Package ‘onemapsgapi’

May 30, 2025

Type Package

Title R Wrapper for the 'OneMap.Sg API'

Version 2.0.0

Maintainer Jolene Lim <jolene.lim14@gmail.com>

Description An R wrapper for the 'OneMap.Sg' API <https://www.onemap.gov.sg/docs/>.
Functions help users query data from the API and return raw JSON data
in ``tidy'' formats. Support is also available for users to retrieve data from multiple API calls
and integrate results into single dataframes, without
needing to clean and merge the data themselves. This package is best suited for users who would
like to perform analyses with Singapore's spatial data without having to perform exces-
sive data cleaning.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports rlang, httr2, dplyr, purrr, stringr, tidyr, future, furrr

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, sf, googlePolylines

VignetteBuilder knitr

NeedsCompilation no

Author Jolene Lim [aut, cre]

Repository CRAN

Date/Publication 2025-05-30 09:20:02 UTC

Contents

`geocode_onemap` *Geocode a dataframe of keywords*

Description

This function is a wrapper for the Search API. It allows for geocoding of data such as postal codes or address information. Users input a dataframe with a column to geocode (e.g. postal codes, address information). It returns a tibble with additional columns of coordinate data. Optionally, it can also return the output as an `sf` object.

Usage

```
geocode_onemap(
  df,
  search_val,
  return_geom = FALSE,
  address_details = FALSE,
  return_spatial = FALSE,
  spatial_lnglat = TRUE,
  parallel = FALSE
)
```

Arguments

`df`	Input tibble with column to be geocoded
`search_val`	Column name containing keyword(s) to be geocoded, e.g. column of postal codes
`return_geom`	Default = `FALSE`. Whether to return the coordinate information
`address_details`	Default = `FALSE`. Whether to return address information
`return_spatial`	Default = `FALSE`. Whether to return the output as an `sf` object. Please ensure
`spatial_lnglat`	Default = `TRUE`. If `TRUE`, the WGS84 coordinates will be used to create the `sf` tibble. If `FALSE`, the SVY21 coordinates will be used.
`parallel`	Default = `FALSE`. Whether to run API calls in parallel or sequentially (default).

Value

Please note only the top result matching the search will be returned. If no error occurs:

SEARCH_VAL Detailed search name

X Longitude in SVY21. Returned only if `return_geom = TRUE`

Y Latitude in SVY21. Returned only if `return_geom = TRUE`

LONGITUDE Longitude in WGS84. Returned only if `return_geom = TRUE`

LATITUDE Latitude in WGS84. Returned only if `return_geom = TRUE`

BLK_NO Block number

ROAD_NAME Road Name

BUILDING Building Name

ADDRESS Address

POSTAL Postal Code

If an error occurs, an empty result will be returned for that row. A warning message will be printed with the serach value, API error message and status code.

Examples

```
# sample dataframe. the last record does not return any API results.
df <- data.frame(
  places = c("a", "b", "c", "d"),
  address = c("raffles place mrt", "suntec city", "nus", "100353")
)

# Returns the original df with additional columns
## Not run: geocode_onemap(df, "address",
  return_geom=TRUE, address_details = TRUE, return_spatial=TRUE)

## End(Not run)
# If an error occurs for any of the rows, an empty row will be returned.
```

`get_planning_areas` *Get Planning Areas (All)*

Description

This function is a wrapper for the Planning Area Polygons API. It returns the data either in raw format or a combined sf object.

Usage

```
get_planning_areas(token, year = NULL, return_spatial = FALSE)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`year`	Optional, check documentation for valid options. Invalid requests will are ignored by the API.
`return_spatial`	Optional, whether to return the result as a `sf` tibble instead of JSON object. Default value is `FALSE`

Value

If the parameter `read` is not specified, the function returns a raw JSON object with planning names and geojson string vectors.

If `return_spatial` = `TRUE`, the function returns a single "sf" tibble with 2 columns: "name" (name of planning area) and "geometry", which contains the simple features.

If an error occurs, the function throws an error with the API error message and status code.

Note

If the user specifies `return_spatial` = `TRUE` but does not have the `sf` package installed, the function will return the raw JSON and print a warning message.

Examples

