RemoteRef HEAD

RemoteSha 64dc9ce31a2d3cdcaf92cf7f6702198590a799f9

Contents

---

cmd_options   *Format command line options*

Description

Helper function to format options for command line calls. The function accepts key-value pairs
where the parameter name is the name of the option and the parameter value is the value of the
option. Arguments are formatted according to the following rules:

- If a value is TRUE, add parameter name as flag.

- If a value is FALSE, do not add parameter name as flag.

- If a value has length(x) > 1, collapse it as a CSV.

- If a parameter name is missing, take the value as the flag name.

- If a parameter name is given, replace underscores with hyphens.

Usage

```
cmd_options(..., use_double_hyphens = FALSE)
```

Arguments

...   Key-value pairs of command line options.

use_double_hyphens

                    If TRUE, uses double hyphens to designate non-abbreviated command line op-
                    tions and single-hyphens to designate abbreviated ones. If FALSE, always uses
                    single hyphens. Defaults to FALSE as both Java and photon use single hyphens.

Value

A character vector of formatted command line options that can be used as input to system2 or run.

Examples

```
# converts R parameters to CMD options
# parameters for the ping command
cmd_options(n = 1, w = 5, "127.0.0.1")

# sometimes, it is necessary to use double hyphens
# options for the docker ps command
cmd_options("ps", all = TRUE, format = "json", use_double_hyphens = TRUE)

# particularly useful together with photon
# the following options can be used for the `photon_opts` argument
# of photon$start()
cmd_options(cors_any = TRUE, data_dir = "path/to/dir")
```

---

download_photon   *Download photon*

Description

Download the photon executable from GitHub.

Usage

```
download_photon(
  path = ".",
  version = NULL,
  opensearch = FALSE,
  only_url = FALSE,
  quiet = FALSE
)
```

Arguments

path	Path to a directory to store the executable.
version	Version tag of the photon release. If NULL, downloads the latest known version.
opensearch	If TRUE, downloads the OpenSearch version of photon if available. OpenSearch versions are available for photon >= 0.6.0.
only_url	If TRUE, performs a download. Otherwise, only returns a link to the file.
quiet	If TRUE, suppresses all informative messages.

Value

If only_url = FALSE, returns a character string giving the path to the downloaded file. Otherwise, returns the URL to be downloaded.

Examples

```
download_photon(tempdir(), version = "0.4.1")
```

---

download_searchindex   *Download search index*

Description

Finds and downloads the Elasticsearch index database necessary to set up Photon locally.

Usage

```
download_searchindex(
  path = ".",
  country = "Monaco",
  date = "latest",
  exact = FALSE,
  section = NULL,
  only_url = FALSE,
  quiet = FALSE
)
```

Arguments

path	Path to a directory where the identified file should be stored.
country	Character string that can be identified by [countryname](#) as a country. An extract for this country will be downloaded. If "planet", downloads a global search index.
date	Character string or date-time object used to specify the creation date of the search index. If "latest", will download the file tagged with "latest". If a character string, the value should be parseable by [as.POSIXct](#). If exact = FALSE, the input value is compared to all available dates and the closest date will be selected. Otherwise, a file will be selected that exactly matches the input to date.
exact	If TRUE, exactly matches the date. Otherwise, selects the date with lowest difference to the date parameter.
section	Subdirectory of the download server from which to select a search index. If "experimental", selects a dump made for the master version of photon. If "archived", selects a dump made for an older version of photon. If NULL, selects a dump made for the current release. Defaults to NULL.
only_url	If TRUE, performs a download. Otherwise, only returns a link to the file.
quiet	If TRUE, suppresses all informative messages.

Value

If only_url = FALSE, returns the local path to the downloaded file. Otherwise, returns the URL to the remote file.

Note

Depending on the country, search index databases tend to be very large. The global search index is about 75 GB of size (10/2024). Keep that in mind when running this function.

Examples

```
# download the latest extract of Monaco
download_searchindex(path = tempdir())

# download the latest extract of American Samoa
download_searchindex(path = tempdir(), country = "Samoa")

# download an extract from a month ago
download_searchindex(
  path = tempdir(),
  country = "Monaco",
  date = Sys.time() - 2629800
 )

# if possible, download an extract from today
try(download_searchindex(
  path = tempdir(),
  country = "Monaco",
  date = Sys.Date(),
  exact = TRUE
))

# get the latest global coverage
# NOTE: the file to be downloaded is several tens of gigabytes of size!
## Not run:
download_searchindex(path = tempdir(), country = "planet")
## End(Not run)
```

---

geocode   *Unstructured geocoding*

Description

Geocode arbitrary text strings. Unstructured geocoding is more flexible but generally less accurate than structured geocoding.

