Package 'rmapzen'

March 7, 2023

Type Package

Title Client for 'Mapzen' and Related Map APIs

Version 0.4.4

Maintainer Tarak Shah <tarak.shah@gmail.com>

Description Provides an interface to 'Mapzen'-based APIs (including geocode.earth, Nextzen, and NYC GeoSearch) for geographic search and geocoding, isochrone calculation, and vector data to draw map tiles. See <https://www.mapzen.com/documentation/> for more information. The original Mapzen has gone out of business, but 'rmapzen' can be set up to work with any provider who implements the Mapzen API.

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 2.10)

Imports tibble, httr, jsonlite, maps, dplyr, assertthat, geojsonio, tidyr, purrr, sp, rgdal, digest, maptools, sf (>= 1.0.0), utils, rgeos

RoxygenNote 7.2.2

Suggests testthat, covr, knitr, rmarkdown, rlang

URL https://tarakc02.github.io/rmapzen/

BugReports https://github.com/tarakc02/rmapzen/issues

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author Tarak Shah [aut, cre], Daniel Possenriede [ctb]

Repository CRAN

Date/Publication 2023-03-07 19:20:02 UTC

R topics documented:

`as_sf` *Coerce a Mapzen response to a simple features object*

Description

Coerces responses to class sf. See `vignette("sf1", package = "sf")` for more information about Simple Features for R.

Usage

```
as_sf(geo, ...)

## S3 method for class 'geo_list'
as_sf(geo, ...)

## S3 method for class 'mapzen_vector_layer'
as_sf(geo, ...)
```

Arguments

geo	The object to be converted
...	not currently used

---

as_sp *Coerce a Mapzen response to an Spatial*DataFrame*

Description

Coerces responses to SpatialPoints, SpatialLines, or SpatialPolygons data frames.

Usage

```
as_sp(geo, ...)

## S3 method for class 'geo_list'
as_sp(geo, ...)

## S3 method for class 'mapzen_vector_layer'
as_sp(geo, ..., geometry_type = NULL)
```

Arguments

geo	The object to be converted
...	not currently used
geometry_type	"point", "line", or "polygon" – can be left NULL and only needs to be specified when an object contains multiple geometry types.

---

ca_tiles *Vector tiles the contain California*

Description

Vector tiles the contain California

Usage

```
ca_tiles
```

Format

An object of class `mapzen_vector_tiles` (inherits from `list`) of length 9.

Source

Mapzen, OpenStreetMap contributors, Who's On First, Natural Earth, and openstreetmapdata.com

costing_models *Costing model constructors and helpers*

Description

Mapzen's Isochrone service (mz_isochrone) as well as other mobility services (currently not implemented in this package, read more at https://valhalla.readthedocs.io/en/latest/) require users to specify a "costing model." See https://valhalla.readthedocs.io/en/latest/ for details. These can be difficult to construct correctly, so the objects mz_costing and mz_costing_options exist to make that process less error-prone and more convenient.

Usage

```
mz_costing

mz_costing_options
```

Format

An object of class list of length 4.

An object of class list of length 4.

See Also

mz_isochrone

Examples

```
## creates a pedestrian costing model with walking speed of 2 km/hr
## that also avoids alleys.
## non-multimodal costing models will accept 0 or more options from the
## appropriate list.
mz_costing$pedestrian(
    mz_costing_options$pedestrian$walking_speed(2.0),
    mz_costing_options$pedestrian$alley_factor(0)
)

## creates a multimodal costing model that favors buses over rails, and
## has a slower than default walking speed
## (note multimodal has named arguments requiring list inputs)
mz_costing$multimodal(
    transit = list(
        mz_costing_options$transit$use_bus(1.0),
        mz_costing_options$transit$use_rail(5)
    ),
    pedestrian = list(
        mz_costing_options$pedestrian$walking_speed(4.1)
    )
)
```

mapzen_references *Reference lists*

Description

Lists of sources, layers, and countries, as they are expected to appear in the mz_search functions. These data objects are provided as a convenience, to be able to quickly and easily look up acceptable values for the optional arguments of search functions. Object names match the argument names for which they are appropriate. So mz_sources provide acceptable arguments for the source argument in mz_search, mz_layers for the layer argument, and mz_countries for the boundary.country argument. Mapzen's documentation (https://github.com/pelias/documentation/) explains more about each of these arguments.