```
# returns raw JSON object
## Not run: get_planning_areas(token)
## Not run: get_planning_areas(token, 2008)

# returns dataframe of class "sf"
## Not run: get_planning_areas(token, return_spatial=TRUE)

# error: output is NULL, warning message shows status code
## Not run: get_planning_areas("invalid_token")
```

`get_planning_names` *Get Planning Area Names*

Description

This function is a wrapper for the Names of Planning Area API. It returns the data as a tibble.

Usage

```
get_planning_names(token, year = NULL)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`year`	Optional, check documentation for valid options. Invalid requests will are ignored by the API.

Value

A tibble with 2 columns:

id Planning area id

pln_area_n Planning area name

Examples

```
# returns tibble
## Not run: get_planning_names(token)
## Not run: get_planning_names(token, 2008)

# error: output is NULL, warning message shows status code
## Not run: get_planning_names("invalid_token")
```

`get_planning_polygon` *Get Planning Polygon for a Specific Point*

Description

This function is a wrapper for the Planning Area Query API. It returns the spatial polygon data matching the specified location point, either in raw format or as an sf tibble.

Usage

```
get_planning_polygon(token, lat, lon, year = NULL, return_spatial = FALSE)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`lat`	Latitude of location point
`lon`	Longitude of location point
`year`	Optional, check documentation for valid options. Invalid requests will are ignored by the API.
`return_spatial`	Optional, defaults to `FALSE`. If `TRUE`, result will be returned as a `sf` tibble, otherwise the raw JSON object will be returned.

Value

If the parameter `read` is not specified, the function returns a raw JSON object a list containing the planning area name and a geojson string representing the polygon.

If `read = "sf"`, the function returns a 1 x 2 "sf" dataframe: "name" (name of planning area) and "geometry", which contains the simple feature.

If an error occurs, the function throws an error with the API error message and status code.

Note

If the user specifies a `return_spatial = TRUE` but does not have the `sf` package installed, the function will return the raw JSON and print a warning message.

Examples

```
# returns raw JSON object
## Not run: get_planning_polygon(token, lat = 1.429443081, lon = 103.835005)
## Not run: get_planning_polygon(token, lat = 1.429443081, lon = 103.835005, year = 2008)

# returns dataframe of class "sf"
## Not run: get_planning_polygon(token, lat = 1.429443081, lon = 103.835005, return_spatial = TRUE)

# error: output is NULL, warning message shows status code
## Not run: get_planning_polygon("invalid_token", lat = 1.429443081, lon = 103.835005)
## Not run: get_planning_polygon(token, "invalidlat", "invalidlon")
```

`get_pop_queries` *Get Population Data (Multiple)*

Description

This function is a wrapper for the Population Query API. It allows for querying of multiple Popquery data types for multiple towns and years.

Usage

```
get_pop_queries(
  token,
  data_types,
  planning_areas,
  years,
  gender = NULL,
  parallel = FALSE
)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`data_types`	Type of data to be retrieved, should correspond to one of the API endpoints. E.g. to get economic status data, `data_type = "getEconomicStatus"`. The API endpoints can be found on the documentation page.
`planning_areas`	Town for which the data should be retrieved.
`years`	Year for which the data should be retrieved.
`gender`	Optional, if specified only records for that gender will be returned. This parameter is only valid for the `"getEconomicStatus"`, `"getEthnicGroup"`, `"getMaritalStatus"` and `"getPopulationAgeGroup"` endpoints. If specified for other endpoints, the parameter will be dropped.
`parallel`	Default = `FALSE`. Whether to run API calls in parallel or sequentially (default). Enabling parallel iterations is highly recommended for when querying multiple data types/years/towns.

Value

A tibble with each row representing a town in a particular year for a particular gender, and columns with the variables returned by the API endpoint. If any API call returns no data, the values will be NA but the row will be returned. However, if all data_types do not return data for that town and year, no row will be returned for it.