Usage

```
geocode(
  texts,
  limit = 3,
  lang = "en",
  bbox = NULL,
  osm_tag = NULL,
```

```
    layer = NULL,
    locbias = NULL,
    locbias_scale = NULL,
    zoom = NULL,
    progress = interactive()
)
```

Arguments

texts	Character vector of a texts to geocode.
limit	Number of results to return. Defaults to 3.
lang	Language of the results.
bbox	Any object that can be parsed by [st_bbox](#). Results must lie within this bbox.
osm_tag	Character string giving an [OSM tag](#) to filter the results by. See details.
layer	Character string giving a layer to filter the results by. Can be one of ″house″, ″street″, ″locality″, ″district″, ″city″, ″county″, ″state″, ″country″, or ″other″.
locbias	Numeric vector of length 2 or any object that can be coerced to a length-2 numeric vector (e.g. a list or sfg object). Specifies a location bias for geocoding in the format c(lon, lat). Geocoding results are biased towards this point. The radius of the bias is controlled through zoom and the weight of place prominence through location_bias_scale.
locbias_scale	Numeric vector specifying the importance of prominence in locbias. A higher prominence scale gives more weight to important places. Defaults to 0.2.
zoom	Numeric specifying the radius for which the locbias is effective. Corresponds to the zoom level in OpenStreetMap. The exact relation to locbias is $0.25 \text{ km} \cdot 2^{(18-\text{zoom})}$. Defaults to 16.
progress	If TRUE, shows a progress bar for longer queries.

Details

Filtering by OpenStreetMap tags follows a distinct syntax explained on https://github.com/komoot/photon. In particular:

- Include places with tag: key:value

- Exclude places with tag: !key:value

- Include places with tag key: key

- Include places with tag value: :value

- Exclude places with tag key: !key

- Exclude places with tag value: :!value

Value

An sf dataframe or tibble containing the following columns:

- idx: Internal ID specifying the index of the texts parameter.
- osm_type: Type of OSM element, one of N (node), W (way), R (relation), or P (polygon).
- osm_id: OpenStreetMap ID of the matched element.
- country: Country of the matched place.
- city: City of the matched place.
- osm_key: OpenStreetMap key.
- countrycode: ISO2 country code.
- housenumber: House number, if applicable.
- postcode: Post code, if applicable.
- locality: Locality, if applicable.
- street: Street, if applicable.
- district: District name, if applicable.
- osm_value: OpenStreetMap tag value.
- name: Place name.
- type: Layer type as described for the layer parameter.
- extent: Boundary box of the match.

Examples

```
# an instance must be mounted first
photon <- new_photon()

# geocode a city
geocode("Berlin")

# return more results
geocode("Berlin", limit = 10)

# return the results in german
geocode("Berlin", limit = 10, lang = "de")

# limit to cities
geocode("Berlin", layer = "city")

# limit to European cities
geocode("Berlin", bbox = c(xmin = -71.18, ymin = 44.46, xmax = 13.39, ymax = 52.52))

# search for museums in berlin
geocode("Berlin", osm_tag = "tourism:museum")

# search for touristic attractions in berlin
geocode("Berlin", osm_tag = "tourism")
```

```
# search for anything but tourism
geocode("Berlin", osm_tag = "!tourism")

# use location biases to match Berlin, IL instead of Berlin, DE
geocode("Berlin", locbias = c(-100, 40), locbias_scale = 0.1, zoom = 7, osm_tag = "place")
```

---

get_instance   *Photon utilities*

Description

Utilities to manage photon instances. These functions operate on mounted photon instances which
can be initialized using new_photon.

- get_instance() retrieves the active photon instance.

- get_photon_url() retrieves the photon URL to send requests.

Usage

```
get_instance()

get_photon_url()
```

Value

get_instance returns a R6 object of class photon. get_photon_url() returns a URL string.

Examples

```
# make a new photon instance
new_photon()

# retrieve it from the cache
get_instance()

# get the server url
get_photon_url()
```

---

has_java *Is Java installed?*

Description

Utility function to check if Java is installed and if it has the right version.

