Package 'googleway'

June 25, 2017

Type Package

Title Accesses Google Maps APIs to Retrieve Data and Plot Maps

Version 2.2.0

Date 2017-06-25

Description Provides a mechanism to plot a Google Map from R and overlay it with shapes and markers. Also provides access to Google Maps APIs, including places, directions, roads, distances, geocoding, elevation and timezone.

License GPL-3

LazyData TRUE

Depends R (>= 3.3.1)

Imports Rcpp (>= 0.12.5), jsonlite (>= 0.9.20), curl, htmlwidgets, htmltools, magrittr, shiny, jpeg

LinkingTo Rcpp

RoxygenNote 5.0.1

BugReports https://github.com/SymbolixAU/googleway/issues

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation yes

Author David Cooley [aut, cre],
Paulo Barcelos [ctb] (Author of c++ decode_pl),
Rstudio [ctb] (Functions written for the Leaflet pacakge)

Maintainer David Cooley <dcooley@symbolix.com.au>

Repository CRAN

Date/Publication 2017-06-25 07:26:01 UTC

R topics documented:

add_bicycling . . . 3
add_circles . . . 3
add_fusion . . . 5
add_heatmap . . . 6
add_kml . . . 8
add_markers . . . 9
add_overlay . . . 10
add_polygons . . . 11
add_polylines . . . 14
add_rectangles . . . 17
add_traffic . . . 19
add_transit . . . 19
clear . . . 20
clear_search . . . 21
decode_pl . . . 21
encode_pl . . . 22
get_route . . . 23
google_directions . . . 23
google_dispatch . . . 26
google_distance . . . 27
google_elevation . . . 28
google_geocode . . . 30
google_map . . . 31
google_map-shiny . . . 33
google_map_update . . . 34
google_nearestRoads . . . 35
google_places . . . 36
google_place_autocomplete . . . 39
google_place_details . . . 40
google_reverse_geocode . . . 41
google_snapToRoads . . . 42
google_speedLimits . . . 44
google_streetview . . . 44
google_timezone . . . 46
map_styles . . . 47
melbourne . . . 48
tram_route . . . 49
tram_stops . . . 49
update_circles . . . 50
update_heatmap . . . 51
update_polygons . . . 51
update_polylines . . . 53
update_rectangles . . . 55
update_style . . . 56
%>% . . . 57

Index **58**

add_bicycling *Add bicycling*

Description

Adds bicycle route information to a googleway map object

Usage

```
add_bicycling(map)
```

Arguments

map	a googleway map object created from google_map()

Examples

```
## Not run:

google_map(key = "your_api_key") %>%
  add_bicycling()


## End(Not run)
```

add_circles *Add circle*

Description

Add circles to a google map

Usage

```
add_circles(map, data = get_map_data(map), id = NULL, lat = NULL,
  lon = NULL, radius = NULL, draggable = NULL, stroke_colour = NULL,
  stroke_opacity = NULL, stroke_weight = NULL, fill_colour = NULL,
  fill_opacity = NULL, mouse_over = NULL, mouse_over_group = NULL,
  info_window = NULL, layer_id = NULL, z_index = NULL, digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least two columns, one specifying the latitude coordinates, and the other specifying the longitude. If Null, the data passed into google_map() will be used.
id	string specifying the column containing an identifier for a circle
lat	string specifying the column of data containing the 'latitude' coordinates. If left NULL, a best-guess will be made
lon	string specifying the column of data containing the 'longitude' coordinates. If left NULL, a best-guess will be made
radius	either a string specifying the column of data containing the radius of each circle, OR a numeric value specifying the radius of all the circles (radius is expressed in metres)
draggable	string specifying the column of data defining if the circle is 'draggable' (either TRUE or FALSE)
stroke_colour	either a string specifying the column of data containing the stroke colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the circles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
stroke_weight	either a string specifying the column of data containing the stroke weight of each circle, or a number indicating the width of pixels in the line to be applied to all the circles
fill_colour	either a string specifying the column of data containing the fill colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the cirlces
fill_opacity	either a string specifying the column of data containing the fill opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
mouse_over	string specifying the column of data to display when the mouse rolls over the circle
mouse_over_group	string specifying the column of data specifying which groups of circles to highlight on mouseover
info_window	string specifying the column of data to display in an info window when a circle is clicked
layer_id	single value specifying an id for the layer. layer.
z_index	single value specifying where the circles appear in the layering of the map objects. Layers with a higher z_index appear on top of those with a lower z_index. See details.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Details

z_index values define the order in which objects appear on the map. Those with a higher value appear on top of those with a lower value. The default order of objects is (1 being underneath all other objects)

- 1. Polygon
- 2. Rectangle
- 3. Polyline
- 4. Circle

Markers are always the top layer

Examples

```
## Not run:

google_map(key = map_key, data = tram_stops) %>%
 add_circles(lat = "stop_lat", lon = "stop_lon")


## End(Not run)
```

---

add_fusion *Add Fusion*

Description

Adds a fusion table layer to a map.

Usage

```
add_fusion(map, query, styles = NULL, heatmap = FALSE, layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
query	a data.frame of 2 or 3 columns, and only 1 row. Two columns must be 'select' and 'from', and the third 'where'. The 'select' value is the column name (from the fusion table) containing the location information, and the 'from' value is the encrypted table Id. The 'where' value is a string specifying the 'where' condition on the data query.
styles	a list object used to apply colour, stroke weight and opacity to lines and polygons. See examples to see how the list should be constructed.
heatmap	logical indicating whether to show a heatmap.
layer_id	single value specifying an id for the layer.

Examples

```
## Not run:

mapKey <- 'your_api_key'

qry <- data.frame(select = 'address',
    from = '1d7qpn60tAvG4LEg4jvClZbc1ggp8fIGGvpMGzA',
    where = 'ridership > 200')

google_map(key = mapKey, location = c(41.8, -87.7), zoom = 9) %>%
  add_fusion(query = qry)



qry <- data.frame(select = 'geometry',
   from = '1ertEwm-1bMBhpEwHhtNYT47HQ9k2ki_6sRa-UQ')

styles <- list(
  list(
    polygonOptions = list( fillColor = "#00FF00", fillOpacity = 0.3)
    ),
  list(
    where = "birds > 300",
    polygonOptions = list( fillColor = "#0000FF" )
    ),
  list(
    where = "population > 5",
    polygonOptions = list( fillOpacity = 1.0 )
 )
)

google_map(key = mapKey, location = c(-25.3, 133), zoom = 4) %>%
  add_fusion(query = qry, styles = styles)

qry <- data.frame(select = 'location',
    from = '1xWyeuAhIFK_aED1ikkQEGmR8mINSCJO9Vq-BPQ')

google_map(key = mapKey, location = c(0, 0), zoom = 1) %>%
  add_fusion(query = qry, heatmap = T)


## End(Not run)
```

---

add_heatmap *Add heatmap*

Description

Adds a heatmap to a google map

Usage

```
add_heatmap(map, data = get_map_data(map), lat = NULL, lon = NULL,
  weight = NULL, option_gradient = NULL, option_dissipating = FALSE,
  option_radius = 0.01, option_opacity = 0.6, layer_id = NULL,
  digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least two columns, one specifying the latitude coordinates, and the other specifying the longitude. If Null, the data passed into google_map() will be used.
lat	string specifying the column of data containing the 'latitude' coordinates. If left NULL, a best-guess will be made
lon	string specifying the column of data containing the 'longitude' coordinates. If left NULL, a best-guess will be made
weight	string specifying the column of data containing the 'weight' associated with each point. If NULL, each point will get a weight of 1.
option_gradient	vector of colours to use as the gradient colours. see Details
option_dissipating	logical Specifies whether heatmaps dissipate on zoom. When dissipating is FALSE the radius of influence increases with zoom level to ensure that the color intensity is preserved at any given geographic location. Defaults to FALSE
option_radius	numeric. The radius of influence for each data point, in pixels.
option_opacity	The opacity of the heatmap, expressed as a number between 0 and 1. Defaults to 0.6.
layer_id	single value specifying an id for the layer.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Details

option_gradient colours can be two of the R colour specifications; either a colour name (as listed by colors(), or a hexadecimal string of the form "#rrggbb"). The first colour in the vector will be used as the colour that fades to transparent, while the last colour in the vector will be use in the centre of the 'heat'.

Examples

```
## Not run:

map_key <- 'your_api_key'

set.seed(20170417)
df <- tram_route
```

```
df$weight <- sample(1:10, size = nrow(df), replace = T)

google_map(key = map_key, data = df) %>%
 add_heatmap(lat = "shape_pt_lat", lon = "shape_pt_lon", weight = "weight",
              option_radius = 0.001)

## specifying different colour gradient
option_gradient <- c('orange', 'blue', 'mediumpurple4', 'snow4', 'thistle1')

google_map(key = map_key, data = df) %>%
 add_heatmap(lat = "shape_pt_lat", lon = "shape_pt_lon", weight = "weight",
              option_radius = 0.001, option_gradient = option_gradient)


## End(Not run)
```

---

add_kml *Add KML*

Description

Adds a KML layer to a map.

Usage

```
add_kml(map, kml_url, layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
kml_url	URL string specifying the location of the kml layer
layer_id	single value specifying an id for the layer.