Examples

```
# output with no NA
## Not run: get_pop_queries(token, c("getReligion", "getLanguageLiterate"),
    c("Bedok", "Yishun"), "2010")
## End(Not run)
## Not run: get_pop_queries(token, c("getEconomicStatus", "getEthnicGroup"),
    "Yishun", "2010", "female")
## End(Not run)

## note behaviour if data types is a mix of those that accept gender params
### only total will have all records
## Not run: get_pop_queries(token, c("getEconomicStatus", "getOccupation", "getLanguageLiterate"),
    "Bedok", "2010")
## End(Not run)
### data type that does not accept gender params will be in gender = Total
## Not run: get_pop_queries(token, c("getEconomicStatus", "getOccupation", "getLanguageLiterate"),
    "Bedok", "2010", gender = "female")
## End(Not run)

# output with some town-year queries without record due to no data
# warning message will show data_type/town/year/gender for which an error occurred
## Not run: get_pop_queries(token, c("getEconomicStatus", "getOccupation"),
    "Bedok", c("2010", "2012"))
## End(Not run) # no records for 2012
```

get_pop_query *Get Population Data*

Description

This function is a wrapper for the Population Query API. It only allows for querying of one data type (i.e. one of the API endpoints) for a particular town and year.

Usage

```
get_pop_query(token, data_type, planning_area, year, gender = NULL)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`data_type`	Type of data to be retrieved, should correspond to one of the API endpoints. E.g. to get economic status data, `data_type = "getEconomicStatus"`. The API endpoints can be found on the documentation page.
`planning_area`	Town for which the data should be retrieved.
`year`	Year for which the data should be retrieved.
`gender`	Optional, valid values include `male` and `female`. If specified, only records for that gender will be returned. This parameter is only valid for the `"getEconomicStatus"`, `"getEthnicGroup"`, `"getMaritalStatus"` and `"getPopulationAgeGroup"` endpoints. If specified for other endpoints, the parameter will be dropped. If gender is not specified for valid endpoints, records for total, male and female will be returned.

Value

A tibble with 1 row and values for all the corresponding variables returned by the API endpoint. If an error occurs, the function returns NULL and a warning message. This differs from the error handling of other functions in this package to prevent collective failure for `get_pop_queries`.

Examples

```
# output with no NA
## Not run: get_pop_query(token, "getReligion", "Yishun", "2010")
## Not run: get_pop_query(token, "getModeOfTransportSchool", "Bishan", "2015", "female")

# if gender parameter is not specified, results for both genders and total will be returned
## Not run: get_pop_query(token, "getMaritalStatus", "Bedok", "2010")
## Not run: get_pop_query(token, "getEthnicGroup", "Bedok", "2010")
## Not run: get_pop_query(token, "getPopulationAgeGroup", "Bedok", "2010")

# output due to error
## Not run: get_pop_query(token, "getSpokenAtHome", "Bedok", "2043")
```

`get_summ_route` *Get Summary Route Information*

Description

This function is a wrapper for the Route Service API. However, it only returns the total time, distance and optionally the route geometry between two points. If `route = "pt"`, only the best route is chosen (i.e. `n_itineraries = 1`).

Usage

```
get_summ_route(
  token,
  start,
  end,
  route,
  date = format(Sys.Date(), "%m-%d-%Y"),
  time = format(Sys.time(), "%T"),
  mode = NULL,
  max_dist = NULL,
  route_geom = FALSE
)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`start`	Vector of c(lat, lon) coordinates for the route start point
`end`	Vector of c(lat, lon) coordinates for the route end point
`route`	Type of route. Accepted values are `walk`, `drive`, `pt` (public transport), or `cycle`
`date`	Default = current date. Date for which route is requested.
`time`	Default = current time. Time for which route is requested.
`mode`	Required if `route = "pt"`. Accepted values are `TRANSIT`, `BUS` or `RAIL`
`max_dist`	Optional if `route = "pt"`. Maximum walking distance
`route_geom`	Default = FALSE. Whether to return decoded route_geometry. Please ensure packages **googlePolylines** and **sf** are installed and note that this is a lossy conversion.

Value

If no error occurs, a tibble of 1 x 2 with the variables:

total_time The total time taken for this route

total_dist The total distance travelled for this route

If an error occurs, the output will be `NA`, along with a warning message.