Usage

```
has_java(version = NULL)
```

Arguments

version   Character string specifying the minimum version of Java. If the installed Java version is lower than this, returns FALSE. If NULL, only checks if any kind of Java is installed on the system.

Value

A logical vector of length 1.

Examples

```
has_java() # Is Java installed?
has_java("11") # Is Java > 11 installed?
```

---

new_photon *Initialize a photon instance*

Description

Initialize a photon instance by creating a new photon object. This object is stored in the R session and can be used to perform geocoding requests.

Instances can either local or remote. Remote instances require nothing more than a URL that geocoding requests are sent to. Local instances require the setup of the photon executable, a search index, and Java. See [photon_local](#) for details.

Usage

```
new_photon(
  path = NULL,
  url = NULL,
  photon_version = NULL,
  country = NULL,
  date = "latest",
  exact = FALSE,
```

```
    section = NULL,
    opensearch = FALSE,
    overwrite = FALSE,
    quiet = FALSE
)
```

Arguments

path	Path to a directory where the photon executable and data should be stored. Defaults to a directory "photon" in the current working directory. If NULL, a remote instance is set up based on the url parameter.
url	URL of a photon server to connect to. If NULL and path is also NULL, connects to the public API under https://photon.komoot.io/.
photon_version	Version of photon to be used. A list of all releases can be found here: https://github.com/komoot/photon/releases/. Ignored if jar is given. If NULL, uses the latest known version.
country	Character string that can be identified by countryname as a country. An extract for this country will be downloaded. If NULL, downloads a global search index.
date	Character string or date-time object used to specify the creation date of the search index. If "latest", will download the file tagged with "latest". If a character string, the value should be parseable by as.POSIXct. If exact = FALSE, the input value is compared to all available dates and the closest date will be selected. Otherwise, a file will be selected that exactly matches the input to date.
exact	If TRUE, exactly matches the date. Otherwise, selects the date with lowest difference to the date parameter.
section	Subdirectory of the download server from which to select a search index. If "experimental", selects a dump made for the master version of photon. If "archived", selects a dump made for an older version of photon. If NULL (or any arbitrary string), selects a dump made for the current release. Defaults to NULL.
opensearch	If TRUE, looks for an OpenSearch version of photon in the specified path. Opensearch-based photon supports structured geocoding queries but is currently only experimental. Defaults to FALSE. See vignette("nominatim-import", package = "photon") for details.
overwrite	If TRUE, overwrites existing jar files and search indices when initializing a new instance. Defaults to FALSE.
quiet	If TRUE, suppresses all informative messages.

Value

An R6 object of class photon.

Examples

```
# connect to public API
photon <- new_photon()
```

```
# connect to arbitrary server
photon <- new_photon(url = "photonserver.org")

if (has_java("11")) {
# set up a local instance in the current working directory
photon <- new_photon("photon", country = "Monaco")
}
```

---

photon_local   *Local photon instance*

Description

This R6 class is used to initialize and manage local photon instances. It can download and setup the Java, the photon executable, and the necessary ElasticSearch search index. It can start, stop, and query the status of the photon instance. It is also the basis for geocoding requests at it is used to retrieve the URL for geocoding.

ElasticSearch / OpenSearch

The standard version of photon uses ElasticSearch indices to geocode. These search indices can be self-provided by importing an existing Nominatim database or they can be downloaded from the [Photon download server](#). Use nominatim = TRUE to indicate that no ElasticSearch indices should be downloaded. See vignette("nominatim-import", package = "photon") for details on how to import from Nominatim.

To enable structured geocoding, the photon geocoder needs to be built to support OpenSearch. Since photon 0.6.0, OpenSearch jar files are included in the photon releases. OpenSearch indices can also be downloaded, but do not support structured geocoding as of yet. To enable structured geocoding, indices have to be imported from an existing Nominatim database.

Super class

[photon::photon](#) -> photon_local

Public fields

path Path to the directory where the photon instance is stored.

proc [process](#) object that handles the external process running photon.

Methods

Public methods:

- [photon_local$new()](#)
- [photon_local$mount()](#)
- [photon_local$info()](#)
- [photon_local$purge()](#)

- photon_local$import()
- photon_local$start()
- photon_local$stop()
- photon_local$download_data()
- photon_local$remove_data()
- photon_local$is_running()
- photon_local$is_ready()
- photon_local$get_url()
- photon_local$get_logs()
- photon_local$clone()

Method new(): Initialize a local photon instance. If necessary, downloads the photon executable, the search index, and Java.

