

Package ‘mapgl’

October 2, 2026

Title Interactive Maps with 'Mapbox GL JS' and 'MapLibre GL JS'

Version 0.5.2

Date 2026-10-02

Description Provides an interface to the 'Mapbox GL JS' (<<https://docs.mapbox.com/mapbox-gl-js/guides>>) and the 'MapLibre GL JS' (<<https://maplibre.org/maplibre-gl-js/docs/>>) interactive mapping libraries to help users create custom interactive maps in R. Users can create interactive globe visualizations; layer 'sf' objects to create filled maps, circle maps, 'heatmaps', and three-dimensional graphics; and customize map styles and views. The package also includes utilities to use 'Mapbox' and 'MapLibre' maps in 'Shiny' web applications.

URL <https://walker-data.com/mapgl/>

BugReports <https://github.com/walkerke/mapgl/issues>

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports htmlwidgets, geojsonsf, sf, rlang, htmltools, grDevices, base64enc, terra, classInt, shiny, viridisLite, png, jsonlite, httpuv

Suggests chromote, knitr, mapboxapi, testthat (>= 3.0.0), usethis, leaflet, withr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Kyle Walker [aut, cre],
Egor Kotov [ctb]

Maintainer Kyle Walker <kyle@walker-data.com>

Repository CRAN

Date/Publication 2026-10-02 14:50:02 UTC

Contents

add_bivariate_legend	5
add_circle_layer	6
add_control	10
add_coordinates_control	11
add_draw_control	12
add_features_to_draw	17
add_fill_extrusion_layer	18
add_fill_layer	21
add_flowmap	24
add_fullscreen_control	28
add_geocoder_control	28
add_geolocate_control	30
add_globe_control	31
add_globe_minimap	32
add_h3j_source	33
add_h3t_source	34
add_heatmap_layer	35
add_image	37
add_image_source	38
add_layer	39
add_layers_control	41
add_line_layer	44
add_markers	47
add_navigation_control	49
add_pmtiles_source	50
add_raster_dem_source	51
add_raster_layer	52
add_raster_source	54
add_reset_control	55
add_scale_control	56
add_screenshot_control	57
add_slider_control	58
add_source	63
add_symbol_layer	64
add_terradraw_control	70
add_vector_source	72
add_video_source	73
add_view	74
as_time_property	75
basemap_style	76
bivariate_palettes	77
bivariate_scale	77
bixi_flows	78
bixi_locations	79
carto_style	80
classification_helpers	80

clear_controls	82
clear_drawn_features	83
clear_layer	84
clear_legend	85
clear_markers	86
cluster_options	86
compare	89
concat	92
conditional_expressions	93
draw_attribute	95
ease_to	96
enable_shiny_hover	97
esri_open_style	98
esri_style	99
fit_bounds	101
flowmap_color_schemes	102
fly_to	102
get_column	103
get_drawn_features	103
get_queried_features	104
interpolate	105
interpolate_palette	106
jump_to	108
legend_style	108
mapboxgl	111
mapboxglCompareOutput	112
mapboxglOutput	113
mapboxgl_compare_proxy	113
mapboxgl_proxy	114
mapboxgl_view	114
mapbox_style	116
maplibre	116
maplibreCompareOutput	118
maplibreOutput	119
maplibre_compare_proxy	119
maplibre_proxy	120
maplibre_view	120
maptiler_style	122
map_legends	122
match_expr	132
move_layer	133
number_format	134
on_section	136
openfreemap_style	136
palette_to_lut	137
print_map	138
query_rendered_features	139
renderMapboxgl	142

renderMapboxglCompare	142
renderMaplibre	143
renderMaplibreCompare	143
save_map	144
set_config_property	145
set_filter	146
set_flowmap_filter	147
set_flowmap_settings	148
set_fog	149
set_layout_property	149
set_paint_property	150
set_popup	151
set_projection	151
set_rain	152
set_snow	153
set_source	155
set_style	155
set_terrain	156
set_tooltip	157
set_view	158
slider_style	158
step_classification	161
step_expr	163
story_leaflet	164
story_map	165
story_maplibre	166
story_section	167
terrdraw_options	167
tooltip_style	170
turf_area	172
turf_buffer	172
turf_center_of_mass	174
turf_centroid	175
turf_concave_hull	176
turf_convex_hull	177
turf_difference	178
turf_distance	179
turf_filter	180
turf_intersect	181
turf_union	182
turf_voronoi	182
update_slider_control	183

add_bivariate_legend *Add a bivariate legend*

Description

Add a bivariate legend

Usage

```
add_bivariate_legend(
  map,
  scale,
  legend_title = NULL,
  x_title = NULL,
  y_title = NULL,
  position = "top-left",
  width = NULL,
  style = NULL,
  add = FALSE,
  unique_id = NULL,
  layer_id = NULL,
  target = "compare",
  draggable = FALSE,
  collapsible = FALSE,
  collapsed = FALSE,
  min_zoom = NULL,
  max_zoom = NULL
)
```

Arguments

map	A map object created by mapboxgl() or maplibre(), a compare object, or a proxy object.
scale	A mapgl_bivariate_scale object from bivariate_scale().
legend_title	Optional legend title.
x_title	Label for the horizontal axis. Defaults to the x column name.
y_title	Label for the vertical axis. Defaults to the y column name.
position	The legend position.
width	Legend width. Defaults to "fit-content" (capped at 340px) so titles and axis labels stay on one line.
style	Optional styling options from legend_style() or a list.
add	Logical, whether to add to existing legends.
unique_id	Optional unique legend ID.
layer_id	Optional associated layer ID for layer-control show/hide.

target	For compare objects, one of "compare", "before", or "after"; for synced grids with more than two maps, a map identifier such as "map3" can also be used.
draggable	Logical, whether the legend can be dragged.
collapsible	Logical, whether the legend can collapse.
collapsed	Logical, whether the legend starts collapsed.
min_zoom	The minimum zoom level at which the legend is displayed.
max_zoom	The maximum zoom level at which the legend is displayed. The legend is hidden when the map zoom is at or above this value.

Value

The updated map object.

add_circle_layer	<i>Add a circle layer to a Mapbox GL map</i>
------------------	--

Description

Add a circle layer to a Mapbox GL map

Usage

```
add_circle_layer(  
  map,  
  id,  
  source,  
  source_layer = NULL,  
  circle_blur = NULL,  
  circle_color = NULL,  
  circle_emissive_strength = NULL,  
  circle_opacity = NULL,  
  circle_pitch_alignment = NULL,  
  circle_pitch_scale = NULL,  
  circle_radius = NULL,  
  circle_sort_key = NULL,  
  circle_stroke_color = NULL,  
  circle_stroke_opacity = NULL,  
  circle_stroke_width = NULL,  
  circle_translate = NULL,  
  circle_translate_anchor = "map",  
  visibility = "visible",  
  slot = NULL,  
  min_zoom = NULL,  
  max_zoom = NULL,  
  popup = NULL,
```

```

    tooltip = NULL,
    tooltip_style = NULL,
    popup_style = NULL,
    hover_options = NULL,
    before_id = NULL,
    filter = NULL,
    cluster_options = NULL
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies type and url for a remote source.
source_layer	The source layer (for vector sources).
circle_blur	Amount to blur the circle.
circle_color	The color of the circle.
circle_emissive_strength	Controls the intensity of light emitted on the source features. Requires 3D lights.
circle_opacity	The opacity at which the circle will be drawn.
circle_pitch_alignment	Orientation of circles when the map is pitched. One of "map" or "viewport".
circle_pitch_scale	Controls the scaling behavior of circles when the map is pitched. One of "map" (scaled by distance) or "viewport" (not scaled).
circle_radius	Circle radius.
circle_sort_key	Sorts features in ascending order based on this value.
circle_stroke_color	The color of the circle's stroke.
circle_stroke_opacity	The opacity of the circle's stroke.
circle_stroke_width	The width of the circle's stroke.
circle_translate	The geometry's offset. Values are c(x, y) where negatives indicate left and up.
circle_translate_anchor	Controls the frame of reference for circle-translate.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.

popup	Popup content shown on click: a column name, a {brace} template (e.g. "{name}": {value}"), or a concat()/number_format() expression. Columns containing HTML are parsed.
tooltip	Tooltip content shown on hover; same forms as popup.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a <code>tooltip_style()</code> object. When omitted, the native (unstyled) appearance is kept.
hover_options	A named list of options for highlighting features in the layer on hover.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.
cluster_options	A list of options for clustering circles, created by the <code>cluster_options()</code> function. Two input shapes are supported: pass an <code>sf/sfc</code> object as source for native live clustering (a GeoJSON source is injected automatically), or pass the id of an already-registered vector source (e.g. from <code>add_pmtiles_source()</code>) along with <code>source_layer</code> to use pre-clustered vector tiles such as those produced by the <code>freestiler</code> package. In the latter case the cluster-count label is abbreviated client-side from <code>point_count</code>. Set <code>donut_column</code> in <code>cluster_options()</code> to draw clusters as donut charts of category shares; see <code>cluster_options()</code> for the properties pre-clustered tiles need. Updating a clustered layer in Shiny: the shortcut creates three layers ("id", "id-clusters", "id-cluster-count") on top of one source. For reactive data updates the recommended pattern is <code>set_source()</code>, which replaces the source's data and lets Mapbox/MapLibre re-cluster automatically without tearing down the layers: <code>mapboxgl_proxy("map") > set_source(layer_id = "circles", source = filtered())</code>. If you need to remove a clustered layer entirely (e.g. before switching backends), pass the full trio to <code>clear_layer()</code>: <code>clear_layer(proxy, c("circles", "circles-clusters", "circles-cluster-count"))</code>.

Value

The modified map object with the new circle layer added.

Examples

```
## Not run:
library(mapgl)
library(sf)
library(dplyr)

# Set seed for reproducibility
set.seed(1234)

# Define the bounding box for Washington DC (approximately)
bbox <- st_bbox(
  c(
```

```

        xmin = -77.119759,
        ymin = 38.791645,
        xmax = -76.909393,
        ymax = 38.995548
    ),
    crs = st_crs(4326)
)

# Generate 30 random points within the bounding box
random_points <- st_as_sf(
  data.frame(
    id = 1:30,
    lon = runif(30, bbox["xmin"], bbox["xmax"]),
    lat = runif(30, bbox["ymin"], bbox["ymax"])
  ),
  coords = c("lon", "lat"),
  crs = 4326
)

# Assign random categories
categories <- c("music", "bar", "theatre", "bicycle")
random_points <- random_points %>%
  mutate(category = sample(categories, n(), replace = TRUE))

# Map with circle layer
mapboxgl(style = mapbox_style("light")) %>%
  fit_bounds(random_points, animate = FALSE) %>%
  add_circle_layer(
    id = "poi-layer",
    source = random_points,
    circle_color = match_expr(
      "category",
      values = c(
        "music", "bar", "theatre",
        "bicycle"
      ),
    ),
    stops = c(
      "#1f78b4", "#33a02c",
      "#e31a1c", "#ff7f00"
    )
  ),
  circle_radius = 8,
  circle_stroke_color = "#ffffff",
  circle_stroke_width = 2,
  circle_opacity = 0.8,
  tooltip = "category",
  hover_options = list(
    circle_radius = 12,
    circle_color = "#ffff99"
  )
) %>%
  add_categorical_legend(
    legend_title = "Points of Interest",

```

```

        values = c("Music", "Bar", "Theatre", "Bicycle"),
        colors = c("#1f78b4", "#33a02c", "#e31a1c", "#ff7f00"),
        circular_patches = TRUE
    )

## End(Not run)

```

add_control

Add a custom control to a map

Description

This function adds a custom control to a Mapbox GL or MapLibre GL map. It allows you to create custom HTML element controls and add them to the map.

Usage

```

add_control(
  map,
  html,
  position = "top-right",
  className = NULL,
  id = NULL,
  ...
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
html	Character string containing the HTML content for the control.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".
className	Optional CSS class name for the control container.
id	Optional unique identifier for the control. If not provided, defaults to "custom". This ID can be used with clear_controls() to selectively remove this specific control.
...	Additional arguments passed to the JavaScript side.

Value

The modified map object with the custom control added.

Examples

```
## Not run:
library(mapgl)

# Basic custom control
maplibre() |>
  add_control(
    html = "<div style='background-color: white; padding: 5px;'>
      <p>Custom HTML</p>
      <img src='path/to/image.png' alt='image' />
    </div>",
    position = "top-left"
  )

# Custom control with specific ID for selective removal
maplibre() |>
  add_control(
    html = "<div style='background: blue; color: white; padding: 10px;'>
      My Control
    </div>",
    position = "top-right",
    id = "my_custom_control"
  )

# Later, remove only this specific control
maplibre_proxy("map") |>
  clear_controls("my_custom_control")

## End(Not run)
```

add_coordinates_control

Add a coordinates control to a map

Description

This function adds a compact control that displays the cursor position as longitude and latitude in WGS84 coordinates.

Usage

```
add_coordinates_control(
  map,
  position = "bottom-right",
  format = c("decimal", "dms"),
  precision = NULL,
  label = NULL,
  empty_text = "Move cursor over map",
  wrap = TRUE
)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "bottom-right".
format	Coordinate display format. One of "decimal" for decimal degrees or "dms" for degrees, minutes, and seconds.
precision	Number of decimal places to display. If NULL, defaults to 5 for decimal degrees and 1 for DMS seconds. For format = "dms", this controls decimal places for seconds.
label	Optional label shown above the coordinates. Default is NULL.
empty_text	Text shown before the cursor enters the map, and after it leaves the map.
wrap	Logical. If TRUE, longitudes are wrapped to the standard [-180, 180] range. Default is TRUE.

Value

The modified map object with the coordinates control added.

Examples

```
## Not run:
library(mapgl)

maplibre() |>
  add_coordinates_control()

mapboxgl() |>
  add_coordinates_control(
    position = "bottom-left",
    format = "dms",
    precision = 2,
    label = "Longitude, latitude"
  )

## End(Not run)
```

add_draw_control	<i>Add a draw control to a map</i>
------------------	------------------------------------

Description

Add a draw control to a map

Usage

```

add_draw_control(
  map,
  position = "top-left",
  freehand = FALSE,
  simplify_freehand = FALSE,
  rectangle = FALSE,
  radius = FALSE,
  bezier = FALSE,
  bezier_polygon = FALSE,
  orientation = "vertical",
  source = NULL,
  attributes = NULL,
  point_color = "#3bb2d0",
  line_color = "#3bb2d0",
  fill_color = "#3bb2d0",
  fill_opacity = 0.1,
  active_color = "#fbb03b",
  vertex_radius = 5,
  line_width = 2,
  download_button = FALSE,
  download_filename = "drawn-features",
  show_measurements = FALSE,
  measurement_units = "both",
  provider = c("mapbox-gl-draw", "terra-draw"),
  modes = NULL,
  options = NULL,
  ...
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	A string specifying the position of the draw control. One of "top-right", "top-left", "bottom-right", or "bottom-left".
freehand	Logical, whether to enable freehand drawing mode. Default is FALSE.
simplify_freehand	Logical, whether to apply simplification to freehand drawings. Default is FALSE.
rectangle	Logical, whether to enable rectangle drawing mode. Default is FALSE.
radius	Logical, whether to enable radius/circle drawing mode. Default is FALSE.
bezier	Logical, whether to enable Bezier curve drawing mode. Default is FALSE.
bezier_polygon	Logical, whether to enable Bezier polygon drawing mode. Default is FALSE.
orientation	A string specifying the orientation of the draw control. Either "vertical" (default) or "horizontal".
source	A character string specifying a source ID to add to the draw control. Default is NULL.

attributes	Optional named list defining editable feature attributes. Use <code>draw_attribute()</code> to define fields.
point_color	Color for point features. Default is "#3bb2d0" (light blue).
line_color	Color for line features. Default is "#3bb2d0" (light blue).
fill_color	Fill color for polygon features. Default is "#3bb2d0" (light blue).
fill_opacity	Fill opacity for polygon features. Default is 0.1.
active_color	Color for active (selected) features. Default is "#fbb03b" (orange).
vertex_radius	Radius of vertex points in pixels. Default is 5.
line_width	Width of lines in pixels. Default is 2.
download_button	Logical, whether to add a download button to export drawn features as GeoJSON. Default is FALSE.
download_filename	Base filename for downloaded GeoJSON (without extension). Default is "drawn-features".
show_measurements	Logical, whether to show live measurements while drawing. Default is FALSE.
measurement_units	Units for measurements. Either "metric", "imperial", or "both". Default is "both".
provider	The drawing engine to use: "mapbox-gl-draw" (the default, current behavior) or "terra-draw" to use the Terra Draw library. Terra Draw works identically on Mapbox and MapLibre maps and offers additional drawing modes plus a richer select/edit mode.
modes	For provider = "terra-draw", a character vector of drawing modes to expose as toolbar buttons. Valid modes are "point", "linestring", "polygon", "rectangle", "circle", "freehand", "freehand-linestring", "angled-rectangle", "sector", "sensor", "curve", "curve-linestring", and "select". When NULL (the default), the mode set is derived from the legacy freehand, rectangle, and radius arguments plus "point", "linestring", "polygon", and "select". For provider = "mapbox-gl-draw", a supplied modes value is forwarded to the MapboxDraw constructor unchanged (equivalent to passing it via ...).
options	For provider = "terra-draw", an object created by <code>terradraw_options()</code> with advanced Terra Draw settings (snapping, select-mode editing flags, per-mode overrides).
...	Additional named arguments for the default "mapbox-gl-draw" provider. See https://github.com/mapbox/mapbox-gl-draw/blob/main/docs/API.md#options for a list of options. Not supported with provider = "terra-draw"; use options there instead.

Details

The terra-draw provider:

With provider = "terra-draw", mapgl renders its own toolbar (Terra Draw is a headless library) with one button per requested mode, a trash button, and an optional download button.

`add_terra draw_control()` is an equivalent convenience wrapper whose signature contains only the arguments that apply to this provider. Behavioral notes:

- The trash button deletes the currently selected feature and does nothing when no feature is selected; use `clear_drawn_features()` to remove everything.
- When the mode set includes "select", finishing a shape returns to the select tool with the new feature selected (set `terra draw_options(keep_mode_active = TRUE)` to keep drawing instead).
- Colors are coerced to 6-digit hex (Terra Draw requires hex), so R color names work but alpha channels are ignored; use `fill_opacity` for transparency.
- Features loaded via source or `add_features_to_draw()` are adapted to Terra Draw's constraints: Multi* geometries are split into single-part features, coordinates are rounded to 9 decimal places (Terra Draw's precision limit, about 0.1 mm), and polygon interior rings (holes) are removed.
- Changing the map style preserves drawn features, but discards an unfinished drawing and clears the current selection and undo history.
- The "curve" and "curve-linestring" modes draw shapes that mix straight and curved (cubic Bezier) edges, pen-tool style: click places a corner point; click-and-drag places an anchor and pulls out symmetric curve handles (drag distance sets the curvature); moving the mouse previews the pending segment; click the first point (the last point for lines) or press Enter to finish; Escape cancels; Backspace removes the last point. The stored feature uses the rendered curved coordinates, so measurements, the download button, and `get_drawn_features()` work unchanged, and the curve's control points are preserved in a `curveNodes` JSON-string property. In select mode curve features can be moved as a whole (their control points move with them) but vertex editing, rotate, and scale are disabled for them, and self-crossing curve outlines cannot be finished as polygons. While a curve tool is active, left-drag places curved anchors, so map panning is suspended until you switch tools.
- `bezier`, `bezier_polygon`, and `simplify_freehand` are specific to mapbox-gl-draw and error under terra-draw. `attributes` and `show_measurements` work with both providers or standalone widgets (as with the default provider, neither is available in compare views).

Bezier modes (mapbox-gl-draw provider only):

Bezier drawing modes are supported when the draw control is added to the original map widget or later through a regular Shiny map proxy. Compare widgets and compare proxies are not yet supported for Bezier modes.

To draw Bezier curves, click the Bezier button, then use **Alt + left-drag** to create nodes with handles. A plain left-click creates nodes without handles. Press Enter, or click the last node, to finish the curve. In direct select mode, select a node and drag its handles to edit the curve; use **Alt + drag** on a handle to break handle symmetry.

Retrieved Bezier features are returned to R as standard sf geometries using the rendered curved coordinates: Bezier curves become LineString features and Bezier polygons become Polygon features. The Bezier control metadata is also preserved in feature-property columns so the browser widget can continue to edit those features as Bezier objects.

When `attributes` is supplied, selecting exactly one drawn feature opens a small attribute editor. Click Save to write values to the feature properties; `get_drawn_features()` returns those properties as sf columns. The editor works for newly drawn features and features loaded into the draw control with source or `add_features_to_draw()`. Compare widgets are not yet supported for attribute editing.

Value

The modified map object with the draw control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl(
  style = mapbox_style("streets"),
  center = c(-74.50, 40),
  zoom = 9
) |>
  add_draw_control()

# With initial features from a source
library(tigris)
tx <- counties(state = "TX", cb = TRUE)
mapboxgl(bounds = tx) |>
  add_source(id = "tx", data = tx) |>
  add_draw_control(source = "tx")

# With custom styling
mapboxgl() |>
  add_draw_control(
    point_color = "#ff0000",
    line_color = "#00ff00",
    fill_color = "#0000ff",
    fill_opacity = 0.3,
    active_color = "#ff00ff",
    vertex_radius = 7,
    line_width = 3
  )

# Enable rectangle drawing mode
mapboxgl() |>
  add_draw_control(rectangle = TRUE)

# Enable radius/circle drawing mode
mapboxgl() |>
  add_draw_control(radius = TRUE)

# Enable Bezier curve drawing mode
mapboxgl() |>
  add_draw_control(bezier = TRUE)

# Add an attribute editor for classification workflows
mapboxgl() |>
  add_draw_control(
    attributes = list(
      class = draw_attribute(
        "select",
```

```

        choices = c("forest", "water", "urban"),
        required = TRUE
    ),
    notes = draw_attribute("textarea"),
    confidence = draw_attribute(
        "numeric",
        min = 0,
        max = 1,
        step = 0.1,
        default = 1
    )
)
)

# Enable multiple drawing modes
mapboxgl() |>
  add_draw_control(
    freehand = TRUE,
    rectangle = TRUE,
    radius = TRUE,
    bezier = TRUE
  )

# Use the Terra Draw engine (works on Mapbox and MapLibre maps)
maplibre() |>
  add_draw_control(
    provider = "terra-draw",
    modes = c("point", "polygon", "rectangle", "circle", "select"),
    options = terradraw_options(snap_to_coordinates = TRUE)
  )

## End(Not run)

```

add_features_to_draw *Add features to an existing draw control*

Description

This function adds features from an existing source to a draw control on a map.

Usage

```
add_features_to_draw(map, source, clear_existing = FALSE)
```

Arguments

map	A map object with a draw control already added
source	Character string specifying a source ID to get features from
clear_existing	Logical, whether to clear existing drawn features before adding new ones. Default is FALSE.

Value

The modified map object

Examples

```
## Not run:
library(mapgl)
library(tigris)

# Add features from an existing source
tx <- counties(state = "TX", cb = TRUE)
mapboxgl(bounds = tx) |>
  add_source(id = "tx", data = tx) |>
  add_draw_control() |>
  add_features_to_draw(source = "tx")

# In a Shiny app
observeEvent(input$load_data, {
  mapboxgl_proxy("map") |>
    add_features_to_draw(
      source = "dynamic_data",
      clear_existing = TRUE
    )
})

## End(Not run)
```

add_fill_extrusion_layer

Add a fill-extrusion layer to a Mapbox GL map

Description

Add a fill-extrusion layer to a Mapbox GL map

Usage

```
add_fill_extrusion_layer(
  map,
  id,
  source,
  source_layer = NULL,
  fill_extrusion_ambient_occlusion_intensity = NULL,
  fill_extrusion_ambient_occlusion_radius = NULL,
  fill_extrusion_base = NULL,
  fill_extrusion_cast_shadows = NULL,
  fill_extrusion_color = NULL,
  fill_extrusion_cutoff_fade_range = NULL,
```

```

    fill_extrusion_emissive_strength = NULL,
    fill_extrusion_height = NULL,
    fill_extrusion_opacity = NULL,
    fill_extrusion_pattern = NULL,
    fill_extrusion_translate = NULL,
    fill_extrusion_translate_anchor = "map",
    fill_extrusion_vertical_gradient = NULL,
    visibility = "visible",
    slot = NULL,
    min_zoom = NULL,
    max_zoom = NULL,
    popup = NULL,
    tooltip = NULL,
    tooltip_style = NULL,
    popup_style = NULL,
    hover_options = NULL,
    before_id = NULL,
    filter = NULL
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies type and url for a remote source.
source_layer	The source layer (for vector sources).
fill_extrusion_ambient_occlusion_intensity	Controls the intensity of ambient occlusion shading. Value between 0 and 1; around 0.3 provides the most plausible results for buildings.
fill_extrusion_ambient_occlusion_radius	Shades area near ground and concave angles between walls. Default 3.0 corresponds to one floor height.
fill_extrusion_base	The base height of the fill extrusion.
fill_extrusion_cast_shadows	If TRUE (the default), the fill extrusion casts shadows.
fill_extrusion_color	The color of the fill extrusion.
fill_extrusion_cutoff_fade_range	Defines the fade-out range before automatic content cutoff on pitched views. Value between 0 and 1; 0 disables cutoff.
fill_extrusion_emissive_strength	Controls the intensity of light emitted on the source features. Requires 3D lights.
fill_extrusion_height	The height of the fill extrusion.

fill_extrusion_opacity	The opacity of the fill extrusion.
fill_extrusion_pattern	Name of image in sprite to use for drawing image fills.
fill_extrusion_translate	The geometry's offset. Values are c(x, y) where negatives indicate left and up.
fill_extrusion_translate_anchor	Controls the frame of reference for fill-extrusion-translate.
fill_extrusion_vertical_gradient	If TRUE (the default), sides will be shaded slightly darker farther down.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.
popup	Popup content shown on click: a column name, a {brace} template (e.g. "{name}:{value}"), or a concat()/number_format() expression. Columns containing HTML are parsed.
tooltip	Tooltip content shown on hover; same forms as popup.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a tooltip_style() object. When omitted, the native (unstyled) appearance is kept.
hover_options	A named list of options for highlighting features in the layer on hover.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.