Usage

```
mz_sources

mz_layers

mz_countries
```

Format

An object of class list of length 8.

An object of class list of length 13.

An object of class list of length 552.

Examples

```
## Not run:
# look for YMCAs in Jamaica:
# Note that boundary.country is supplied via ISO3166 code,
# but mz_countries will look up the code
mz_search("YMCA",
          boundary.country = mz_countries$Jamaica,
          layers = c(mz_layers$venue, mz_layers$address))

## End(Not run)
```

marina_walks *Pedestrian isochrones from the Berkeley Marina for 10 and 15 minutes*

Description

Isochrone results from Mapzen as of January 8, 2017. The location for the isochrones is the Berkeley Marina, lat 37.86613, lon -122.3151

Usage

```
marina_walks
```

Format

An object of class `mapzen_isochrone_list` (inherits from `geo_list`) of length 3.

Source

Mapzen, OpenStreetMap, British Oceanographic Data Centre, NASA, USGS, and Transitland.

marina_walks_polygons *Pedestrian isochrones from the Berkeley Marina for 10 and 15 minutes, as polygons*

Description

Polygon Isochrone results (using `polygons = TRUE`) from Mapzen as of January 10, 2017. The location for the isochrones is the Berkeley Marina, lat 37.86613, lon -122.3151, and the contours are 10 and 15 minutes for a pedestrian costing model.

Usage

```
marina_walks_polygons
```

Format

An object of class `mapzen_isochrone_list` (inherits from `geo_list`) of length 3.

Source

Mapzen, OpenStreetMap, British Oceanographic Data Centre, NASA, USGS, and Transitland.

`mz_autocomplete` *Mapzen search API*

Description

Functions to access the various endpoints from the Mapzen Search API. For more details, see https://github.com/pelias/documentation/. If your data is already split up by street, city, state, zip, etc., then you might find `mz_structured_search` to be more precise. All arguments besides `text` (`point` in the case of `mz_reverse_geocode`) are optional. If you have parsed addresses (e.g. for geocoding), use `mz_structured_search`

Usage

```
mz_autocomplete(
  text,
  boundary.country = NULL,
  boundary.rect = NULL,
  focus.point = NULL,
  sources = NULL,
  layers = NULL,
  api_key = NULL
)

mz_reverse_geocode(
  point,
  size = NULL,
  layers = NULL,
  sources = NULL,
  boundary.country = NULL,
  api_key = NULL
)

mz_search(
  text,
  size = 10,
  boundary.country = NULL,
  boundary.rect = NULL,
  boundary.circle = NULL,
  focus.point = NULL,
  sources = NULL,
  layers = NULL,
  api_key = NULL
)
```

Arguments

`text`	Search string

boundary.country
: ISO-3166 country code to narrow the search. See mz_countries

boundary.rect
: 4 corners that define a box to narrow the search. Can be the result of mz_bbox. Should have named elements with names "min_lon", "min_lat", "max_lon", "max_lat" – can be created using mz_rect.

focus.point
: A point to "focus" the search. Can be created with mz_location or mz_geocode, otherwise should have names "lat" and "lon"

sources
: The originating source of the data (to filter/narrow search results). See mz_sources

layers
: Which layers (types of places) to search. See https://github.com/pelias/documentation/ for definitions, and use mz_layers for convenience

api_key
: Your Mapzen API key. The default is to look for the key within the provider information that was set up with 'mz_set_host'.

point
: For reverse geocoding, the location to reverse geocode. Can be created with mz_location or mz_geocode, otherwise should have names "lat" and "lon"

size
: Number of search results requested

boundary.circle
: A circle to narrow the search. Should have named elements with names "lon", "lat", and "radius"

See Also

mz_place, mz_structured_search, mz_countries, mz_sources, mz_layers

Examples

```
## Not run:
# hard rock cafes in sweden:
mz_search("Hard Rock Cafe", boundary.country = "SE")

# autocompletions when the user types in "Union Square"
# prioritizing San Francisco results first:
mz_autocomplete("Union Square",
                focus.point = mz_geocode("San Francisco, CA"))

## End(Not run)
```

---

mz_bbox *Get the bounding box*

Description

Returns the bottom left and top right corners of the box that contains a mapzen object (mz_geo_list, mz_isochrone_list, or mapzen_vector_tiles). In the case of mz_rect, creates such a box from the specified coordinates. The returned value can be used directly as the boundary.rect parameter for search functions, as well as converted to x, y, zoom coordinates to use with mz_vector_tiles.