Examples

```
## Not run:

map_key <- 'your_api_key'

kmlUrl <- paste0('https://developers.google.com/maps/',
'documentation/javascript/examples/kml/westcampus.kml')

google_map(key = map_key) %>%
  add_kml(kml_url = kmlUrl)


## End(Not run)
```

add_markers *Add markers*

Description

Add markers to a google map

Usage

```
add_markers(map, data = get_map_data(map), id = NULL, colour = NULL,
  lat = NULL, lon = NULL, title = NULL, draggable = NULL,
  opacity = NULL, label = NULL, cluster = FALSE, info_window = NULL,
  mouse_over = NULL, mouse_over_group = NULL, marker_icon = NULL,
  layer_id = NULL, digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least two columns, one specifying the latitude coordinates, and the other specifying the longitude. If Null, the data passed into google_map() will be used.
id	string specifying the column containing an identifier for a marker
colour	string specifying the column containing the 'colour' to use for the markers. One of 'red', 'blue', 'green' or 'lavender'.
lat	string specifying the column of data containing the 'latitude' coordinates. If left NULL, a best-guess will be made
lon	string specifying the column of data containing the 'longitude' coordinates. If left NULL, a best-guess will be made
title	string specifying the column of data containing the 'title' of the markers. The title is displayed when you hover over a marker. If blank, no title will be displayed for the markers.
draggable	string specifying the column of data defining if the marker is 'draggable' (either TRUE or FALSE)
opacity	string specifying the column of data defining the 'opacity' of the maker. Values must be between 0 and 1 (inclusive).
label	string specifying the column of data defining the character to appear in the centre of the marker. Values will be coerced to strings, and only the first character will be used.
cluster	logical indicating if co-located markers should be clustered when the map zoomed out
info_window	string specifying the column of data to display in an info window when a marker is clicked
mouse_over	string specifying the column of data to display when the mouse rolls over the marker

mouse_over_group
: string specifying the column of data specifying which groups of circles to highlight on mouseover

marker_icon
: string specifying the column of data containing a link/URL to an image to use for a marker

layer_id
: single value specifying an id for the layer.

digits
: integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Examples

```
## Not run:

map_key <- "your api key"

google_map(key = map_key, data = tram_stops) %>%
 add_markers(lat = "stop_lat", lon = "stop_lon", info_window = "stop_name")


## using marker icons
iconUrl <- paste0("https://developers.google.com/maps/documentation/",
"javascript/examples/full/images/beachflag.png")

tram_stops$icon <- iconUrl

google_map(key = map_key, data = tram_stops) %>%
  add_markers(lat = "stop_lat", lon = "stop_lon", marker_icon = "icon")


## End(Not run)
```

---

add_overlay *Add Overlay*

Description

Adds a ground overlay to a map. The overlay can only be added from a URL

Usage

```
add_overlay(map, north, east, south, west, overlay_url, layer_id = NULL,
  digits = 4)
```

Arguments

map
: a googleway map object created from google_map()

north
: northern-most latitude coordinate

east
: eastern-most longitude

south	southern-most latitude coordinate
west	western-most longitude
overlay_url	URL string specifying the location of the overlay layer
layer_id	single value specifying an id for the layer.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Examples

```
## Not run:

map_key <- 'your_api_key'

google_map(key = map_key) %>%
  add_overlay(north = 40.773941,south = 40.712216, east = -74.12544, west = -74.22655,
            overlay_url = "https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg")



## End(Not run)
```

---

add_polygons *Add polygon*

Description

Add a polygon to a google map.

Usage

```
add_polygons(map, data = get_map_data(map), polyline = NULL, lat = NULL,
  lon = NULL, id = NULL, pathId = NULL, stroke_colour = NULL,
  stroke_weight = NULL, stroke_opacity = NULL, fill_colour = NULL,
  fill_opacity = NULL, info_window = NULL, mouse_over = NULL,
  mouse_over_group = NULL, draggable = NULL, editable = NULL,
  update_map_view = TRUE, layer_id = NULL, z_index = NULL, digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least a polyline column, or a lat and a lon column. If Null, the data passed into google_map() will be used.
polyline	string specifying the column of data containing the encoded polyline
lat	string specifying the column of data containing the 'latitude' coordinates. Co-ordinates must be in the order that defines the path.

lon	string specifying the column of data containing the 'longitude' coordinates. Coordinates must be in the order that defines the path.
id	string specifying the column containing an identifier for a polygon.
pathId	string specifying the column containing an identifer for each path that forms the complete polygon. Not required when using polyline, as each polyline is itself a path.
stroke_colour	either a string specifying the column of data containing the stroke colour of each polygon, or a valid hexadecimal numeric HTML style to be applied to all the polygons
stroke_weight	either a string specifying the column of data containing the stroke weight of each polygon, or a number indicating the width of pixels in the line to be applied to all the polygons
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each polygon, or a value between 0 and 1 that will be applied to all the polygons
fill_colour	either a string specifying the column of data containing the fill colour of each polygon, or a valid hexadecimal numeric HTML style to be applied to all the polygons
fill_opacity	either a string specifying the column of data containing the fill opacity of each polygon, or a value between 0 and 1 that will be applied to all the polygons
info_window	string specifying the column of data to display in an info window when a polygon is clicked
mouse_over	string specifying the column of data to display when the mouse rolls over the polygon
mouse_over_group	string specifying the column of data specifying which groups of polygons to highlight on mouseover
draggable	string specifying the column of data defining if the polygon is 'draggable'. The column of data should be logical (either TRUE or FALSE)
editable	string specifying the column of data defining if the polygon is 'editable' (either TRUE or FALSE)
update_map_view	logical specifying if the map should re-centre according to the polyline.
layer_id	single value specifying an id for the layer.
z_index	single value specifying where the polygons appear in the layering of the map objects. Layers with a higher z_index appear on top of those with a lower z_index. See details.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Details

z_index values define the order in which objects appear on the map. Those with a higher value appear on top of those with a lower value. The default order of objects is (1 being underneath all other objects)

- 1. Polygon
- 2. Rectangle
- 3. Polyline
- 4. Circle

Markers are always the top layer

Note

A polygon represents an area enclosed by a closed path. Polygon objects are similar to polylines in that they consist of a series of coordinates in an ordered sequence. Polygon objects can describe complex shapes, including

- Multiple non-contiguous areas defined by a single polygon
- Areas with holes in them
- Intersections of one or more areas

To define a complex shape, you use a polygon with multiple paths.

To create a hole in a polygon, you need to create two paths, one inside the other. To create the hole, the coordinates of the inner path must be wound in the opposite order to those defining the outer path. For example, if the coordinates of the outer path are in clockwise order, then the inner path must be anti-clockwise.

You can represent a polygon in one of three ways

- as a series of coordinates defining a path (or paths) with both an `id` and `pathId` argument that make up the polygon
- as an encoded polyline using an `id` column to specify multiple polylines for a polygon
- as a list column in a data.frame, where each row of the data.frame contains the polylines that comprise the polygon

See Examples

See Also

encode_pl

Examples

```
## Not run:

map_key <- 'your_api_key'

## polygon with a hole - Bermuda triangle
## using one row per polygon, and a list-column of encoded polylines
pl_outer <- encode_pl(lat = c(25.774, 18.466,32.321),
      lon = c(-80.190, -66.118, -64.757))

pl_inner <- encode_pl(lat = c(28.745, 29.570, 27.339),
       lon = c(-70.579, -67.514, -66.668))
```

```
df <- data.frame(id = c(1, 1),
       polyline = c(pl_outer, pl_inner),
       stringsAsFactors = FALSE)

df <- aggregate(polyline ~ id, data = df, list)

google_map(key = map_key, height = 800) %>%
    add_polygons(data = df, polyline = "polyline")

## the same polygon, but using an 'id' to specify the polygon
df <- data.frame(id = c(1,1),
       polyline = c(pl_outer, pl_inner),
       stringsAsFactors = FALSE)

google_map(key = map_key, height = 800) %>%
    add_polygons(data = df, polyline = "polyline", id = "id")

## the same polygon, specified using coordinates, and with a second independent
## polygon
df <- data.frame(myId = c(1,1,1,1,1,1,2,2,2),
      lineId = c(1,1,1,2,2,2,1,1,1),
      lat = c(26.774, 18.466, 32.321, 28.745, 29.570, 27.339, 22, 23, 22),
      lon = c(-80.190, -66.118, -64.757, -70.579, -67.514, -66.668, -50, -49, -51),
      colour = c(rep("#00FF0F", 6), rep("#FF00FF", 3)),
      stringsAsFactors = FALSE)

google_map(key = map_key) %>%
  add_polygons(data = df, lat = 'lat', lon = 'lon', id = 'myId', pathId = 'lineId',
               fill_colour = 'colour')




## End(Not run)
```