Value

The modified map object with the new fill-extrusion layer added.

Examples

```
## Not run:
library(mapgl)

maplibre(
  style = maptiler_style("basic"),
  center = c(-74.0066, 40.7135),
  zoom = 15.5,
  pitch = 45,
  bearing = -17.6
) |>
  add_vector_source(
    id = "openmaptiles",
    url = paste0(
      "https://api.maptiler.com/tiles/v3/tiles.json?key=",
```

```

        Sys.getenv("MAPTILER_API_KEY")
    )
) |>
add_fill_extrusion_layer(
  id = "3d-buildings",
  source = "openmaptiles",
  source_layer = "building",
  fill_extrusion_color = interpolate(
    column = "render_height",
    values = c(0, 200, 400),
    stops = c("lightgray", "royalblue", "lightblue")
  ),
  fill_extrusion_height = list(
    "interpolate",
    list("linear"),
    list("zoom"),
    15,
    0,
    16,
    list("get", "render_height")
  )
)

## End(Not run)

```

add_fill_layer	<i>Add a fill layer to a map</i>
----------------	----------------------------------

Description

Add a fill layer to a map

Usage

```

add_fill_layer(
  map,
  id,
  source,
  source_layer = NULL,
  fill_antialias = TRUE,
  fill_color = NULL,
  fill_emissive_strength = NULL,
  fill_opacity = NULL,
  fill_outline_color = NULL,
  fill_pattern = NULL,
  fill_pattern_cross_fade = NULL,
  fill_sort_key = NULL,
  fill_translate = NULL,
  fill_translate_anchor = "map",

```

```

    fill_z_offset = NULL,
    visibility = "visible",
    slot = NULL,
    min_zoom = NULL,
    max_zoom = NULL,
    popup = NULL,
    tooltip = NULL,
    tooltip_style = NULL,
    popup_style = NULL,
    hover_options = NULL,
    before_id = NULL,
    filter = NULL
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies type and url for a remote source.
source_layer	The source layer (for vector sources).
fill_antialias	Whether or not the fill should be antialiased.
fill_color	The color of the filled part of this layer.
fill_emissive_strength	Controls the intensity of light emitted on the source features.
fill_opacity	The opacity of the entire fill layer.
fill_outline_color	The outline color of the fill.
fill_pattern	Name of image in sprite to use for drawing image fills.
fill_pattern_cross_fade	Controls the transition progress between image variants of fill_pattern. Value between 0 and 1.
fill_sort_key	Sorts features in ascending order based on this value.
fill_translate	The geometry's offset. Values are c(x, y) where negatives indicate left and up.
fill_translate_anchor	Controls the frame of reference for fill-translate.
fill_z_offset	Specifies an uniform elevation in meters.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.

popup	Popup content shown on click: a column name, a {brace} template (e.g. "{name}:{value}"), or a concat()/number_format() expression. Columns containing HTML are parsed.
tooltip	Tooltip content shown on hover; same forms as popup.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a <code>tooltip_style()</code> object. When omitted, the native (unstyled) appearance is kept.
hover_options	A named list of options for highlighting features in the layer on hover.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.

Value

The modified map object with the new fill layer added.

Examples

```
## Not run:
library(tidycensus)

fl_age <- get_acs(
  geography = "tract",
  variables = "B01002_001",
  state = "FL",
  year = 2022,
  geometry = TRUE
)

mapboxgl() |>
  fit_bounds(fl_age, animate = FALSE) |>
  add_fill_layer(
    id = "fl_tracts",
    source = fl_age,
    fill_color = interpolate(
      column = "estimate",
      values = c(20, 80),
      stops = c("lightblue", "darkblue"),
      na_color = "lightgrey"
    ),
    fill_opacity = 0.5
  )

## End(Not run)
```

add_flowmap	<i>Adds a FlowmapGL layer for visualizing origin-destination flows between point locations.</i>
-------------	---

Description

Adds a FlowmapGL layer for visualizing origin-destination flows between point locations.

Usage

```
add_flowmap(
  map,
  id,
  locations,
  flows,
  flow_color_scheme = "Teal",
  flow_opacity = 1,
  flow_dark_mode = "auto",
  flow_blend = "auto",
  flow_fade_amount = 50,
  flow_highlight_color = "#ff9b29",
  flow_locations_enabled = TRUE,
  flow_location_totals_enabled = TRUE,
  flow_location_labels_enabled = FALSE,
  flow_lines_rendering_mode = c("straight", "animated-straight", "curved"),
  flow_line_thickness_scale = 1,
  flow_line_curviness = 1,
  flow_clustering_enabled = TRUE,
  flow_clustering_auto = TRUE,
  flow_clustering_level = NULL,
  flow_fade_enabled = TRUE,
  flow_fade_opacity_enabled = FALSE,
  flow_adaptive_scales_enabled = TRUE,
  flow_temporal_scale_domain = c("selected", "all"),
  flow_max_top_flows_display_num = 5000,
  flow_endpoints_in_viewport_mode = c("any", "both"),
  flow_time_column = NULL,
  flow_selected_time_range = NULL,
  flow_selected_locations = NULL,
  flow_location_filter_mode = c("ALL", "INCOMING", "OUTGOING", "BETWEEN"),
  tooltip = TRUE,
  popup = FALSE,
  tooltip_style = NULL,
  popup_style = NULL,
  visibility = c("visible", "none"),
  before_id = NULL,
  slot = NULL
```

)

Arguments

map	A map object created by <code>mapboxgl()</code> or <code>maplibre()</code> .
id	A unique layer ID.
locations	A data frame or sf point object with location data. Data frames must include id, lat, and lon columns. sf point objects must include id; coordinates are transformed to EPSG:4326 and serialized as lon/lat.
flows	A data frame with origin, dest, and count columns.
flow_color_scheme	FlowMapGL preset color scheme name, a character vector of at least two CSS colors, or a <code>mapgl_continuous_scale</code> object created by <code>interpolate_palette()</code> . Preset names are case-sensitive; use <code>flowmap_color_schemes()</code> to list them.
flow_opacity	Layer opacity between 0 and 1.
flow_dark_mode	Logical (TRUE or FALSE), or "auto"; whether to use FlowMapGL dark-mode colors. If "auto", the mode is dynamically detected based on the map style.
flow_blend	Logical (TRUE or FALSE), "auto", or a character string specifying a CSS mix-blend-mode. Valid modes are: "normal", "multiply", "screen", "overlay", "darken", "lighten", "color-dodge", "color-burn", "hard-light", "soft-light", "difference", "exclusion", "hue", "saturation", "color", and "luminosity".

Recommendations:

- On **dark basemaps**: "screen" looks best, creating a glowing additive effect where flows overlap.
- On **light basemaps**: "multiply" or "darken" looks best, increasing contrast against the light background.

If "auto", automatically chooses "screen" for dark styles and "multiply" for light styles. If `before_id` or `slot` is specified (interleaved mode), "auto" quietly disables blending (FALSE) without throwing a warning. If TRUE, defaults to "screen" when `flow_dark_mode` is TRUE, and "multiply" when FALSE. If FALSE, no blending is applied. Note: CSS blending requires a standalone canvas overlay and is ignored when `before_id` or `slot` is specified.

flow_fade_amount	Controls how much lower-magnitude flows fade compared to higher ones. Range: 0-100.
flow_highlight_color	Color used for highlighting hovered elements.
flow_locations_enabled	Whether to show location circles.
flow_location_totals_enabled	Whether to show incoming/outgoing totals as concentric circles at each location.
flow_location_labels_enabled	Whether to show text labels at locations.

flow_lines_rendering_mode	Controls how flow lines are rendered: "straight", "animated-straight", or "curved".
flow_line_thickness_scale	Multiplier for flow line thickness.
flow_line_curviness	Multiplier for flow line curviness (only used when flow_lines_rendering_mode is "curved").
flow_clustering_enabled	Whether to cluster nearby locations when zoomed out.
flow_clustering_auto	Whether to automatically adjust clustering level based on zoom.
flow_clustering_level	Fixed clustering zoom level. Only used when flow_clustering_auto is FALSE.
flow_fade_enabled	Whether to apply color fading to lower-magnitude flows.
flow_fade_opacity_enabled	Whether to also fade opacity for lower-magnitude flows.
flow_adaptive_scales_enabled	Whether to adapt flow thickness and color scales to the current viewport. This controls the spatial scale domain while panning and zooming.
flow_temporal_scale_domain	For temporal flowmaps, whether flow thickness and color scales use only the currently selected time range ("selected") or all flow data across the full time extent ("all").
flow_max_top_flows_display_num	Maximum number of flows to display.
flow_endpoints_in_viewport_mode	Controls when a flow is considered visible based on endpoint locations: "any" or "both".
flow_time_column	Optional column name in flows for time data.
flow_selected_time_range	Optional vector of two dates (or strings) for initial time filtering.
flow_selected_locations	Optional vector of location IDs to select.
flow_location_filter_mode	Optional location filter mode: "ALL", "INCOMING", "OUTGOING", or "BETWEEN".
tooltip	Hover tooltip content. Use FALSE/NULL to disable, TRUE for the default, a column name, a {brace} template, a concat()/number_format() expression, or a named list(location = , flow =) to set different content for hovered locations vs flows. Available fields are, for locations: name, id, lat, lon, incoming, outgoing, internal (plus any locations columns); for flows: count, origin_id, origin_name, dest_id, dest_name (plus any flows columns).
popup	Click popup content. Same forms as tooltip; defaults to FALSE (disabled).

tooltip_style	Tooltip appearance. A preset string ("light", "dark", or "auto") or a <code>tooltip_style()</code> object (colors, border, radius, font, padding, shadow, and position = "floating"/"anchored"). Defaults to a light/dark preset matching flow_dark_mode.
popup_style	Popup appearance; same forms as tooltip_style.
visibility	Whether the layer is initially "visible" or "none".
before_id	Optional map layer ID to render before.
slot	Optional Mapbox Standard slot.

Details

Mapbox and MapLibre layer paint arguments such as `fill_color`, `circle_color`, and `line_color` require a scalar CSS color or a style expression. Use `interpolate_palette(...)$expression` for data-driven layer color ramps. FlowMapGL's `flow_color_scheme` accepts a preset name such as "Teal", a plain color ramp such as `c("red", "white", "blue")`, or an `interpolate_palette(...)` scale object.

Flow scale domains have separate spatial and temporal controls. `flow_adaptive_scales_enabled = TRUE` rescales flow thickness and color for the current viewport; `FALSE` keeps the scale tied to the broader map extent. For temporal flowmaps, `flow_temporal_scale_domain = "selected"` rescales within the selected time-control interval, while "all" keeps the scale comparable across the full time extent.

Value

The modified map object with the flowmap layer added.

Examples

```
# Create a flowmap centered on Montréal using the bundled datasets
maplibre(
  style = carto_style("dark-matter"),
  center = c(-73.58, 45.50),
  zoom = 11,
  projection = "mercator"
) |>
  add_flowmap(
    id = "bixi-rides",
    locations = bixi_locations,
    flows = bixi_flows,
    flow_time_column = "time",
    flow_color_scheme = "Teal",
    flow_dark_mode = TRUE
  )

# To animate the flows over time, pair the flowmap with a window-mode
# slider that targets the same layer id; see [add_slider_control()].
```

add_fullscreen_control

Add a fullscreen control to a map

Description

Add a fullscreen control to a map

Usage

```
add_fullscreen_control(map, position = "top-right")
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	A string specifying the position of the fullscreen control. One of "top-right", "top-left", "bottom-right", or "bottom-left".

Value

The modified map object with the fullscreen control added.

Examples

```
## Not run:
library(mapgl)

maplibre(
  style = maptiler_style("streets"),
  center = c(11.255, 43.77),
  zoom = 13
) |>
  add_fullscreen_control(position = "top-right")

## End(Not run)
```

add_geocoder_control *Add a geocoder control to a map*

Description

This function adds a Geocoder search bar to a Mapbox GL or MapLibre GL map. By default, a marker will be added at the selected location and the map will fly to that location. The results of the geocode are accessible in a Shiny session at `input$MAPID_geocoder$result`, where MAPID is the name of your map.

Usage

```
add_geocoder_control(
  map,
  position = "top-right",
  placeholder = "Search",
  collapsed = FALSE,
  provider = NULL,
  maptiler_api_key = NULL,
  ...
)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".
placeholder	A string to use as placeholder text for the search bar. Default is "Search".
collapsed	Whether the control should be collapsed until hovered or clicked. Default is FALSE.
provider	The geocoding provider to use for MapLibre maps. Either "osm" for OpenStreetMap/Nominatim or "maptiler" for MapTiler geocoding. If NULL (default), MapLibre maps will use "osm". Mapbox maps will always use the Mapbox geocoder, regardless of this parameter.
maptiler_api_key	Your MapTiler API key (required when provider is "maptiler" for MapLibre maps). Can also be set with <code>MAPTILER_API_KEY</code> environment variable. Mapbox maps will always use the Mapbox API key set at the map level.
...	Additional parameters to pass to the Geocoder.

Value

The modified map object with the geocoder control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl() |>
  add_geocoder_control(position = "top-left", placeholder = "Enter an address")

maplibre() |>
  add_geocoder_control(position = "top-right", placeholder = "Search location")

# Using MapTiler geocoder
maplibre() |>
  add_geocoder_control(provider = "maptiler", maptiler_api_key = "YOUR_API_KEY")

## End(Not run)
```

add_geolocate_control *Add a geolocate control to a map*

Description

This function adds a Geolocate control to a Mapbox GL or MapLibre GL map. The geolocate control allows users to track their current location on the map.

Usage

```
add_geolocate_control(
  map,
  position = "top-right",
  track_user = FALSE,
  show_accuracy_circle = TRUE,
  show_user_location = TRUE,
  show_user_heading = FALSE,
  fit_bounds_options = list(maxZoom = 15),
  position_options = list(enableHighAccuracy = FALSE, timeout = 6000)
)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".
track_user	Whether to actively track the user's location. If TRUE, the map will continuously update as the user moves. Default is FALSE.
show_accuracy_circle	Whether to show a circle indicating the accuracy of the location. Default is TRUE.
show_user_location	Whether to show a dot at the user's location. Default is TRUE.
show_user_heading	Whether to show an arrow indicating the device's heading when tracking location. Only works when track_user is TRUE. Default is FALSE.
fit_bounds_options	A list of options for fitting bounds when panning to the user's location. Default maxZoom is 15.
position_options	A list of Geolocation API position options. Default has enableHighAccuracy=FALSE and timeout=6000.

Value

The modified map object with the geolocate control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl() |>
  add_geolocate_control(
    position = "top-right",
    track_user = TRUE,
    show_user_heading = TRUE
  )

## End(Not run)
```

add_globe_control	<i>Add a globe control to a map</i>
-------------------	-------------------------------------

Description

This function adds a globe control to a MapLibre GL map that allows toggling between "mercator" and "globe" projections with a single click.

Usage

```
add_globe_control(map, position = "top-right")
```

Arguments

map	A map object created by the maplibre function.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".

Value

The modified map object with the globe control added.

Examples

```
## Not run:
library(mapgl)

maplibre() |>
  add_globe_control(position = "top-right")

## End(Not run)
```

add_globe_minimap	<i>Add a Globe Minimap to a map</i>
-------------------	-------------------------------------

Description

This function adds a globe minimap control to a Mapbox GL or Maplibre map.

Usage

```
add_globe_minimap(
  map,
  position = "bottom-right",
  globe_size = 82,
  land_color = "white",
  water_color = "rgba(30 40 70/60%)",
  marker_color = "#ff2233",
  marker_size = 1
)
```

Arguments

map	A mapboxgl or maplibre object.
position	A string specifying the position of the minimap.
globe_size	Number of pixels for the diameter of the globe. Default is 82.
land_color	HTML color to use for land areas on the globe. Default is 'white'.
water_color	HTML color to use for water areas on the globe. Default is 'rgba(30 40 70/60%)'.
marker_color	HTML color to use for the center point marker. Default is '#ff2233'.
marker_size	Scale ratio for the center point marker. Default is 1.

Value

The modified map object with the globe minimap added.

Examples

```
## Not run:
library(mapgl)

m <- mapboxgl() %>%
  add_globe_minimap()

m <- maplibre() %>%
  add_globe_minimap()

## End(Not run)
```

add_h3j_source	<i>Add a hexagon source from the H3 geospatial indexing system.</i>
----------------	---

Description

Add a hexagon source from the H3 geospatial indexing system.

Usage

```
add_h3j_source(map, id, url)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the vector tile source.

References

<https://h3geo.org>, <https://github.com/INSPIDE/h3j-h3t>

Examples

```
url = "https://inspide.github.io/h3j-h3t/examples/h3j/sample.h3j"
maplibre(center=c(-3.704, 40.417), zoom=15, pitch=30) |>
  add_h3j_source("h3j_testsource",
                url = url
  ) |>
  add_fill_extrusion_layer(
    id = "h3j_testlayer",
    source = "h3j_testsource",
    fill_extrusion_color = interpolate(
      column = "value",
      values = c(0, 21.864),
      stops = c("#430254", "#f83c70")
    ),
    fill_extrusion_height = list(
      "interpolate",
      list("linear"),
      list("zoom"),
      14,
      0,
      15.05,
      list("*", 10, list("get", "value"))
    ),
    fill_extrusion_opacity = 0.7
  )
```

add_h3t_source

Add a tiled hexagon source from the H3 geospatial indexing system.

Description

Wraps the **h3t** tile protocol, which registers a `h3tiles://` MapLibre protocol and fetches H3J-formatted JSON tiles from a `{z}/{x}/{y}` endpoint. Unlike `add_h3j_source()`, which pulls a single H3J file, this source lets the map request only the cells visible in the current viewport — on pan/zoom the protocol handler fires a fresh request per tile.

Usage

```
add_h3t_source(
  map,
  id,
  tiles,
  sourcelayer = id,
  geometry_type = c("Polygon", "Point"),
  minzoom = 0,
  maxzoom = 14,
  promote_id = TRUE,
  debug = FALSE
)
```

Arguments

map	A map object created by <code>maplibre()</code> or a <code>maplibre_proxy</code> .
id	Unique source ID.
tiles	A character vector of tile URL templates, each using the <code>h3tiles://</code> scheme. The tokens <code>{z}</code> , <code>{x}</code> , <code>{y}</code> are substituted by MapLibre on each tile request. Example: <code>"h3tiles://h3t.example.com/{z}/{x}/{y}.h3t?q=..."</code> .
sourcelayer	Name of the source layer that downstream <code>add_fill_layer()</code> (or similar) calls reference via <code>source_layer</code> . Defaults to <code>id</code> .
geometry_type	Either <code>"Polygon"</code> (hex boundaries) or <code>"Point"</code> (cell centroids). Defaults to <code>"Polygon"</code> .
minzoom, maxzoom	Zoom bounds for the source (MapLibre semantics).
promote_id	Whether to promote the <code>h3id</code> property to the feature ID.
debug	If <code>TRUE</code> , the protocol handler logs per-tile timing to the browser console.

References

<https://github.com/INSPIDE/h3j-h3t>

Examples

```
maplibre(center = c(-119, 34), zoom = 5) |>
  add_h3t_source(
    id = "sardine",
    tiles = "h3tiles://h3t.example.com/{z}/{x}/{y}.h3t?q=<base64-SELECT>"
  ) |>
  add_fill_layer(
    id = "sardine",
    source = "sardine",
    source_layer = "sardine",
    fill_color = interpolate(
      column = "value", values = c(0, 100),
      stops = c("#ffffcc", "#e31a1c")
    ),
    fill_opacity = 0.7
  )
```

add_heatmap_layer	<i>Add a heatmap layer to a Mapbox GL map</i>
-------------------	---

Description

Add a heatmap layer to a Mapbox GL map

Usage

```
add_heatmap_layer(
  map,
  id,
  source,
  source_layer = NULL,
  heatmap_color = NULL,
  heatmap_intensity = NULL,
  heatmap_opacity = NULL,
  heatmap_radius = NULL,
  heatmap_weight = NULL,
  visibility = "visible",
  slot = NULL,
  min_zoom = NULL,
  max_zoom = NULL,
  before_id = NULL,
  filter = NULL
)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an <code>sf</code> object (which will be converted to a GeoJSON source) or a named list that specifies <code>type</code> and <code>url</code> for a remote source.
source_layer	The source layer (for vector sources).
heatmap_color	The color of the heatmap points.
heatmap_intensity	The intensity of the heatmap points.
heatmap_opacity	The opacity of the heatmap layer.
heatmap_radius	The radius of influence of each individual heatmap point.
heatmap_weight	The weight of each individual heatmap point.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.

Value

The modified map object with the new heatmap layer added.

Examples

```
## Not run:
library(mapgl)

mapboxgl(
  style = mapbox_style("dark"),
  center = c(-120, 50),
  zoom = 2
) |>
  add_heatmap_layer(
    id = "earthquakes-heat",
    source = list(
      type = "geojson",
      data = "https://docs.mapbox.com/mapbox-gl-js/assets/earthquakes.geojson"
    ),
    heatmap_weight = interpolate(
      column = "mag",
      values = c(0, 6),
      stops = c(0, 1)
    )
  )
```

```

    ),
    heatmap_intensity = interpolate(
        property = "zoom",
        values = c(0, 9),
        stops = c(1, 3)
    ),
    heatmap_color = interpolate(
        property = "heatmap-density",
        values = seq(0, 1, 0.2),
        stops = c(
            "rgba(33,102,172,0)", "rgb(103,169,207)",
            "rgb(209,229,240)", "rgb(253,219,199)",
            "rgb(239,138,98)", "rgb(178,24,43)"
        )
    ),
    heatmap_opacity = 0.7
)

## End(Not run)

```

add_image

Add an image to the map

Description

This function adds an image to the map's style. The image can be used with icon-image, background-pattern, fill-pattern, or line-pattern. It can also be used with a style list, such as one created by [basemap_style\(\)](#), before the style is supplied to [maplibre\(\)](#) or [mapboxgl\(\)](#).

Usage

```

add_image(
  map,
  id,
  url,
  content = NULL,
  pixel_ratio = 1,
  sdf = FALSE,
  stretch_x = NULL,
  stretch_y = NULL
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions, a map proxy, or a style list.
id	A string specifying the ID of the image.

url	A string specifying the URL of the image to be loaded or a path to a local image file. Must be PNG or JPEG format.
content	A vector of four numbers c(x1, y1, x2, y2) defining the part of the image that can be covered by the content in text-field if icon-text-fit is used.
pixel_ratio	A number specifying the ratio of pixels in the image to physical pixels on the screen.
sdf	A logical value indicating whether the image should be interpreted as an SDF image.
stretch_x	A list of number pairs defining the part(s) of the image that can be stretched horizontally.
stretch_y	A list of number pairs defining the part(s) of the image that can be stretched vertically.

Value

The modified map object or style list with the image added.

Examples

```
## Not run:
library(mapgl)

# Path to your local image file OR a URL to a remote image file
# that is not blocked by CORS restrictions
image_path <- "/path/to/your/image.png"

pts <- tigris::landmarks("DE")[1:100, ]

maplibre(bounds = pts) |>
  add_image("local_icon", image_path) |>
  add_symbol_layer(
    id = "local_icons",
    source = pts,
    icon_image = "local_icon",
    icon_size = 0.5,
    icon_allow_overlap = TRUE
  )

## End(Not run)
```

add_image_source	<i>Add an image source to a Mapbox GL or Maplibre GL map</i>
------------------	--

Description

Add an image source to a Mapbox GL or Maplibre GL map

Usage

```
add_image_source(
  map,
  id,
  url = NULL,
  data = NULL,
  coordinates = NULL,
  colors = NULL
)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the image source.
data	A SpatRaster object from the terra package or a RasterLayer object.
coordinates	A list of coordinates specifying the image corners in clockwise order: top left, top right, bottom right, bottom left. For SpatRaster or RasterLayer objects, this will be extracted for you.
colors	A vector of colors to use for the raster image.

Value

The modified map object with the new source added.

add_layer	<i>Add a layer to a map from a source</i>
-----------	---

Description

In many cases, you will use add_layer() internal to other layer-specific functions in mapgl. Advanced users will want to use add_layer() for more fine-grained control over the appearance of their layers.

Usage

```
add_layer(
  map,
  id,
  type = "fill",
  source,
  source_layer = NULL,
  paint = list(),
  layout = list(),
  slot = NULL,
```

```

    min_zoom = NULL,
    max_zoom = NULL,
    popup = NULL,
    tooltip = NULL,
    hover_options = NULL,
    before_id = NULL,
    filter = NULL,
    tooltip_style = NULL,
    popup_style = NULL,
    metadata = NULL
  )

```

Arguments

map	A map object created by the <code>mapboxgl()</code> or <code>maplibre()</code> functions.
id	A unique ID for the layer.
type	The type of the layer (e.g., "fill", "line", "circle").
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies <code>type</code> and <code>url</code> for a remote source.
source_layer	The source layer (for vector sources).
paint	A list of paint properties for the layer.
layout	A list of layout properties for the layer.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.
popup	Popup content shown on click: a column name, a {brace} template (e.g. "{name}:{value}"), or a <code>concat()</code> / <code>number_format()</code> expression. Columns containing HTML are parsed.
tooltip	Tooltip content shown on hover; same forms as popup.
hover_options	A named list of options for highlighting features in the layer on hover.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a <code>tooltip_style()</code> object. When omitted, the native (unstyled) appearance is kept.
metadata	An optional named list stored as the layer's metadata style property. It is not rendered, but is available to JavaScript through <code>map.getLayer(id).metadata</code> .

Value

The modified map object with the new layer added.