Usage:
```
photon_local$new(
  path,
  photon_version = NULL,
  country = NULL,
  date = "latest",
  exact = FALSE,
  section = NULL,
  opensearch = FALSE,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments:

path Path to a directory where the photon executable and data should be stored. Defaults to a directory "photon" in the current working directory.

photon_version Version of photon to be used. A list of all releases can be found here: https://github.com/komoot/photon/releases/. Ignored if jar is given. If NULL, uses the latest known version.

country Character string that can be identified by countryname as a country. An extract for this country will be downloaded. If NULL, downloads a global search index.

date Character string or date-time object used to specify the creation date of the search index. If "latest", will download the file tagged with "latest". If a character string, the value should be parseable by as.POSIXct. If exact = FALSE, the input value is compared to all available dates and the closest date will be selected. Otherwise, a file will be selected that exactly matches the input to date.

exact If TRUE, exactly matches the date. Otherwise, selects the date with lowest difference to the date parameter.

section Subdirectory of the download server from which to select a search index. If "experimental", selects a dump made for the master version of photon. If "archived", selects a dump made for an older version of photon. If NULL (or any arbitrary string), selects a dump made for the current release. Defaults to NULL.

opensearch If TRUE, looks for an OpenSearch version of photon in the specified path. Opensearch-based photon supports structured geocoding queries but has to be built manually using gradle. Hence, it cannot be downloaded directly. If no OpenSearch executable is found in the search path, then this parameter is set to FALSE. Defaults to FALSE.

overwrite If TRUE, overwrites existing jar files and search indices when initializing a new instance. Defaults to FALSE.

quiet If TRUE, suppresses all informative messages.

Method mount(): Attach the object to the session. If mounted, all geocoding functions send their requests to the URL of this instance. Manually mounting is useful if you want to switch between multiple photon instances.

Usage:
```
photon_local$mount()
```

Method info(): Retrieve metadata about the java and photon version used as well as the country and creation date of the Eleasticsearch search index.

Usage:
```
photon_local$info()
```

Returns: A list containing the java version, the photon version, and if applicable, the spatial and temporal coverage of the search index.

Method purge(): Kill the photon process and remove the directory. Useful to get rid of an instance entirely.

Usage:
```
photon_local$purge(ask = TRUE)
```

Arguments:

ask If TRUE, asks for confirmation before purging the instance.

Returns: NULL, invisibly.

Method import(): Import a Postgres Nominatim database to photon. Runs the photon jar file using the additional parameter -nominatim-import. Requires a running Nominatim database that can be connected to.

Usage:
```
photon_local$import(
  host = "127.0.0.1",
  port = 5432,
  database = "nominatim",
  user = "nominatim",
  password = "",
  structured = FALSE,
  update = FALSE,
  enable_update_api = FALSE,
  languages = c("en", "fr", "de", "it"),
  countries = NULL,
  extra_tags = NULL,
  json = FALSE,
```

```
  timeout = 60,
  java_opts = NULL,
  photon_opts = NULL
)
```

Arguments:

host Postgres host of the database. Defaults to `"127.0.0.1"`.

port Postgres port of the database. Defaults to `5432`.

database Postgres database name. Defaults to `"nominatim"`.

user Postgres database user. Defaults to `"nominatim"`.

password Postgres database password. Defaults to `""`.

structured If `TRUE`, enables structured query support when importing the database. This allows the usage of [`structured`](). Structured queries are only supported in the OpenSearch version of photon. See section "OpenSearch" above. Defaults to `FALSE`.

update If `TRUE`, fetches updates from the Nominatim database, updating the search index without offering an API. If `FALSE`, imports the database an deletes the previous index. Defaults to `FALSE`.

enable_update_api If `TRUE`, enables an additional endpoint `/nominatim-update`, which allows updates from Nominatim databases.

languages Character vector specifying the languages to import from the Nominatim databases. Defaults to English, French, German, and Italian.

countries Character vector specifying the country codes to import from the Nominatim database. Defaults to all country codes.

extra_tags Character vector specifying extra OSM tags to import from the Nominatim database. These tags are used to augment geocoding results. Defaults to `NULL`.

json If `TRUE`, dumps the imported Nominatim database to a JSON file and returns the path to the output file. Defaults to `FALSE`.

timeout Time in seconds before the java process aborts. Defaults to 60 seconds.

java_opts List of further flags passed on to the `java` command.

photon_opts List of further flags passed on to the photon jar in the java command. See [`cmd_options`]() for a helper function to import external Nominatim databases.

Method `start()`: Start a local instance of the Photon geocoder. Runs the jar executable located in the instance directory.

Usage:

```
photon_local$start(
  host = "0.0.0.0",
  port = "2322",
  ssl = FALSE,
  timeout = 60,
  java_opts = NULL,
  photon_opts = NULL
)
```

Arguments:

host Character string of the host name that the geocoder should be opened on.

port Port that the geocoder should listen to.

ssl If TRUE, uses https, otherwise http. Defaults to FALSE.

timeout Time in seconds before the java process aborts. Defaults to 60 seconds.

java_opts List of further flags passed on to the java command.

photon_opts List of further flags passed on to the photon jar in the java command. See cmd_options for a helper function to import external Nominatim databases.

Details: While there is a certain way to determine if a photon instance is ready, there is no clear way as of yet to determine if a photon setup has failed. Due to this, a failing setup is mostly indicated by the setup hanging after emitting a warning. In this case, the setup has to be interrupted manually.

Method stop(): Kills the running photon process.

Usage:
photon_local$stop()

Method download_data(): Downloads a search index using download_searchindex.

Usage:
```
photon_local$download_data(
  country = NULL,
  date = "latest",
  exact = FALSE,
  section = NULL
)
```
Arguments:

country Character string that can be identified by countryname as a country. An extract for this country will be downloaded. If NULL, downloads a global search index.

date Character string or date-time object used to specify the creation date of the search index. If "latest", will download the file tagged with "latest". If a character string, the value should be parseable by as.POSIXct. If exact = FALSE, the input value is compared to all available dates and the closest date will be selected. Otherwise, a file will be selected that exactly matches the input to date.

exact If TRUE, exactly matches the date. Otherwise, selects the date with lowest difference to the date parameter.

section Subdirectory of the download server from which to select a search index. If "experimental", selects a dump made for the master version of photon. If "archived", selects a dump made for an older version of photon. If NULL (or any arbitrary string), selects a dump made for the current release. Defaults to NULL.

Method remove_data(): Removes the data currently used in the photon directory. This only affects the unpacked photon_data directory, not archived files.

Usage:
photon_local$remove_data()

Method is_running(): Checks whether the photon instance is running and ready. The difference to $is_ready() is that $is_running() checks specifically if the running photon instance is managed by a process from its own photon object. In other words, $is_running() returns TRUE if both $proc$is_alive() and $is_ready() return TRUE. This method is useful if you want to ensure that the photon object can control its photon server (mostly internal use).

Usage:

```
photon_local$is_running()
```

Returns: A logical of length 1.

Method `is_ready()`: Checks whether the photon instance is ready to take requests. This is the case if the photon server returns a HTTP 400 when sending a queryless request. This method is useful if you want to check whether you can send requests.

Usage:

```
photon_local$is_ready()
```

Returns: A logical of length 1.

Method `get_url()`: Constructs the URL that geocoding requests should be sent to.

Usage:

```
photon_local$get_url()
```

Returns: A URL to send requests to.

Method `get_logs()`: Retrieve the logs of previous photon runs.

Usage:

```
photon_local$get_logs()
```

Returns: Returns a dataframe containing the run ID (`rid`, the highest number is the most recent run), a timestamp (`ts`), the thread, the log type (INFO, WARN, or ERROR), the class trace and the error message.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
photon_local$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
if (has_java("11")) {
dir <- file.path(tempdir(), "photon")

# start a new instance using a Monaco extract
photon <- new_photon(path = dir, country = "Monaco")

# start a new instance with an older photon version
photon <- new_photon(path = dir, photon_version = "0.4.1")}

## Not run:
# import a nominatim database using OpenSearch photon
# this example requires the OpenSearch version of photon and a running
# Nominatim server.
photon <- new_photon(path = dir, opensearch = TRUE)
photon$start(photon_options = cmd_options(port = 29146, password = "pgpass"))
## End(Not run)
```

---

purge_java   *Purge Java processes*

Description

Kill all or selected running Java processes. This function is useful to stop Photon instances when not being able to kill the [process](#) objects. Be aware that you can also kill Java processes other than the photon application using this function!

Usage

```
purge_java(pids = NULL, consent = FALSE)
```

Arguments

pids	PIDs to kill. The PIDs should be Java processes. If NULL, tries to kill all Java processes.
consent	If FALSE, asks for consent before killing the processes. Defaults to FALSE.