---

add_polylines *Add polyline*

Description

Add a polyline to a google map

Usage

```
add_polylines(map, data = get_map_data(map), polyline = NULL, lat = NULL,
  lon = NULL, id = NULL, geodesic = NULL, stroke_colour = NULL,
  stroke_weight = NULL, stroke_opacity = NULL, info_window = NULL,
  mouse_over = NULL, mouse_over_group = NULL, update_map_view = TRUE,
  layer_id = NULL, z_index = NULL, digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least a polyline column, or a lat and a lon column. If Null, the data passed into google_map() will be used.
polyline	string specifying the column of data containing the encoded 'polyline'.
lat	string specifying the column of data containing the 'latitude' coordinates. Coordinates must be in the order that defines the path.
lon	string specifying the column of data containing the 'longitude' coordinates. Coordinates must be in the order that defines the path.
id	string specifying the column containing an identifier for a polyline
geodesic	logical
stroke_colour	either a string specifying the column of data containing the stroke colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the circles
stroke_weight	either a string specifying the column of data containing the stroke weight of each circle, or a number indicating the width of pixels in the line to be applied to all the circles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
info_window	string specifying the column of data to display in an info window when a polyline is clicked
mouse_over	string specifying the column of data to display when the mouse rolls over the polyline
mouse_over_group	string specifying the column of data specifying which groups of polylines to highlight on mouseover
update_map_view	logical specifying if the map should re-centre according to the polyline.
layer_id	single value specifying an id for the layer.
z_index	single value specifying where the polylines appear in the layering of the map objects. Layers with a higher z_index appear on top of those with a lower z_index. See details.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Details

z_index values define the order in which objects appear on the map. Those with a higher value appear on top of those with a lower value. The default order of objects is (1 being underneath all other objects)

- 1. Polygon
- 2. Rectangle

- 3. Polyline
- 4. Circle

Markers are always the top layer

Note

The lines can be generated by either using an encoded polyline, or by a set of lat/lon coordinates. You sould specify either the column containing an encoded polyline, OR the lat / lon colulmns.

Using `update_map_view = TRUE` for multiple polylines may be slow, so it may be more appropriate to set the view of the map using the location argument of `google_map()`

Examples

```
## Not run:

## using lat/lon coordinates

map_key <- "your_api_key"

google_map(data = tram_route, key = map_key) %>%
  add_polylines(lat = "shape_pt_lat", lon = "shape_pt_lon")


## using encoded polyline and various colour / fill options
url <- 'https://raw.githubusercontent.com/plotly/datasets/master/2011_february_aa_flight_paths.csv'
flights <- read.csv(url)
flights$id <- seq_len(nrow(flights))


## encode the routes as polylines
lst <- lapply(unique(flights$id), function(x){
  lat = c(flights[flights["id"] == x, c("start_lat")], flights[flights["id"] == x, c("end_lat")])
  lon = c(flights[flights["id"] == x, c("start_lon")], flights[flights["id"] == x, c("end_lon")])
  data.frame(id = x, polyline = encode_pl(lat = lat, lon = lon))
})

flights <- merge(flights, do.call(rbind, lst), by = "id")

style <- map_styles()$night

google_map(key = map_key, style = style) %>%
  add_polylines(data = flights, polyline = "polyline", mouse_over_group = "airport1",
                stroke_weight = 1, stroke_opacity = 0.3, stroke_colour = "#ccffff")



## End(Not run)
```

add_rectangles *Add Rectangles*

Description

Adds a rectangle to a google map

Usage

```
add_rectangles(map, data = get_map_data(map), north, east, south, west,
  id = NULL, draggable = NULL, editable = NULL, stroke_colour = NULL,
  stroke_opacity = NULL, stroke_weight = NULL, fill_colour = NULL,
  fill_opacity = NULL, mouse_over = NULL, mouse_over_group = NULL,
  info_window = NULL, layer_id = NULL, z_index = NULL, digits = 4)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing the bounds for the rectangles
north	String specifying the column of data that contains the northern most latitude coordinate
east	String specifying the column of data that contains the eastern most longitude
south	String specifying the column of data that contains the southern most latitude coordinate
west	String specifying the column of data that contains the western most longitude
id	string specifying the column containing an identifier for a rectangle
draggable	string specifying the column of data defining if the rectangle is 'draggable' (either TRUE or FALSE)
editable	string specifying the column of data defining if the rectangle is 'editable' (either TRUE or FALSE)
stroke_colour	either a string specifying the column of data containing the stroke colour of each rectangle, or a valid hexadecimal numeric HTML style to be applied to all the rectangle
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each rectangle, or a value between 0 and 1 that will be applied to all the rectangle
stroke_weight	either a string specifying the column of data containing the stroke weight of each rectangle, or a number indicating the width of pixels in the line to be applied to all the rectangle
fill_colour	either a string specifying the column of data containing the fill colour of each rectangle, or a valid hexadecimal numeric HTML style to be applied to all the rectangle
fill_opacity	either a string specifying the column of data containing the fill opacity of each rectangle, or a value between 0 and 1 that will be applied to all the rectangles

mouse_over	string specifying the column of data to display when the mouse rolls over the rectangle
mouse_over_group	string specifying the column of data specifying which groups of rectangle to highlight on mouseover
info_window	string specifying the column of data to display in an info window when a rectangle is clicked
layer_id	single value specifying an id for the layer.
z_index	single value specifying where the rectangles appear in the layering of the map objects. Layers with a higher z_index appear on top of those with a lower z_index. See details.
digits	integer. Use this parameter to specify how many digits (decimal places) should be used for the latitude / longitude coordinates.

Details

z_index values define the order in which objects appear on the map. Those with a higher value appear on top of those with a lower value. The default order of objects is (1 being underneath all other objects)

- 1. Polygon
- 2. Rectangle
- 3. Polyline
- 4. Circle

Markers are always the top layer

Examples

```
## Not run:

map_key <- 'your_api_key'

df <- data.frame(north = 33.685, south = 33.671, east = -116.234, west = -116.251)

google_map(key = map_key) %>%
  add_rectangles(data = df, north = 'north', south = 'south',
                 east = 'east', west = 'west')

## editable rectangle
df <- data.frame(north = -37.8459, south = -37.8508, east = 144.9378,
                  west = 144.9236, editable = T, draggable = T)

google_map(key = map_key) %>%
  add_rectangles(data = df, north = 'north', south = 'south',
                 east = 'east', west = 'west',
                 editable = 'editable', draggable = 'draggable')


## End(Not run)
```

add_traffic *Add Traffic*

Description

Adds live traffic information to a googleway map object

Usage

```
add_traffic(map)
```

Arguments

map	a googleway map object created from google_map()

Examples

```
## Not run:

google_map(key = "your_api_key") %>%
  add_traffic()


## End(Not run)
```

add_transit *Add transit*

Description

Adds public transport information to a googleway map object

Usage

```
add_transit(map)
```

Arguments

map	a googleway map object created from google_map()

Examples

```
## Not run:

google_map(key = "your_api_key") %>%
  add_transit()


## End(Not run)
```

clear *clear map elements*

Description

clears elements from a map

Usage

```
clear_markers(map, layer_id = NULL)

clear_circles(map, layer_id = NULL)

clear_heatmap(map, layer_id = NULL)

clear_traffic(map)

clear_transit(map)

clear_bicycling(map)

clear_polylines(map, layer_id = NULL)

clear_polygons(map, layer_id = NULL)

clear_rectangles(map, layer_id = NULL)

clear_fusion(map, layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
layer_id	id value of the layer to be removed from the map

Note

These operations are intended for use in conjunction with google_map_update in an interactive shiny environment

clear_search *Clear search*

Description

clears the markers placed on the map after using the search box

Usage

```
clear_search(map)
```

Arguments

map	a googleway map object created from google_map()

decode_pl *Decode PL*

Description

Decodes an encoded polyline into the series of lat/lon coordinates that specify the path

Usage

```
decode_pl(encoded)
```

Arguments

encoded	String. An encoded polyline

Value

data.frame of lat/lon coordinates

Note

An encoded polyline is generated from google's polyline encoding algorithm (https://developers.google.com/maps/documentation/utilities/polylinealgorithm).

See Also

encode_pl, google_directions

Examples

```
## polyline joining the capital cities of Australian states
pl <- "nnseFmpzsZgalNytrXetrG}krKsaif@kivIccvzAvvqfClp~uBlymzA~ocQ}_}iCthxo@srst@"

df_polyline <- decode_pl(pl)
df_polyline
```

---

encode_pl *Encode PL*

Description

Encodes a series of lat/lon coordinates that specify a path into an encoded polyline

Usage

```
encode_pl(lat, lon)
```

Arguments

lat	vector of latitude coordinates
lon	vector of longitude coordinates

Value

string encoded polyline

Note

An encoded polyline is generated from google's polyline encoding algorithm (https://developers.google.com/maps/documentation/utilities/polylinealgorithm).

See Also

decode_pl

Examples

```
encode_pl(lat = c(38.5, 40.7, 43.252), lon = c(-120.2, -120.95, -126.453))
## "_p~iF~ps|U_ulLnnqC_mqNvxq`@"
```

get_route *GetRoute*

Description

This function is deprecated. Please use google_directions

Usage

```
get_route(...)
```

Arguments

...	arguments passed to google_directions. Legacy - so that old 'get_route() calls will still enter this function.

google_directions *Google Directions*

Description

The Google Maps Directions API is a service that calculates directions between locations. You can search for directions for several modes of transportation, including transit, driving, walking, or cycling.

Usage