Examples

```
## Not run:
# Load necessary libraries
library(mapgl)
library(tigris)

# Load geojson data for North Carolina tracts
nc_tracts <- tracts(state = "NC", cb = TRUE)

# Create a Mapbox GL map
map <- mapboxgl(
  style = mapbox_style("light"),
  center = c(-79.0193, 35.7596),
  zoom = 7
)

# Add a source and fill layer for North Carolina tracts
map %>%
  add_source(
    id = "nc-tracts",
    data = nc_tracts
  ) %>%
  add_layer(
    id = "nc-layer",
    type = "fill",
    source = "nc-tracts",
    paint = list(
      "fill-color" = "#888888",
      "fill-opacity" = 0.4
    )
  )

## End(Not run)
```

add_layers_control	<i>Add a layers control to the map</i>
--------------------	--

Description

The layers control is a native map control: it stacks alongside other controls (navigation, fullscreen, etc.) in its corner rather than overlapping them, following the order in which controls are added to the map. Its default appearance matches the other map controls (white background, dark monochrome items). To restore the blue active style from earlier versions of mapgl, set `active_color = "#4a90e2"` and `active_text_color = "#ffffff"`.

Usage

```
add_layers_control(
  map,
```

```

position = "top-left",
layers = NULL,
collapsible = TRUE,
use_icon = TRUE,
background_color = NULL,
active_color = NULL,
hover_color = NULL,
active_text_color = NULL,
inactive_text_color = NULL,
margin_top = NULL,
margin_right = NULL,
margin_bottom = NULL,
margin_left = NULL,
mode = c("multiple", "single")
)

```

Arguments

map	A map object.
position	The position of the control on the map (one of "top-left", "top-right", "bottom-left", "bottom-right").
layers	Either a character vector of layer IDs to include in the control, a named list/vector where names are labels and values are layer IDs, or a named list where values can be vectors to group multiple layers together. If NULL, all layers will be included.
collapsible	Whether the control should be collapsible.
use_icon	Whether to use a stacked layers icon instead of the "Layers" text when collapsed. Only applies when collapsible = TRUE.
background_color	The background color for the layers control; this will be the color used for inactive layer items.
active_color	The background color for active layer items.
hover_color	The background color for layer items when hovered.
active_text_color	The text color for active layer items.
inactive_text_color	The text color for inactive layer items.
margin_top	Optional top margin in pixels, applied to the control within the native control stack. Rarely needed now that the control no longer overlaps other controls; NULL (the default) uses standard control spacing.
margin_right	Optional right margin in pixels. Default is NULL.
margin_bottom	Optional bottom margin in pixels. Default is NULL.
margin_left	Optional left margin in pixels. Default is NULL.

mode How many entries can be visible at once. In "multiple" mode (the default), entries toggle independently. In "single" mode, activating an entry turns the others off — useful for flipping through alternative analytical layers or raster imagery where only one should show at a time. Clicking the active entry does nothing. When the control is created in "single" mode with several entries visible, the first visible entry stays on and the rest are turned off. To combine both behaviors on one map, add two controls: one "single"-mode control for the alternatives and one "multiple"-mode control for independent overlays.

Value

The modified map object with the layers control added.

Examples

```
## Not run:
library(tigris)
options(tigris_use_cache = TRUE)

rds <- roads("TX", "Tarrant")
tr <- tracts("TX", "Tarrant", cb = TRUE)
cty <- counties("TX", cb = TRUE)

maplibre() |>
  fit_bounds(rds) |>
  add_fill_layer(
    id = "Census tracts",
    source = tr,
    fill_color = "purple",
    fill_opacity = 0.6
  ) |>
  add_line_layer(
    "Local roads",
    source = rds,
    line_color = "pink"
  ) |>
  add_layers_control(
    position = "top-left",
    background_color = "#ffffff",
    active_color = "#4a90e2"
  )

# With custom labels
maplibre() |>
  add_fill_layer(id = "tract-fill", source = tr) |>
  add_line_layer(id = "tract-line", source = tr) |>
  add_layers_control(
    layers = list(
      "Census Tracts" = "tract-fill",
      "Tract Borders" = "tract-line"
    )
  )
```

```
# Group multiple layers together
maplibre(bounds = cty) |>
  add_fill_layer(id = "county-fill", source = cty, fill_opacity = 0.3) |>
  add_line_layer(
    id = "county-outline",
    source = cty,
    line_color = "yellow",
    line_width = 3
  ) |>
  add_line_layer(
    id = "roads-layer",
    source = rds,
    line_color = "blue"
  ) |>
  add_layers_control(
    layers = list(
      "Counties" = c("county-fill", "county-outline"),
      "Roads" = "roads-layer"
    )
  )

## End(Not run)
```

add_line_layer	<i>Add a line layer to a map</i>
----------------	----------------------------------

Description

Add a line layer to a map

Usage

```
add_line_layer(
  map,
  id,
  source,
  source_layer = NULL,
  line_blur = NULL,
  line_cap = NULL,
  line_color = NULL,
  line_dasharray = NULL,
  line_elevation_ground_scale = NULL,
  line_elevation_reference = NULL,
  line_emissive_strength = NULL,
  line_gap_width = NULL,
  line_gradient = NULL,
  line_join = NULL,
  line_miter_limit = NULL,
```

```

    line_occlusion_opacity = NULL,
    line_offset = NULL,
    line_opacity = NULL,
    line_pattern = NULL,
    line_pattern_cross_fade = NULL,
    line_round_limit = NULL,
    line_sort_key = NULL,
    line_translate = NULL,
    line_translate_anchor = "map",
    line_trim_color = NULL,
    line_trim_fade_range = NULL,
    line_trim_offset = NULL,
    line_width = NULL,
    line_z_offset = NULL,
    visibility = "visible",
    slot = NULL,
    min_zoom = NULL,
    max_zoom = NULL,
    popup = NULL,
    tooltip = NULL,
    tooltip_style = NULL,
    popup_style = NULL,
    hover_options = NULL,
    before_id = NULL,
    filter = NULL
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies type and url for a remote source.
source_layer	The source layer (for vector sources).
line_blur	Amount to blur the line, in pixels.
line_cap	The display of line endings. One of "butt", "round", "square".
line_color	The color with which the line will be drawn.
line_dasharray	Specifies the lengths of the alternating dashes and gaps that form the dash pattern.
line_elevation_ground_scale	Controls how much the elevation of lines scales with terrain exaggeration when line_elevation_reference is "sea". Value between 0 and 1; 0 keeps the line at fixed altitude, 1 scales proportionally.
line_elevation_reference	Selects the base of line elevation. One of "none", "sea", "ground", or "hd-road-markup".

line_emissive_strength	Controls the intensity of light emitted on the source features.
line_gap_width	Draws a line casing outside of a line's actual path. Value indicates the width of the inner gap.
line_gradient	A gradient used to color a line feature at various distances along its length.
line_join	The display of lines when joining.
line_miter_limit	Used to automatically convert miter joins to bevel joins for sharp angles.
line_occlusion_opacity	Opacity multiplier of the line part that is occluded by 3D objects.
line_offset	The line's offset.
line_opacity	The opacity at which the line will be drawn.
line_pattern	Name of image in sprite to use for drawing image lines.
line_pattern_cross_fade	Controls the transition progress between image variants of line_pattern. Value between 0 and 1.
line_round_limit	Used to automatically convert round joins to miter joins for shallow angles.
line_sort_key	Sorts features in ascending order based on this value.
line_translate	The geometry's offset. Values are c(x, y) where negatives indicate left and up, respectively.
line_translate_anchor	Controls the frame of reference for line-translate.
line_trim_color	The color to be used for rendering the trimmed line section.
line_trim_fade_range	The fade range for the trim-start and trim-end points.
line_trim_offset	The line part between c(trim_start, trim_end) will be painted using line_trim_color.
line_width	Stroke thickness.
line_z_offset	Vertical offset from ground, in meters.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.
popup	Popup content shown on click: a column name, a {brace} template (e.g. "{name}:{value}"), or a concat()/number_format() expression. Columns containing HTML are parsed.
tooltip	Tooltip content shown on hover; same forms as popup.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a tooltip_style() object. When omitted, the native (unstyled) appearance is kept.

hover_options	A named list of options for highlighting features in the layer on hover.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels)
filter	An optional filter expression to subset features in the layer.

Value

The modified map object with the new line layer added.

Examples

```
## Not run:
library(mapgl)
library(tigris)

loving_roads <- roads("TX", "Loving")

maplibre(style = maptiler_style("backdrop")) |>
  fit_bounds(loving_roads) |>
  add_line_layer(
    id = "tracks",
    source = loving_roads,
    line_color = "navy",
    line_opacity = 0.7
  )

## End(Not run)
```

add_markers

Add markers to a Mapbox GL or Maplibre GL map

Description

Add markers to a Mapbox GL or Maplibre GL map

Usage

```
add_markers(
  map,
  data,
  color = "red",
  rotation = 0,
  popup = NULL,
  marker_id = NULL,
  draggable = FALSE,
  ...
)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> functions.
data	A length-2 numeric vector of coordinates, a list of length-2 numeric vectors, or an <code>sf</code> POINT object.
color	The color of the marker (default is "red").
rotation	The rotation of the marker (default is 0).
popup	A column name for popups (if data is an <code>sf</code> object) or a string for a single popup (if data is a numeric vector or list of vectors).
marker_id	A unique ID for the marker. For lists, names will be inherited from the list names. For <code>sf</code> objects, this should be a column name.
draggable	A boolean indicating if the marker should be draggable (default is FALSE).
...	Additional options passed to the marker.

Value

The modified map object with the markers added.

Examples

```
## Not run:
library(mapgl)
library(sf)

# Create a map object
map <- mapboxgl(
  style = mapbox_style("streets"),
  center = c(-74.006, 40.7128),
  zoom = 10
)

# Add a single draggable marker with an ID
map <- add_markers(
  map,
  c(-74.006, 40.7128),
  color = "blue",
  rotation = 45,
  popup = "A marker",
  draggable = TRUE,
  marker_id = "marker1"
)

# Add multiple markers from a named list of coordinates
coords_list <- list(marker2 = c(-74.006, 40.7128),
                    marker3 = c(-73.935242, 40.730610))
map <- add_markers(
  map,
  coords_list,
  color = "green",
  popup = "Multiple markers",
```

```

    draggable = TRUE
  )

  # Create an sf POINT object
  points_sf <- st_as_sf(data.frame(
    id = c("marker4", "marker5"),
    lon = c(-74.006, -73.935242),
    lat = c(40.7128, 40.730610)
  ), coords = c("lon", "lat"), crs = 4326)
  points_sf$popup <- c("Point 1", "Point 2")

  # Add multiple markers from an sf object with IDs from a column
  map <- add_markers(
    map,
    points_sf,
    color = "red",
    popup = "popup",
    draggable = TRUE,
    marker_id = "id"
  )

  ## End(Not run)

```

add_navigation_control

Add a navigation control to a map

Description

Add a navigation control to a map

Usage

```

add_navigation_control(
  map,
  show_compass = TRUE,
  show_zoom = TRUE,
  visualize_pitch = FALSE,
  position = "top-right",
  orientation = "vertical"
)

```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
show_compass	Whether to show the compass button.
show_zoom	Whether to show the zoom-in and zoom-out buttons.

visualize_pitch	Whether to visualize the pitch by rotating the X-axis of the compass.
position	The position on the map where the control will be added. Possible values are "top-left", "top-right", "bottom-left", and "bottom-right".
orientation	The orientation of the navigation control. Can be "vertical" (default) or "horizontal".

Value

The updated map object with the navigation control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl() |>
  add_navigation_control(visualize_pitch = TRUE)

## End(Not run)
```

add_pmtiles_source	<i>Add a PMTiles source to a Mapbox GL or Maplibre GL map</i>
--------------------	---

Description

Add a PMTiles source to a Mapbox GL or Maplibre GL map

Usage

```
add_pmtiles_source(
  map,
  id,
  url,
  source_type = "vector",
  maxzoom = 22,
  tileSize = 256,
  promote_id = NULL,
  ...
)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the PMTiles archive.
source_type	The source type. Either "vector" (default) or "raster".

maxzoom	Only used when source_type is "raster". The maximum zoom level for the PMTiles source. Defaults to 22.
tilesize	Only used when source_type is "raster". The size of the tiles in the PMTiles source. Defaults to 256.
promote_id	An optional property name to use as the feature ID. This is required for hover effects on vector sources.
...	Additional arguments to be passed to the JavaScript addSource method.

Value

The modified map object with the new source added.

Examples

```
## Not run:

# Visualize the Overture Maps places data as PMTiles
# Works with either `maplibre()` or `mapboxgl()`

library(mapgl)

maplibre(style = maptiler_style("basic", variant = "dark")) |>
  set_projection("globe") |>
  add_pmtiles_source(
    id = "places-source",
    url = "https://overturemaps-tiles-us-west-2-beta.s3.amazonaws.com/2025-06-25/places.pmtiles"
  ) |>
  add_circle_layer(
    id = "places-layer",
    source = "places-source",
    source_layer = "place",
    circle_color = "cyan",
    circle_opacity = 0.7,
    circle_radius = 4,
    tooltip = concat(
      "Name: ",
      get_column("@name"),
      "<br>Confidence: ",
      number_format(get_column("confidence"), maximum_fraction_digits = 2)
    )
  )

## End(Not run)
```

add_raster_dem_source *Add a raster DEM source to a Mapbox GL or Maplibre GL map*

Description

Add a raster DEM source to a Mapbox GL or Maplibre GL map

Usage

```
add_raster_dem_source(map, id, url, tileSize = 512, maxzoom = NULL, ...)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the raster DEM source.
tileSize	The size of the raster tiles.
maxzoom	The maximum zoom level for the raster tiles.
...	Additional arguments to be passed to the JavaScript addSource method.

Value

The modified map object with the new source added.

add_raster_layer	<i>Add a raster layer to a Mapbox GL map</i>
------------------	--

Description

Add a raster layer to a Mapbox GL map

Usage

```
add_raster_layer(
  map,
  id,
  source,
  source_layer = NULL,
  raster_brightness_max = NULL,
  raster_brightness_min = NULL,
  raster_color = NULL,
  raster_color_mix = NULL,
  raster_color_range = NULL,
  raster_contrast = NULL,
  raster_emissive_strength = NULL,
  raster_fade_duration = NULL,
  raster_hue_rotate = NULL,
  raster_opacity = NULL,
  raster_resampling = NULL,
  raster_saturation = NULL,
  visibility = "visible",
  slot = NULL,
  min_zoom = NULL,
```

```

    max_zoom = NULL,
    before_id = NULL
)

```

Arguments

<code>map</code>	A map object created by the <code>mapboxgl</code> function.
<code>id</code>	A unique ID for the layer.
<code>source</code>	The ID of the source.
<code>source_layer</code>	The source layer (for vector sources).
<code>raster_brightness_max</code>	The maximum brightness of the image.
<code>raster_brightness_min</code>	The minimum brightness of the image.
<code>raster_color</code>	Defines a color map for colorizing single-band raster data, parameterized by <code>["raster-value"]</code> . Use an interpolate expression over <code>raster-value</code> to define the color ramp. Requires <code>raster_color_range</code> to be set.
<code>raster_color_mix</code>	Specifies the combination of source RGB channels used to compute the raster value when <code>raster_color</code> is active. A numeric vector of length 4: <code>c(r, g, b, offset)</code> . Defaults to <code>c(0.2126, 0.7152, 0.0722, 0)</code> (RGB luminosity).
<code>raster_color_range</code>	Specifies the value range over which <code>raster_color</code> is tabulated. A numeric vector of length 2: <code>c(min, max)</code> .
<code>raster_contrast</code>	Increase or reduce the brightness of the image.
<code>raster_emissive_strength</code>	Controls the intensity of light emitted on the source features. Requires 3D lights.
<code>raster_fade_duration</code>	The duration of the fade-in/fade-out effect.
<code>raster_hue_rotate</code>	Rotates hues around the color wheel.
<code>raster_opacity</code>	The opacity at which the raster will be drawn.
<code>raster_resampling</code>	The resampling/interpolation method to use for overscaling. Options are <code>"linear"</code> (bilinear, the MapLibre/Mapbox default) and <code>"nearest"</code> (nearest-neighbor). Use <code>"nearest"</code> for categorical or classified rasters (e.g. land cover) to preserve crisp category boundaries when zooming.
<code>raster_saturation</code>	Increase or reduce the saturation of the image.
<code>visibility</code>	Whether this layer is displayed.
<code>slot</code>	An optional slot for layer order.
<code>min_zoom</code>	The minimum zoom level for the layer.
<code>max_zoom</code>	The maximum zoom level for the layer.
<code>before_id</code>	The name of the layer that this layer appears <code>"before"</code> , allowing you to insert layers below other layers in your basemap (e.g. labels).

Value

The modified map object with the new raster layer added.

Examples

```
## Not run:
mapboxgl(
  style = mapbox_style("dark"),
  zoom = 5,
  center = c(-75.789, 41.874)
) |>
  add_image_source(
    id = "radar",
    url = "https://docs.mapbox.com/mapbox-gl-js/assets/radar.gif",
    coordinates = list(
      c(-80.425, 46.437),
      c(-71.516, 46.437),
      c(-71.516, 37.936),
      c(-80.425, 37.936)
    )
  ) |>
  add_raster_layer(
    id = "radar-layer",
    source = "radar",
    raster_fade_duration = 0
  )

## End(Not run)
```

add_raster_source	<i>Add a raster tile source to a Mapbox GL or Maplibre GL map</i>
-------------------	---

Description

Add a raster tile source to a Mapbox GL or Maplibre GL map

Usage

```
add_raster_source(
  map,
  id,
  url = NULL,
  tiles = NULL,
  tileSize = 256,
  maxzoom = 22,
  ...
)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the raster tile source. (optional)
tiles	A vector of tile URLs for the raster source. (optional)
tileSize	The size of the raster tiles.
maxzoom	The maximum zoom level for the raster tiles.
...	Additional arguments to be passed to the JavaScript addSource method.

Value

The modified map object with the new source added.

add_reset_control	<i>Add a reset control to a map</i>
-------------------	-------------------------------------

Description

This function adds a reset control to a Mapbox GL or MapLibre GL map. The reset control allows users to return to the original zoom level and center.

Usage

```
add_reset_control(map, position = "top-right", animate = TRUE, duration = NULL)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".
animate	Whether or not to animate the transition to the original map view; defaults to TRUE. If FALSE, the view will "jump" to the original view with no transition.
duration	The length of the transition from the current view to the original view, specified in milliseconds. This argument only works with animate is TRUE.

Value

The modified map object with the reset control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl() |>
  add_reset_control(position = "top-left")

## End(Not run)
```

add_scale_control	<i>Add a scale control to a map</i>
-------------------	-------------------------------------

Description

This function adds a scale control to a Mapbox GL or Maplibre GL map.

Usage

```
add_scale_control(
  map,
  position = "bottom-left",
  unit = "metric",
  max_width = 100
)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "bottom-left".
unit	The unit of the scale. Can be either "imperial", "metric", or "nautical". Default is "metric".
max_width	The maximum length of the scale control in pixels. Default is 100.

Value

The modified map object with the scale control added.

Examples

```
## Not run:
library(mapgl)

mapboxgl() |>
  add_scale_control(position = "bottom-right", unit = "imperial")

## End(Not run)
```

add_screenshot_control

Add a screenshot control to a map

Description

This function adds a screenshot control to a Mapbox GL or MapLibre GL map. The screenshot control allows users to capture the map along with legends and attribution as a PNG image download.

Usage

```
add_screenshot_control(
  map,
  position = "top-right",
  filename = "map-screenshot",
  include_legend = TRUE,
  hide_controls = TRUE,
  include_scale_bar = TRUE,
  basemap_color = NULL,
  image_scale = 1,
  button_title = "Capture screenshot"
)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
position	The position of the control. Can be one of "top-left", "top-right", "bottom-left", or "bottom-right". Default is "top-right".
filename	The base filename for the downloaded image (without extension). Default is "map-screenshot".
include_legend	Logical, whether to include legends in the screenshot. Default is TRUE.
hide_controls	Logical, whether to hide interactive controls (navigation, fullscreen, etc.) during screenshot capture. Default is TRUE.
include_scale_bar	Logical, whether to keep the scale bar visible in the screenshot when hide_controls = TRUE. Default is TRUE. The scale bar is the only interactive control that renders correctly and provides useful context in static images.
basemap_color	Character string or NULL. If specified, basemap tiles are removed from the screenshot and replaced with this background color (e.g., "white", "lightgrey", "#f0f0f0"). Use "transparent" for no background. Default NULL (keep basemap).
image_scale	Numeric, the scale factor for the output image resolution. Default is 1. Higher values (2 or 3) produce sharper text and legend elements but increase file size. Scale 2 produces 4x larger files, scale 3 produces 9x larger files.
button_title	The tooltip title for the button. Default is "Capture screenshot".

Details

The screenshot is captured using `html2canvas`, which renders the map container including legends and attribution. Attribution is always included in screenshots to comply with map provider terms of service.

Most interactive controls (navigation, fullscreen, etc.) do not render correctly in screenshots due to SVG rendering limitations and will appear as blank boxes. The scale bar is an exception and renders correctly, which is why it is preserved by default via `include_scale_bar = TRUE`.

Value

The modified map object with the screenshot control added.

Examples

```
## Not run:
library(mapgl)

# Basic usage
maplibre(style = carto_style("positron")) |>
  add_screenshot_control()

# With scale control (recommended for screenshots)
maplibre() |>
  add_scale_control(position = "bottom-left") |>
  add_screenshot_control()

# With custom filename
maplibre() |>
  add_fill_layer(
    id = "counties",
    source = list(type = "geojson", data = counties_sf)
  ) |>
  add_legend("Median Income", values = c("Low", "High")) |>
  add_screenshot_control(
    filename = "county-map",
    position = "top-left"
  )

# Exclude legend from screenshot
maplibre() |>
  add_screenshot_control(include_legend = FALSE)

## End(Not run)
```

Description

Adds an interactive slider to a Mapbox GL or MapLibre GL map that steps through a sequence of values and filters one or more layers by a numeric feature property. Works source-agnostically: GeoJSON, vector tile, and PMTiles layers all compose through the same filter pipeline. Optional play button animates through the steps; "cumulative" mode accumulates features through the range rather than swapping them.

Usage

```
add_slider_control(  
  map,  
  layers,  
  property = NULL,  
  time_unit = NULL,  
  values = NULL,  
  labels = NULL,  
  min = NULL,  
  max = NULL,  
  step = 1,  
  mode = c("sequential", "cumulative", "window"),  
  window = NULL,  
  presentation = c("compact", "timeline"),  
  window_behavior = c("auto", "resizable", "fixed"),  
  histogram = FALSE,  
  histogram_data = NULL,  
  counts = NULL,  
  initial_value = NULL,  
  paint_property = NULL,  
  paint_expressions = NULL,  
  paint_properties = NULL,  
  play_button = FALSE,  
  animation_duration = 1000,  
  loop = TRUE,  
  title = NULL,  
  show_value = TRUE,  
  slider_style = NULL,  
  width = 280,  
  position = "top-left",  
  draggable = FALSE,  
  background_color = "#ffffffcc",  
  text_color = "#404040",  
  accent_color = "#4a90e2"  
)
```

Arguments

map	A map object created by <code>mapboxgl()</code> or <code>maplibre()</code> , or a proxy object from <code>mapboxgl_proxy()</code> / <code>maplibre_proxy()</code> .
-----	---

layers	Character vector of layer IDs the slider should affect. Layers need not exist yet — if a matching layer is added later via proxy, the slider's current filter and/or paint expression is applied on its first paint.
property	Name of the numeric feature property to filter on. Optional: if NULL, no filter behavior. At least one of property or paint_property must be supplied.
time_unit	Optional. Only consequential when a target layer is a flowmap (<code>add_flowmap()</code>): ordinary layers filter on the numeric property directly, but flowmap layers need an absolute timestamp range, so in "window" mode the numeric value is converted to a flowmap time range using this unit. One of "seconds", "date", or "year" (the absolute units produced by <code>as_time_property()</code>). "month"/"day" are calendar indices, not instants, and cannot drive a flowmap window.
values	Numeric vector of steps. Required unless min and max are supplied.
labels	Optional character vector of display labels (one per value). When omitted, labels default to <code>as.character(values)</code> , except when time_unit is "seconds" or "date": then the numeric values are formatted back into readable timestamps ("%Y-%m-%d %H:%M" in UTC for seconds, "%Y-%m-%d" for dates) so a time-driven slider does not display raw epoch numbers. Pass an explicit vector to localize or change the format.
min, max, step	Numeric range specification used when values is not supplied. values is generated via <code>seq(min, max, by = step)</code> .
mode	One of "sequential" (default) — each step shows only features matching that exact value; "cumulative" — shows everything up through the current value; or "window" — shows a moving [start, end] range (the end is the current value). Window mode emits a range filter for ordinary layers and a <code>selectedTimeRange</code> for flowmap layers, making it the mode used to drive temporal flowmaps. Only applies when property is set.
window	Only used when mode = "window". NULL (default) makes a cumulative range [<code>min(values)</code> , T]; a positive number makes a sliding window [T - window, T], where window is expressed in the same units as values (e.g. for <code>as_time_property()</code> unit = "date", window = 7 is a 7-day window).
presentation	Visual style of the control. "compact" (default) is a small slider (optionally with a density strip behind it). "timeline" renders a prominent, brushable histogram as the control itself — drag the selected window across the bars, or drag its edges to resize it (as in FlowMapBlue's time control). "timeline" requires histogram data (counts or histogram_data) and implies histogram = TRUE.
window_behavior	Only used when mode = "window". "auto" (default) uses a fixed-duration window when window is supplied and a resizable range otherwise. "fixed" pins the window width to window and moves the whole [end - window, end] band. "resizable" lets the user drag either edge to resize the selected range.
histogram	Logical; if TRUE, draw a density strip behind the slider showing how many features fall at each value (a data-density backdrop, most useful with mode = "window" where the selected band highlights the covered bars). Loads d3 on demand. Supply the bar heights via counts or histogram_data. Default FALSE.
histogram_data	Numeric vector of the raw property values across your features; binned in R to the nearest values step to produce the histogram bar heights. Use this when you have the underlying data to hand. Ignored unless histogram = TRUE.

counts	Numeric vector of pre-computed bar heights, one per values. An alternative to histogram_data when you already have per-step counts. Ignored unless histogram = TRUE.
initial_value	Value to start on. In "window" mode this is the window end value and defaults to the last value; other modes default to the first value.
paint_property	Optional Mapbox paint property the slider should animate, supplied in snake_case to match the rest of the package (e.g. "fill_color", "fill_extrusion_height", "circle_radius"). Kebab-case is also accepted for convenience when copying from Mapbox docs. Shortcut for the single-property case; when set, paint_expressions must also be supplied. For animating more than one paint property together (e.g. height and color), use paint_properties instead.
paint_expressions	List of Mapbox expressions, one per value in values. On each step, the expression at the matching index is applied to each target layer via map.setPaintProperty(). Build expressions with interpolate() , match_expr() , or raw lists.
paint_properties	Optional named list for animating multiple paint properties simultaneously. Shape: list(fill_color = list(<expr_0>, <expr_1>, ...), fill_extrusion_height = list(...)) — names are snake_case (kebab-case also accepted), each value is a list of expressions the same length as values. Cannot be combined with paint_property/paint_expressions. Each property's pre-slider value is captured on mount and restored on removal, independently.
play_button	Logical; include a play/pause button next to the slider. Default FALSE.
animation_duration	Milliseconds to wait between steps while playing. Default 1000.
loop	Logical; when playback reaches the last value, return to the first. Default TRUE.
title	Optional title shown above the slider.
show_value	Logical; show the current label beside the slider. Default TRUE.
slider_style	Optional appearance for the slider: a preset string ("light", "dark", or "auto") or a slider_style() object. When omitted, the legacy width, background_color, text_color, and accent_color arguments are used.
width	Slider container width in pixels. Default 280.
position	One of "top-left", "top-right", "bottom-left", "bottom-right". Default "top-left".
draggable	Logical, whether the slider panel can be dragged to a new position on the map by the user. Default FALSE, keeping it docked at position.
background_color, text_color, accent_color	Styling overrides. Defaults match the package's other controls. For full control of the slider appearance, use slider_style.