Examples

```
# returns output tibble
## Not run: get_summ_route(token, c(1.320981, 103.844150), c(1.326762, 103.8559), "drive")
## Not run: get_summ_route(token, c(1.320981, 103.844150), c(1.326762, 103.8559), "pt",
    mode = "bus", max_dist = 300)
## End(Not run)

# returns output sf dataframe
## Not run: get_summ_route(token, c(1.320981, 103.844150), c(1.326762, 103.8559),
    "drive", route_geom = TRUE)
## End(Not run)
## Not run: get_summ_route(token, c(1.320981, 103.844150), c(1.326762, 103.8559), "pt",
    mode = "bus", max_dist = 300, route_geom = TRUE)
## End(Not run)
#'
# error: output is NULL, warning message shows status code
## Not run: get_summ_route("invalid_token", c(1.320981, 103.844150), c(1.326762, 103.8559), "drive")

# error: output is NULL, warning message shows error message from request
## Not run: get_summ_route(token, c(300, 300), c(400, 500), "cycle")
## Not run: get_summ_route(token, c(1.320981, 103.844150), c(1.326762, 103.8559), "fly")
```

---

get_theme *Get Theme Data from OneMap.Sg*

Description

This function is a wrapper for the Retrieve Theme API. It returns the data as cleaned tibbles.

Usage

```
get_theme(
  token,
  theme,
  extents = NULL,
  return_info = FALSE,
  return_spatial = FALSE
)
```

Arguments

token	User's API token. This can be retrieved using `get_token`
theme	OneMap theme in its `QUERYNAME` format. A tibble of available themes can be retrieved using `search_themes`
extents	Optional, Location Extents for search. This should be in the format "Lat1,%20Lng1,Lat2,%20Lng2". For more information, consult the API Documentation.

`return_info`	Default = `FALSE`. If `FALSE`, function only returns a tibble for query results. If `TRUE`, function returns output as a list containing a tibble for query information and a tibble for query results.
`return_spatial`	Default = `FALSE`. If `FALSE`, function returns a tibble. If `TRUE`, function returns an `sf` tibble.

Value

If no error occurs:

query_info A 1 x 7 tibble containing information about the query. The variables are `FeatCount`, `Theme_Name`, `Category`, `Owner`, `DateTime.date`, `DateTime.timezone_type`, `DateTime.timezone`

query_result Returned if return_info = `TRUE`. A tibble containing the data retrieved from the query. The columns and rows vary depending on theme and user specification, however all tibbles will contain the variables: `NAME`, `DESCRIPTION`, `ADDRESSPOSTALCODE`, `ADDRESSSTREETNAME`, `Lat`, `Lng`, `ICON_NAME`

If an error occurs, an error will be raised, along with the API's error message and status code. For non-error queries where 0 results are returned, the output will be `query_info`, along with a warning message.

Examples

```
# returns a tibble of output
## Not run: get_theme(token, "hotels")
## Not run: get_theme(token, "monuments",
    extents = "1.291789,%20103.7796402,1.3290461,%20103.8726032")
## End(Not run)

# returns a sf dataframe
## Not run: get_theme(token, "hotels", return_spatial = TRUE)

# returns a list of status tibble and output tibble
## Not run: get_theme(token, "funeralparlours", return_info = TRUE)

# error: throws an error with error message and status code
## Not run: get_theme("invalid_token", "hotels")

# error: throws an error with error message and status code
## Not run: get_theme(token, "non-existent-theme")

# error: output is \code{query_info}, warning message query did not return any records
## Not run: get_theme(token, "ura_parking_lot", "1.291789,%20103.7796402,1.3290461,%20103.8726032")
```

`get_theme_info` *Get Theme Information*

Description

This function is a wrapper for the Get Theme Info API. It returns a named character vector of Theme Name and Query Name.