```
google_directions(origin, destination, mode = c("driving", "walking",
  "bicycling", "transit"), departure_time = NULL, arrival_time = NULL,
  waypoints = NULL, optimise_waypoints = FALSE, alternatives = FALSE,
  avoid = NULL, units = c("metric", "imperial"), traffic_model = NULL,
  transit_mode = NULL, transit_routing_preference = NULL, language = NULL,
  region = NULL, key, simplify = TRUE, curl_proxy = NULL)
```

Arguments

origin	numeric vector of lat/lon coordinates, or an address string
destination	numeric vector of lat/lon coordinates, or an address string
mode	string. One of 'driving', 'walking', 'bicycling' or 'transit'.
departure_time	POSIXct. Specifies the desired time of departure. Must be in the future (i.e. greater than sys.time()). If no value is specified it defaults to Sys.time()
arrival_time	POSIXct. Specifies the desired time of arrival. Note you can only specify one of arrival_time or departure_time, not both. If both are supplied, departure_time will be used.

waypoints — list of waypoints, expressed as either a vector of lat/lon coordinates, or a string address to be geocoded. Only available for driving, walking or bicycling modes. List elements must be named either 'stop' or 'via', where 'stop' is used to indicate a stopover for a waypoint, and 'via' will not stop at the waypoint. See https://developers.google.com/maps/documentation/directions/intro#Waypoints for details

optimise_waypoints — boolean allow the Directions service to optimize the provided route by rearranging the waypoints in a more efficient order. (This optimization is an application of the Travelling Salesman Problem.) Travel time is the primary factor which is optimized, but other factors such as distance, number of turns and many more may be taken into account when deciding which route is the most efficient. All waypoints must be stopovers for the Directions service to optimize their route.

alternatives — logical If set to true, specifies that the Directions service may provide more than one route alternative in the response

avoid — character vector stating which features should be avoided. One of 'tolls', 'highways', 'ferries' or 'indoor'

units — string metric or imperial. Note: Only affects the text displayed within the distance field. The values are always in metric

traffic_model — string - one of 'best_guess', 'pessimistic' or 'optimistic'. Only valid with a departure time

transit_mode — vector of strings, either 'bus', 'subway', 'train', 'tram' or 'rail'. Only vaid where mode = 'transit'. Note that 'rail' is equivalent to transit_mode=c("train", "tram", "subwa

transit_routing_preference — vector of strings - one of 'less_walking' and 'fewer_transfers'. specifies preferences for transit routes. Only valid for transit directions.

language — string - specifies the language in which to return the results. See the list of supported languages: https://developers.google.com/maps/faq#using-google-maps-apis If no langauge is supplied, the service will attempt to use the language of the domain from which the request was sent

region — string - specifies the region code, specified as a ccTLD ("top-level domain"). See region basing for details https://developers.google.com/maps/documentation/directions/intro#RegionBiasing

key — string - a valid Google Developers Directions API key

simplify — logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string

curl_proxy — a curl proxy object

Value

Either list or JSON string of the route between origin and destination

Examples

```
## Not run:
## using lat/long coordinates
```

```
google_directions(origin = c(-37.8179746, 144.9668636),
          destination = c(-37.81659, 144.9841),
          mode = "walking",
          key = "<your valid api key>")


## using address string
google_directions(origin = "Flinders Street Station, Melbourne",
         destination = "MCG, Melbourne",
         mode = "walking",
         key = "<your valid api key>")


google_directions(origin = "Melbourne Airport, Australia",
         destination = "Portsea, Melbourne, Australia",
         departure_time =  Sys.time() + (24 * 60 * 60),
         waypoints = list(c(-37.81659, 144.9841),
                          via = "Ringwood, Victoria"),
         mode = "driving",
         alternatives = FALSE,
         avoid = c("TOLLS", "highways"),
         units = "imperial",
         key = "<your valid api key>",
         simplify = TRUE)

## using bus and less walking
google_directions(origin = "Melbourne Airport, Australia",
         destination = "Portsea, Melbourne, Australia",
         departure_time =  Sys.time() + (24 * 60 * 60),
         mode = "transit",
         transit_mode = "bus",
         transit_routing_preference = "less_walking",
         key = "<your valid api key>",
         simplify = FALSE)

## using arrival time
google_directions(origin = "Melbourne Airport, Australia",
         destination = "Portsea, Melbourne, Australia",
         arrival_time =  Sys.time() + (24 * 60 * 60),
         mode = "transit",
         transit_mode = "bus",
         transit_routing_preference = "less_walking",
         key = "<your valid api key>",
         simplify = FALSE)

## return results in French
google_directions(origin = "Melbourne Airport, Australia",
         destination = "Portsea, Melbourne, Australia",
         arrival_time =  Sys.time() + (24 * 60 * 60),
         mode = "transit",
         transit_mode = "bus",
         transit_routing_preference = "less_walking",
         language = "fr",
```

```
          key = key,
          simplify = FALSE)


## End(Not run)
```

`google_dispatch` *Google dispatch*

Description

Extension points for plugins

Usage

```
google_dispatch(map, funcName, google_map = stop(paste(funcName,
  "requires a map update object")), google_map_update = stop(paste(funcName,
  "does not support map udpate objects")))

invoke_method(map, data, method, ...)
```

Arguments

`map`	a map object, as returned from `google_map`
`funcName`	the name of the function that the user called that caused this `google_dispatch` call; for error message purposes
`google_map`	an action to be performed if the map is from `google_map`
`google_map_update`	an action to be performed if the map is from `google_map_update`
`data`	a data object that will be used when evaluating formulas in `...`
`method`	the name of the JavaScript method to invoke
`...`	unnamed arguments to be passed to the JavaScript method

Value

`google_dispatch` returns the value of `google_map` or or an error. `invokeMethod` returns the `map` object that was passed in, possibly modified.

google_distance *Google Distance*

Description

The Google Maps Distance Matrix API is a service that provides travel distance and time for a matrix of origins and destinations, based on the recommended route between start and end points.

Usage

```
google_distance(origins, destinations, mode = c("driving", "walking",
  "bicycling", "transit"), departure_time = NULL, arrival_time = NULL,
  avoid = NULL, units = c("metric", "imperial"), traffic_model = NULL,
  transit_mode = NULL, transit_routing_preference = NULL, language = NULL,
  key, simplify = TRUE, curl_proxy = NULL)
```

Arguments

origins	list of unnamed elements, each element is either a numeric vector of lat/lon coordinates, or an address string
destinations	list of unnamed elements, each element is either a vector of lat/lon coordinates, or an address string
mode	string One of 'driving', 'walking', 'bicycling' or 'transit'.
departure_time	POSIXct. Specifies the desired time of departure. Must be in the future (i.e. greater than sys.time()). If no value is specified it defaults to Sys.time()
arrival_time	POSIXct. Specifies teh desired time of arrival. Note you can only specify one of arrival_time or departure_time, not both. If both are supplied, departure_time will be used.
avoid	character vector stating which features should be avoided. One of 'tolls', 'highways', 'ferries' or 'indoor'
units	string metric or imperial. Note: Only affects the text displayed within the distance field. The values are always in metric
traffic_model	string. One of 'best_guess', 'pessimistic' or 'optimistic'. Only valid with a departure time
transit_mode	vector of strings, either 'bus', 'subway', 'train', 'tram' or 'rail'. Only vaid where mode = 'transit'. Note that 'rail' is equivalent to transit_mode=c("train", "tram", "subwa
transit_routing_preference	vector strings - one of 'less_walking' and 'fewer_transfers'. specifies preferences for transit routes. Only valid for transit directions.
language	string. Specifies the language in which to return the results. See the list of supported languages: https://developers.google.com/maps/faq#using-google-maps-apis If no langauge is supplied, the service will attempt to use the language of the domain from which the request was sent
key	string. A valid Google Developers Distance API key

simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
curl_proxy	a curl proxy object

Value

Either list or JSON string of the distance between origins and destinations

Examples

```
## Not run:
google_distance(origins = list(c("Melbourne Airport, Australia"),
                               c("MCG, Melbourne, Australia"),
                               c(-37.81659, 144.9841)),
                               destinations = c("Portsea, Melbourne, Australia"),
                               key = "<your valid api key>",
                               simplify = FALSE)




## End(Not run)
```

---

google_elevation *Google elevation*

Description

The Google Maps Elevation API provides elevation data for all locations on the surface of the earth, including depth locations on the ocean floor (which return negative values).

Usage

```
google_elevation(df_locations = NULL, polyline = NULL,
  location_type = c("individual", "path"), samples = NULL, key,
  simplify = TRUE)
```

Arguments

df_locations	data.frame of with two columns called 'lat' and 'lon' (or 'latitude' / 'longitude') used as the locations
polyline	string encoded polyline
location_type	string Specifies the results to be returned as individual locations or as a path. One of 'individual' or 'path'. If 'path', the data.frame df_locations must contain at least two rows. The order of the path is determined by the order of the rows.

samples	integer Required if location_type == "path". Specifies the number of sample points along a path for which to return elevation data. The samples parameter divides the given path into an ordered set of equidistant points along the path.
key	string A valid Google Developers Elevation API key
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string

Details

Locations can be specified as either a data.frame containing both a lat/latitude and lon/longitude column, or a single encoded polyline

Value

Either list or JSON string of the elevation data

Examples

```
## Not run:

## elevation data for the MCG in Melbourne
df <- data.frame(lat = -37.81659,
                 lon = 144.9841)

google_elevation(df_locations = df,
                 key = "<your valid api key>",
                 simplify = TRUE)



## elevation data from the MCG to the beach at Elwood (due south)
df <- data.frame(lat = c(-37.81659, -37.88950),
                 lon = c(144.9841, 144.9841))

df <- google_elevation(df_locations = df,
                       location_type = "path",
                       samples = 20,
                       key = "<your valid api key>",
                       simplify = TRUE)

## plot results
library(ggplot2)
df_plot <- data.frame(elevation = df$results$elevation,
                       location = as.integer(rownames(df$results)))

ggplot(data = df_plot, aes(x = location, y = elevation)) +
 geom_line()

## End(Not run)
```

google_geocode *Google geocoding*

Description

Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which you can use to place markers on a map, or position the map.

Usage