Details

Values commonly represent time (years, months, epoch seconds) but need not — any monotonic numeric sequence works (magnitude thresholds, percentile coverage, depth bins, etc.).

The slider composes with other filter sources rather than replacing them: a layer's initial `filter =` argument, a subsequent `set_filter()` call, and an interactive `add_legend()` can all be active at the same time as the slider, and the layer's resolved filter is the `["all", ...]` intersection.

Value

The modified map or proxy object, invisibly.

Date and time values

The slider requires numeric values. Because `sf` objects are serialized to GeoJSON via `geojsonsf::sf_geojson()`, Date and POSIXct columns become JSON strings, not numbers — so you must pre-coerce your time column to numeric before adding the layer. Use `as_time_property()` for the common cases (year, month index, epoch seconds, days-since-epoch).

Positioning and collisions

The slider is implemented as a native control and stacks beside other native controls (navigation, scale, geolocate, fullscreen, `add_layers_control()`) via Mapbox/MapLibre's built-in positioning. Overlays such as `add_legend()` are absolutely-positioned and do not participate in that flow — placing a slider in the same corner as a legend will overlap it. Choose a different corner or adjust the overlay's margins.

See Also

`update_slider_control()`, `as_time_property()`

Examples

```
## Not run:
library(mapgl)
library(sf)

quakes <- sf::read_sf(
  "https://docs.mapbox.com/mapbox-gl-js/assets/significant-earthquakes-2015.geojson"
)
quakes$month <- as.POSIXlt(quakes$time / 1000, origin = "1970-01-01")$mon

mapboxgl(center = c(-45, 0), zoom = 0.25) |>
  add_circle_layer(
    id = "quakes",
    source = quakes,
    circle_color = "#FCA107",
    circle_radius = 20,
    circle_opacity = 0.75
  ) |>
  add_slider_control(
    layers = "quakes",
    property = "month",
    values = 0:11,
    labels = month.name,
```

```

    title = "Significant earthquakes in 2015",
    play_button = TRUE
  )

# --- Paint animation: step through a choropleth's color over years
# Given an sf object with columns pop2020, pop2021, ..., pop2024:
# mapboxgl() |>
#   add_fill_layer(id = "tracts", source = tracts, fill_opacity = 0.85) |>
#   add_slider_control(
#     layers = "tracts",
#     values = 2020:2024,
#     paint_property = "fill_color",
#     paint_expressions = lapply(2020:2024, function(y) {
#       interpolate(
#         column = paste0("pop", y),
#         values = c(0, 5e5, 2e6),
#         stops = c("#f7f4f9", "#67a9cf", "#02818a")
#       )
#     }),
#     play_button = TRUE
#   )

## End(Not run)

```

add_source

Add a GeoJSON or sf source to a Mapbox GL or Maplibre GL map

Description

Add a GeoJSON or sf source to a Mapbox GL or Maplibre GL map

Usage

```
add_source(map, id, data, ...)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
data	An sf object or a URL pointing to a remote GeoJSON file.
...	Additional arguments to be passed to the JavaScript addSource method.

Value

The modified map object with the new source added.

add_symbol_layer	<i>Add a symbol layer to a map</i>
------------------	------------------------------------

Description

Add a symbol layer to a map

Usage

```
add_symbol_layer(  
    map,  
    id,  
    source,  
    source_layer = NULL,  
    icon_allow_overlap = NULL,  
    icon_anchor = NULL,  
    icon_color = NULL,  
    icon_color_brightness_max = NULL,  
    icon_color_brightness_min = NULL,  
    icon_color_contrast = NULL,  
    icon_color_saturation = NULL,  
    icon_emissive_strength = NULL,  
    icon_halo_blur = NULL,  
    icon_halo_color = NULL,  
    icon_halo_width = NULL,  
    icon_ignore_placement = NULL,  
    icon_image = NULL,  
    icon_image_cross_fade = NULL,  
    icon_keep_upright = NULL,  
    icon_occlusion_opacity = NULL,  
    icon_offset = NULL,  
    icon_opacity = NULL,  
    icon_optional = NULL,  
    icon_padding = NULL,  
    icon_pitch_alignment = NULL,  
    icon_rotate = NULL,  
    icon_rotation_alignment = NULL,  
    icon_size = NULL,  
    icon_text_fit = NULL,  
    icon_text_fit_padding = NULL,  
    icon_translate = NULL,  
    icon_translate_anchor = NULL,  
    symbol_avoid_edges = NULL,  
    symbol_placement = NULL,  
    symbol_sort_key = NULL,  
    symbol_spacing = NULL,  
    symbol_z_elevate = NULL,
```

```
symbol_z_offset = NULL,  
symbol_z_order = NULL,  
text_allow_overlap = NULL,  
text_anchor = NULL,  
text_color = "black",  
text_emissive_strength = NULL,  
text_field = NULL,  
text_font = NULL,  
text_halo_blur = NULL,  
text_halo_color = NULL,  
text_halo_width = NULL,  
text_ignore_placement = NULL,  
text_justify = NULL,  
text_keep_upright = NULL,  
text_letter_spacing = NULL,  
text_line_height = NULL,  
text_max_angle = NULL,  
text_max_width = NULL,  
text_occlusion_opacity = NULL,  
text_offset = NULL,  
text_opacity = NULL,  
text_optional = NULL,  
text_padding = NULL,  
text_pitch_alignment = NULL,  
text_radial_offset = NULL,  
text_rotate = NULL,  
text_rotation_alignment = NULL,  
text_size = NULL,  
text_transform = NULL,  
text_translate = NULL,  
text_translate_anchor = NULL,  
text_variable_anchor = NULL,  
text_writing_mode = NULL,  
visibility = "visible",  
slot = NULL,  
min_zoom = NULL,  
max_zoom = NULL,  
popup = NULL,  
tooltip = NULL,  
tooltip_style = NULL,  
popup_style = NULL,  
hover_options = NULL,  
before_id = NULL,  
filter = NULL,  
cluster_options = NULL  
)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
id	A unique ID for the layer.
source	The ID of the source, alternatively an sf object (which will be converted to a GeoJSON source) or a named list that specifies type and url for a remote source.
source_layer	The source layer (for vector sources).
icon_allow_overlap	If TRUE, the icon will be visible even if it collides with other previously drawn symbols.
icon_anchor	Part of the icon placed closest to the anchor.
icon_color	The color of the icon. This is not supported for many Mapbox icons; read more at https://docs.mapbox.com/help/troubleshooting/using-recolorable-images-in-mapbox-m
icon_color_brightness_max	The maximum brightness of the icon color.
icon_color_brightness_min	The minimum brightness of the icon color.
icon_color_contrast	The contrast of the icon color.
icon_color_saturation	The saturation of the icon color.
icon_emissive_strength	The strength of the icon's emissive color.
icon_halo_blur	The blur applied to the icon's halo.
icon_halo_color	The color of the icon's halo.
icon_halo_width	The width of the icon's halo.
icon_ignore_placement	If TRUE, the icon will be visible even if it collides with other symbols.
icon_image	Name of image in sprite to use for drawing an image background. To use values in a column of your input dataset, use get_column('YOUR_ICON_COLUMN_NAME'). Images can also be loaded with the add_image() function which should precede the add_symbol_layer() function.
icon_image_cross_fade	The cross-fade parameter for the icon image.
icon_keep_upright	If TRUE, the icon will be kept upright.
icon_occlusion_opacity	The opacity at which the icon will be drawn when occluded by 3D objects. Value between 0 and 1; 0 hides occluded icons.
icon_offset	Offset distance of icon.
icon_opacity	The opacity at which the icon will be drawn.

icon_optional	If TRUE, the icon will be optional.
icon_padding	Padding around the icon.
icon_pitch_alignment	Alignment of the icon with respect to the pitch of the map.
icon_rotate	Rotates the icon clockwise.
icon_rotation_alignment	Alignment of the icon with respect to the map.
icon_size	The size of the icon, specified relative to the original size of the image. For example, a value of 5 would make the icon 5 times larger than the original size, whereas a value of 0.5 would make the icon half the size of the original.
icon_text_fit	Scales the text to fit the icon.
icon_text_fit_padding	Padding for text fitting the icon.
icon_translate	The offset distance of the icon.
icon_translate_anchor	Controls the frame of reference for icon-translate.
symbol_avoid_edges	If TRUE, the symbol will be avoided when near the edges.
symbol_placement	Placement of the symbol on the map.
symbol_sort_key	Sorts features in ascending order based on this value.
symbol_spacing	Spacing between symbols.
symbol_z_elevate	If TRUE, positions the symbol on top of a fill-extrusion layer. Requires symbol_placement to be set to "point" and symbol-z-order to be set to "auto".
symbol_z_offset	The elevation of the symbol, in meters. Use get_column() to get elevations from a column in the dataset.
symbol_z_order	Orders the symbol z-axis.
text_allow_overlap	If TRUE, the text will be visible even if it collides with other previously drawn symbols.
text_anchor	Part of the text placed closest to the anchor.
text_color	The color of the text.
text_emissive_strength	The strength of the text's emissive color.
text_field	Value to use for a text label.
text_font	Font stack to use for displaying text.
text_halo_blur	The blur applied to the text's halo.
text_halo_color	The color of the text's halo.

text_halo_width	The width of the text's halo.
text_ignore_placement	If TRUE, the text will be visible even if it collides with other symbols.
text_justify	The justification of the text.
text_keep_upright	If TRUE, the text will be kept upright.
text_letter_spacing	Spacing between text letters.
text_line_height	Height of the text lines.
text_max_angle	Maximum angle of the text.
text_max_width	Maximum width of the text.
text_occlusion_opacity	The opacity at which the text will be drawn when occluded by 3D objects. Value between 0 and 1; 0 hides occluded text.
text_offset	Offset distance of text.
text_opacity	The opacity at which the text will be drawn.
text_optional	If TRUE, the text will be optional.
text_padding	Padding around the text.
text_pitch_alignment	Alignment of the text with respect to the pitch of the map.
text_radial_offset	Radial offset of the text.
text_rotate	Rotates the text clockwise.
text_rotation_alignment	Alignment of the text with respect to the map.
text_size	The size of the text.
text_transform	Transform applied to the text.
text_translate	The offset distance of the text.
text_translate_anchor	Controls the frame of reference for text-translate.
text_variable_anchor	Variable anchor for the text.
text_writing_mode	Writing mode for the text.
visibility	Whether this layer is displayed.
slot	An optional slot for layer order.
min_zoom	The minimum zoom level for the layer.
max_zoom	The maximum zoom level for the layer.
popup	A column name containing information to display in a popup on click. Columns containing HTML will be parsed.

tooltip	A column name containing information to display in a tooltip on hover. Columns containing HTML will be parsed.
tooltip_style, popup_style	Optional appearance for the tooltip/popup: a preset string ("light" or "dark") or a <code>tooltip_style()</code> object. When omitted, the native (unstyled) appearance is kept.
hover_options	A named list of options for highlighting features in the layer on hover. Not all elements of SVG icons can be styled.
before_id	The name of the layer that this layer appears "before", allowing you to insert layers below other layers in your basemap (e.g. labels).
filter	An optional filter expression to subset features in the layer.
cluster_options	A list of options for clustering symbols, created by the <code>cluster_options()</code> function. Two input shapes are supported: pass an <code>sf/sfc</code> object as source for native live clustering (a GeoJSON source is injected automatically), or pass the id of an already-registered vector source (e.g. from <code>add_pmtiles_source()</code>) along with <code>source_layer</code> to use pre-clustered vector tiles such as those produced by the <code>freestiler</code> package. In the latter case the cluster-count label is abbreviated client-side from <code>point_count</code>. Set <code>donut_column</code> in <code>cluster_options()</code> to draw clusters as donut charts of category shares; see <code>cluster_options()</code> for the properties pre-clustered tiles need. Updating a clustered layer in Shiny: the shortcut creates three layers ("id", "id-clusters", "id-cluster-count") on top of one source. For reactive data updates the recommended pattern is <code>set_source()</code>, which replaces the source's data and lets Mapbox/MapLibre re-cluster automatically without tearing down the layers: <code>mapboxgl_proxy("map") > set_source(layer_id = "pts", source = filtered())</code>. If you need to remove a clustered layer entirely (e.g. before switching backends), pass the full trio to <code>clear_layer()</code>: <code>clear_layer(proxy, c("pts", "pts-clusters", "pts-cluster-count"))</code>.

Value

The modified map object with the new symbol layer added.

Examples

```
## Not run:
library(mapgl)
library(sf)
library(dplyr)

# Set seed for reproducibility
set.seed(1234)

# Define the bounding box for Washington DC (approximately)
bbox <- st_bbox(
  c(
    xmin = -77.119759,
    ymin = 38.791645,
```

```

      xmax = -76.909393,
      ymax = 38.995548
    ),
    crs = st_crs(4326)
  )

  # Generate 30 random points within the bounding box
  random_points <- st_as_sf(
    data.frame(
      id = 1:30,
      lon = runif(30, bbox["xmin"], bbox["xmax"]),
      lat = runif(30, bbox["ymin"], bbox["ymax"])
    ),
    coords = c("lon", "lat"),
    crs = 4326
  )

  # Assign random icons
  icons <- c("music", "bar", "theatre", "bicycle")
  random_points <- random_points |>
    mutate(icon = sample(icons, n(), replace = TRUE))

  # Map with icons
  mapboxgl(style = mapbox_style("light")) |>
    fit_bounds(random_points, animate = FALSE) |>
    add_symbol_layer(
      id = "points-of-interest",
      source = random_points,
      icon_image = c("get", "icon"),
      icon_allow_overlap = TRUE,
      tooltip = "icon"
    )

  ## End(Not run)

```

add_terradraw_control *Add a Terra Draw control to a map*

Description

A convenience wrapper around [add_draw_control\(\)](#) with provider = "terra-draw", exposing only the arguments that apply to the Terra Draw engine. The two forms are equivalent; see [add_draw_control\(\)](#) for full details of the terra-draw provider's behavior, and [terrardraw_options\(\)](#) for advanced configuration.

Usage

```

add_terradraw_control(
  map,
  position = "top-left",

```

```

    modes = NULL,
    options = NULL,
    source = NULL,
    attributes = NULL,
    point_color = "#3bb2d0",
    line_color = "#3bb2d0",
    fill_color = "#3bb2d0",
    fill_opacity = 0.1,
    active_color = "#fbb03b",
    vertex_radius = 5,
    line_width = 2,
    download_button = FALSE,
    download_filename = "drawn-features",
    show_measurements = FALSE,
    measurement_units = "both",
    orientation = "vertical"
)

```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> functions.
position	A string specifying the position of the draw control. One of "top-right", "top-left", "bottom-right", or "bottom-left".
modes	A character vector of drawing modes to expose as toolbar buttons: any of "point", "linestring", "polygon", "rectangle", "circle", "freehand", "freehand-linestring", "angled-rectangle", "sector", "sensor", "curve", "curve-linestring", and "select". When NULL (the default), the control shows "point", "linestring", "polygon", and "select".
options	An object created by <code>terradraw_options()</code> with advanced Terra Draw settings (snapping, select-mode editing flags, per-mode overrides).
source	A character string specifying a source ID to add to the draw control. Default is NULL.
attributes	Optional named list defining editable feature attributes. Use <code>draw_attribute()</code> to define fields.
point_color	Color for point features. Default is "#3bb2d0" (light blue).
line_color	Color for line features. Default is "#3bb2d0" (light blue).
fill_color	Fill color for polygon features. Default is "#3bb2d0" (light blue).
fill_opacity	Fill opacity for polygon features. Default is 0.1.
active_color	Color for active (selected) features. Default is "#fbb03b" (orange).
vertex_radius	Radius of vertex points in pixels. Default is 5.
line_width	Width of lines in pixels. Default is 2.
download_button	Logical, whether to add a download button to export drawn features as GeoJSON. Default is FALSE.

download_filename	Base filename for downloaded GeoJSON (without extension). Default is "drawn-features".
show_measurements	Logical, whether to show live measurements while drawing. Default is FALSE.
measurement_units	Units for measurements. Either "metric", "imperial", or "both". Default is "both".
orientation	A string specifying the orientation of the draw control. Either "vertical" (default) or "horizontal".

Value

The modified map object with the draw control added.

Examples

```
## Not run:
library(mapgl)

maplibre() |>
  add_terrdraw_control()

maplibre() |>
  add_terrdraw_control(
    modes = c("polygon", "rectangle", "circle", "freehand", "select"),
    options = terrdraw_options(snap_to_coordinates = TRUE),
    download_button = TRUE
  )

## End(Not run)
```

add_vector_source	<i>Add a vector tile source to a Mapbox GL or Maplibre GL map</i>
-------------------	---

Description

Add a vector tile source to a Mapbox GL or Maplibre GL map

Usage

```
add_vector_source(map, id, url = NULL, tiles = NULL, promote_id = NULL, ...)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
url	A URL pointing to the vector tile source.
tiles	A vector of tile URLs, typically in the format "https://example.com/{z}/{x}/{y}.mvt" or similar.
promote_id	An optional property name to use as the feature ID. This is required for hover effects on vector tiles.
...	Additional arguments to be passed to the JavaScript addSource method.

Value

The modified map object with the new source added.

add_video_source	<i>Add a video source to a Mapbox GL or Maplibre GL map</i>
------------------	---

Description

Add a video source to a Mapbox GL or Maplibre GL map

Usage

```
add_video_source(map, id, urls, coordinates)
```

Arguments

map	A map object created by the mapboxgl or maplibre function.
id	A unique ID for the source.
urls	A vector of URLs pointing to the video sources.
coordinates	A list of coordinates specifying the video corners in clockwise order: top left, top right, bottom right, bottom left.

Value

The modified map object with the new source added.

add_view	<i>Add a visualization layer to an existing map</i>
----------	---

Description

This function allows you to add additional data layers to existing maps created with `mapboxgl_view()` or `maplibre_view()`, enabling composition of multiple datasets on a single map.

Usage

```
add_view(
  map,
  data,
  color = "gold",
  column = NULL,
  n = NULL,
  palette = viridisLite::viridis,
  layer_id = NULL,
  legend = FALSE,
  legend_position = "bottom-left"
)
```

Arguments

map	A map object created by <code>mapboxgl_view()</code> , <code>maplibre_view()</code> , <code>mapboxgl()</code> , or <code>maplibre()</code>
data	An sf object, SpatRaster, or RasterLayer to visualize
color	The color used to visualize points, lines, or polygons if column is NULL. Defaults to "navy".
column	The name of the column to visualize. If NULL (default), geometries are shown with default styling.
n	Number of quantile breaks for numeric columns. If specified, uses <code>step_expr()</code> instead of <code>interpolate()</code> .
palette	Color palette function that takes n and returns a character vector of colors. Defaults to <code>viridisLite::viridis</code> .
layer_id	The layer ID to use for the visualization. If NULL, a unique ID will be auto-generated.
legend	Logical, whether to add a legend when a column is specified. Defaults to FALSE for subsequent layers to avoid overwriting existing legends.
legend_position	The position of the legend on the map. Defaults to "bottom-left".

Value

The map object with the new layer added

Examples

```
## Not run:
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))

# Basic layering
mapboxgl_view(nc) |>
  add_view(nc[1:10, ], color = "red", layer_id = "subset")

# Layer different geometries
mapboxgl_view(polygons) |>
  add_view(points, color = "blue") |>
  add_view(lines, color = "green")

# Add raster data
mapboxgl_view(boundaries) |>
  add_view(elevation_raster, layer_id = "elevation")

## End(Not run)
```

as_time_property

*Coerce a time column to a numeric property for filtering***Description**

Helper for preparing a Date or POSIXct column for use with [add_slider_control\(\)](#). Because sf objects are serialized to GeoJSON as JSON strings for date/time columns, the slider needs a numeric counterpart to filter on.

Usage

```
as_time_property(x, unit = c("year", "month", "day", "date", "seconds"))
```

Arguments

x	A Date, POSIXct, or numeric vector.
unit	One of: • "year" — calendar year as integer (e.g. 2026). • "month" — zero-based month index (0 = January). • "day" — day of month. • "date" — days since 1970-01-01. • "seconds" — seconds since the Unix epoch.

Value

An integer or numeric vector appropriate for attaching to an sf object as a new column before calling [add_slider_control\(\)](#).

Examples

```

dates <- as.Date(c("2015-01-15", "2015-06-30", "2015-12-31"))
as_time_property(dates, "month")
as_time_property(dates, "year")
as_time_property(dates, "date")

```

basemap_style	Create a blank basemap style
---------------	------------------------------

Description

Creates a minimal map style with only a solid background color (or pattern) and no basemap tiles. Useful when you want to display data layers without any underlying map features.

Usage

```
basemap_style(color = "white", pattern = NULL)
```

Arguments

color	Character string. The background color. Default "white". Accepts any CSS color value (e.g., "#f0f0f0", "lightgrey", "rgba(0,0,0,0)"). Also used as a fallback behind transparent areas of a pattern.
pattern	Character string or NULL. The ID of an image to use as a repeating background pattern. The image must be loaded with add_image() before it can be referenced. Default NULL (solid color only).

Value

A list representing a minimal map style, suitable for passing to the style parameter of [maplibre\(\)](#) or [mapboxgl\(\)](#).

Examples

```

## Not run:
library(mapgl)

# Solid color background
maplibre(style = basemap_style("lightgrey")) |>
  add_fill_layer(
    id = "data",
    source = my_sf_data,
    fill_color = "steelblue"
  )

# Background pattern (image must be loaded with add_image())
maplibre(style = basemap_style(pattern = "parchment")) |>

```

```

add_image("parchment", "parchment.jpg") |>
add_line_layer(
  id = "borders",
  source = my_sf_data,
  line_color = "#2c1810"
)

## End(Not run)

```

bivariate_palettes	<i>Bivariate color palettes</i>
--------------------	---------------------------------

Description

Bivariate color palettes

Usage

```
bivariate_palettes(palette = NULL)
```

Arguments

palette Optional palette name. If NULL, returns all palettes.

Value

A named list of 3 by 3 color matrices, or one matrix if palette is provided.

bivariate_scale	<i>Create a bivariate color scale</i>
-----------------	---------------------------------------

Description

Create a bivariate color scale

Usage

```

bivariate_scale(
  data = NULL,
  x,
  y,
  x_values = NULL,
  y_values = NULL,
  x_breaks = NULL,
  y_breaks = NULL,
  method = "quantile",
  colors = NULL,

```

```

    palette = "blue_pink",
    na_color = "lightgrey"
  )

```

Arguments

<code>data</code>	A data frame or sf object containing the variables.
<code>x</code>	The name of the first numeric column.
<code>y</code>	The name of the second numeric column.
<code>x_values</code>	Optional numeric vector for the first variable.
<code>y_values</code>	Optional numeric vector for the second variable.
<code>x_breaks</code>	Optional numeric vector of four increasing break values for the x variable. If NULL, breaks are computed from <code>x_values</code> .
<code>y_breaks</code>	Optional numeric vector of four increasing break values for the y variable. If NULL, breaks are computed from <code>y_values</code> .
<code>method</code>	Classification method. The MVP supports "quantile".
<code>colors</code>	A 3 by 3 matrix or 9-color vector. If NULL, a built-in palette is used.
<code>palette</code>	Built-in palette name. Defaults to "blue_pink". Use <code>bivariate_palettes()</code> to inspect available palettes.
<code>na_color</code>	Color for missing values. Defaults to "lightgrey".

Value

A `mapgl_bivariate_scale` object.

bixi_flows	<i>BIXI Montréal Hourly Bike Sharing Flows (July 1-7, 2019)</i>
------------	---

Description

A dataset containing hourly aggregated bike sharing trips between BIXI stations in Montréal during the week of July 1 to July 7, 2019.

Usage