Details

A list of running Java tasks is retrieved using ps (on Linux and MacOS) or tasklist (on Windows). Tasks are killed using pkill (on Linux and MacOS) or Taskkill (on Windows).

Value

An integer vector of the pkill / Taskkill status codes or NULL if not running Java processes are found.

Examples

```
# NOTE: These examples should only be run interactively or when you are
# sure that no other java processes are running simultaneously!
## Not run:
purge_java() # does nothing if no java processes are running

# start a new photon instance
dir <- file.path(tempdir(), "photon")
photon <- new_photon(dir, country = "Samoa")
photon$start()

# kill photon using a sledgehammer
purge_java()

photon$start()

# kill photon using a scalpel
library(ps)
p <- ps_handle(photon$proc$get_pid())
```

```
pids <- sapply(ps_children(p), ps::ps_pid)
purge_java(pids)
## End(Not run)
```

---

reverse   *Reverse geocoding*

Description

Reverse geocode a set of points to retrieve their corresponding place names. To geocode a place
name or an address, see unstructured or structured geocoding.

Usage

```
reverse(
  .data,
  radius = NULL,
  limit = 3,
  lang = "en",
  osm_tag = NULL,
  layer = NULL,
  locbias = NULL,
  locbias_scale = NULL,
  zoom = NULL,
  distance_sort = TRUE,
  progress = interactive()
)
```

Arguments

.data	A dataframe or list with names lon and lat, or an sfc or sf object containing point geometries.
radius	Numeric specifying the range around the points in .data that is used for searching.
limit	Number of results to return. Defaults to 3.
lang	Language of the results.
osm_tag	Character string giving an OSM tag to filter the results by. See details.
layer	Character string giving a layer to filter the results by. Can be one of "house", "street", "locality", "district", "city", "county", "state", "country", or "other".
locbias	Numeric vector of length 2 or any object that can be coerced to a length-2 numeric vector (e.g. a list or sfg object). Specifies a location bias for geocoding in the format c(lon, lat). Geocoding results are biased towards this point. The radius of the bias is controlled through zoom and the weight of place prominence through location_bias_scale.

locbias_scale	Numeric vector specifying the importance of prominence in locbias. A higher prominence scale gives more weight to important places. Defaults to 0.2.
zoom	Numeric specifying the radius for which the locbias is effective. Corresponds to the zoom level in OpenStreetMap. The exact relation to locbias is $0.25 \text{ km} \cdot 2^{(18-\text{zoom})}$. Defaults to 16.
distance_sort	If TRUE, sorts the reverse geocoding results based on the distance to the input point. Defaults to TRUE.
progress	If TRUE, shows a progress bar for longer queries.

Details

Filtering by OpenStreetMap tags follows a distinct syntax explained on https://github.com/komoot/photon. In particular:

- Include places with tag: key:value
- Exclude places with tag: !key:value
- Include places with tag key: key
- Include places with tag value: :value
- Exclude places with tag key: !key
- Exclude places with tag value: :!value

Value

An sf dataframe or tibble containing the following columns:

- idx: Internal ID specifying the index of the texts parameter.
- osm_type: Type of OSM element, one of N (node), W (way), R (relation), or P (polygon).
- osm_id: OpenStreetMap ID of the matched element.
- country: Country of the matched place.
- city: City of the matched place.
- osm_key: OpenStreetMap key.
- countrycode: ISO2 country code.
- housenumber: House number, if applicable.
- postcode: Post code, if applicable.
- locality: Locality, if applicable.
- street: Street, if applicable.
- district: District name, if applicable.
- osm_value: OpenStreetMap tag value.
- name: Place name.
- type: Layer type as described for the layer parameter.
- extent: Boundary box of the match.

Examples

```
# an instance must be mounted first
photon <- new_photon()

# works with sf objects
sf_data <- sf::st_sfc(sf::st_point(c(8, 52)), sf::st_point(c(7, 52)))
reverse(sf_data)

# ... but also with simple dataframes
df_data <- data.frame(lon = c(8, 7), lat = c(52, 52))
reverse(df_data)

# limit search radius to 10m
reverse(df_data, radius = 10)
```

---

structured   *Structured geocoding*

Description

Geocode a set of place information such as street, house number, or post code. Structured geocoding is generally more accurate but requires more information than unstructured geocoding.