Usage

```
get_theme_info(token, theme)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`theme`	Query name of theme. Themes' query names can be retrieved using `search_themes`.

Value

A named character vector of Theme Name and Query Name. If an error occurred, the function throws an error with the status code and API's error message.

Examples

```
# returns named character vector
## Not run: get_theme_info(token, "kindergartens")

# throws an error with error message and status code
## Not run: get_theme_info(token, "invalid_theme")

# throws an error with error message and status code
## Not run: get_theme_info("invalid_token", "blood_bank")
```

`get_theme_status` *Check Theme Status*

Description

This function is a wrapper for the Check Theme Status API. It returns a named logical indicating if the theme is updated at a specific date.

Usage

```
get_theme_status(
  token,
  theme,
  date = Sys.Date(),
  time = format(Sys.time(), format = "%T")
)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`theme`	Query name of theme. Themes' query names can be retrieved using `search_themes`.
`date`	Default = current date. Date to check for updates. Format YYYY-MM-DD
`time`	Default = current time. Time to check for updates. Format: HH:MM:SS:FFFZ

Value

A named logical indicating if the theme is updated at a specific date. If an error occurred, the function throws an error with the status code and API's error message.

Examples

```
# returns named logical
## Not run: get_theme_status(token, "kindergartens")
## Not run: get_theme_status(token, "hotels", "2020-01-01", "12:00:00")

# throws an error with error message and status code
## Not run: get_theme_status("invalid_token", "blood_bank")

# throws an error with error message and status code
## Not run: get_theme_status(token, "invalid_theme")
```

`get_token` *Extract API token from OneMap.Sg*

Description

This function is a wrapper for the OneMap Authentication Service API. It allows users to generate a API token from OneMap.Sg. Using the API requires that users have a registered email address with Onemap.Sg. Users can register themselves using OneMap.Sg's form.

Usage

```
get_token(email, password, hide_message = FALSE)
```

Arguments

`email`	User's registered email address.
`password`	User's password.
`hide_message`	Default = `FALSE`. Whether to hide message telling user when the token expires.

Value

API token, or NULL if an error occurs. If error occurs, a warning message will be printed with the error code.

Examples

```
## Not run: get_token("user@example.com",  "password")
```

---

get_travel *Get Travel Time, Distance and Route*

Description

This function is a wrapper for the Route Service API. It takes in a dataframe of start and end coordinates and returns the same dataframe with total time, total distance and optionally route geometry. The function also accepts multiple arguments for ‘route‘ and ‘pt_mode‘, allowing users to compare various route options.

Note that if ‘as_wide = TRUE‘ is selected, any columns with identical names as the additional output columns will be overwritten. Also, if `as_wide = TRUE`, only unique pairs of start and end points should be used. Regardless, using only unique pairs and joining data back is also a generally recommended workflow to reduce computation time.

Usage

```
get_travel(
  token,
  df,
  origin_lat,
  origin_lon,
  destination_lat,
  destination_lon,
  routes,
  date = format(Sys.Date(), "%m-%d-%Y"),
  time = format(Sys.time(), format = "%T"),
  pt_mode = "TRANSIT",
  pt_max_dist = NULL,
  as_wide = TRUE,
  parallel = FALSE,
  route_geom = FALSE
)
```

Arguments

`token`	User’s API token. This can be retrieved using `get_token`
`df`	The input dataframe of start and end coordinates (the dataframe can have additional variables)
`origin_lat`	Name of the dataframe column with the start point latitude.
`origin_lon`	Name of the dataframe column with the start point longitude.
`destination_lat`	Name of the dataframe column with the end point latitude.

`destination_lon`	Name of the dataframe column with the end point longitude.
`routes`	Vector of the types of routes desired. Accepted values are `walk`, `drive`, `pt` (public transport), or `cycle`
`date`	Default = current date. Date for which route is requested.
`time`	Default = current time. Time for which route is requested.
`pt_mode`	Vector of public transport modes required. Default = `route` = `c("transit")`. Accepted values are `transit`, `bus` or `rail`
`pt_max_dist`	Optional if `route` = `"pt"`. Maximum walking distance
`as_wide`	Default = `TRUE`. Whether to return output as a list as a long tibble with each row a route, or a wide tibble with the same number of rows as the input tibble.
`parallel`	Default = `FALSE`. Whether to run API calls in parallel or sequentially (default).
`route_geom`	Default = `FALSE`. Whether to return decoded route_geometry. Will only be returned if `as_wide` = `FALSE`. Please ensure packages `googlePolylines` and `sf` are installed and note that this is a lossy conversion.