Usage

```
mz_bbox(geo)

## S3 method for class 'mapzen_geo_list'
mz_bbox(geo)

## S3 method for class 'mapzen_isochrone_list'
mz_bbox(geo)

mz_rect(min_lon, min_lat, max_lon, max_lat)
```

Arguments

geo	A mapzen geo list or isochrone list
min_lon, min_lat, max_lon, max_lat	The bottom left and top right corners, expressed as latitude and longitude, of a rectangle.

Value

A single-row tibble with columns `min_lon`, `min_lat`, `max_lon`, `max_lat`.

Examples

```
mz_rect(min_lon = -122.2856, min_lat = 37.73742, max_lon = -122.1749, max_lat = 37.84632)
mz_bbox(oakland_public)
```

---

mz_check_usage *Check usage statistics*

Description

Prints out remaining queries for various time periods. `rmapzen` manages rate limiting for the per-second limits, but does not keep track of the daily limits.

Usage

```
mz_check_usage()
```

Details

This function is populated from the headers of responses to various API requests. If no queries have been made, or if the only queries so far have hit cache servers, then no information will be available.

mz_contours *Create an mz_contours object*

Description

Contours are given as inputs to mz_isochrone. This function makes it convenient to construct them.

Usage

```
mz_contours(times, colors = NULL)
```

Arguments

times	Times in minutes for the contour. Up to a maximum of 4 numbers.
colors	Colors for the contours. By default, a palette will be constructed from the Colorbrewer 4-class oranges palette.

mz_coordinates *Extract a data frame of coordinates from a* mapzen_geo_list

Description

Extract a data frame of coordinates from a mapzen_geo_list

Usage

```
mz_coordinates(geo)

## S3 method for class 'mapzen_geo_list'
mz_coordinates(geo)
```

Arguments

geo	A mapzen geo list

Value

A tibble, with columns lon and lat.

Examples

```
mz_coordinates(oakland_public)
```

mz_date_time *Create mz_date_time objects*

Description

Mobility services (such as `mz_isochrone`) take, optionally, a date_time argument that specifies the date and time along with type (departure/arrival). This function constructs the appropriate objects to use as date_time arguments.

Usage

```
mz_date_time(date_time, type = "departure")
```

Arguments

`date_time`	A POSIXt date-time object
`type`	"departure" or "arrival"

mz_geocode *Geocode an address or other location*

Description

This is a convenience function that calls `mz_search` to retrieve latitude and longitude.

Usage

```
mz_geocode(location, ...)
```

Arguments

`location`	An address or other suitably specific search string
`...`	Additional arguments passed on to `mz_search`

Value

A tibble, with the parsed address used to retrieve the geocode, lat/lon, and the confidence (between 0 and 1)

See Also

`mz_search`, `mz_reverse_geocode`

Examples

```
## Not run:
mz_geocode("1600 Pennsylvania Ave., Washington DC")

# can also be a landmark
mz_geocode("Statue of Liberty, New York")

## End(Not run)
```

`mz_geocode_structured` *Geocode a structured address*

Description

`mz_geocode` allows you to search using an unstructured string of text, but if your address data has more structure (eg separate columns for address, city, state, zip), then using the structured search service may provide more precision. For more information, see https://github.com/pelias/documentation/. Note that all of the arguments are optional, but at least one of them must be non-NULL. Furthermore, `postalcode` can not be used by itself.