```
google_geocode(address, bounds = NULL, key, language = NULL,
  region = NULL, components = NULL, simplify = TRUE)
```

Arguments

address	string. The street address that you want to geocode, in the format used by the national postal service of the country concerned
bounds	list of two, each element is a vector of lat/lon coordinates representing the south-west and north-east bounding box
key	string. A valid Google Developers Geocode API key
language	string. Specifies the language in which to return the results. See the list of supported languages: https://developers.google.com/maps/faq#using-google-maps-apis. If no langauge is supplied, the service will attempt to use the language of the domain from which the request was sent
region	string. Specifies the region code, specified as a ccTLD ("top-level domain"). See region basing for details https://developers.google.com/maps/documentation/directions/intro#RegionBiasing
components	data.frame of two columns, component and value. Restricts the results to a specific area. One or more of "route","locality","administrative_area", "postal_code","country"
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string

Value

Either list or JSON string of the geocoded address

Examples

```
## Not run:
df <- google_geocode(address = "MCG, Melbourne, Australia",
                     key = "<your valid api key>",
                     simplify = TRUE)

df$results$geometry$location
        lat      lng
```

```
1 -37.81659 144.9841

## using bounds
bounds <- list(c(34.172684,-118.604794),
               c(34.236144,-118.500938))

js <- google_geocode(address = "Winnetka",
                     bounds = bounds,
                     key = "<your valid api key>",
                     simplify = FALSE)

## using components
components <- data.frame(component = c("postal_code", "country"),
                         value = c("3000", "AU"))

df <- google_geocode(address = "Flinders Street Station",
                   key = "<your valid api key>",
                   components = components,
                   simplify = FALSE)


## End(Not run)
```

`google_map` *Google map*

Description

Generates a google map object

Usage

```
google_map(key, data = NULL, location = NULL, zoom = NULL, width = NULL,
  height = NULL, padding = 0, styles = NULL, search_box = FALSE,
  zoom_control = TRUE, map_type_control = TRUE, scale_control = FALSE,
  street_view_control = TRUE, rotate_control = TRUE,
  fullscreen_control = TRUE)
```

Arguments

`key`	A valid Google Maps API key. see Details
`data`	data to be used on the map. This will likely contain two columns for latitude and longitude, and / or encoded polylines for plotting polylines and polygons
`location`	`numeric` vector of latitude/longitude (in that order) coordinates for the initial starting position of the map. The map will automatically set the location and zoom if markers are supplied through add_markers. If null, the map will default to Melbourne, Australia.
`zoom`	`integer` representing the zoom level of the map (0 is fully zoomed out)

width	the width of the map
height	the height of the map
padding	the padding of the map
styles	JSON string representation of a valid Google Maps styles Array. See the Google documentation for details https://developers.google.com/maps/documentation/javascript/styling
search_box	boolean indicating if a search box should be placed on the map
zoom_control	logical
map_type_control	logical
scale_control	logical
street_view_control	logical
rotate_control	logical
fullscreen_control	logical

Details

The data argument is only needed if you call other functions to add layers to the map, such as add_markers() or add_polylines. However, the data argument can also be passed into those functions as well.

In order to use Google Maps you need a valid Google Maps Web JavaScript API key. See the Google Maps API documentation https://developers.google.com/maps/

Examples

```
## Not run:

map_key <- "your_api_key"
df <- structure(list(lat = c(-37.8201904296875, -37.8197288513184,
-37.8191299438477, -37.8187675476074, -37.8186187744141, -37.8181076049805
), lon = c(144.968612670898, 144.968414306641, 144.968139648438,
144.967971801758, 144.967864990234, 144.967636108398), weight = c(31.5698964400217,
97.1629025738221, 58.9051092562731, 76.3215389118996, 37.8982300488278,
77.1501972114202), opacity = c(0.2, 0.2, 0.2, 0.2, 0.2, 0.2)), .Names = c("lat",
"lon", "weight", "opacity"), row.names = 379:384, class = "data.frame")

google_map(key = map_key, data = df_line) %>%
 add_markers() %>%
 add_heatmap() %>%
 add_traffic()

## style map using 'cobalt simplified' style
style <- '[{"featureType":"all","elementType":"all","stylers":[{"invert_lightness":true},
{"saturation":10},{"lightness":30},{"gamma":0.5},{"hue":"#435158"}]},
{"featureType":"road.arterial","elementType":"all","stylers":[{"visibility":"simplified"}]},
{"featureType":"transit.station","elementType":"labels.text","stylers":[{"visibility":"off"}]}]'
```

```
google_map(key = map_key, styles = style)


## End(Not run)
```

---

google_map-shiny *Shiny bindings for google_map*

Description

Output and render functions for using google_map within Shiny applications and interactive Rmd documents.

Usage

```
google_mapOutput(outputId, width = "100%", height = "400px")

renderGoogle_map(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a google_map
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Examples

```
## Not run:
library(shiny)
library(googleway)

ui <- fluidPage(google_mapOutput("map"))

server <- function(input, output, session){

api_key <- "your_api_key"

  df <- structure(list(lat = c(-37.8201904296875, -37.8197288513184,
  -37.8191299438477, -37.8187675476074, -37.8186187744141, -37.8181076049805
  ), lon = c(144.968612670898, 144.968414306641, 144.968139648438,
  144.967971801758, 144.967864990234, 144.967636108398), weight = c(31.5698964400217,
  97.1629025738221, 58.9051092562731, 76.3215389118996, 37.8982300488278,
```

```
  77.1501972114202), opacity = c(0.2, 0.2, 0.2, 0.2, 0.2, 0.2)), .Names = c("lat",
  "lon", "weight", "opacity"), row.names = 379:384, class = "data.frame")

  output$map <- renderGoogle_map({
    google_map(key = api_key)
  })
}

shinyApp(ui, server)

## End(Not run)
```

---

google_map_update *Google map update*

Description

Update a Google map in a shiny app. Use this function whenever the map needs to respond to reactive content.

Usage

```
google_map_update(map_id, session = shiny::getDefaultReactiveDomain(),
  data = NULL, deferUntilFlush = TRUE)
```

Arguments

map_id	string containing the output ID of the map in a shiny application.
session	the Shiny session object to which the map belongs; usually the default value will suffice.
data	data to be used in the map. See the details section for google_map.
deferUntilFlush	indicates whether actions performed against this instance should be carried out right away, or whether they should be held until after the next time all of the outputs are updated; defaults to TRUE.

Examples

```
## Not run:

library(shiny)
library(googleway)

ui <- pageWithSidebar(
  headerPanel("Toggle markers"),
  sidebarPanel(
    actionButton(inputId = "markers", label = "toggle markers")
```

```
  ),
  mainPanel(
    google_mapOutput("map")
  )
)

server <- function(input, output, session){

  # api_key <- "your_api_key"

  df <- structure(list(lat = c(-37.8201904296875, -37.8197288513184,
  -37.8191299438477, -37.8187675476074, -37.8186187744141, -37.8181076049805
  ), lon = c(144.968612670898, 144.968414306641, 144.968139648438,
 144.967971801758, 144.967864990234, 144.967636108398), weight = c(31.5698964400217,
 97.1629025738221, 58.9051092562731, 76.3215389118996, 37.8982300488278,
 77.1501972114202), opacity = c(0.2, 0.2, 0.2, 0.2, 0.2, 0.2)), .Names = c("lat",
 "lon", "weight", "opacity"), row.names = 379:384, class = "data.frame")


  output$map <- renderGoogle_map({
    google_map(key = api_key)
  })

  observeEvent(input$markers,{

    if(input$markers %% 2 == 1){
      google_map_update(map_id = "map") %>%
        add_markers(data = df)
    }else{
      google_map_update(map_id = "map") %>%
        clear_markers()
    }
  })
 }
shinyApp(ui, server)

## End(Not run)
```

---

google_nearestRoads *Nearest Roads*

Description

Takes up to 100 independent coordinates and returns the closest road segment for each point. The points passed do not need to be part of a continuous path.

Usage

```
google_nearestRoads(df_points, lat = NULL, lon = NULL, simplify = TRUE,
  key)
```

Arguments

df_points	data.frame with at least two columns specifying the latitude & longitude coordinates, with a maximum of 100 pairs of coordinates.
lat	string specifying the column of df_path containing the 'latitude' coordinates. If left NULL, a best-guess will be made
lon	string specifying the column of df_path containing the 'longitude' coordinates. If left NULL, a best-guess will be made
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Places API key

See Also

google_snapToRoads

Examples

```
## Not run:

key <- 'your_api_key'

df_points <- read.table(text = "lat lon
 60.1707 24.9426
 60.1708 24.9424
 60.1709 24.9423", header = T)

google_nearestRoads(df_points, key = key)


## End(Not run)
```

---

google_places *Google places*

Description

The Google Places API Web Service allows you to query for place information on a variety of categories, such as: establishments, prominent points of interest, geographic locations, and more.