Note that structured geocoding must be specifically enabled when building a Nominatim database. It is generally not available on komoot's public API and on pre-built search indices through download_searchindex. See vignette("nominatim-import", package = "photon") for details. You can use the helper function has_structured_support() to check if the current API supports structured geocoding.

Usage

```
structured(
  .data,
  limit = 3,
  lang = "en",
  bbox = NULL,
  osm_tag = NULL,
  layer = NULL,
  locbias = NULL,
  locbias_scale = NULL,
  zoom = NULL,
  progress = interactive()
)

has_structured_support()
```

Arguments

`.data`	Dataframe or list containing structured information on a place to geocode. Can contain the columns `street`, `housenumber`, `postcode`, `city`, `district`, `county`, `state`, and `countrycode`. At least one of these columns must be present in the dataframe. Note that countries must be passed as ISO-2 country codes.
`limit`	Number of results to return. Defaults to 3.
`lang`	Language of the results.
`bbox`	Any object that can be parsed by [st_bbox](). Results must lie within this bbox.
`osm_tag`	Character string giving an [OSM tag]() to filter the results by. See details.
`layer`	Character string giving a layer to filter the results by. Can be one of `"house"`, `"street"`, `"locality"`, `"district"`, `"city"`, `"county"`, `"state"`, `"country"`, or `"other"`.
`locbias`	Numeric vector of length 2 or any object that can be coerced to a length-2 numeric vector (e.g. a list or `sfg` object). Specifies a location bias for geocoding in the format `c(lon, lat)`. Geocoding results are biased towards this point. The radius of the bias is controlled through `zoom` and the weight of place prominence through `location_bias_scale`.
`locbias_scale`	Numeric vector specifying the importance of prominence in `locbias`. A higher prominence scale gives more weight to important places. Defaults to 0.2.
`zoom`	Numeric specifying the radius for which the `locbias` is effective. Corresponds to the zoom level in OpenStreetMap. The exact relation to `locbias` is $0.25 \text{ km} \cdot 2^{(18-\text{zoom})}$. Defaults to 16.
`progress`	If `TRUE`, shows a progress bar for longer queries.

Details

Filtering by OpenStreetMap tags follows a distinct syntax explained on https://github.com/komoot/photon. In particular:

- Include places with tag: `key:value`
- Exclude places with tag: `!key:value`
- Include places with tag key: `key`
- Include places with tag value: `:value`
- Exclude places with tag key: `!key`
- Exclude places with tag value: `:!value`

Value

An sf dataframe or tibble containing the following columns:

- `idx`: Internal ID specifying the index of the `texts` parameter.
- `osm_type`: Type of OSM element, one of N (node), W (way), R (relation), or P (polygon).
- `osm_id`: OpenStreetMap ID of the matched element.

- country: Country of the matched place.
- city: City of the matched place.
- osm_key: OpenStreetMap key.
- countrycode: ISO2 country code.
- housenumber: House number, if applicable.
- postcode: Post code, if applicable.
- locality: Locality, if applicable.
- street: Street, if applicable.
- district: District name, if applicable.
- osm_value: OpenStreetMap tag value.
- name: Place name.
- type: Layer type as described for the layer parameter.
- extent: Boundary box of the match.

Examples

```
## Not run:
# structured() requires an OpenSearch instance with structured support
# the following code will not work off the shelf
# refer to vignette("nominatim-import") for details
dir <- file.path(tempdir(), "photon")
photon <- new_photon(dir, opensearch = TRUE)
photon$import(password = "psql_password", structured = TRUE)
photon$start()

# check if structured() is supported
has_structured_support()

# structured() works on dataframes containing structurized data
place_data <- data.frame(
  housenumber = c(NA, "77C", NA),
  street = c("Falealilli Cross Island Road", "Main Beach Road", "Le Mafa Pass Road"),
  state = c("Tuamasaga", "Tuamasaga", "Atua")
)
structured(place_data, limit = 1)

# countries must be specified as iso2 country codes
structured(data.frame(countrycode = "ws"))

# traditional parameters from geocode() can also be used but are much more niche
structured(data.frame(city = "Apia"), layer = "house") # matches nothing

# structured geocoding can discern small differences in places
safune <- data.frame(
  city = c("Safune", "Safune"),
  state = c("Gaga'ifomauga", "Tuamasaga")
)
structured(safune, limit = 1)
```

```
## End(Not run)
```

Index