Usage

```
mz_geocode_structured(
  address = NULL,
  neighbourhood = NULL,
  borough = NULL,
  locality = NULL,
  county = NULL,
  region = NULL,
  postalcode = NULL,
  country = NULL,
  ...
)
```

Arguments

`address`	Can be a numbered street address or just the name of the street
`neighbourhood`	Neighborhood name (eg "Notting Hill" in London)
`borough`	eg "Manhattan"
`locality`	The city (eg "Oakland")
`county`	The county
`region`	States in the case of US/Canada, or state-like administrative division in other countries
`postalcode`	AKA the zip code. Can not be used alone, must have at least one other argument
`country`	The country - Can be the full name or the abbreviation from `mz_countries`
`...`	Arguments passed on to `mz_structured_search`

Value

A tibble, with the parsed address used to retrieve the geocode, lat/lon, and the confidence (between 0 and 1)

See Also

mz_geocode, mz_structured_search

`mz_isochrone` *Retrieve isochrones*

Description

From https://valhalla.readthedocs.io/en/latest/: "An isochrone is a line that connects points of equal travel time about a given location, from the Greek roots of 'iso' for equal and 'chrone' for time. The Mapzen Isochrone service computes areas that are reachable within specified time intervals from a location, and returns the reachable regions as contours of polygons or lines that you can display on a map."

Usage

```
mz_isochrone(
  locations,
  costing_model,
  contours,
  date_time = NULL,
  polygons = NULL,
  denoise = NULL,
  generalize = NULL,
  id = "my-iso",
  api_key = NULL
)
```

Arguments

`locations`	An `mz_location`, or something that can be coerced to an `mz_location`, as the departure point for the isochrone. This can be the result of `mz_geocode`. Despite the argument name, the isochrone service currently can only accept a single location
`costing_model`	The costing model, see `mz_costing`
`contours`	Up to 4 contours, see `mz_contours`
`date_time`	The local date and time at the location, and whether it is the departure or arrival time. See `mz_date_time`
`polygons`	Whether to return polygons (TRUE) or linestrings (FALSE, default)

denoise	A value between 0 and 1 (default 1) to remove smaller contours. A value of 1 will only return the largest contour for a given time value. A value of 0.5 drops any contours that are less than half the area of the largest contour.
generalize	Tolerance in meters for the Douglas-Peucker generalization.
id	A descriptive identifier, the response will contain the id as an element.
api_key	Your Mapzen API key. The default is to look for the key within the provider information that was set up with 'mz_set_host'.

Value

A mapzen_isochrone_list, which can be converted to sf or sp using as_sf or as_sp.

See Also

mz_costing

Examples

```
## Not run:
mz_isochrone(
    mz_location(lat = 37.87416, lon = -122.2544),
    costing_model = mz_costing$auto(),
    contours = mz_contours(c(10, 20, 30))
)

# departure point can be specified as a geocode result
mz_isochrone(
    mz_geocode("UC Berkeley"),
    costing_model = mz_costing$pedestrian(),
    contours = mz_contours(c(10, 20, 30))
)

## End(Not run)
```

---

mz_location *Create/extract lat/lon location information*

Description

mz_location constructs a new mz_location object, which can be used with functions such as mz_isochrone or mz_reverse_geocode. as.mz_location coerces eligible objects to mz_locations.

Usage

```
mz_location(lat, lon)

as.mz_location(x, ...)

## Default S3 method:
as.mz_location(x, ...)

## S3 method for class 'mz_geocode_result'
as.mz_location(x, ...)
```

Arguments

lat	Latitude
lon	Longitude
x	An object that has location information
...	Not currently used

See Also

mz_isochrone For using the Mapzen isochrone service mz_contours, mz_costing, and mz_costing_options for other argument constructors

---

mz_place *Get details on a place*

Description

Search functions (e.g. mz_search) return identification numbers, or gids. Use mz_place to retrieve more details about the place. See https://github.com/pelias/documentation/ for details. This function is generic, and can take a character vector of IDs, or a mapzen_geo_list as returned by mz_search and friends.

Usage

```
mz_place(ids, ..., api_key = NULL)

## S3 method for class 'character'
mz_place(ids, ..., api_key = NULL)

## S3 method for class 'mapzen_geo_list'
mz_place(ids, ..., gid = "gid", api_key = NULL)
```

Arguments

ids	A character vector of gids (see details), or a mapzen_geo_list
...	Arguments passed on to methods
api_key	Your Mapzen API key. The default is to look for the key within the provider information that was set up with 'mz_set_host'.
gid	The name of the gid field to use. Search results may include, in addition to the gid for the search result itself (the default), the gids for the country, region, county, locality and neighborhood.

---

mz_provider *Configure provider information*

Description

rmapzen works with most implementations of PELIAS. This function defines the base URL for a particular API provider, and can be used to provider the provider argument to mz_set_host.