```
bixi_flows
```

Format

A data frame with 6,092 rows and 4 variables:

time Hourly timestamp (POSIXct, UTC)
origin Origin station ID (factor, matching `bixi_locations$id`)
dest Destination station ID (factor, matching `bixi_locations$id`)
count Aggregated number of bike sharing trips in that hour (integer)

Details

To minimize the package footprint, the dataset has been truncated to a **minimum of three trips** (retaining only flows where count > 2), which reduces the rows from 213,227 to 6,092, and compresses the final size to just **~22 KB**.

Source

BIXI Montréal Open Data (<https://bixi.com/fr/donnees-ouvertes>). Prepared for <https://github.com/FlowmapBlue/FlowmapBlue> by Ilya Boyandin (<https://github.com/ilyabo>). Data © BIXI Montréal, released under the BIXI Montréal Open Data license (<https://bixi.com/fr/donnees-ouvertes>). Original interactive visualization on Flowmap.blue: <https://www.flowmap.blue/1qTV0zkPB7U1ySI4g4uPtVBzzEDCI8n1WXAmQeZL15fE>

See Also

[bixi_locations](#)

Examples

```
# Check first few records
print(head(bixi_flows))
```

bixi_locations	<i>BIXI Montréal Bike Share Stations (2019)</i>
----------------	---

Description

A dataset containing the names and coordinates of BIXI bicycle sharing stations in Montréal, Quebec, Canada.

Usage

```
bixi_locations
```

Format

A data frame with 618 rows and 4 variables:

id Unique station ID (character, e.g., "4000", "MTL-EC05.1-01")

name Station name (character, e.g., "Jeanne-d'Arc / Ontario")

lat Latitude coordinate (numeric)

lon Longitude coordinate (numeric)

Source

BIXI Montréal Open Data (<https://bixi.com/fr/donnees-ouvertes>). Prepared for <https://github.com/FlowmapBlue/FlowmapBlue> by Ilya Boyandin (<https://github.com/ilyabo>). Data © BIXI Montréal, released under the BIXI Montréal Open Data license (<https://bixi.com/fr/donnees-ouvertes>).

See Also

[bixi_flows](#)

Examples

```
# Convert to sf object to view on a map
if (requireNamespace("sf", quietly = TRUE)) {
  bixi_sf <- sf::st_as_sf(bixi_locations, coords = c("lon", "lat"), crs = 4326)
  print(head(bixi_sf))
}
```

carto_style	<i>Get CARTO Style URL</i>
-------------	----------------------------

Description

Get CARTO Style URL

Usage

```
carto_style(style_name)
```

Arguments

style_name The name of the style (e.g., "voyager", "positron", "dark-matter").

Value

The style URL corresponding to the given style name.

classification_helpers	<i>Extract information from classification and continuous scale objects</i>
------------------------	---

Description

These functions extract different components from `mapgl_classification` objects (created by `step_equal_interval()`, `step_quantile()`, `step_jenks()`) and `mapgl_continuous_scale` objects (created by `interpolate_palette()`).

Usage

```

get_legend_labels(
  scale,
  format = "none",
  currency_symbol = "$",
  digits = 2,
  big_mark = ",",
  suffix = "",
  prefix = ""
)

get_legend_colors(scale)

get_breaks(scale)

## S3 method for class 'mapgl_classification'
print(x, format = "none", ...)

## S3 method for class 'mapgl_continuous_scale'
print(x, format = "none", ...)

```

Arguments

scale	A mapgl_classification or mapgl_continuous_scale object.
format	A character string specifying the format type for labels. Options include: • "none" (default): No special formatting • "currency": Format as currency (e.g., "\$1,234") • "percent": Format as percentage (e.g., "12.3%") • "scientific": Format in scientific notation (e.g., "1.2e+03") • "compact": Format with abbreviated units (e.g., "1.2K", "3.4M")
currency_symbol	The currency symbol to use when format = "currency". Defaults to "\$".
digits	The number of decimal places to display. Defaults to 2.
big_mark	The character to use as thousands separator. Defaults to ",".
suffix	An optional suffix to add to all values (e.g., "km", "mph").
prefix	An optional prefix to add to all values (useful for compact currency like "\$1.2K").
x	A mapgl_classification or mapgl_continuous_scale object to print.
...	Additional arguments passed to formatting functions.

Value

get_legend_labels() A character vector of formatted legend labels

get_legend_colors() A character vector of colors

get_breaks() A numeric vector of break values

Examples

```
## Not run:
# Texas county income data
library(tidycensus)
tx <- get_acs(geography = "county", variables = "B19013_001",
             state = "TX", geometry = TRUE)

# Classification examples
eq_class <- step_equal_interval("estimate", tx$estimate, n = 4)
labels <- get_legend_labels(eq_class, format = "currency")
colors <- get_legend_colors(eq_class)
breaks <- get_breaks(eq_class)

# Continuous scale examples
scale <- interpolate_palette("estimate", tx$estimate, method = "quantile", n = 5)
labels <- get_legend_labels(scale, format = "compact", prefix = "$")
colors <- get_legend_colors(scale)

## End(Not run)
```

clear_controls

Clear controls from a Mapbox GL or Maplibre GL map in a Shiny app

Description

This function allows you to remove specific controls or all controls from a map. You can target controls by their type names, which correspond to the function names used to add them (e.g., "navigation" for controls added with `add_navigation_control`).

Usage

```
clear_controls(map, controls = NULL)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function.
controls	A character vector of control types to remove, or <code>NULL</code> to remove all controls. Control types include: "navigation", "draw", "fullscreen", "scale", "geolocate", "geocoder", "layers", "reset", "globe_minimap", or custom control IDs. If <code>NULL</code> (default), all controls will be removed.

Value

The modified map object with specified controls removed.

Examples

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

# Clear all controls
maplibre_proxy("map") |>
  clear_controls()

# Clear specific controls
maplibre_proxy("map") |>
  clear_controls("navigation")

# Clear multiple controls
maplibre_proxy("map") |>
  clear_controls(c("draw", "navigation"))

# Clear a custom control by ID
maplibre_proxy("map") |>
  clear_controls("my_custom_control")

## End(Not run)
```

clear_drawn_features	<i>Clear all drawn features from a map</i>
----------------------	--

Description

This function removes all features that have been drawn using the draw control on a Mapbox GL or MapLibre GL map in a Shiny application.

Usage

```
clear_drawn_features(map)
```

Arguments

map	A proxy object created by the <code>mapboxgl_proxy</code> or <code>maplibre_proxy</code> functions.
-----	---

Value

The modified map object with all drawn features cleared.

Examples

```
## Not run:
# In a Shiny application
library(shiny)
library(mapgl)
```

```

ui <- fluidPage(
  mapboxglOutput("map"),
  actionButton("clear_btn", "Clear Drawn Features")
)

server <- function(input, output, session) {
  output$map <- renderMapboxgl({
    mapboxgl(
      style = mapbox_style("streets"),
      center = c(-74.50, 40),
      zoom = 9
    ) |>
    add_draw_control()
  })

  observeEvent(input$clear_btn, {
    mapboxgl_proxy("map") |>
    clear_drawn_features()
  })
}

shinyApp(ui, server)

## End(Not run)

```

clear_layer

Clear layers from a map using a proxy

Description

This function allows one or more layers to be removed from an existing Mapbox GL map using a proxy object.

Usage

```
clear_layer(proxy, layer_id)
```

Arguments

proxy	A proxy object created by <code>mapboxgl_proxy</code> or <code>maplibre_proxy</code> .
layer_id	A character vector of layer IDs to be removed. Can be a single layer ID or multiple layer IDs.

Value

The updated proxy object.

clear_legend	<i>Clear legends from a map</i>
--------------	---------------------------------

Description

Remove one or more legends from a Mapbox GL or MapLibre GL map in a Shiny application.

Usage

```
clear_legend(map, legend_ids = NULL)
```

Arguments

map	A map proxy object created by <code>mapboxgl_proxy()</code> or <code>maplibre_proxy()</code> .
legend_ids	Optional. A character vector of legend IDs to clear. If not provided, all legends will be cleared.

Value

The updated map proxy object with the specified legend(s) cleared.

Note

This function can only be used with map proxy objects in Shiny applications. It cannot be used with static map objects.

Examples

```
## Not run:
# In a Shiny server function:

# Clear all legends
observeEvent(input$clear_all, {
  mapboxgl_proxy("map") %>%
    clear_legend()
})

# Clear specific legends by ID
observeEvent(input$clear_specific, {
  mapboxgl_proxy("map") %>%
    clear_legend(legend_ids = c("legend-1", "legend-2"))
})

# Clear legend after removing a layer
observeEvent(input$remove_layer, {
  mapboxgl_proxy("map") %>%
    remove_layer("my_layer") %>%
    clear_legend(legend_ids = "my_layer_legend")
})
```

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

clear_markers	<i>Clear markers from a map in a Shiny session</i>
---------------	--

Description

Clear markers from a map in a Shiny session

Usage

```
clear_markers(map)
```

Arguments

map	A map object created by the <code>mapboxgl_proxy</code> or <code>maplibre_proxy</code> function.
-----	--

Value

The modified map object with the markers cleared.

cluster_options	<i>Prepare cluster options for circle layers</i>
-----------------	--

Description

This function creates a list of options for clustering circle layers. Clusters are drawn as circles colored by point count, or, when `donut_column` is set, as donut charts showing the mix of categories inside each cluster.

Usage

```
cluster_options(
  max_zoom = 14,
  cluster_radius = NULL,
  color_stops = c("#51bbd6", "#f1f075", "#f28cb1"),
  radius_stops = c(20, 30, 40),
  count_stops = c(0, 100, 750),
  circle_blur = NULL,
  circle_opacity = NULL,
  circle_stroke_color = NULL,
  circle_stroke_opacity = NULL,
  circle_stroke_width = NULL,
  text_color = "black",
  count_format = c("abbreviated", "grouped", "raw"),
```

```

    donut_column = NULL,
    donut_values = NULL,
    donut_colors = NULL,
    donut_weight = NULL,
    donut_width = 0.35,
    donut_fill = "white",
    donut_resolution = 2
)

```

Arguments

max_zoom	The maximum zoom level at which to cluster points.
cluster_radius	The radius of each cluster when clustering points, in pixels. Defaults to 50 for circle clusters and to 1.5 times the largest of radius_stops (60 by default) for donut clusters, which keeps neighboring donuts from piling on top of each other.
color_stops	A vector of colors for the circle color step expression. Ignored for donut clusters.
radius_stops	A vector of radii for the circle radius step expression. Also sizes donut clusters.
count_stops	A vector of point counts for both color and radius step expressions.
circle_blur	Amount to blur the circle. Ignored for donut clusters.
circle_opacity	The opacity of the circle. For donut clusters, the opacity of the whole donut.
circle_stroke_color	The color of the circle's stroke. For donut clusters, the color of the donut's outer edge (default "white").
circle_stroke_opacity	The opacity of the circle's stroke.
circle_stroke_width	The width of the circle's stroke. For donut clusters, the width of the donut's outer edge in pixels (default 1).
text_color	The color to use for labels on the cluster circles.
count_format	The formatting of the text labels on the cluster circles to represent the counts. "abbreviated" (the default) will use shortened notation, e.g. "11k", "1.7M", or "1.5B". "grouped" will show comma-separated numbers, e.g. "11,000". "raw" shows the raw value.
donut_column	The name of a categorical column. When set, clusters are drawn as donut charts showing the share of each category within the cluster.
donut_values, donut_colors	The categories to show and their colors, one color per entry of donut_values. Pass a list to group several values under one color, e.g. <code>list(c("Oil", "Gas"), "Dry Hole")</code> . When NULL (the default), both are taken from the layer's <code>circle_color</code> if it is a <code>match_expr()</code> on <code>donut_column</code> ; its default color becomes an "other" slice for unlisted values. Required for <code>add_symbol_layer()</code> .
donut_weight	An optional numeric column to sum instead of counting points, e.g. population. The cluster label then shows the weighted total. Missing weights count as zero.
donut_width	The thickness of the donut ring as a fraction of its radius, between 0 and 1.

donut_fill	The color of the donut's center, behind the count label. Use NA for a transparent center.
donut_resolution	The step, in percent, to which category shares are rounded (an integer from 1 to 10). Categories whose share rounds to zero are not drawn.

Details

Donut clusters. Setting `donut_column` computes per-category totals for every cluster and draws each cluster as a donut chart in the category colors. Colors are usually taken from the unclustered layer's `circle_color` so that clusters and points match:

```
add_circle_layer(
  id = "people",
  source = dots,
  circle_color = match_expr(
    "race",
    values = c("White", "Black", "Hispanic", "Asian"),
    stops = c("#1b9e77", "#d95f02", "#7570b3", "#e7298a")
  ),
  cluster_options = cluster_options(donut_column = "race")
)
```

Build a matching legend by passing the same values and colors to `add_categorical_legend()`. Colors taken from `match_expr()` have already had any alpha channel removed; pass `donut_colors` to keep transparency. The ring shares are among the categories drawn: without an "other" slice, points with unlisted or missing categories are left out of the ring (but still count toward `point_count`, and toward the weighted label total when `donut_weight` is set).

Computing category totals makes clustering about two to three times slower than plain clustering with ten categories. Each distinct mix of rounded shares is drawn as its own small image and kept for the life of the map, so a long session panning across many zoom levels accumulates images (tens of MB over a wide sweep of a large dataset). A coarser `donut_resolution` produces fewer distinct images.

Pre-clustered vector tiles. For tiles that are already clustered (e.g. by `freestiler`), donut clusters read these properties from each cluster feature: `point_count`, one `"<donut_column>:<value>"` total per category (a count, or the sum of `donut_weight`), `"<donut_column>:_other"` for the other slice when there is one, and `"<donut_column>:_total"`, the weight summed over all points, when `donut_weight` is set. Numeric categories must be whole numbers and are written without exponents, e.g. `"code:100000"`. Missing properties count as zero. `_other` and `_total` are reserved and can't be used as category values.

Value

A list of cluster options.

Examples

```
cluster_options(
```

```

    max_zoom = 14,
    cluster_radius = 50,
    color_stops = c("#51bbd6", "#f1f075", "#f28cb1"),
    radius_stops = c(20, 30, 40),
    count_stops = c(0, 100, 750),
    circle_blur = 1,
    circle_opacity = 0.8,
    circle_stroke_color = "ffffff",
    circle_stroke_width = 2
)

# Donut clusters with explicit categories and colors
cluster_options(
  donut_column = "type",
  donut_values = c("Oil", "Gas", "Dry Hole"),
  donut_colors = c("#1B5E20", "#fc8d59", "#cd5c5c")
)

```

compare

Create a Compare widget

Description

This function creates a comparison view between two or more Mapbox GL or Maplibre GL maps, allowing users to either swipe between two maps or view multiple maps side-by-side with synchronized navigation.

Usage

```

compare(
  map1,
  map2,
  ...,
  width = "100%",
  height = NULL,
  elementId = NULL,
  mousemove = FALSE,
  orientation = "vertical",
  mode = "swipe",
  ncol = NULL,
  swiper_color = NULL,
  laser = FALSE,
  laser_color = "#ff2d55",
  laser_size = 14
)

```

Arguments

map1	A mapboxgl or maplibre object representing the first map.
map2	A mapboxgl or maplibre object representing the second map.
...	Additional mapboxgl or maplibre objects to include in the comparison. Supplying more than two maps requires mode = "sync", and the synced maps are arranged in a grid controlled by ncol. When extra maps are supplied, all other arguments (width, height, etc.) must be passed by name.
width	Width of the map container.
height	Height of the map container.
elementId	An optional string specifying the ID of the container for the comparison. If NULL, a unique ID will be generated.
mousemove	A logical value indicating whether to enable swiping during cursor movement (rather than only when clicked). Only applicable when mode="swipe".
orientation	A string specifying the orientation of the swiper or the side-by-side layout, either "horizontal" or "vertical".
mode	A string specifying the comparison mode: "swipe" (default) for a swipeable comparison with a slider, or "sync" for synchronized maps displayed next to each other.
ncol	Number of columns in the synced map grid. Defaults to ceiling(sqrt(n)) for more than two maps; for two maps, orientation controls the layout unless ncol is given. Only applicable when mode = "sync".
swiper_color	An optional CSS color value (e.g., "#000000", "rgb(0,0,0)", "black") to customize the color of the swiper handle. Only applicable when mode="swipe".
laser	Logical; if TRUE, show a laser pointer on the other maps that follows the cursor location. Only applies when mode = "sync".
laser_color	CSS color for the laser pointer.
laser_size	Size of the laser pointer in pixels.

Details**Comparison modes:**

The compare() function supports two modes:

- mode="swipe" (default) - Creates a swipeable interface with a slider to reveal portions of each map. Swipe mode supports exactly two maps.
- mode="sync" - Places the maps next to each other with synchronized navigation. Sync mode supports two or more maps; pass additional maps after map1 and map2 and control the grid layout with ncol.

In both modes, navigation (panning, zooming, rotating, tilting) is synchronized between the maps.

Using the compare widget in Shiny:

The compare widget can be used in Shiny applications with the following functions:

- mapboxglCompareOutput() / renderMapboxglCompare() - For Mapbox GL comparisons
- maplibreCompareOutput() / renderMaplibreCompare() - For Maplibre GL comparisons

- `mapboxgl_compare_proxy()` / `maplibre_compare_proxy()` - For updating maps in a compare widget

After creating a compare widget in a Shiny app, you can use the proxy functions to update either the "before" (left/top) or "after" (right/bottom) map. The proxy objects work with all the regular map update functions like `set_style()`, `set_paint_property()`, etc.

To get a proxy that targets a specific map in the comparison:

```
# Access the left/top map
left_proxy <- maplibre_compare_proxy("compare_id", map_side = "before")
```

```
# Access the right/bottom map
right_proxy <- maplibre_compare_proxy("compare_id", map_side = "after")
```

The compare widget also provides Shiny input values for view state and clicks. For a compare widget with ID "mycompare", you'll have:

- `input$mycompare_before_view` - View state (center, zoom, bearing, pitch) of the left/top map
- `input$mycompare_after_view` - View state of the right/bottom map
- `input$mycompare_before_click` - Click events on the left/top map
- `input$mycompare_after_click` - Click events on the right/bottom map

Comparing more than two maps:

When more than two maps are supplied with `mode = "sync"`, the maps are identified as "map1", "map2", "map3", and so on, in the order they were passed to `compare()`. Use these identifiers (or their position as an integer) as `map_side` in the proxy functions, e.g. `maplibre_compare_proxy("mycompare", map_side = "map3")` or `map_side = 3`. Shiny input values follow the same naming: `input$mycompare_map1_view`, `input$mycompare_map3_click`, etc. Legends can be targeted at individual maps in the grid with `add_legend(..., target = "map3")`.

Value

A comparison widget.

Examples

```
## Not run:
library(mapgl)

m1 <- mapboxgl(style = mapbox_style("light"))
m2 <- mapboxgl(style = mapbox_style("dark"))

# Default swipe mode
compare(m1, m2)

# Synchronized side-by-side mode
compare(m1, m2, mode = "sync")

# Synchronized maps with a laser pointer
compare(m1, m2, mode = "sync", laser = TRUE)
```

```

# Synchronize four maps in a 2 x 2 grid
m3 <- mapboxgl(style = mapbox_style("streets"))
m4 <- mapboxgl(style = mapbox_style("satellite"))
compare(m1, m2, m3, m4, mode = "sync", ncol = 2)

# Custom swiper color
compare(m1, m2, swiper_color = "#FF0000") # Red swiper

# Shiny example
library(shiny)

ui <- fluidPage(
  maplibreCompareOutput("comparison")
)

server <- function(input, output, session) {
  output$comparison <- renderMaplibreCompare({
    compare(
      maplibre(style = carto_style("positron")),
      maplibre(style = carto_style("dark-matter")),
      mode = "sync"
    )
  })

  # Update the right map
  observe({
    right_proxy <- maplibre_compare_proxy("comparison", map_side = "after")
    set_style(right_proxy, carto_style("voyager"))
  })

  # Example with custom swiper color
  output$comparison2 <- renderMaplibreCompare({
    compare(
      maplibre(style = carto_style("positron")),
      maplibre(style = carto_style("dark-matter")),
      swiper_color = "#3498db" # Blue swiper
    )
  })
}

## End(Not run)

```

concat

Create a concatenation expression

Description

This function creates a concatenation expression that combines multiple values or expressions into a single string. Useful for creating dynamic tooltips or labels.

Usage

```
concat(...)
```

Arguments

... Values or expressions to concatenate. Can be strings, numbers, or other expressions like `get_column()`.

Value

A list representing the concatenation expression.

Examples

```
# Create a dynamic tooltip
concat("<strong>Name:</strong> ", get_column("name"), "<br>Value: ", get_column("value"))
```

conditional_expressions

Conditional expressions for popups, tooltips, and styling

Description

These helpers build GL-style conditional expressions for use anywhere mapgl accepts an expression - most usefully as popup or tooltip content, where they are evaluated against each feature's properties at render time in the browser. That makes display logic possible without precomputing columns, which matters for remote sources (PMTiles, vector tiles) whose attributes cannot be modified from R. They compose with `concat()`, `get_column()`, `number_format()`, `match_expr()`, and `step_expr()`; `interpolate()` also works in popup/tooltip content, but only with numeric outputs (color stops render as an empty string with a console warning - interpolate colors in layer styling, not popups).

Usage

```
if_else_expr(condition, yes, no = "")
```

```
case_expr(..., default)
```

```
coalesce_expr(...)
```

```
has_column(column)
```

```
is_blank(column)
```

```
html_escape_expr(expression)
```

Arguments

condition	For <code>if_else_expr()</code> , an expression that evaluates to TRUE or FALSE for a feature - e.g. <code>is_blank()</code> , <code>has_column()</code> , or a raw comparison like <code>list(">=", get_column("value"), 100)</code> .
yes, no	The results when condition is true / false. Strings, numbers, or expressions.
...	For <code>case_expr()</code> , condition/output pairs (condition 1, output 1, condition 2, output 2, ...): the output for the first true condition is used. For <code>coalesce_expr()</code> , values or expressions; the first that evaluates to a non-null value is used (<code>0</code> , FALSE, and <code>""</code> count as present).
default	For <code>case_expr()</code> , the result when no condition matches. Required (may be NULL).
column	The name of the feature property to test.
expression	For <code>html_escape_expr()</code> , the expression whose result should be HTML-escaped.

Details

Conditional branches are evaluated lazily: only the branch selected for a feature is computed, so e.g. a `number_format()` in one branch is not evaluated for features that take the other branch.

Popup/tooltip expressions follow the GL expression type system: conditions must evaluate to booleans, math operators require numeric operands (a numeric string from tile data must be converted explicitly, e.g. `list("to-number", get_column("value"))`), and ordered comparisons require two numbers or two strings. A type error, malformed expression, or non-finite arithmetic result renders as an empty string with a one-time console warning rather than breaking the popup.

`is_blank(column)` is true when the property is missing, null, or an empty string (`""`). It does not detect whitespace-only strings or sentinel values like `0` - test those explicitly with a comparison condition.

Note that unlike `"{column}"` brace templates, expression results are inserted into popups and tooltips as raw HTML (that is what makes `concat("<strong>", ...)` work). Use these with trusted feature data, or wrap untrusted values in `html_escape_expr()`, which escapes its result at render time. `html_escape_expr()` is a mapgl extension for popup/tooltip content only - it is not part of the GL style specification, so don't use it in layer styling or filters.

Value

A list representing the expression.

Examples

```
# Show a fallback when a property is missing or empty (e.g. parcels from
# PMTiles where some counties don't report addresses or values)
popup_content <- concat(
  "<strong>", get_column("owner_names"), "</strong><br>",
  if_else_expr(
    is_blank("situs_addr"),
    "Address not available",
    get_column("situs_addr")
  ),
)
```

```

    "<br>",
    case_expr(
      list("=", get_column("county"), "HUDSPETH"), "Value not reported",
      default = concat("Total value: $", number_format("total_value"))
    )
  )

# First non-missing value
coalesce_expr(get_column("preferred_name"), get_column("name"), "Unnamed")

```

draw_attribute	<i>Define an editable draw attribute</i>
----------------	--

Description

This helper creates one field definition for the `attributes` argument in `add_draw_control()`. The field name comes from the name used in the `attributes` list.

Usage

```

draw_attribute(
  type = NULL,
  label = NULL,
  choices = NULL,
  default = NULL,
  required = FALSE,
  placeholder = NULL,
  min = NULL,
  max = NULL,
  step = NULL
)

```

Arguments

<code>type</code>	Input type for the editor. Supported values are "text", "textarea", "select", "number", and "checkbox". "numeric" is an alias for "number"; "logical", "bool", and "boolean" are aliases for "checkbox". If NULL, the type is inferred from choices or default.
<code>label</code>	Optional label shown in the editor. Defaults to the field name.
<code>choices</code>	Values for "select" fields. Names, if present, are used as labels and values are written to feature properties.
<code>default</code>	Optional default value. Defaults are applied to newly drawn features only; existing feature properties are preserved.
<code>required</code>	Logical; whether the browser should require a value before saving.
<code>placeholder</code>	Optional placeholder for text, textarea, and number inputs.
<code>min, max, step</code>	Optional numeric input constraints for "number" fields.

Value

A list suitable for one entry in `add_draw_control(attributes = )`.

Examples

```
draw_attribute("select", choices = c("candidate", "active", "rejected"))
draw_attribute("textarea", label = "Notes")
draw_attribute("numeric", min = 0, max = 1, step = 0.1, default = 1)

## Not run:
mapboxgl() |>
  add_draw_control(
    attributes = list(
      status = draw_attribute(
        "select",
        choices = c(Candidate = "candidate", Active = "active")
      ),
      notes = draw_attribute("textarea"),
      value = draw_attribute("numeric")
    )
  )

## End(Not run)
```

ease_to

Ease to a given view

Description

Ease to a given view

Usage

```
ease_to(map, center, zoom = NULL, ...)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function or a proxy object.
center	A numeric vector of length 2 specifying the target center of the map (longitude, latitude).
zoom	The target zoom level.
...	Additional named arguments for easing to the view.