Value

Original dataframe with total time and total distance for each route type.

If an error occurs, the output row will be have `NA`s for the additional variables, along with a warning message.

Examples

```
# sample dataframe
sample <- data.frame(start_lat = c(1.3746617, 1.3567797, 1.3361976, 500),
    start_lon = c(103.8366159, 103.9347695, 103.6957732, 501),
    end_lat = c(1.429443081, 1.380298287, 1.337586882, 601),
    end_lon = c(103.835005, 103.7452918, 103.6973215, 600),
    add_info = c("a", "b", "c", "d"))

# no error, wide format
## Not run: get_travel(token, sample[1:3, ],
    "start_lat", "start_lon", "end_lat", "end_lon",
    routes = c("cycle", "walk"))
## End(Not run)
## Not run: get_travel(token, sample[1:3, ],
    "start_lat", "start_lon", "end_lat", "end_lon",
    routes = c("drive", "pt"), pt_mode = c("bus", "transit"))
## End(Not run)

# no error, long format
## Not run: get_travel(token, sample[1:3, ],
    "start_lat", "start_lon", "end_lat", "end_lon",
    routes = c("walk", "pt"), pt_mode = c("bus", "transit"),
    as_wide = FALSE)
## End(Not run)
```

```
# no error, sf dataframe
## Not run: get_travel(token, sample[1:3, ],
    "start_lat", "start_lon", "end_lat", "end_lon",
    routes = c("drive", "pt"), pt_mode = c("bus", "transit"),
    as_wide = FALSE, route_geom = TRUE)
## End(Not run)

# with error
# warning message will show start/end/route/pt_mode for which an error occurred
## Not run: get_travel(token, sample,
    "start_lat", "start_lon", "end_lat", "end_lon",
    routes = c("cycle", "walk"))
## End(Not run)
```

---

search_geo *Get Location Data from keyword*

Description

This function is a wrapper for the Search API. It allows for geocoding of data such as postal codes or address information. This is an internal function for the geocode_onemap function.

Usage

```
search_geo(search_val = NULL, return_geom = FALSE, address_details = FALSE)
```

Arguments

search_val	Keyword(s) to be geocoded, e.g. column of postal codes
return_geom	Default = FALSE. Whether to return the coordinate information
address_details	Default = FALSE. Whether to return address information

---

search_themes *Search for Themes available on OneMap.Sg*

Description

This function is a wrapper for the Get All Themes Info API. It allows users to get a tibble of all available themes, and their details, in the OneMap.Sg API. It also provides an additional functionality where users can subset their results using search terms.

Usage

```
search_themes(token, ..., more_info = FALSE)
```

Arguments

`token`	User's API token. This can be retrieved using `get_token`
`...`	Optional Search terms to subset results; results with any of search terms will be returned. Search terms are not case-sensitive.
`more_info`	Whether more information should be queried, default = `FALSE`. If `FALSE`, output will contain Theme Name, Query Name and Icon information. If `TRUE`, output will additionally contain Category and Theme Owner information.

Value

If no error occurs, a tibble with the following variables:

THEMENAME Name of the Theme

QUERYNAME Query name of the Theme

ICON Name of image file used as Icon in OneMap Web Map

EXPIRY_DATE Expiry Date of the Theme

PUBLISHED_DATE Published Date of the Theme

CATEGORY Returned only if `more_info = TRUE`. Topic that Theme relates to, e.g. Health, Sports, Environment, etc.

THEME_OWNER Returned only if `more_info = TRUE`. Government Agency who Owns the Dataset

If an error occurs, the function throws an error with error message and status code.

Examples

```
# valid
## Not run: search_themes(token)
## Not run: search_themes(token, "hdb", "parks")
## Not run: search_themes(token, more_info = TRUE)

# error
## Not run: search_themes("my_invalid_token")
```

Index