Usage

```
mz_provider(hostname, path = NULL, key = NULL, scheme = "https")
```

Arguments

hostname	The hostname in the API URL, for instance www.example.com
path	Specific path that all API requests must include, e.g. "v1"
key	API key for this provider, if required
scheme	The scheme for the URL, should always be "https"

See Also

mz_set_host

---

mz_set_host *Set up a host provider for a PELIAS service*

Description

rmapzen works with most implementations of PELIAS. Use this function to set up the basic information required to connect to a particular provider. Provider-specific setup functions include information to set up known providers.

Usage

```
mz_set_host(which, provider)

mz_get_host(which)

mz_set_search_host_geocode.earth(key = Sys.getenv("GEOCODE.EARTH_KEY"))

mz_set_search_host_nyc_geosearch()

mz_set_tile_host_nextzen(key = Sys.getenv("NEXTZEN_KEY"))
```

Arguments

which	One of "search", "matrix", or "tile"
provider	A provider, created using mz_provider
key	API key

See Also

mz_provider

---

mz_structured_search *Structured search*

Description

mz_search allows you to search using an unstructured string of text, but if your address data has more structure (eg separate columns for address, city, state, zip), then using the structured search service may provide more precision. For more information, see https://github.com/pelias/documentation. Note that all of the arguments are optional, but at least one of them must be non-NULL. Furthermore, postalcode can not be used by itself.

Usage

```
mz_structured_search(
  address = NULL,
  neighbourhood = NULL,
  borough = NULL,
  locality = NULL,
  county = NULL,
  region = NULL,
  postalcode = NULL,
  country = NULL,
  api_key = NULL,
  ...
)
```

Arguments

address	Can be a numbered street address or just the name of the street
neighbourhood	Neighborhood name (eg "Notting Hill" in London)
borough	eg "Manhattan"
locality	The city (eg "Oakland")
county	The county
region	States in the case of US/Canada, or state-like administrative division in other countries
postalcode	AKA the zip code. Can not be used alone, must have at least one other argument
country	The country - Can be the full name or the abbreviation from mz_countries
api_key	Your Mapzen API key. The default is to look for the key within the provider information that was set up with ‘mz_set_host‘.
...	Any of the parameters, other than "text", that appear in mz_search, can appear here, for example size, boundary.country, etc.

See Also

mz_search

---

mz_tile_coordinates *Specify tile coordinates*

Description

mz_vector_tiles requires tile coordinates or some other specification of the region that is to be drawn. mz_vector_tiles will automatically convert its inputs to vector tiles, so you generally won't need to use this function directly.

Usage

```
mz_tile_coordinates(x, y, z)

as.mz_tile_coordinates(obj, ...)

## S3 method for class 'mz_tile_coordinates'
as.mz_tile_coordinates(obj, ...)

## S3 method for class 'mz_bbox'
as.mz_tile_coordinates(obj, ..., z = NULL, height = NULL, width = NULL)

## S3 method for class 'mz_location'
as.mz_tile_coordinates(obj, ..., z = 15L)

## S3 method for class 'mz_geocode_result'
as.mz_tile_coordinates(obj, ..., z = 15L)
```

Arguments

x	integer vector of x-coordinates
y	integer vector of y-coordinates
z	integer between 0 and 19 specifying the zoom level
obj	An object that can be converted to tile coordinates
...	Other arguments passed on to methods
height	Height in pixels
width	Width in pixels

See Also

mz_vector_tiles, mz_bbox

Examples

```
mz_tile_coordinates(19293, 24641, 16)

## can specify multiple contiguous tiles:
mz_tile_coordinates(19293:19294, 24641:24642, 16)

## a rectangular bounding box can be converted to tile coordinates:
as.mz_tile_coordinates(mz_rect(min_lon = -122.2856,
                               min_lat = 37.73742,
                               max_lon = -122.1749,
                               max_lat = 37.84632))

## zoom level is calculated based on desired pixel dimensions of the map:
as.mz_tile_coordinates(mz_rect(min_lon = -122.2856,
                               min_lat = 37.73742,
                               max_lon = -122.1749,
                               max_lat = 37.84632), height = 750, width = 1000)

## a bounding box can also be calculated:
as.mz_tile_coordinates(mz_bbox(oakland_public))
```

---

mz_vector_tiles *Request vector tile data*

Description

From https://tilezen.readthedocs.io/en/latest/: "Vector tiles are square-shaped collections of geographic data that contain the map feature geometry, such as lines and points."