Value

The updated map object.

enable_shiny_hover	<i>Enable hover events for Shiny applications</i>
--------------------	---

Description

This function enables hover functionality for maplibre and mapboxgl widgets in Shiny applications, providing `_hover` and `_feature_hover` input values.

Usage

```
enable_shiny_hover(map, coordinates = TRUE, features = TRUE, layer_id = NULL)
```

Arguments

<code>map</code>	A maplibre or mapboxgl widget object.
<code>coordinates</code>	Logical. If TRUE, provides general mouse coordinates via <code>_hover</code> input. Defaults to TRUE.
<code>features</code>	Logical. If TRUE, provides feature information via <code>_feature_hover</code> input when hovering over map features. Defaults to TRUE.
<code>layer_id</code>	Character. If provided, only features from the specified layer will be included in the <code>_feature_hover</code> input. Defaults to NULL. For multiple layers, provide a vector of layer IDs.

Value

The modified map object with hover events enabled.

Examples

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

ui <- fluidPage(
  maplibreOutput("map"),
  verbatimTextOutput("hover_info")
)

server <- function(input, output) {
  output$map <- renderMaplibre({
    maplibre() |>
      enable_shiny_hover()
  })

  output$hover_info <- renderText({
    paste("Mouse at:", input$map_hover$lng, input$map_hover$lat)
  })
}
```

```
shinyApp(ui, server)

## End(Not run)
```

esri_open_style	<i>Get Esri Open Basemap Style URL</i>
-----------------	--

Description

Generates a style URL for the ArcGIS Open Basemap Styles. These styles use open data from Overture Maps and OpenStreetMap. An ArcGIS access token is required.

Usage

```
esri_open_style(
  style_name,
  variant = NULL,
  token = NULL,
  language = NULL,
  worldview = NULL,
  places = NULL
)
```

Arguments

style_name	The name of the style. Available styles: "osm-style", "osm-style-relief", "navigation", "navigation-dark", "streets", "streets-relief", "streets-night", "hybrid", "light-gray", "dark-gray", "blueprint".
variant	An optional variant for the style. Not all styles support variants. Use the style table in Details to see which variants are available.
token	An ArcGIS access token (character) or an httr2_token object as returned by arcgisutils::auth_user() and similar functions. If not provided, the function will attempt to use the ARCGIS_API_KEY environment variable.
language	An optional language code for map labels (e.g., "fr", "zh-CN").
worldview	An optional worldview for boundary representation.
places	An optional POI visibility setting: "all", "attributed", or "none".

Details

The following styles and variant options are available:

Style	Variants
osm-style	(none)
osm-style-relief	base
navigation	(none)

navigation-dark	(none)
streets	(none)
streets-relief	base
streets-night	(none)
hybrid	detail
light-gray	base, labels
dark-gray	base, labels
blueprint	(none)

Value

A style URL string for use with `maplibre`.

Examples

```
## Not run:
# Basic usage
maplibre(style = esri_open_style("streets"))

# With a variant
maplibre(style = esri_open_style("light-gray", variant = "labels"))

# Dark navigation style
maplibre(style = esri_open_style("navigation-dark"))

## End(Not run)
```

esri_style	<i>Get Esri ArcGIS Basemap Style URL</i>
------------	--

Description

Generates a style URL for the ArcGIS Basemap Styles Service (v2). These styles use authoritative Esri data sources (TomTom, Garmin, USGS, etc.). An ArcGIS access token is required.

Usage

```
esri_style(
  style_name,
  variant = NULL,
  token = NULL,
  language = NULL,
  worldview = NULL,
  places = NULL
)
```

Arguments

style_name	The name of the style. Available styles: "navigation", "navigation-night", "streets", "streets-night", "streets-relief", "community", "outdoor", "topographic", "terrain", "imagery", "light-gray", "dark-gray", "oceans", "hillshade", "human-geography", "human-geography-dark", "charted-territory", "colored-pencil", "nova", "modern-antique", "midcentury", "newspaper".
variant	An optional variant for the style. Not all styles support variants. Use the style table in Details to see which variants are available.
token	An ArcGIS access token (character) or an http2_token object as returned by arcgisutils::auth_user() and similar functions. If not provided, the function will attempt to use the ARCGIS_API_KEY environment variable.
language	An optional language code for map labels (e.g., "fr", "zh-CN").
worldview	An optional worldview for boundary representation.
places	An optional POI visibility setting: "all", "attributed", or "none".

Details

The following styles and variant options are available:

Style	Variants
navigation	(none)
navigation-night	(none)
streets	(none)
streets-night	(none)
streets-relief	base
community	(none)
outdoor	(none)
topographic	base
terrain	base, detail
imagery	standard, labels
light-gray	base, labels
dark-gray	base, labels
oceans	base, labels
hillshade	light, dark
human-geography	base, detail, labels
human-geography-dark	base, detail, labels
charted-territory	base
colored-pencil	(none)
nova	(none)
modern-antique	base
midcentury	(none)
newspaper	(none)

Value

A style URL string for use with [maplibre](#).

Examples

```
## Not run:
# Basic usage
maplibre(style = esri_style("streets"))

# With a variant
maplibre(style = esri_style("topographic", variant = "base"))

# With language and places
maplibre(style = esri_style("navigation", language = "fr", places = "all"))

## End(Not run)
```

fit_bounds	<i>Fit the map to a bounding box</i>
------------	--------------------------------------

Description

Fit the map to a bounding box

Usage

```
fit_bounds(map, bbox, animate = FALSE, ...)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function or a proxy object.
bbox	A bounding box specified as a numeric vector of length 4 (<code>minLng</code> , <code>minLat</code> , <code>maxLng</code> , <code>maxLat</code>), or an <code>sf</code> object from which a bounding box will be calculated.
animate	A logical value indicating whether to animate the transition to the new bounds. Defaults to <code>FALSE</code> .
...	Additional named arguments for fitting the bounds.

Value

The updated map object.

flowmap_color_schemes	<i>FlowMapGL color scheme names</i>
-----------------------	-------------------------------------

Description

Returns the FlowMapGL 9.3.0 preset color scheme names supported by `add_flowmap()`. These names are case-sensitive.

Usage

```
flowmap_color_schemes()
```

Details

The bundled FlowMapGL presets are: Blues, BluGrn, BluYl, BrwnYl, BuGn, BuPu, Burg, BurgYl, Cool, DarkMint, Emrld, GnBu, Grayish, Greens, Greys, Inferno, Magenta, Magma, Mint, Oranges, OrRd, OrYel, Peach, Plasma, PinkYl, PuBu, PuBuGn, PuRd, Purp, Purples, PurpOr, RdPu, RedOr, Reds, Sunset, SunsetDark, Teal, TealGrn, Viridis, Warm, YlGn, YlGnBu, YlOrBr, and YlOrRd.

Value

A character vector of FlowMapGL preset names.

Examples

```
flowmap_color_schemes()
```

fly_to	<i>Fly to a given view</i>
--------	----------------------------

Description

Fly to a given view

Usage

```
fly_to(map, center, zoom = NULL, ...)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function or a proxy object.
center	A numeric vector of length 2 specifying the target center of the map (longitude, latitude).
zoom	The target zoom level.
...	Additional named arguments for flying to the view.

Value

The updated map object.

get_column	<i>Get column or property for use in mapping</i>
------------	--

Description

This function returns a an expression to get a specified column from a dataset (or a property from a layer).

Usage

```
get_column(column)
```

Arguments

column	The name of the column or property to get.
--------	--

Value

A list representing the expression to get the column.

get_drawn_features	<i>Get drawn features from the map</i>
--------------------	--

Description

Get drawn features from the map

Usage

```
get_drawn_features(map)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a map proxy.
-----	--

Details

In non-Shiny sessions, retrieval requires a map that was built by piping the original widget object through `add_draw_control()`. Non-Shiny proxy updates and compare widgets are not yet supported.

With `add_draw_control(provider = "terra-draw")`, features include a mode column recording which drawing tool created each feature (e.g. "polygon", "circle", "freehand").

Value

An sf object containing the drawn features. Feature properties are preserved as columns and the CRS is EPSG:4326. If the drawn features do not include an id property, an integer id column is added. If no features are available, a 0-row sf object with an id column is returned.

Examples

```
## Not run:
# In a Shiny application
library(shiny)
library(mapgl)

ui <- fluidPage(
  mapboxglOutput("map"),
  actionButton("get_features", "Get Drawn Features"),
  verbatimTextOutput("feature_output")
)

server <- function(input, output, session) {
  output$map <- renderMapboxgl({
    mapboxgl(
      style = mapbox_style("streets"),
      center = c(-74.50, 40),
      zoom = 9
    ) |>
    add_draw_control()
  })

  observeEvent(input$get_features, {
    drawn_features <- get_drawn_features(mapboxgl_proxy("map"))
    output$feature_output <- renderPrint({
      print(drawn_features)
    })
  })
}

shinyApp(ui, server)

## End(Not run)
```

get_queried_features *Get queried features from a map as an sf object*

Description

This function retrieves the results of a feature query triggered by query_rendered_features(). It returns the features as a deduplicated sf object. Note that only features that were visible in the viewport at the time of the query will be included.

Usage

```
get_queried_features(map)
```

Arguments

map A map object (mapboxgl, maplibre) or proxy object (mapboxgl_proxy, maplibre_proxy, mapboxgl_compare_proxy, maplibre_compare_proxy)

Value

An sf object containing the queried features, or an empty sf object if no features were found

Examples

```
## Not run:
# In a Shiny server function:
observeEvent(input$query_button, {
  proxy <- maplibre_proxy("map")
  query_rendered_features(proxy, layer_id = "counties")
  features <- get_queried_features(proxy)
  print(nrow(features))
})

## End(Not run)
```

interpolate

Create an interpolation expression

Description

This function generates an interpolation expression that can be used to style your data.

Usage

```
interpolate(
  column = NULL,
  property = NULL,
  type = "linear",
  values,
  stops,
  na_color = NULL
)
```

Arguments

column	The name of the column to use for the interpolation. If specified, property should be NULL.
property	The name of the property to use for the interpolation. If specified, column should be NULL.
type	The interpolation type. Can be one of "linear", list("exponential", base) where base specifies the rate at which the output increases, or list("cubic-bezier", x1, y1, x2, y2) where you define a cubic bezier curve with control points.
values	A numeric vector of values at which stops occur.
stops	A vector of corresponding stops (colors, sizes, etc.) for the interpolation.
na_color	The color to use for missing values. Mapbox GL JS defaults to black if this is not supplied.

Value

A list representing the interpolation expression.

Examples

```
interpolate(
  column = "estimate",
  type = "linear",
  values = c(1000, 200000),
  stops = c("#eff3ff", "#08519c")
)
```

interpolate_palette	<i>Create an interpolation expression with automatic palette and break calculation</i>
---------------------	--

Description

This function creates an interpolation expression by automatically calculating break points using different methods and applying a color palette. It handles the values/stops matching automatically and supports the same classification methods as the step functions.

Usage

```
interpolate_palette(
  data = NULL,
  column,
  data_values = NULL,
  method = "equal",
  n = 5,
  palette = NULL,
  colors = NULL,
```

```

    na_color = "grey",
    color_ramps = NULL,
    selected_ramp = NULL
  )

```

Arguments

<code>data</code>	A data frame or sf object containing the data. If provided, <code>data_values</code> will be extracted from <code>data[[column]]</code> . Either <code>data</code> or <code>data_values</code> must be provided.
<code>column</code>	The name of the column to use for the interpolation.
<code>data_values</code>	A numeric vector of the actual data values used to calculate breaks. If NULL and <code>data</code> is provided, will be extracted from <code>data[[column]]</code> .
<code>method</code>	The method for calculating breaks. Options are "equal" (equal intervals), "quantile" (quantile breaks), or "jenks" (Jenks natural breaks). Defaults to "equal".
<code>n</code>	The number of break points to create. Defaults to 5.
<code>palette</code>	A function that takes <code>n</code> and returns a character vector of colors. If NULL and <code>colors</code> is also NULL, defaults to <code>viridisLite::viridis</code> .
<code>colors</code>	A character vector of colors to use. If provided, these colors will be interpolated to match the number of breaks if needed. Either <code>palette</code> or <code>colors</code> should be provided, but not both.
<code>na_color</code>	The color to use for missing values. Defaults to "grey".
<code>color_ramps</code>	Optional list of color vectors for downstream legend color-ramp pickers. Ramps are interpolated to the calculated breaks. Named lists use the names as picker labels; unnamed lists get generated labels.
<code>selected_ramp</code>	Optional name or index of the initially selected ramp.

Value

A list of class "mapgl_continuous_scale" containing the interpolation expression and metadata.

Examples

```

## Not run:
# Create continuous color scale - using palette function
my_data <- data.frame(value = c(10, 25, 30, 45, 60, 75, 90))
scale1 <- interpolate_palette(data = my_data, column = "value",
                             method = "equal", n = 5, palette = viridisLite::plasma)

# Using specific colors (will interpolate to 5 if needed)
scale2 <- interpolate_palette(data = my_data, column = "value",
                             method = "equal", n = 5, colors = c("red", "yellow", "blue"))

# Or with piping
scale3 <- my_data |> interpolate_palette("value", method = "equal", n = 5)

# Use in a layer
add_fill_layer(map, fill_color = scale1$expression)

```

```
# Extract legend information
labels <- get_legend_labels(scale1, format = "currency")
colors <- scale1$colors

## End(Not run)
```

jump_to	<i>Jump to a given view</i>
---------	-----------------------------

Description

Jump to a given view

Usage

```
jump_to(map, center, zoom = NULL, ...)
```

Arguments

map	A map object created by the mapboxgl or maplibre function or a proxy object.
center	A numeric vector of length 2 specifying the target center of the map (longitude, latitude).
zoom	The target zoom level.
...	Additional named arguments for jumping to the view.

Value

The updated map object.

legend_style	<i>Create custom styling for map legends</i>
--------------	--

Description

This function creates a styling object that can be passed to legend functions to customize the appearance of legends, including colors, fonts, borders, and shadows.

Usage

```

legend_style(
  background_color = NULL,
  background_opacity = NULL,
  border_color = NULL,
  border_width = NULL,
  border_radius = NULL,
  text_color = NULL,
  text_size = NULL,
  title_color = NULL,
  title_size = NULL,
  font_family = NULL,
  title_font_family = NULL,
  font_weight = NULL,
  title_font_weight = NULL,
  element_border_color = NULL,
  element_border_width = NULL,
  shadow = NULL,
  shadow_color = NULL,
  shadow_size = NULL,
  padding = NULL
)

```

Arguments

<code>background_color</code>	Background color for the legend container (e.g., "white", "#ffffff").
<code>background_opacity</code>	Opacity of the legend background (0-1, where 1 is fully opaque).
<code>border_color</code>	Color of the legend border (e.g., "black", "#000000").
<code>border_width</code>	Width of the legend border in pixels.
<code>border_radius</code>	Border radius for rounded corners in pixels.
<code>text_color</code>	Color of the legend text (e.g., "black", "#000000").
<code>text_size</code>	Size of the legend text in pixels.
<code>title_color</code>	Color of the legend title text.
<code>title_size</code>	Size of the legend title text in pixels.
<code>font_family</code>	Font family for legend text (e.g., "Arial", "Times New Roman", "Open Sans").
<code>title_font_family</code>	Font family for legend title (defaults to <code>font_family</code> if not specified).
<code>font_weight</code>	Font weight for legend text (e.g., "normal", "bold", "lighter", or numeric like 400, 700).
<code>title_font_weight</code>	Font weight for legend title (defaults to <code>font_weight</code> if not specified).
<code>element_border_color</code>	Color for borders around legend elements (color bar for continuous, patches/circles for categorical).

element_border_width	Width in pixels for borders around legend elements.
shadow	Logical, whether to add a drop shadow to the legend.
shadow_color	Color of the drop shadow (e.g., "black", "rgba(0,0,0,0.3)").
shadow_size	Size/blur radius of the drop shadow in pixels.
padding	Internal padding of the legend container in pixels.

Value

A list of class "mapgl_legend_style" containing the styling options.

Examples

```
## Not run:
# Create a dark theme legend style
dark_style <- legend_style(
  background_color = "#2c3e50",
  text_color = "white",
  title_color = "white",
  font_family = "Arial",
  title_font_weight = "bold",
  element_border_color = "white",
  element_border_width = 1,
  shadow = TRUE,
  shadow_color = "rgba(0,0,0,0.3)",
  shadow_size = 6
)

# Use the style in a legend
add_categorical_legend(
  map = map,
  legend_title = "Categories",
  values = c("A", "B", "C"),
  colors = c("red", "green", "blue"),
  style = dark_style
)

# Create a minimal style with just borders
minimal_style <- legend_style(
  element_border_color = "gray",
  element_border_width = 1
)

## End(Not run)
```

`mapboxgl`*Initialize a Mapbox GL Map*

Description

Initialize a Mapbox GL Map

Usage

```
mapboxgl(  
  style = NULL,  
  center = c(0, 0),  
  zoom = 0,  
  bearing = 0,  
  pitch = 0,  
  projection = "globe",  
  parallels = NULL,  
  access_token = NULL,  
  bounds = NULL,  
  width = "100%",  
  height = NULL,  
  ...  
)
```

Arguments

<code>style</code>	The Mapbox style to use.
<code>center</code>	A numeric vector of length 2 specifying the initial center of the map.
<code>zoom</code>	The initial zoom level of the map.
<code>bearing</code>	The initial bearing (rotation) of the map, in degrees.
<code>pitch</code>	The initial pitch (tilt) of the map, in degrees.
<code>projection</code>	The map projection to use (e.g., "mercator", "globe").
<code>parallels</code>	A vector of two numbers representing the standard parallels of the projection. Only available when the projection is "albers" or "lambertConformalConic".
<code>access_token</code>	Your Mapbox access token.
<code>bounds</code>	The bounding box to fit the map to. Accepts one of the following: • <code>sf</code> object; • output of <code>st_bbox()</code>; • unnamed numeric vector of the form <code>c(xmin, ymin, xmax, ymax)</code>.
<code>width</code>	The width of the output <code>htmlwidget</code> .
<code>height</code>	The height of the output <code>htmlwidget</code> .

... Additional named parameters to be passed to the Mapbox GL JS Map. See the Mapbox GL JS documentation for a full list of options: <https://docs.mapbox.com/mapbox-gl-js/api/map/#map-parameters>. Common options include:

- minZoom / maxZoom: Minimum and maximum zoom levels (0-24).
- maxBounds: Restrict panning to a bounding box, specified as `list(c(sw_lng, sw_lat), c(ne_lng, ne_lat))`.
- dragRotate: If FALSE, disables rotation via mouse drag (default TRUE).
- touchZoomRotate: If FALSE, disables pinch-to-rotate on touch (default TRUE).
- scrollZoom: If FALSE, disables scroll wheel zoom (default TRUE).

Value

An HTML widget for a Mapbox GL map.

Examples

```
## Not run:
# Basic map
mapboxgl(projection = "globe")

# Constrained map with zoom limits and disabled rotation
mapboxgl(
  bounds = my_sf_object,
  minZoom = 5,
  maxZoom = 12,
  dragRotate = FALSE,
  touchZoomRotate = FALSE
)

## End(Not run)
```

mapboxglCompareOutput *Create a Mapbox GL Compare output element for Shiny*

Description

Create a Mapbox GL Compare output element for Shiny

Usage

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

Arguments

outputId	The output variable to read from
width	The width of the element
height	The height of the element

Value

A Mapbox GL Compare output element for use in a Shiny UI

mapboxglOutput	Create a Mapbox GL output element for Shiny
----------------	---

Description

Create a Mapbox GL output element for Shiny

Usage

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

Arguments

outputId	The output variable to read from
width	The width of the element
height	The height of the element

Value

A Mapbox GL output element for use in a Shiny UI

mapboxgl_compare_proxy	Create a proxy object for a Mapbox GL Compare widget in Shiny
------------------------	---

Description

This function allows updates to be sent to an existing Mapbox GL Compare widget in a Shiny application.

Usage

```
mapboxgl_compare_proxy(  
  compareId,  
  session = shiny::getDefaultReactiveDomain(),  
  map_side = "before"  
)
```

Arguments

compareId	The ID of the compare output element.
session	The Shiny session object.
map_side	Which map to target in the compare widget: "before" or "after" for two-map widgets, or a map identifier such as "map3" (or its position as an integer, e.g. 3) for synced grids with more than two maps. "before" and "after" are aliases for the first and second maps.

Value

A proxy object for the Mapbox GL Compare widget.

mapboxgl_proxy	<i>Create a proxy object for a Mapbox GL map in Shiny</i>
----------------	---

Description

This function allows updates to be sent to an existing Mapbox GL map in a Shiny application without redrawing the entire map.

Usage

```
mapboxgl_proxy(mapId, session = shiny::getDefaultReactiveDomain())
```

Arguments

mapId	The ID of the map output element.
session	The Shiny session object.

Value

A proxy object for the Mapbox GL map.

mapboxgl_view	<i>Quick visualization of geometries with Mapbox GL</i>
---------------	---

Description

This function provides a quick way to visualize sf geometries and raster data using Mapbox GL JS. It automatically detects the geometry type and applies appropriate styling.

Usage

```
mapboxgl_view(
  data,
  color = "navy",
  column = NULL,
  n = NULL,
  palette = viridisLite::viridis,
  style = mapbox_style("light"),
  layer_id = "quickview",
  legend = TRUE,
  legend_position = "top-left",
  interactive_legend = FALSE,
  ...
)
```

Arguments

<code>data</code>	An sf object, SpatRaster, or RasterLayer to visualize
<code>color</code>	The color used to visualize points, lines, or polygons if column is NULL. Defaults to "navy".
<code>column</code>	The name of the column to visualize. If NULL (default), geometries are shown with default styling.
<code>n</code>	Number of quantile breaks for numeric columns. If specified, uses <code>step_expr()</code> instead of <code>interpolate()</code> .
<code>palette</code>	Color palette function that takes <code>n</code> and returns a character vector of colors. Defaults to <code>viridisLite::viridis</code> .
<code>style</code>	The Mapbox style to use. Defaults to <code>mapbox_style("light")</code> .
<code>layer_id</code>	The layer ID to use for the visualization. Defaults to "quickview".
<code>legend</code>	Logical, whether to add a legend when a column is specified. Defaults to TRUE.
<code>legend_position</code>	The position of the legend on the map. Defaults to "top-left".
<code>interactive_legend</code>	Logical, whether to make the legend interactive. When TRUE, categorical legends allow clicking to toggle visibility, and continuous legends show a range slider. Defaults to FALSE.
<code>...</code>	Additional arguments passed to <code>mapboxgl()</code>

Value

A Mapbox GL map object

Examples

```
## Not run:
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
```

```
# Basic view
mapboxgl_view(nc)

# View with column visualization
mapboxgl_view(nc, column = "AREA")

# View with quantile breaks
mapboxgl_view(nc, column = "AREA", n = 5)

# Custom palette examples
mapboxgl_view(nc, column = "AREA", palette = viridisLite::mako)
mapboxgl_view(nc, column = "AREA", palette = function(n) RColorBrewer::brewer.pal(n, "RdYlBu"))
mapboxgl_view(nc, column = "AREA", palette = colorRampPalette(c("red", "white", "blue"))))

## End(Not run)
```

mapbox_style	<i>Get Mapbox Style URL</i>
--------------	-----------------------------

Description

Get Mapbox Style URL

Usage

```
mapbox_style(style_name)
```

Arguments

style_name The name of the style (e.g., "standard", "streets", "outdoors", etc.).

Value

The style URL corresponding to the given style name.

maplibre	<i>Initialize a Maplibre GL Map</i>
----------	-------------------------------------

Description

Initialize a Maplibre GL Map

Usage

```
maplibre(
  style = openfreemap_style("bright"),
  center = c(0, 0),
  zoom = 0,
  bearing = 0,
  pitch = 0,
  projection = "globe",
  bounds = NULL,
  width = "100%",
  height = NULL,
  ...
)
```

Arguments

<code>style</code>	The style JSON to use.
<code>center</code>	A numeric vector of length 2 specifying the initial center of the map.
<code>zoom</code>	The initial zoom level of the map.
<code>bearing</code>	The initial bearing (rotation) of the map, in degrees.
<code>pitch</code>	The initial pitch (tilt) of the map, in degrees.
<code>projection</code>	The map projection to use (e.g., "mercator", "globe").
<code>bounds</code>	The bounding box to fit the map to. Accepts one of the following: • <code>sf</code> object; • output of <code>st_bbox()</code>; • unnamed numeric vector of the form <code>c(xmin, ymin, xmax, ymax)</code>.
<code>width</code>	The width of the output <code>htmlwidget</code> .
<code>height</code>	The height of the output <code>htmlwidget</code> .
<code>...</code>	Additional named parameters to be passed to the MapLibre GL JS Map. See the MapLibre GL JS documentation for a full list of options: https://maplibre.org/maplibre-gl-js/docs/API/type-aliases/MapOptions/ . Common options include: • <code>minZoom</code> / <code>maxZoom</code>: Minimum and maximum zoom levels (0-24). • <code>maxBounds</code>: Restrict panning to a bounding box, specified as <code>list(c(sw_lng, sw_lat), c(ne_lng, ne_lat))</code>. • <code>dragRotate</code>: If <code>FALSE</code>, disables rotation via mouse drag (default <code>TRUE</code>). • <code>touchZoomRotate</code>: If <code>FALSE</code>, disables pinch-to-rotate on touch (default <code>TRUE</code>). • <code>scrollZoom</code>: If <code>FALSE</code>, disables scroll wheel zoom (default <code>TRUE</code>).

Value

An HTML widget for a MapLibre GL map.