Usage

```
google_places(search_string = NULL, location = NULL, radar = FALSE,
  radius = NULL, rankby = NULL, keyword = NULL, language = NULL,
  name = NULL, place_type = NULL, price_range = NULL, open_now = NULL,
  page_token = NULL, simplify = TRUE, key)
```

Arguments

search_string	string A search term representing a place for which to search. If blank, the location argument must be used.
location	numeric vector of latitude/longitude coordinates (in that order) around which to retrieve place information. If blank, the search_string argument must be used. If used in conjunction with search_string it represents the latitude/longitude around which to retrieve place information.
radar	boolean The Google Places API Radar Search Service allows you to search for up to 200 places at once, but with less detail than is typically returned from a Text Search (search_string) or Nearby Search (location) request. A radar search must contain a location and radius, and one of keyword, name or type. A radar search will not use a search_string
radius	numeric Defines the distance (in meters) within which to return place results. Required if only a location search is specified. The maximum allowed radius is 50,000 meters. Radius must not be included if rankby="distance" is specified. see Details.
rankby	string Specifies the order in which results are listed. Possible values are "prominence", "distance" or "location". If rankby = distance, then one of keyword, name or place_type must be specified. If a search_string is used then rankby is ignored.
keyword	string A term to be matched against all content that Google has indexed for this place, including but not limited to name, type, and address, as well as customer reviews and other third-party content.
language	string The language code, indicating in which language the results should be returned, if possible. Searches are also biased to the selected language; results in the selected language may be given a higher ranking. See the list of supported languages and their codes https://developers.google.com/maps/faq#languagesupport.
name	string vector One or more terms to be matched against the names of places. Ignored when used with a search_string. Results will be restricted to those containing the passed name values. Note that a place may have additional names associated with it, beyond its listed name. The API will try to match the passed name value against all of these names. As a result, places may be returned in the results whose listed names do not match the search term, but whose associated names do.
place_type	string Restricts the results to places matching the specified type. Only one type may be specified. For a list of valid types, please visit https://developers.google.com/places/supported_types.
price_range	numeric vector Specifying the minimum and maximum price ranges. Values range between 0 (most affordable) and 4 (most expensive).
open_now	logical Returns only those places that are open for business at the time the query is sent. Places that do not specify opening hours in the Google Places database will not be returned if you include this parameter in your query.
page_token	string Returns the next 20 results from a previously run search. Setting a page_token parameter will execute a search with the same parameters used in

	a previous search. All parameters other than page_token will be ignored. The page_token can be found in the result set of a previously run query.
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string into a list.
key	string A valid Google Developers Places API key.

Details

With the Places service you can perform four kinds of searches:

- Nearby Search
- Text Search
- Radar Sarch
- Place Details request

A Nearby search lets you search for places within a specified area or by keyword. A Nearby search must always include a location, which can be specified as a point defined by a pair of lat/lon coordinates, or a circle defined by a point and a radius.

A Text search returns information about a set of places based on the search_string. The service responds with a list of places matching the string and any location bias that has been set.

A Radar search lets you search for places within a specified search radius by keyword, type or name. The Radar search returns more results than a Nearby or Text search, but the results contain fewer fields.

A Place Detail search (using google_place_details) can be performed when you have a given place_id from one of the other three search methods.

radius - Required when only using a location search, radius defines the distance (in meters) within which to return place results. The maximum allowed radius is 50,000 meters. Note that radius must not be included if rankby = distance is specified.

radius - Optional when using a search_string. Defines the distance (in meters) within which to bias place results. The maximum allowed radius is 50,000 meters. Results inside of this region will be ranked higher than results outside of the search circle; however, prominent results from outside of the search radius may be included.

Note

The Google Places API Web Service enforces a default limit of 1,000 free requests per 24 hour period, calculated as the sum of client-side and server-side requets. See https://developers.google.com/places/web-service/usage for details.

Use of the Places Library must be in accordance with the polices described for the Google Places API Web Service https://developers.google.com/places/web-service/policies

See Also

google_place_details

Examples

```
## Not run:

## query restaurants in Melbourne (will return 20 results)
key <- 'your_api_key'

res <- google_places(search_string = "Restaurants in Melbourne, Australia",
                     key = key)

## use the 'next_page_token' from the previous search to get the next 20 results
res_next <- google_places(search_string = "Restaurants in Melbourne, Australia",
                          page_token = res$next_page_token,
                          key = key)

## search for a specific place type
google_places(location = c(-37.817839,144.9673254),
              place_type = "bicycle_store",
              radius = 20000,
              key = key)

## search for places that are open at the time of query
 google_places(search_string = "Bicycle shop, Melbourne, Australia",
               open_now = TRUE,
               key = key)


## End(Not run)
```

---

google_place_autocomplete

Google place autocomplete

Description

The Place Autocomplete service is a web service that returns place predictions in response to an HTTP request. The request specifies a textual search string and optional geographic bounds. The service can be used to provide autocomplete functionality for text-based geographic searches, by returning places such as businesses, addresses and points of interest as a user types.

Usage

```
google_place_autocomplete(place_input, location = NULL, radius = NULL,
  language = NULL, place_type = NULL, components = NULL,
  simplify = TRUE, key)
```

Arguments

place_input	`string` The text string on which to search. The Place Autocomplete service will return candidate matches based on this string and order results based on their perceived relevance.

location	numeric vector of latitude/longitude coordinates (in that order) the point around which you wish to retrieve place information
radius	numeric The distance (in meters) within which to return place results. Note that setting a radius biases results to the indicated area, but may not fully restrict results to the specified area
language	string The language code, indicating in which language the results should be returned, if possible. Searches are also biased to the selected language; results in the selected language may be given a higher ranking. See the list of supported languages and their codes https://developers.google.com/maps/faq#languagesupport
place_type	string Restricts the results to places matching the specified type. Only one type may be specified (if more than one type is provided, all types following the first entry are ignored). For a list of valid types, please visit https://developers.google.com/places/web-service/autocomplete#place_types
components	string of length 1 which identifies a grouping of places to which you would like to restrict your results. Currently, you can use components to filter by country only. The country must be passed as a two character, ISO 3166-1 Alpha-2 compatible country code. For example: components=country:fr would restrict your results to places within France.
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Places API key

Examples

```
## Not run:

## search for 'Maha' Restaurant, Melbourne
google_place_autocomplete("Maha Restaurant", key = key)

## search for 'Maha' Restaurant, exclusively in Australia
google_place_autocomplete("maha Restaurant", component = "au", key = key)


## End(Not run)
```

---

google_place_details *Google place details*

Description

Once you have a place_id from a Place Search, you can request more details about a particular establishment or point of interest by initiating a Place Details request. A Place Details request returns more comprehensive information about the indicated place such as its complete address, phone number, user rating and reviews.

Usage

```
google_place_details(place_id, language = NULL, simplify = TRUE, key)
```

Arguments

place_id	string A textual identifier that uniquely identifies a place, usually of the form ChIJrTLr-GyuEmsRBfy61i59si0, returned from a place search
language	string The language code, indicating in which language the results should be returned, if possible. Searches are also biased to the selected language; results in the selected language may be given a higher ranking. See the list of supported languages and their codes https://developers.google.com/maps/faq#languagesupport
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Places API key

See Also

google_places

Examples

```
## Not run:
## search for a specific restaurant, Maha, in Melbourne, firstly using google_places()
res <- google_places(search_string = "Maha Restaurant, Melbourne, Australia",
                     radius = 1000,
                     key = key)

## request more details about the restaurant using google_place_details()
google_place_details(place_id = res$results$place_id, key = key)


## End(Not run)
```

---

google_reverse_geocode

Google reverse geocoding

Description

Reverse geocoding is the process of converting geographic coordinates into a human-readable address.

Usage

```
google_reverse_geocode(location, result_type = NULL, location_type = NULL,
  language = NULL, key, simplify = TRUE)
```

Arguments

location	numeric vector of lat/lon coordinates.
result_type	string vector - one or more address types. See https://developers.google.com/maps/documentation/geocoding/intro#Types for list of available types.
location_type	string vector specifying a location type will restrict the results to this type. If multiple types are specified, the API will return all addresses that match any of the types
language	string specifies the language in which to return the results. See the list of supported languages: https://developers.google.com/maps/faq#using-google-maps-apis. If no langauge is supplied, the service will attempt to use the language of the domain from which the request was sent
key	string. A valid Google Developers Geocode API key
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string

Value

Either list or JSON string of the geocoded address

Examples

```
## Not run:
## searching for the street address for the rooftop location type
google_reverse_geocode(location = c(-37.81659, 144.9841),
                       result_type = c("street_address"),
                       location_type = "rooftop",
                       key = "<your valid api key>")

## End(Not run)
```

---

google_snapToRoads *Snap To Roads*

Description

Takes up to 100 GPS coordinates collected along a route and returns a similar set of data, with the points snapped to the most likely roads the vehicle was treveling along

Usage

```
google_snapToRoads(df_path, lat = NULL, lon = NULL, interpolate = FALSE,
  simplify = TRUE, key)
```

Arguments

df_path	data.frame with at least two columns specifying the latitude & longitude coordinates, with a maximum of 100 pairs of coordinates.
lat	string specifying the column of df_path containing the 'latitude' coordinates. If left NULL, a best-guess will be made.
lon	string specifying the column of df_path containing the ' longitude' coordinates. If left NULL, a best-guess will be made.
interpolate	logical indicating whether to interpolate a path to include all points forming the full road-geometry. When TRUE, additional interpolated points will also be returned, resulting in a path that smoothly follows the geometry of the road, even around corners and through tunnels. Interpolated paths will most likely contain more ponts that the original path.
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Places API key

Note

The snapping algorithm works best for points that are not too far apart. If you observe odd snapping behaviour, try creating paths that have points closer together. To ensure the best snap-to-road quality, you should aim to provide paths on which consecutive pairs of points are within 300m of each other. This will also help in handling any isolated, long jumps between consecutive points caused by GPS signal loss or noise.

See Also

google_nearestRoads

Examples

```
## Not run:

key <- 'your_api_key'

df_path <- read.table(text = "lat lon
-35.27801 149.12958
-35.28032 149.12907
-35.28099 149.12929
-35.28144 149.12984
-35.28194 149.13003
-35.28282 149.12956
-35.28302 149.12881
-35.28473 149.12836", header = T)

google_snapToRoads(df_path, key = key, interpolate = TRUE, simplify = TRUE)



## End(Not run)
```

google_speedLimits *Speed Limits*

Description

Returns the posted speed limit for a given road segment. In the case of road segments with variable speed limits, the default speed limit for the segment is returned. The speed limits service is only available to Google Maps API Premium Plan customers with an Asset Tracking license.