Usage

```
mz_vector_tiles(tile_coordinates, ..., Origin = NULL)
```

Arguments

tile_coordinates	an mz_tile_coordinates object, or something that can be coerced to one (including the output of mz_bbox)
...	Arguments passed on to as.mz_tile_coordinates.
Origin	optional, specify Origin URL in request header

Details

Multiple tiles are stitched together and returned as one object. Individual layers can be converted to sf or sp, making it possible to draw each layer with custom styles.

Value

A list of tile layers (such as "water", "buildings", "roads", etc.). Each layer is an object of class mapzen_vector_layer, which can be converted to sf or sp using as_sf or as_sp

See Also

mz_tile_coordinates

Examples

```
## Not run:
# vector tile at x = 19293, y = 24641, and zoom level 16
mz_vector_tiles(mz_tile_coordinates(19293, 24641, 16))

# multiple contiguous tiles will be stitched together
# this returns the result of stitching together 4 tiles
mz_vector_tiles(mz_tile_coordinates(19293:19294, 24641:24642, 16))

# can also use a bounding box:
mz_vector_tiles(mz_rect(min_lon = -122.2856,
                        min_lat = 37.73742,
                        max_lon = -122.1749,
                        max_lat = 37.84632))

# mz_bbox returns a bounding box for any Mapzen object
mz_vector_tiles(mz_bbox(oakland_public))

# bounding boxes are automatically converted to tile coordinates,
# with the zoom level based on the desired size in pixels of the final map
mz_vector_tiles(mz_bbox(oakland_public), height = 750, width = 1000)

## End(Not run)
```

oakland_public *25 search results for "Oakland Public library branch"*

Description

Contains the search results from Mapzen's search service for the query "Oakland public library branch" as of January 8, 2017.

Usage

```
oakland_public
```

Format

A `mapzen_geo_list` with 25 locations

Source

Mapzen, OpenStreetMap, OpenAddresses, GeoNames, WhosOnFirst, see https://www.mapzen.com/rights/

rmapzen *rmapzen: A client application for the 'Mapzen' API.*

Description

The rmapzen package provides interfaces to the Search (https://github.com/pelias/documentation/), Isochrone (https://valhalla.readthedocs.io/en/latest/), and Vector Tile (https://tilezen.readthedocs.io/en/latest/) services from 'Mapzen', via the following functions:

Search

All functionality described in https://github.com/pelias/documentation/ are supported:

- mz_search
- mz_reverse_geocode
- mz_autocomplete
- mz_place
- mz_structured_search

Additionally, mz_geocode is useful for a common application of search, that of just obtaining latitude and longitude for a given address or place.

Isochrone

Isochrones are the areas reachable from a given location within a specified period of time. Mapzen's Isochrone service can calculate isochrones for driving, walking, cycling, or multimodal forms of transport:

- `mz_isochrone`
- `mz_costing`: for constructing "costing models" that describe method of transport along with speed and other options relevant to the calculation of the isochrone
- `mz_costing_options`: for selecting specific options when constructing a costing model

Vector Tiles

- `mz_vector_tiles`: Request one or more adjacent tiles. Multiple map tiles will be stitched together before being returned as a single object.
- `mz_tile_coordinates`: When using `mz_vector_tiles`, you must specify the geographic area for which you want tile data. One way to do so is using the x, y, z tile naming system (see https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames).
- `mz_rect`: Alternatively, you can specify the lower left and top-right points of a bounding box, which will automatically be converted to tile-coordinates when you use `mz_vector_tiles`
- `mz_bbox`: This is a generic function which will return the bounding box of any Mapzen object. In this way, you can request vector tiles for a region defined as the bounding box of an existing object.

Data types and conversion

Objects returned by rmapzen can be converted to both Spatial*DataFrames and simple features (sf) via the generic functions `as_sp` (for Spatial*DataFrames) and `as_sf` (for simple features). Search and Isochrone objects can additionally be converted to ordinary data.frames via `as.data.frame`.

See Also

- https://tarakc02.github.io/rmapzen/ contains detailed examples
- https://www.mapzen.com/documentation/ 'Mapzen' documentation

Index