Examples

```
## Not run:
# Basic map
maplibre()

# Constrained map with zoom limits and disabled rotation
maplibre(
  bounds = my_sf_object,
  minZoom = 5,
  maxZoom = 12,
  dragRotate = FALSE,
  touchZoomRotate = FALSE
)

## End(Not run)
```

maplibreCompareOutput *Create a Maplibre GL Compare output element for Shiny*

Description

Create a Maplibre GL Compare output element for Shiny

Usage

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

Arguments

outputId	The output variable to read from
width	The width of the element
height	The height of the element

Value

A Maplibre GL Compare output element for use in a Shiny UI

maplibreOutput	Create a Maplibre GL output element for Shiny
----------------	---

Description

Create a Maplibre GL output element for Shiny

Usage

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

Arguments

outputId	The output variable to read from
width	The width of the element
height	The height of the element

Value

A Maplibre GL output element for use in a Shiny UI

maplibre_compare_proxy	Create a proxy object for a Maplibre GL Compare widget in Shiny
------------------------	---

Description

This function allows updates to be sent to an existing Maplibre GL Compare widget in a Shiny application.

Usage

```
maplibre_compare_proxy(
  compareId,
  session = shiny::getDefaultReactiveDomain(),
  map_side = "before"
)
```

Arguments

compareId	The ID of the compare output element.
session	The Shiny session object.
map_side	Which map to target in the compare widget: "before" or "after" for two-map widgets, or a map identifier such as "map3" (or its position as an integer, e.g. 3) for synced grids with more than two maps. "before" and "after" are aliases for the first and second maps.

Value

A proxy object for the Maplibre GL Compare widget.

maplibre_proxy	<i>Create a proxy object for a Maplibre GL map in Shiny</i>
----------------	---

Description

This function allows updates to be sent to an existing Maplibre GL map in a Shiny application without redrawing the entire map.

Usage

```
maplibre_proxy(mapId, session = shiny::getDefaultReactiveDomain())
```

Arguments

mapId	The ID of the map output element.
session	The Shiny session object.

Value

A proxy object for the Maplibre GL map.

maplibre_view	<i>Quick visualization of geometries with MapLibre GL</i>
---------------	---

Description

This function provides a quick way to visualize sf geometries and raster data using MapLibre GL JS. It automatically detects the geometry type and applies appropriate styling.

Usage

```
maplibre_view(
  data,
  color = "navy",
  column = NULL,
  n = NULL,
  palette = viridisLite::viridis,
  style = openfreemap_style("positron"),
  layer_id = "quickview",
  legend = TRUE,
  legend_position = "top-left",
  interactive_legend = FALSE,
  ...
)
```

Arguments

<code>data</code>	An sf object, SpatRaster, or RasterLayer to visualize
<code>color</code>	The color used to visualize points, lines, or polygons if column is NULL. Defaults to "navy".
<code>column</code>	The name of the column to visualize. If NULL (default), geometries are shown with default styling.
<code>n</code>	Number of quantile breaks for numeric columns. If specified, uses <code>step_expr()</code> instead of <code>interpolate()</code> .
<code>palette</code>	Color palette function that takes <code>n</code> and returns a character vector of colors. Defaults to <code>viridisLite::viridis</code> .
<code>style</code>	The MapLibre style to use. Defaults to <code>carto_style("positron")</code> .
<code>layer_id</code>	The layer ID to use for the visualization. Defaults to "quickview".
<code>legend</code>	Logical, whether to add a legend when a column is specified. Defaults to TRUE.
<code>legend_position</code>	The position of the legend on the map. Defaults to "top-left".
<code>interactive_legend</code>	Logical, whether to make the legend interactive. When TRUE, categorical legends allow clicking to toggle visibility, and continuous legends show a range slider. Defaults to FALSE.
<code>...</code>	Additional arguments passed to <code>maplibre()</code>

Value

A MapLibre GL map object

Examples

```
## Not run:
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))

# Basic view
maplibre_view(nc)

# View with column visualization
maplibre_view(nc, column = "AREA")

# View with quantile breaks
maplibre_view(nc, column = "AREA", n = 5)

# Custom palette examples
maplibre_view(nc, column = "AREA", palette = viridisLite::mako)
maplibre_view(nc, column = "AREA", palette = function(n) RColorBrewer::brewer.pal(n, "RdYlBu"))
maplibre_view(nc, column = "AREA", palette = colorRampPalette(c("red", "white", "blue"))))

## End(Not run)
```

maptiler_style	<i>Get MapTiler Style URL</i>
----------------	-------------------------------

Description

Get MapTiler Style URL

Usage

```
maptiler_style(style_name, variant = NULL, api_key = NULL)
```

Arguments

style_name	The name of the style (e.g., "basic", "streets", "toner", etc.).
variant	The color variant of the style. Options are "dark", "light", or "pastel". Default is NULL (standard variant). Not all styles support all variants.
api_key	Your MapTiler API key (required)

Value

The style URL corresponding to the given style name and variant.

map_legends	<i>Add legends to Mapbox GL and MapLibre GL maps</i>
-------------	--

Description

These functions add categorical and continuous legends to maps. Use `legend_style()` to customize appearance and `clear_legend()` to remove legends.

Usage

```
add_legend(
  map,
  legend_title,
  values = NULL,
  colors = NULL,
  type = c("continuous", "categorical"),
  circular_patches = FALSE,
  patch_shape = "square",
  position = "top-left",
  sizes = NULL,
  add = FALSE,
  unique_id = NULL,
  width = NULL,
```

```

    layer_id = NULL,
    margin_top = NULL,
    margin_right = NULL,
    margin_bottom = NULL,
    margin_left = NULL,
    style = NULL,
    target = NULL,
    interactive = FALSE,
    filter_column = NULL,
    filter_values = NULL,
    classification = NULL,
    breaks = NULL,
    color_ramps = NULL,
    selected_ramp = NULL,
    ramp_picker = !is.null(color_ramps),
    ramp_labels = TRUE,
    color_column = NULL,
    color_property = NULL,
    na_color = NULL,
    draggable = FALSE,
    collapsible = FALSE,
    collapsed = FALSE,
    patch_spacing = c("uniform", "proportional"),
    min_zoom = NULL,
    max_zoom = NULL
  )

```

```

add_categorical_legend(
  map,
  legend_title,
  values,
  colors,
  circular_patches = FALSE,
  patch_shape = "square",
  position = "top-left",
  unique_id = NULL,
  sizes = NULL,
  add = FALSE,
  width = NULL,
  layer_id = NULL,
  margin_top = NULL,
  margin_right = NULL,
  margin_bottom = NULL,
  margin_left = NULL,
  style = NULL,
  interactive = FALSE,
  filter_column = NULL,
  filter_values = NULL,

```

```

    breaks = NULL,
    draggable = FALSE,
    collapsible = FALSE,
    collapsed = FALSE,
    patch_spacing = c("uniform", "proportional"),
    min_zoom = NULL,
    max_zoom = NULL
  )

```

```

add_continuous_legend(
  map,
  legend_title,
  values,
  colors,
  position = "top-left",
  unique_id = NULL,
  add = FALSE,
  width = NULL,
  layer_id = NULL,
  margin_top = NULL,
  margin_right = NULL,
  margin_bottom = NULL,
  margin_left = NULL,
  style = NULL,
  interactive = FALSE,
  filter_column = NULL,
  filter_values = NULL,
  draggable = FALSE,
  color_ramps = NULL,
  selected_ramp = NULL,
  ramp_picker = !is.null(color_ramps),
  ramp_labels = TRUE,
  color_column = NULL,
  color_property = NULL,
  na_color = NULL,
  collapsible = FALSE,
  collapsed = FALSE,
  min_zoom = NULL,
  max_zoom = NULL
)

```

```

## S3 method for class 'mapboxgl_compare'
add_legend(
  map,
  legend_title,
  values,
  colors,
  type = c("continuous", "categorical"),

```

```

    circular_patches = FALSE,
    patch_shape = "square",
    position = "top-left",
    sizes = NULL,
    add = FALSE,
    unique_id = NULL,
    width = NULL,
    layer_id = NULL,
    margin_top = NULL,
    margin_right = NULL,
    margin_bottom = NULL,
    margin_left = NULL,
    style = NULL,
    target = "compare",
    interactive = FALSE,
    filter_column = NULL,
    filter_values = NULL,
    classification = NULL,
    breaks = NULL,
    color_ramps = NULL,
    selected_ramp = NULL,
    ramp_picker = !is.null(color_ramps),
    ramp_labels = TRUE,
    color_column = NULL,
    color_property = NULL,
    na_color = NULL,
    draggable = FALSE,
    collapsible = FALSE,
    collapsed = FALSE,
    patch_spacing = c("uniform", "proportional"),
    min_zoom = NULL,
    max_zoom = NULL
  )

```

```

## S3 method for class 'maplibre_compare'
add_legend(
  map,
  legend_title,
  values,
  colors,
  type = c("continuous", "categorical"),
  circular_patches = FALSE,
  patch_shape = "square",
  position = "top-left",
  sizes = NULL,
  add = FALSE,
  unique_id = NULL,
  width = NULL,

```

```

layer_id = NULL,
margin_top = NULL,
margin_right = NULL,
margin_bottom = NULL,
margin_left = NULL,
style = NULL,
target = "compare",
interactive = FALSE,
filter_column = NULL,
filter_values = NULL,
classification = NULL,
breaks = NULL,
color_ramps = NULL,
selected_ramp = NULL,
ramp_picker = !is.null(color_ramps),
ramp_labels = TRUE,
color_column = NULL,
color_property = NULL,
na_color = NULL,
draggable = FALSE,
collapsible = FALSE,
collapsed = FALSE,
patch_spacing = c("uniform", "proportional"),
min_zoom = NULL,
max_zoom = NULL
)

```

```

## S3 method for class 'maplibregl_compare'
add_legend(
  map,
  legend_title,
  values,
  colors,
  type = c("continuous", "categorical"),
  circular_patches = FALSE,
  patch_shape = "square",
  position = "top-left",
  sizes = NULL,
  add = FALSE,
  unique_id = NULL,
  width = NULL,
  layer_id = NULL,
  margin_top = NULL,
  margin_right = NULL,
  margin_bottom = NULL,
  margin_left = NULL,
  style = NULL,
  target = "compare",

```

```

    interactive = FALSE,
    filter_column = NULL,
    filter_values = NULL,
    classification = NULL,
    breaks = NULL,
    color_ramps = NULL,
    selected_ramp = NULL,
    ramp_picker = !is.null(color_ramps),
    ramp_labels = TRUE,
    color_column = NULL,
    color_property = NULL,
    na_color = NULL,
    draggable = FALSE,
    collapsible = FALSE,
    collapsed = FALSE,
    patch_spacing = c("uniform", "proportional"),
    min_zoom = NULL,
    max_zoom = NULL
  )

```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function.
legend_title	The title of the legend.
values	The values being represented on the map (either a vector of categories or a vector of stops).
colors	The corresponding colors for the values (either a vector of colors, a single color, or an interpolate function).
type	One of "continuous" or "categorical" (for <code>add_legend</code> only).
circular_patches	(Deprecated) Logical, whether to use circular patches in the legend. Use <code>patch_shape = "circle"</code> instead.
patch_shape	Character or <code>sf</code> object, the shape of patches to use in categorical legends. Can be one of the built-in shapes ("square", "circle", "line", "hexagon"), a custom SVG string, or an <code>sf</code> object with POLYGON or MULTIPOLYGON geometry (which will be automatically converted to SVG). Default is "square".
position	The position of the legend on the map (one of "top-left", "bottom-left", "top-right", "bottom-right").
sizes	An optional numeric vector of sizes for the legend patches, or a single numeric value (only for categorical legends). For line patches, this controls the line thickness.
add	Logical, whether to add this legend to existing legends (TRUE) or replace existing legends (FALSE). Default is FALSE.
unique_id	Optional. A unique identifier for the legend. If not provided, a random ID will be generated.

width	The width of the legend. Can be specified in pixels (e.g., "250px") or as "auto". Default is NULL, which uses the built-in default.
layer_id	The ID of the layer (or a character vector of layer IDs) that this legend is associated with. If provided, the legend will be shown/hidden when the layer visibility is toggled. When multiple layer IDs are provided with <code>interactive = TRUE</code> , the legend will filter all specified layers simultaneously.
margin_top	Custom top margin in pixels, allowing for fine control over legend positioning. Default is NULL (uses standard positioning).
margin_right	Custom right margin in pixels. Default is NULL.
margin_bottom	Custom bottom margin in pixels. Default is NULL.
margin_left	Custom left margin in pixels. Default is NULL.
style	Optional styling options created by <code>legend_style()</code> or a list of style options.
target	For compare objects only: where to place the legend. Can be "compare" (attached to compare container, persists during swipe), "before" (attached to left/top map), or "after" (attached to right/bottom map). Default is "compare". For synced grids with more than two maps, a map identifier such as "map3" can also be used.
interactive	Logical, whether to make the legend interactive. For categorical legends, clicking on legend items will toggle the visibility of the corresponding features. For continuous legends, a range slider will appear allowing users to filter features by value. Default is FALSE. Note: interactive legends are not yet supported for compare maps.
filter_column	Character, the name of the data column to use for filtering when interactive is TRUE. If NULL (default), the column will be auto-detected from the layer's paint expression.
filter_values	For interactive legends, the actual data values to filter on. For categorical legends, use this when your display labels differ from the data values (e.g., <code>values = c("Music", "Bar")</code> for display, <code>filter_values = c("music", "bar")</code> for filtering). For continuous legends, provide numeric break values when using formatted display labels (e.g., <code>values = get_legend_labels(scale)</code> , <code>filter_values = get_breaks(scale)</code>). If NULL (default), uses values.
classification	A <code>mapgl_classification</code> object (from <code>step_quantile</code> , <code>step_equal_interval</code> , etc.) to use for the legend. When provided, values and colors will be automatically extracted. For interactive legends, range-based filtering will be used based on the classification breaks.
breaks	Numeric vector of break points for filtering with classification-based legends. Typically extracted automatically from the classification object. Only needed if you want to override the default breaks.
color_ramps	For continuous legends, a list of color vectors to expose in a color-ramp picker. Named lists use the names as picker labels; unnamed lists get generated labels.
selected_ramp	The initially selected ramp name or index when <code>color_ramps</code> is provided.
ramp_picker	Logical, whether to show the continuous legend color-ramp picker.
ramp_labels	Logical, whether to show palette labels in the color-ramp picker.

color_column	Character, the data column to use when restyling the layer. If NULL, mapgl attempts to auto-detect it from the layer paint expression.
color_property	Character, the paint property to restyle. If NULL, mapgl attempts to auto-detect one of fill-color, circle-color, line-color, or fill-extrusion-color.
na_color	Color to use for missing values when rebuilding the interpolation expression.
draggable	Logical, whether the legend can be dragged to a new position by the user. Default is FALSE.
collapsible	Logical, whether to render a toggle button that collapses the legend to a header-only view. Default is FALSE. Most useful for categorical legends with tall bodies on small viewports.
collapsed	Logical, whether the legend starts in the collapsed state. Only applies when collapsible = TRUE. Default is FALSE.
patch_spacing	For categorical legends with per-item sizes, either "uniform" (default; every row uses the largest symbol's height) or "proportional" (each row's height tracks its own symbol size, so vertical spacing scales with symbol size - useful for graduated-symbol legends).
min_zoom	The minimum zoom level at which the legend is displayed. The legend is hidden when the map zoom is below this value.
max_zoom	The maximum zoom level at which the legend is displayed. The legend is hidden when the map zoom is at or above this value.

Details

Zoom-based visibility. min_zoom and max_zoom show or hide the legend as the user zooms, with the same semantics as the layer arguments of the same names: the legend is visible when zoom >= min_zoom and zoom < max_zoom. Pass the same values you gave a layer's min_zoom/max_zoom and the legend will track that layer's rendering - for example, a county legend that gives way to a parcel legend as the user zooms in.

Automatic legend stacking. When multiple legends share the same corner position, they are automatically stacked so they do not overlap, and they reflow when a legend is shown or hidden (by zoom, by a layers control, or by collapsing). Legends with any explicit margin_* value, and legends the user has dragged, are left exactly where they were placed and are ignored by the stacking (they may overlap a stacked group).

Collapsible legends. When collapsible = TRUE, a 26x26px toggle button is rendered in the legend's top-right corner. Collapsed, only the title heading and the toggle button remain visible; every other direct child of the legend (subtitles, swatches, item labels, the reset-filter button from interactive legends, any user-appended source footers) is hidden via CSS. The toggle button inherits border_color and text_color from legend_style() so it picks up your legend theme.

If you inject your own title block via htmlwidgets::onRender() – for example, to add a styled heading above the default title – mark that element with class="mapgl-legend-title" so it stays visible when collapsed.

Value

The updated map object with the legend added.

Examples

```
## Not run:
# Basic categorical legend
add_legend(map, "Population",
            values = c("Low", "Medium", "High"),
            colors = c("blue", "yellow", "red"),
            type = "categorical")

# Continuous legend with custom styling
add_legend(map, "Income",
            values = c(0, 50000, 100000),
            colors = c("blue", "yellow", "red"),
            type = "continuous",
            style = list(
                background_color = "white",
                background_opacity = 0.9,
                border_width = 2,
                border_color = "navy",
                text_color = "darkblue",
                font_family = "Times New Roman",
                title_font_weight = "bold"
            ))

# Legend with custom styling using a list
add_legend(map, "Temperature",
            values = c(0, 50, 100),
            colors = c("blue", "yellow", "red"),
            type = "continuous",
            style = list(
                background_color = "#f0f0f0",
                title_size = 16,
                text_size = 12,
                shadow = TRUE,
                shadow_color = "rgba(0,0,0,0.1)",
                shadow_size = 8
            ))

# Dark legend with white element borders
add_legend(map, "Elevation",
            values = c(0, 1000, 2000, 3000),
            colors = c("#2c7bb6", "#abd9e9", "#fdae61", "#d7191c"),
            type = "continuous",
            style = list(
                background_color = "#2c3e50",
                text_color = "white",
                title_color = "white",
                element_border_color = "white",
                element_border_width = 1
            ))

# Categorical legend with circular patches
add_categorical_legend(
```

```

    map = map,
    legend_title = "Population",
    values = c("Low", "Medium", "High"),
    colors = c("#FED976", "#FEB24C", "#FD8D3C"),
    patch_shape = "circle",
    sizes = c(10, 15, 20),
    style = list(
      background_opacity = 0.95,
      border_width = 1,
      border_color = "gray",
      title_color = "navy",
      element_border_color = "black",
      element_border_width = 1
    )
  )

# Legend with line patches for line layers
add_categorical_legend(
  map = map,
  legend_title = "Road Type",
  values = c("Highway", "Primary", "Secondary"),
  colors = c("#000000", "#333333", "#666666"),
  patch_shape = "line",
  sizes = c(5, 3, 1) # Line thickness in pixels
)

# Legend with hexagon patches (e.g., for H3 data)
add_categorical_legend(
  map = map,
  legend_title = "H3 Hexagon Categories",
  values = c("Urban", "Suburban", "Rural"),
  colors = c("#8B0000", "#FF6347", "#90EE90"),
  patch_shape = "hexagon",
  sizes = 25
)

# Custom SVG shapes - star
add_categorical_legend(
  map = map,
  legend_title = "Ratings",
  values = c("5 Star", "4 Star", "3 Star"),
  colors = c("#FFD700", "#FFA500", "#FF6347"),
  patch_shape = paste0('<path d="M50,5 L61,35 L95,35 L68,57 L79,91 L50,70 ',
    'L21,91 L32,57 L5,35 L39,35 Z" />')
)

# Using sf objects directly as patch shapes
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
county_shape <- nc[1, ] # Get first county

add_categorical_legend(
  map = map,

```

```

    legend_title = "County Types",
    values = c("Rural", "Urban", "Suburban"),
    colors = c("#228B22", "#8B0000", "#FFD700"),
    patch_shape = county_shape # sf object automatically converted to SVG
  )

# For advanced users needing custom conversion options
custom_svg <- mapgl:::sf_to_svg(county_shape, simplify = TRUE,
                                tolerance = 0.001, fit_viewbox = TRUE)

add_categorical_legend(
  map = map,
  legend_title = "Custom Converted Shape",
  values = c("Type A"),
  colors = c("#4169E1"),
  patch_shape = custom_svg
)

# Compare view legends
compare_view <- compare(map1, map2)

# Add persistent legend (stays visible during swipe)
compare_view |>
  add_legend("Persistent Legend",
    values = c("Low", "High"),
    colors = c("blue", "red"),
    type = "categorical",
    target = "compare",
    position = "top-left")

# Add legends to specific maps
compare_view |>
  add_legend("Left Map",
    values = c("A", "B"),
    colors = c("green", "orange"),
    type = "categorical",
    target = "before",
    position = "bottom-left") |>
  add_legend("Right Map",
    values = c("X", "Y"),
    colors = c("purple", "yellow"),
    type = "categorical",
    target = "after",
    position = "bottom-right")

## End(Not run)

```

Description

This function generates a match expression that can be used to style your data.

Usage

```
match_expr(column = NULL, property = NULL, values, stops, default = "#cccccc")
```

Arguments

column	The name of the column to use for the match expression. If specified, property should be NULL.
property	The name of the property to use for the match expression. If specified, column should be NULL.
values	A vector of values to match against.
stops	A vector of corresponding stops (colors, etc.) for the matched values.
default	A default value to use if no matches are found.

Value

A list representing the match expression.

Examples

```
match_expr(
  column = "category",
  values = c("A", "B", "C"),
  stops = c("#ff0000", "#00ff00", "#0000ff"),
  default = "#cccccc"
)
```

move_layer

Move a layer to a different z-position

Description

This function allows a layer to be moved to a different z-position in a Mapbox GL or Maplibre GL map. For initial maps, the operation is queued and executed during map initialization. For proxy objects, the operation is executed immediately.

Usage

```
move_layer(map, layer_id, before_id = NULL)
```

Arguments

map	A map object created by mapboxgl or maplibre, or a proxy object created by mapboxgl_proxy or maplibre_proxy.
layer_id	The ID of the layer to move.
before_id	The ID of an existing layer to insert the new layer before. Important: this means that the layer will appear <i>immediately behind</i> the layer defined in before_id. If omitted, the layer will be appended to the end of the layers array and appear above all other layers.

Value

The updated map or proxy object.

number_format	<i>Create a number formatting expression</i>
---------------	--

Description

This function creates a number formatting expression that formats numeric values according to locale-specific conventions. It can be used in tooltips, popups, and text fields for symbol layers.

Usage

```
number_format(
  column,
  locale = "en-US",
  style = "decimal",
  currency = NULL,
  unit = NULL,
  minimum_fraction_digits = NULL,
  maximum_fraction_digits = NULL,
  minimum_integer_digits = NULL,
  use_grouping = NULL,
  notation = NULL,
  compact_display = NULL
)
```

Arguments

column	The name of the column containing the numeric value to format. Can also be an expression that evaluates to a number.
locale	A string specifying the locale to use for formatting (e.g., "en-US", "de-DE", "fr-FR"). Defaults to "en-US".
style	The formatting style to use. Options include: "decimal" (default): Plain number formatting

	• "currency": Currency formatting (requires currency parameter) • "percent": Percentage formatting (multiplies by 100 and adds %) • "unit": Unit formatting (requires unit parameter)
currency	For style = "currency", the ISO 4217 currency code (e.g., "USD", "EUR", "GBP").
unit	For style = "unit", the unit to use (e.g., "kilometer", "mile", "liter").
minimum_fraction_digits	The minimum number of fraction digits to display.
maximum_fraction_digits	The maximum number of fraction digits to display.
minimum_integer_digits	The minimum number of integer digits to display.
use_grouping	Whether to use grouping separators (e.g., thousands separators). Defaults to TRUE.
notation	The formatting notation. Options include: • "standard" (default): Regular notation • "scientific": Scientific notation • "engineering": Engineering notation • "compact": Compact notation (e.g., "1.2K", "3.4M")
compact_display	For notation = "compact", whether to use "short" (default) or "long" form.

Value

A list representing the number-format expression.

Examples

```
# Basic number formatting with thousands separators
number_format("population")

# Currency formatting
number_format("income", style = "currency", currency = "USD")

# Percentage with 1 decimal place
number_format("rate", style = "percent", maximum_fraction_digits = 1)

# Compact notation for large numbers
number_format("population", notation = "compact")

# Using within a tooltip
concat("Population: ", number_format("population", notation = "compact"))

# Using with get_column()
number_format(get_column("value"), style = "currency", currency = "EUR")
```

on_section	<i>Observe events on story map section transitions</i>
------------	--

Description

For a given `story_section()`, you may want to trigger an event when the section becomes visible. This function wraps `shiny::observeEvent()` to allow you to modify the state of your map or invoke other Shiny actions on user scroll.

Usage

```
on_section(map_id, section_id, handler)
```

Arguments

<code>map_id</code>	The ID of your map output
<code>section_id</code>	The ID of the section to trigger on, defined in <code>story_section()</code>
<code>handler</code>	Expression to execute when section becomes visible.

openfreemap_style	<i>Get OpenFreeMap Style URL</i>
-------------------	----------------------------------

Description

Get OpenFreeMap Style URL

Usage

```
openfreemap_style(style_name)
```

Arguments

<code>style_name</code>	The name of the style (e.g., "bright", "positron", "liberty", "dark", or "fiord").
-------------------------	--

Value

The style URL corresponding to the given style name.

palette_to_lut	<i>Convert R color palette to mapgl LUT</i>
----------------	---

Description

This function takes an R color palette and converts it into a base64-encoded LUT (Look-Up Table) image that can be used with Mapbox GL JS v3+ for custom map themes. The LUT applies color transformations to the basemap.