Usage

```
google_speedLimits(df_path = NULL, lat = NULL, lon = NULL,
  placeIds = NULL, units = c("KPH", "MPH"), simplify = TRUE, key)
```

Arguments

df_path	data.frame with at least two columns specifying the latitude & longitude coordinates, with a maximum of 100 pairs of coordinates.
lat	string specifying the latitude column
lon	string specifying the longitude column
placeIds	vector of Place IDs of the road segments. Place IDs are returned in response to google_snapToRoads and google_nearestRoads reqeusts. You can pass up to 100 placeIds at a time
units	Whether to return speed limits in kilometers or miles per hour
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Places API key

Note

The accuracy of speed limit data returned by Google Maps Roads API can not be guaranteed. The speed limit data provided is not real-time, and may be estimated, inaccurate, incomplete, and / or outdated.

google_streetview *Google street view*

Description

Displays a static street view image from Google Maps Street View Image API

Usage

```
google_streetview(location = NULL, panorama_id = NULL, size = c(400, 400),
  heading = NULL, fov = 90, pitch = 0, output = c("plot", "html"),
  response_check = FALSE, signature = NULL, key)
```

Arguments

location	numeric vector of lat/lon coordinates, or an address string.
panorama_id	a specific panorama ID.
size	numeric vector of length 2, specifying the output size of the image in pixels, given in `width x height`. For example, `c(600, 400)` returns an image 600 pixles wide and 400 pixles high.
heading	indicates the compass heading of the camera. Accepted values are from 0 to 360 (both 0 and 360 indicate north), 90 indicates east, 180 south and 270 west. If no heading is specified a value will be calculated that directs teh camera to wards the specified `location`, from the point at which the closest photograph was taken.
fov	determines the horizontal field of view of the image. The field of view is expressed in degrees, with a maximum allowed value of 120. When dealing with a fixed-size viewport, as with Street View image of a set size, field of view in essence represents zoom, with small numbers indicating a higher level of zoom
pitch	specifies the up or down angle of the camera relative to the Street View vehicle. This is often, but not always, flat horizontal. Positive values angle the camera up (with 90 degrees indicating straight up); negative values angle the camera down (with -90 indicating straight down)
output	specifies whether the result should be displayed in R's viewer, or embedded as HTML inside a webpage.
response_check	logical indicating if the function should first check if the image is available. If TRUE and no image is available, a warning message is printed and no image will be downloaded. if FALSE and no image is available, a blank image will be displayed saying 'Sorry, we have no imagery here'.
signature	a digitial signature used to verify that any site generating requests using your API key is authorised to do so. See Google Documentation for further details https://developers.google.com/maps/documentation/streetview/intro
key	string. A valid Google Developers Street View Image API key

Examples

```
## Not run:

## download and display an image
# key <- "your_api_key"
google_streetview(location = c(-37.817714, 144.96726),
    size = c(400,400), output = "plot",
    key = key)
```

```
## no response check - display 'sorry' message
google_streetview(location = c(-37.8, 144),
   size = c(400,400),
   panorama_id = NULL,
   output = "plot",
   heading = 90,
   fov = 90,
   pitch = 0,
   response_check = FALSE,
   key = key)


## embed an image of Flinders Street Station into a Shiny webpage
library(shiny)
library(googleway)

ui <- fluidPage(
  uiOutput(outputId = "myStreetview")
)

server <- function(input, output){
  key <- "your_api_key"

  output$myStreetview <- renderUI({
    tags$img(src = google_streetview(location = c(-37.817714, 144.96726),
                                     size = c(400,400), output = "html",
                                     key = key),  width = "400px", height = "400px")
  })
}

shinyApp(ui, server)


## End(Not run)
```

---

google_timezone *Google timezone*

Description

The Google Maps Time Zone API provides time offset data for locations on the surface of the earth. You request the time zone information for a specific latitude/longitude pair and date.

Usage

```
google_timezone(location, timestamp = Sys.time(), language = NULL,
  simplify = TRUE, key)
```

Arguments

location	vector of lat/lon pair
timestamp	POSIXct The Google Maps Time Zone API uses the timestamp to determine whether or not Daylight Savings should be applied. Will default to the current system time.
language	string specifies the language in which to return the results. See the list of supported languages: https://developers.google.com/maps/faq#using-google-maps-apis. If no langauge is supplied, the service will attempt to use the language of the domain from which the request was sent.
simplify	logical - TRUE indicates the returned JSON will be coerced into a list. FALSE indicates the returend JSON will be returned as a string
key	string A valid Google Developers Timezone API key.

Value

Either list or JSON string of the timezone

Examples

```
## Not run:
google_timezone(location = c(-37.81659, 144.9841),
                timestamp = as.POSIXct("2016-06-05"),
                key = "<your valid api key>")

## End(Not run)
```

---

map_styles *Map Styles*

Description

Various styles for a google_map() map.

Usage

```
map_styles()
```

Value

list of styles

Note

you can generate your own map styles at https://mapstyle.withgoogle.com/

Examples

```
## Not run:
map_key <- "your_map_key"
google_map(key = map_key, style = map_styles()$silver)


## End(Not run)
```

`melbourne` *Melbourne*

Description

Polygons for Melbourne and the surrounding area

Usage

```
melbourne
```

Format

A data frame with 397 observations and 7 variables

polygonId a unique identifier for each polygon

pathId an identifier for each path that define a polygon

SA2_NAME statistical area 2 name of the polygon

SA3_NAME statistical area 3 name of the polygon

SA4_NAME statistical area 4 name of the polygon

AREASQKM area of the SA2 polygon

polyline encoded polyline that defines each `pathId`

Details

This data set is a subset of the Statistical Area Level 2 (SA2) ASGS Edition 2016 data released by the Australian Bureau of Statistics http://www.abs.gov.au

The data is realsed under a Creative Commons Attribution 2.5 Australia licence https://creativecommons.org/licenses/by/2.5/au/

tram_route *Tram Route*

Description

The latitude and longitude coordinates specifying the path tram 35 follows in Melbourne.

Usage

```
tram_route
```

Format

A data frame with 55 observations and 3 variables

shape_pt_lat the latitude of each point in the route

shape_pt_lon the longitude of each point in the route

shape_pt_sequence the position in the sequence of coordinates for each point

Details

The data is taken from the PTV GTFS data

tram_stops *Tram stops along tram route 35 in Melbourne*

Description

A data set containing the latitude and longitude coordinates of tram stops along route 35 in Melbourne.

Usage

```
tram_stops
```

Format

A data frame with 41 observations and 4 variables

stop_id unique ID for each stop

stop_name the name of each stop

stop_lat the latitude of the stop

stop_lon the longitude of the stop

Details

The data is taken from the PTV GTFS data

update_circles *Update circles*

Description

Updates specific colours and opacities of specified circles Designed to be used in a shiny application.

Usage

```
update_circles(map, data, id, radius = NULL, draggable = NULL,
  stroke_colour = NULL, stroke_weight = NULL, stroke_opacity = NULL,
  fill_colour = NULL, fill_opacity = NULL, layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
data	data.frame containing the new values for the circles
id	string representing the column of data containing the id values for the circles. The id values must be present in the data supplied to add_circles in order for the polygons to be udpated
radius	either a string specifying the column of data containing the radius of each circle, OR a numeric value specifying the radius of all the circles (radius is expressed in metres)
draggable	string specifying the column of data defining if the circle is 'draggable' (either TRUE or FALSE)
stroke_colour	either a string specifying the column of data containing the stroke colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the circles
stroke_weight	either a string specifying the column of data containing the stroke weight of each circle, or a number indicating the width of pixels in the line to be applied to all the circles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
fill_colour	either a string specifying the column of data containing the fill colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the cirlces
fill_opacity	either a string specifying the column of data containing the fill opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
layer_id	single value specifying an id for the layer.

Note

Any circles (as specified by the id argument) that do not exist in the data passed into add_circles() will not be added to the map. This function will only update the circles that currently exist on the map when the function is called.

update_heatmap *update heatmap*

Description

updates a heatmap layer

Usage

```
update_heatmap(map, data, lat = NULL, lon = NULL, weight = NULL,
  layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
data	data frame containing at least two columns, one specifying the latitude coordinates, and the other specifying the longitude. If Null, the data passed into google_map() will be used.
lat	string specifying the column of data containing the 'latitude' coordinates. If left NULL, a best-guess will be made
lon	string specifying the column of data containing the 'longitude' coordinates. If left NULL, a best-guess will be made
weight	string specifying the column of data containing the 'weight' associated with each point. If NULL, each point will get a weight of 1.
layer_id	single value specifying an id for the layer.

update_polygons *Update polygons*

Description

Updates specific colours and opacities of specified polygons. Designed to be used in a shiny application.

Usage

```
update_polygons(map, data, id, stroke_colour = NULL, stroke_weight = NULL,
  stroke_opacity = NULL, fill_colour = NULL, fill_opacity = NULL,
  layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
data	data.frame containing the new values for the polygons
id	string representing the column of data containing the id values for the polygons. The id values must be present in the data supplied to add_polygons in order for the polygons to be udpated
stroke_colour	either a string specifying the column of data containing the stroke colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the circles
stroke_weight	either a string specifying the column of data containing the stroke weight of each circle, or a number indicating the width of pixels in the line to be applied to all the circles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
fill_colour	either a string specifying the column of data containing the fill colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the cirlces
fill_opacity	either a string specifying the column of data containing the fill opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
layer_id	single value specifying an id for the layer.

Note

Any polygons (as specified by the id argument) that do not exist in the data passed into add_polygons() will not be added to the map. This function will only update the polygons that currently exist on the map when the function is called.

Examples

```
## Not run:

map_key <- 'your_api_key'

pl_outer <- encode_pl(lat = c(25.774, 18.466,32.321),
                      lon = c(-80.190, -66.118, -64.757))

pl_inner <- encode_pl(lat = c(28.745, 29.570, 27.339),
                      lon = c(-70.579, -67.514, -66.668))

pl_other <- encode_pl(c(21,23,22), c(-50, -49, -51))

## using encoded polylines
df <- data.frame(id = c(1,1,2),
                 colour = c("#00FF00", "#00FF00", "#FFFF00"),
                 polyline = c(pl_outer, pl_inner, pl_other),
                 stringsAsFactors = FALSE)

google_map(key = map_key) %>%
  add_polygons(data = df, polyline = 'polyline', id = 'id', fill_colour = 'colour')
```

```
df_update <- df[, c("id", "colour")]
df_update$colour <- c("#FFFFFF", "#FFFFFF", "000000")

google_map(key = map_key) %>%
  add_polygons(data = df, polyline = 'polyline', id = 'id', fill_colour = 'colour') %>%
  update_polygons(data = df_update, id = 'id', fill_colour = 'colour')


df <- aggregate(polyline ~ id + colour, data = df, list)

google_map(key = map_key) %>%
  add_polygons(data = df, polyline = 'polyline', fill_colour = 'colour')

google_map(key = map_key) %>%
  add_polygons(data = df, polyline = 'polyline', id = 'id', fill_colour = 'colour') %>%
  update_polygons(data = df_update, id = 'id', fill_colour = 'colour')


## using coordinates
df <- data.frame(id = c(rep(1, 6), rep(2, 3)),
                 lineId = c(rep(1, 3), rep(2, 3), rep(1, 3)),
                 lat = c(25.774, 18.466, 32.321, 28.745, 29.570, 27.339, 21, 23, 22),
              lon = c(-80.190, -66.118, -64.757, -70.579, -67.514, -66.668, -50, -49, -51))

google_map(key = map_key) %>%
  add_polygons(data = df, lat = 'lat', lon = 'lon', id = 'id', pathId = 'lineId')

google_map(key = map_key) %>%
  add_polygons(data = df, lat = 'lat', lon = 'lon', id = 'id', pathId = 'lineId') %>%
  update_polygons(data = df_update, id = 'id', fill_colour = 'colour')


## End(Not run)
```

---

update_polylines *Update polylines*

Description

Updates specific attributes of polylines. Designed to be used in a shiny application.

Usage

```
update_polylines(map, data, id, stroke_colour = NULL, stroke_weight = NULL,
  stroke_opacity = NULL, layer_id = NULL)
```

Arguments

map	a googleway map object created from google_map()
data	data.frame containing the new values for the polylines
id	string representing the column of data containing the id values for the polylines The id values must be present in the data supplied to add_polylines in order for the polylines to be udpated
stroke_colour	either a string specifying the column of data containing the stroke colour of each circle, or a valid hexadecimal numeric HTML style to be applied to all the circles
stroke_weight	either a string specifying the column of data containing the stroke weight of each circle, or a number indicating the width of pixels in the line to be applied to all the circles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each circle, or a value between 0 and 1 that will be applied to all the circles
layer_id	single value specifying an id for the layer.

Note

Any polylines (as specified by the id argument) that do not exist in the data passed into add_polylines() will not be added to the map. This function will only update the polylines that currently exist on the map when the function is called.

Examples

```
## Not run:

map_key <- 'your_api_key'

## coordinate columns
## plot polylines using default attributes
df <- tram_route
df$id <- c(rep(1, 27), rep(2, 28))

df$colour <- c(rep("#00FFFF", 27), rep("#FF00FF", 28))

google_map(key = map_key) %>%
  add_polylines(data = df, lat = 'shape_pt_lat', lon = 'shape_pt_lon',
                stroke_colour = "colour", id = 'id')

## specify width and colour attributes to update
df_update <- data.frame(id = c(1,2),
                        width = c(3,10),
                        colour = c("#00FF00", "#DCAB00"))

google_map(key = map_key) %>%
  add_polylines(data = df, lat = 'shape_pt_lat', lon = 'shape_pt_lon',
                stroke_colour = "colour", id = 'id') %>%
  update_polylines(data = df_update, id = 'id', stroke_weight = "width",
                   stroke_colour = 'colour')
```

```
## encoded polylines
pl <- sapply(unique(df$id), function(x){
  encode_pl(lat = df[ df$id == x , 'shape_pt_lat'], lon = df[ df$id == x, 'shape_pt_lon'])
})

df <- data.frame(id = c(1, 2), polyline = pl)

google_map(key = map_key) %>%
  add_polylines(data = df, polyline = 'polyline')

google_map(key = map_key) %>%
  add_polylines(data = df, polyline = 'polyline') %>%
  update_polylines(data = df_update, id = 'id', stroke_weight = "width",
                   stroke_colour = 'colour')


## End(Not run)
```

`update_rectangles` *Update rectangles*

Description

Updates specific colours and opacities of specified rectangles Designed to be used in a shiny application.

Usage

```
update_rectangles(map, data, id, draggable = NULL, stroke_colour = NULL,
  stroke_weight = NULL, stroke_opacity = NULL, fill_colour = NULL,
  fill_opacity = NULL, layer_id = NULL)
```

Arguments

`map`	a googleway map object created from `google_map()`
`data`	data.frame containing the new values for the rectangles
`id`	string representing the column of `data` containing the id values for the rectangles The id values must be present in the data supplied to `add_rectangles` in order for the polygons to be udpated
`draggable`	string specifying the column of `data` defining if the rectangle is 'draggable' (either TRUE or FALSE)
`stroke_colour`	either a string specifying the column of `data` containing the stroke colour of each rectangle, or a valid hexadecimal numeric HTML style to be applied to all the rectangles

stroke_weight	either a string specifying the column of data containing the stroke weight of each rectangle, or a number indicating the width of pixels in the line to be applied to all the rectangles
stroke_opacity	either a string specifying the column of data containing the stroke opacity of each rectangle, or a value between 0 and 1 that will be applied to all the rectangles
fill_colour	either a string specifying the column of data containing the fill colour of each rectangle, or a valid hexadecimal numeric HTML style to be applied to all the cirlces
fill_opacity	either a string specifying the column of data containing the fill opacity of each rectangle, or a value between 0 and 1 that will be applied to all the rectangles
layer_id	single value specifying an id for the layer.

Note

Any rectangles (as specified by the id argument) that do not exist in the data passed into add_rectangles() will not be added to the map. This function will only update the rectangles that currently exist on the map when the function is called.

---

update_style *Update style*

Description

Updates the map with the given styles

Usage

```
update_style(map, styles = NULL)
```

Arguments

map	a googleway map object created from google_map()
styles	JSON string representation of a valid Google Maps styles Array. See the Google documentation for details https://developers.google.com/maps/documentation/javascript/styling

Note

This function is intended for use with google_map_update in an interactive shiny environment. You can set the styles of the original map using the styles argument of google_map

`%>%` *Pipe*

Description

Uses the pipe operator (%>%) to chain statements. Useful for adding layers to a `google_map`

Arguments

`lhs, rhs`	A google map and a layer to add to it

Examples

```
## Not run:

key <- "your_api_key"
google_map(key = key) %>%
add_traffic()


## End(Not run)
```

Index

∗Topic **datasets**
melbourne, 48
tram_route, 49
tram_stops, 49
%>%, 57

add_bicycling, 3
add_circles, 3
add_fusion, 5
add_heatmap, 6
add_kml, 8
add_markers, 9, *31*
add_overlay, 10
add_polygons, 11
add_polylines, 14
add_rectangles, 17
add_traffic, 19
add_transit, 19

clear, 20
clear_bicycling *(*clear*)*, 20
clear_circles *(*clear*)*, 20
clear_fusion *(*clear*)*, 20
clear_heatmap *(*clear*)*, 20
clear_markers *(*clear*)*, 20
clear_polygons *(*clear*)*, 20
clear_polylines *(*clear*)*, 20
clear_rectangles *(*clear*)*, 20
clear_search, 21
clear_traffic *(*clear*)*, 20
clear_transit *(*clear*)*, 20

decode_pl, 21, *22*

encode_pl, *13*, *21*, 22

get_route, 23
google_directions, *21*, *23*, 23
google_dispatch, 26
google_distance, 27
google_elevation, 28
google_geocode, 30
google_map, *26*, 31, *34*, *56*
google_map-shiny, 33
google_map_update, *20*, *26*, 34, *56*
google_mapOutput *(*google_map-shiny*)*, 33
google_nearestRoads, 35, *43*, *44*
google_place_autocomplete, 39
google_place_details, *38*, 40
google_places, 36, *41*
google_reverse_geocode, 41
google_snapToRoads, *36*, 42, *44*
google_speedLimits, 44
google_streetview, 44
google_timezone, 46
googleway *(*google_map*)*, 31

invoke_method *(*google_dispatch*)*, 26

map_styles, 47
melbourne, 48

renderGoogle_map *(*google_map-shiny*)*, 33

tram_route, 49
tram_stops, 49

update_circles, 50
update_heatmap, 51
update_polygons, 51
update_polylines, 53
update_rectangles, 55
update_style, 56