Usage

```
palette_to_lut(  
  colors,  
  n = 5,  
  method = c("tint", "replace", "duotone", "tritone", "luminosity"),  
  intensity = 0.5,  
  lut_size = 32,  
  reverse = FALSE  
)
```

Arguments

colors	Character vector of colors (hex or R color names) or a function that generates colors (like viridis)
n	Number of colors to sample from the palette (if colors is a function)
method	Method for applying colors to the LUT: • "tint": Applies palette as a color tint/overlay • "replace": Maps grayscale values to palette colors • "duotone": Creates duotone effect with first two colors • "tritone": Creates tritone effect with first three colors • "luminosity": Applies palette based on pixel luminosity
intensity	Strength of the effect (0-1)
lut_size	Size of the LUT (16, 32, or 64)
reverse	Logical; whether to reverse the color palette

Value

Base64-encoded PNG data URI string

Examples

```
## Not run:  
# Using viridis palette  
theme_data <- palette_to_lut(viridisLite::viridis(5))
```

```
# Using a palette function directly
theme_data <- palette_to_lut(viridisLite::plasma, n = 7)

# Using RColorBrewer
theme_data <- palette_to_lut(RColorBrewer::brewer.pal(9, "YlOrRd"))

# Use in mapboxgl (requires Mapbox GL JS v3+)
mapboxgl(
  center = c(139.7, 35.7),
  zoom = 10,
  config = list(
    basemap = list(
      theme = "custom",
      "theme-data" = theme_data
    )
  )
)

## End(Not run)
```

print_map

Render a map as a static image

Description

Renders a mapgl map as a static PNG image for display. When called inside a knitr/Quarto document, the map is included as a static figure via `knitr::include_graphics()`. In an interactive session, the image is displayed in the R graphics device.

Usage

```
print_map(
  map,
  width = 900,
  height = 500,
  include_legend = TRUE,
  hide_controls = TRUE,
  include_scale_bar = TRUE,
  basemap_color = NULL,
  image_scale = 1,
  background = "white",
  delay = NULL
)
```

Arguments

map	A map object created by <code>mapboxgl()</code> or <code>maplibre()</code> .
width	Integer. The width of the map viewport in pixels. Always overrides any width configured when the map widget was created.

height	Integer. The height of the map viewport in pixels. Always overrides any height configured when the map widget was created.
include_legend	Logical. Include the legend in the output? Default TRUE.
hide_controls	Logical. Hide navigation and other interactive controls? Default TRUE.
include_scale_bar	Logical. Include the scale bar? Default TRUE.
basemap_color	Character string or NULL. If specified, basemap tiles are removed and replaced with this background color (e.g., "white", "lightgrey", "#f0f0f0"). Use "transparent" for no background. Default NULL (keep basemap).
image_scale	Numeric. Scale factor for the output image. Use 2 for retina/HiDPI output. Default 1.
background	Character string or NULL. Background color for the output image. Default "white". Set to NULL for a transparent background. Ignored when basemap_color is set (basemap_color controls the background in that case).
delay	Numeric or NULL. Additional delay in seconds to wait after the map reports idle, before capturing. Useful for maps with complex rendering. Default NULL (no extra delay).

Value

In a knitr context, the result of `knitr::include_graphics()`. In an interactive session, the image is displayed and the temporary file path is returned invisibly.

Examples

```
## Not run:
library(mapgl)

map <- maplibre(
  center = c(-96, 37.8),
  zoom = 3
)

# In a Quarto document chunk
print_map(map)

# With custom dimensions
print_map(map, width = 1200, height = 800, image_scale = 2)

## End(Not run)
```

Description

This function queries features that are currently rendered (visible) in the map viewport. Only features within the current viewport bounds will be returned - features outside the visible area or hidden due to zoom constraints will not be included. Use `get_queried_features()` to retrieve the results as an sf object, or use the `callback` parameter to handle results automatically when they're ready.

Usage

```
query_rendered_features(
  proxy,
  geometry = NULL,
  layer_id = NULL,
  filter = NULL,
  callback = NULL
)
```

Arguments

<code>proxy</code>	A MapboxGL or Maplibre proxy object, defined with <code>mapboxgl_proxy()</code> , <code>maplibre_proxy()</code> , <code>mapboxgl_compare_proxy()</code> , or <code>maplibre_compare_proxy()</code>
<code>geometry</code>	The geometry to query. Can be: • <code>NULL</code> (default): Query the entire viewport • A length-2 vector <code>c(x, y)</code>: Query at a single point in pixel coordinates • A length-4 vector <code>c(xmin, ymin, xmax, ymax)</code>: Query within a bounding box in pixel coordinates
<code>layer_id</code>	A character vector of layer names to include in the query. Can be a single layer name or multiple layer names. If <code>NULL</code> (default), all layers are queried.
<code>filter</code>	A filter expression used to filter features in the query. Should be a list representing a Mapbox GL expression. Using this parameter applies the filter during the query WITHOUT changing the map display, avoiding race conditions. If you've called <code>set_filter()</code> separately, you must pass the same filter here to get aligned results.
<code>callback</code>	A function to execute when results are ready. The function will receive the sf object as its argument. If provided, this avoids timing issues by automatically handling results when they're available.

Details**Viewport Limitation:**

This function only queries features that are currently rendered in the map viewport. Features outside the visible area will not be returned, even if they exist in the data source. This includes features that are:

- Outside the current map bounds
- Hidden due to zoom level constraints (minzoom/maxzoom)
- Not yet loaded (if using vector tiles)

Avoiding Race Conditions:

IMPORTANT: `set_filter()` is asynchronous while `query_rendered_features()` is synchronous. Calling `query_rendered_features()` immediately after `set_filter()` will return features from the PREVIOUS filter state, not the new one.

Safe Usage Patterns::

Pattern 1: Query First, Then Filter (Recommended)

```
query_rendered_features(proxy, layer_id = "counties", callback = function(features) {
  # Process features, then update map based on results
  proxy |> set_filter("highlight", list("in", "id", features$id))
})
```

Pattern 2: Use Filter Parameter Instead

```
# Query with filter without changing map display
query_rendered_features(proxy, filter = list(">=", "population", 1000),
  callback = function(features) {
  # Process filtered results without race condition
})
```

What NOT to Do::

```
# WRONG - This will return stale results!
proxy |> set_filter("layer", new_filter)
query_rendered_features(proxy, layer_id = "layer") # Gets OLD filter results
```

Value

The proxy object (invisibly). Use `get_queried_features()` to retrieve the query results manually, or provide a callback function to handle results automatically.

Examples

```
## Not run:
# Pattern 1: Query first, then filter (RECOMMENDED)
proxy <- maplibre_proxy("map")
query_rendered_features(proxy, layer_id = "counties", callback = function(features) {
  if (nrow(features) > 0) {
    # Filter map based on query results - no race condition
    proxy |> set_filter("selected", list("in", "id", features$id))
  }
})

# Pattern 2: Use filter parameter to avoid race conditions
query_rendered_features(proxy,
  filter = list(">=", "population", 50000),
  callback = function(features) {
    # These results are guaranteed to match the filter
    print(paste("Found", nrow(features), "high population areas"))
  })

# Query specific bounding box with callback
query_rendered_features(proxy, geometry = c(100, 100, 200, 200),
  layer_id = "counties", callback = function(features) {
```

```

    print(paste("Found", nrow(features), "features"))
  })

  # ANTI-PATTERN - Don't do this!
  # proxy |> set_filter("layer", new_filter)
  # query_rendered_features(proxy, layer_id = "layer") # Will get stale results!

## End(Not run)

```

renderMapboxgl	<i>Render a Mapbox GL output element in Shiny</i>
----------------	---

Description

Render a Mapbox GL output element in Shiny

Usage

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

Arguments

expr	An expression that generates a Mapbox GL map
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression

Value

A rendered Mapbox GL map for use in a Shiny server

renderMapboxglCompare	<i>Render a Mapbox GL Compare output element in Shiny</i>
-----------------------	---

Description

Render a Mapbox GL Compare output element in Shiny

Usage

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

Arguments

expr	An expression that generates a Mapbox GL Compare map
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression

Value

A rendered Mapbox GL Compare map for use in a Shiny server

renderMaplibre	<i>Render a Maplibre GL output element in Shiny</i>
----------------	---

Description

Render a Maplibre GL output element in Shiny

Usage

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

Arguments

expr	An expression that generates a Maplibre GL map
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression

Value

A rendered Maplibre GL map for use in a Shiny server

renderMaplibreCompare	<i>Render a Maplibre GL Compare output element in Shiny</i>
-----------------------	---

Description

Render a Maplibre GL Compare output element in Shiny

Usage

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

Arguments

expr	An expression that generates a Maplibre GL Compare map
env	The environment in which to evaluate expr
quoted	Is expr a quoted expression

Value

A rendered Maplibre GL Compare map for use in a Shiny server

save_map

*Save a map as a static PNG image***Description**

Renders a mapgl map widget to a static PNG file using headless Chrome via the chromote package. Uses the same html2canvas-based screenshot infrastructure as [add_screenshot_control\(\)](#).

Usage

```
save_map(
  map,
  filename = "map.png",
  width = 900,
  height = 500,
  include_legend = TRUE,
  hide_controls = TRUE,
  include_scale_bar = TRUE,
  basemap_color = NULL,
  image_scale = 1,
  background = "white",
  delay = NULL
)
```

Arguments

map	A map object created by mapboxgl() or maplibre() .
filename	Character string. The output file path. Defaults to "map.png". If the filename does not end in .png, the extension is appended automatically.
width	Integer. The width of the map viewport in pixels. Always overrides any width configured when the map widget was created.
height	Integer. The height of the map viewport in pixels. Always overrides any height configured when the map widget was created.
include_legend	Logical. Include the legend in the output? Default TRUE.
hide_controls	Logical. Hide navigation and other interactive controls? Default TRUE.
include_scale_bar	Logical. Include the scale bar? Default TRUE.
basemap_color	Character string or NULL. If specified, basemap tiles are removed and replaced with this background color (e.g., "white", "lightgrey", "#f0f0f0"). Use "transparent" for no background. Default NULL (keep basemap).
image_scale	Numeric. Scale factor for the output image. Use 2 for retina/HiDPI output. Default 1.
background	Character string or NULL. Background color for the output image. Default "white". Set to NULL for a transparent background. Ignored when basemap_color is set (basemap_color controls the background in that case).

delay Numeric or NULL. Additional delay in seconds to wait after the map reports idle, before capturing. Useful for maps with complex rendering. Default NULL (no extra delay).

Details

This function requires the **chromote** and **httpuv** packages. Install them with `install.packages(c("chromote", "httpuv"))`. **chromote** also requires a Chrome or Chromium browser installation.

The function works by:

1. Saving the map widget to a temporary HTML file
2. Opening it in headless Chrome
3. Waiting for all map tiles and styles to load
4. Using `html2canvas` to capture the rendered map (including legends, attribution, and optionally the scale bar)
5. Decoding the captured image and writing it to the output file

Value

The output file path, invisibly.

Examples

```
## Not run:
library(mapgl)

map <- maplibre(
  center = c(-96, 37.8),
  zoom = 3
)

save_map(map, "us_map.png")
save_map(map, "us_map_retina.png", image_scale = 2)

# Remove basemap, keep only data layers on white
save_map(map, "data_only.png", basemap_color = "white")

## End(Not run)
```

`set_config_property` *Set a configuration property for a Mapbox GL map*

Description

Set a configuration property for a Mapbox GL map

Usage

```
set_config_property(map, import_id, config_name, value)
```

Arguments

map	A map object created by the <code>mapboxgl</code> function or a proxy object defined with <code>mapboxgl_proxy()</code> .
import_id	The name of the imported style to set the config for (e.g., 'basemap').
config_name	The name of the configuration property from the style.
value	The value to set for the configuration property.

Value

The updated map object with the configuration property set.

set_filter	<i>Set a filter on a map layer</i>
------------	------------------------------------

Description

This function sets a filter on a map layer, working with both regular map objects and proxy objects.

Usage

```
set_filter(map, layer_id, filter)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function, or a proxy object.
layer_id	The ID of the layer to which the filter will be applied.
filter	The filter expression to apply.

Value

The updated map object.

Clustered layers

A layer created via the `cluster_options` shortcut in `add_circle_layer()` or `add_symbol_layer()` is actually three layers over one source ("id", "id-clusters", "id-cluster-count"). `set_filter()` targets exactly one of them, so:

- Calling `set_filter("id", ...)` applies only to the unclustered sub-layer. Cluster circles still show the pre-filter counts.

- Cluster points are synthetic (their only properties are `point_count`, `cluster_id`, etc.), so a filter that reads a feature property like "year" cannot be meaningfully applied to the `-clusters` layer — it would evaluate to `FALSE` and hide all clusters.

For the common "filter my data" case on a clustered map, use `set_source()` instead. It replaces the source's data and Mapbox/MapLibre re-cluster automatically:

```
mapboxgl_proxy("map") |>
  set_source(layer_id = "circles", source = filtered())
```

`set_filter()` is still the right tool for cluster-aware filters that read cluster-point properties, e.g. hiding clusters below a count threshold:

```
mapboxgl_proxy("map") |>
  set_filter("circles-clusters", list(">=", get_column("point_count"), 10))
```

set_flowmap_filter	<i>Update flowmap filter</i>
--------------------	------------------------------

Description

Updates the filter state of a flowmap layer, including selected locations and time range.

Usage

```
set_flowmap_filter(
  proxy,
  id,
  selected_locations = NULL,
  location_filter_mode = NULL,
  selected_time_range = NULL
)
```

Arguments

proxy	A map proxy object.
id	The ID of the flowmap layer to update.
selected_locations	Optional vector of location IDs to select.
location_filter_mode	Optional location filter mode: "ALL", "INCOMING", "OUTGOING", or "BETWEEN".
selected_time_range	Optional vector of two dates for time filtering.

Value

The modified map proxy.

set_flowmap_settings *Update a flowmap setting*

Description

Updates one setting of a flowmap layer.

Usage

```
set_flowmap_settings(map, id, name, value)
```

Arguments

map	A map object created by mapboxgl() or maplibre() , or a proxy object created by mapboxgl_proxy() or maplibre_proxy() .
id	The ID of the flowmap layer to update.
name	The setting name to update. Supported canonical FlowMapGL setting names are opacity, colorScheme, darkMode, fadeAmount, highlightColor, locationsEnabled, locationTotalsEnabled, locationLabelsEnabled, flowLinesRenderingMode, flowLineThicknessScale, flowLineCurviness, clusteringEnabled, clusteringAuto, clusteringLevel, fadeEnabled, fadeOpacityEnabled, adaptiveScalesEnabled, temporalScaleDomain, maxTopFlowsDisplayNum, and flowEndpointsInViewportMode. Snake-case aliases such as color_scheme, temporal_scale_domain, and max_top_flows_display_num are accepted and normalized internally. Filter state (selectedTimeRange, selectedLocations, and locationFilterMode) must be updated with set_flowmap_filter() .
value	The setting value.

Details

colorScheme accepts the same values as flow_color_scheme in [add_flowmap\(\)](#): a FlowMapGL preset name, a character vector of at least two CSS colors, or a mapgl_continuous_scale object from [interpolate_palette\(\)](#). opacity must be between 0 and 1. fadeAmount must be between 0 and 100. maxTopFlowsDisplayNum must be positive. clusteringLevel must be numeric or NULL. flowLinesRenderingMode must be "straight", "animated-straight", or "curved". temporalScaleDomain must be "selected" or "all". flowEndpointsInViewportMode must be "any" or "both". Boolean settings must be scalar TRUE or FALSE.

Value

The modified map object.

set_fog	<i>Set fog on a Mapbox GL map</i>
---------	-----------------------------------

Description

Set fog on a Mapbox GL map

Usage

```
set_fog(  
  map,  
  range = NULL,  
  color = NULL,  
  horizon_blend = NULL,  
  high_color = NULL,  
  space_color = NULL,  
  star_intensity = NULL  
)
```

Arguments

map	A map object created by the mapboxgl function or a proxy object.
range	A numeric vector of length 2 defining the minimum and maximum range of the fog.
color	A string specifying the color of the fog.
horizon_blend	A number between 0 and 1 controlling the blending of the fog at the horizon.
high_color	A string specifying the color of the fog at higher elevations.
space_color	A string specifying the color of the fog in space.
star_intensity	A number between 0 and 1 controlling the intensity of the stars in the fog.

Value

The updated map object.

set_layout_property	<i>Set a layout property on a map layer</i>
---------------------	---

Description

Set a layout property on a map layer

Usage

```
set_layout_property(map, layer_id = NULL, name, value, layer = NULL)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a proxy object.
layer_id	The ID of the layer to update.
name	The name of the layout property to set.
value	The value to set the property to.
layer	Deprecated. Use layer_id instead.

Value

The updated map object.

set_paint_property	<i>Set a paint property on a map layer</i>
--------------------	--

Description

Set a paint property on a map layer

Usage

```
set_paint_property(map, layer_id = NULL, name, value, layer = NULL)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a proxy object.
layer_id	The ID of the layer to update.
name	The name of the paint property to set.
value	The value to set the property to.
layer	Deprecated. Use layer_id instead.

Value

The updated map object.

set_popup	<i>Set popup on a map layer</i>
-----------	---------------------------------

Description

Set popup on a map layer

Usage

```
set_popup(map, layer_id = NULL, popup, layer = NULL, style = NULL)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a proxy object.
layer_id	The ID of the layer to update.
popup	Popup content: a column name, a {brace} template, or a concat()/number_format() expression.
layer	Deprecated. Use layer_id instead.
style	Optional popup appearance: a preset string ("light" or "dark") or a tooltip_style()/popup_style() object.

Value

The updated map object.

set_projection	<i>Set Projection for a Mapbox/Maplibre Map</i>
----------------	---

Description

This function sets the projection dynamically after map initialization.

Usage

```
set_projection(map, projection)
```

Arguments

map	A map object created by mapboxgl() or maplibre() functions, or their respective proxy objects
projection	A string representing the projection name (e.g., "mercator", "globe", "albers", "equalEarth", etc.)

Value

The modified map object

set_rain

*Set rain effect on a Mapbox GL map***Description**

Set rain effect on a Mapbox GL map

Usage

```
set_rain(
  map,
  density = 0.5,
  intensity = 1,
  color = "#a8adbc",
  opacity = 0.7,
  center_thinning = 0.57,
  direction = c(0, 80),
  droplet_size = c(2.6, 18.2),
  distortion_strength = 0.7,
  vignette = 1,
  vignette_color = "#464646",
  remove = FALSE
)
```

Arguments

map	A map object created by the <code>mapboxgl</code> function or a proxy object.
density	A number between 0 and 1 controlling the rain particles density. Default is 0.5.
intensity	A number between 0 and 1 controlling the rain particles movement speed. Default is 1.
color	A string specifying the color of the rain droplets. Default is "#a8adbc".
opacity	A number between 0 and 1 controlling the rain particles opacity. Default is 0.7.
center_thinning	A number between 0 and 1 controlling the thinning factor of rain particles from center. Default is 0.57.
direction	A numeric vector of length 2 defining the azimuth and polar angles of the rain direction. Default is <code>c(0, 80)</code> .
droplet_size	A numeric vector of length 2 controlling the rain droplet size (x - normal to direction, y - along direction). Default is <code>c(2.6, 18.2)</code> .
distortion_strength	A number between 0 and 1 controlling the rain particles screen-space distortion strength. Default is 0.7.
vignette	A number between 0 and 1 controlling the screen-space vignette rain tinting effect intensity. Default is 1.0.

vignette_color A string specifying the rain vignette screen-space corners tint color. Default is "#464646".

remove A logical value indicating whether to remove the rain effect. Default is FALSE.

Value

The updated map object.

Examples

```
## Not run:
# Add rain effect with default values
mapboxgl(...) |> set_rain()

# Add rain effect with custom values
mapboxgl(
  style = mapbox_style("standard"),
  center = c(24.951528, 60.169573),
  zoom = 16.8,
  pitch = 74,
  bearing = 12.8
) |>
  set_rain(
    density = 0.5,
    opacity = 0.7,
    color = "#a8adbc"
  )

# Remove rain effect (useful in Shiny)
map_proxy |> set_rain(remove = TRUE)

## End(Not run)
```

set_snow

Set snow effect on a Mapbox GL map

Description

Set snow effect on a Mapbox GL map

Usage

```
set_snow(
  map,
  density = 0.85,
  intensity = 1,
  color = "#ffffff",
  opacity = 1,
  center_thinning = 0.4,
```

```

    direction = c(0, 50),
    flake_size = 0.71,
    vignette = 0.3,
    vignette_color = "#ffffff",
    remove = FALSE
  )

```

Arguments

map	A map object created by the <code>mapboxgl</code> function or a proxy object.
density	A number between 0 and 1 controlling the snow particles density. Default is 0.85.
intensity	A number between 0 and 1 controlling the snow particles movement speed. Default is 1.0.
color	A string specifying the color of the snow particles. Default is "#ffffff".
opacity	A number between 0 and 1 controlling the snow particles opacity. Default is 1.0.
center_thinning	A number between 0 and 1 controlling the thinning factor of snow particles from center. Default is 0.4.
direction	A numeric vector of length 2 defining the azimuth and polar angles of the snow direction. Default is <code>c(0, 50)</code> .
flake_size	A number between 0 and 5 controlling the snow flake particle size. Default is 0.71.
vignette	A number between 0 and 1 controlling the snow vignette screen-space effect. Default is 0.3.
vignette_color	A string specifying the snow vignette screen-space corners tint color. Default is "#ffffff".
remove	A logical value indicating whether to remove the snow effect. Default is FALSE.

Value

The updated map object.

Examples

```

## Not run:
# Add snow effect with default values
mapboxgl(...) |> set_snow()

# Add snow effect with custom values
mapboxgl(
  style = mapbox_style("standard"),
  center = c(24.951528, 60.169573),
  zoom = 16.8,
  pitch = 74,
  bearing = 12.8
) |>

```

```
    set_snow(  
      density = 0.85,  
      flake_size = 0.71,  
      color = "ffffff"  
    )  
  
    # Remove snow effect (useful in Shiny)  
    map_proxy |> set_snow(remove = TRUE)  
  
    ## End(Not run)
```

set_source	<i>Set source of a map layer</i>
------------	----------------------------------

Description

Set source of a map layer

Usage

```
set_source(map, layer_id = NULL, source, layer = NULL)
```

Arguments

map	A map object created by the <code>mapboxgl</code> or <code>maplibre</code> function, or a proxy object.
layer_id	The ID of the layer to update.
source	An sf object (which will be converted to a GeoJSON source).
layer	Deprecated. Use <code>layer_id</code> instead.

Value

The updated map object.

set_style	<i>Update the style of a map</i>
-----------	----------------------------------

Description

Update the style of a map

Usage

```
set_style(map, style, config = NULL, diff = TRUE, preserve_layers = TRUE)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a proxy object.
style	The new style URL to be applied to the map.
config	A named list of options to be passed to the style config.
diff	A boolean that attempts a diff-based update rather than re-drawing the full style. Not available for all styles.
preserve_layers	A boolean that indicates whether to preserve user-added sources and layers when changing styles. Defaults to TRUE.

Value

The modified map object.

Examples

```
## Not run:
map <- mapboxgl(
  style = mapbox_style("streets"),
  center = c(-74.006, 40.7128),
  zoom = 10,
  access_token = "your_mapbox_access_token"
)

# Update the map style in a Shiny app
observeEvent(input$change_style, {
  mapboxgl_proxy("map", session) %>%
    set_style(mapbox_style("dark"), config = list(showLabels = FALSE), diff = TRUE)
})

## End(Not run)
```

set_terrain	<i>Set terrain properties on a map</i>
-------------	--

Description

Set terrain properties on a map

Usage

```
set_terrain(map, source, exaggeration = 1)
```

Arguments

map	A map object created by the mapboxgl or maplibre functions.
source	The ID of the raster DEM source.
exaggeration	The terrain exaggeration factor.

Value

The modified map object with the terrain settings applied.

Examples

```
## Not run:
library(mapgl)

mapboxgl(
  style = mapbox_style("standard-satellite"),
  center = c(-114.26608, 32.7213),
  zoom = 14,
  pitch = 80,
  bearing = 41
) |>
add_raster_dem_source(
  id = "mapbox-dem",
  url = "mapbox://mapbox.mapbox-terrain-dem-v1",
  tileSize = 512,
  maxzoom = 14
) |>
set_terrain(
  source = "mapbox-dem",
  exaggeration = 1.5
)

## End(Not run)
```

set_tooltip

Set tooltip on a map layer

Description

Set tooltip on a map layer

Usage

```
set_tooltip(map, layer_id = NULL, tooltip, layer = NULL, style = NULL)
```

Arguments

map	A map object created by the mapboxgl or maplibre function, or a proxy object.
layer_id	The ID of the layer to update.
tooltip	Tooltip content: a column name, a {brace} template, or a concat()/number_format() expression.
layer	Deprecated. Use layer_id instead.
style	Optional tooltip appearance: a preset string ("light" or "dark") or a tooltip_style() object.

Value

The updated map object.

set_view	<i>Set the map center and zoom level</i>
----------	--

Description

Set the map center and zoom level

Usage

```
set_view(map, center, zoom)
```

Arguments

map	A map object created by the mapboxgl or maplibre function or a proxy object.
center	A numeric vector of length 2 specifying the center of the map (longitude, latitude).
zoom	The zoom level.

Value

The updated map object.

slider_style	<i>Style a slider control</i>
--------------	-------------------------------

Description

Builds an appearance specification for [add_slider_control\(\)](#). Use a preset string ("light", "dark", or "auto") as a starting point and override individual pieces of the control on top of it. The older background_color, text_color, accent_color, and width arguments on [add_slider_control\(\)](#) continue to work for simple styling; use slider_style() when you need control over the container, play button, track, thumb, and histogram.

Usage

```
slider_style(  
  preset = NULL,  
  width = NULL,  
  background_color = NULL,  
  background_opacity = NULL,  
  text_color = NULL,  
  border_color = NULL,  
  border_width = NULL,  
  border_radius = NULL,  
  font_family = NULL,  
  font_size = NULL,  
  font_weight = NULL,  
  padding = NULL,  
  shadow = NULL,  
  shadow_color = NULL,  
  shadow_size = NULL,  
  title_color = NULL,  
  title_size = NULL,  
  title_weight = NULL,  
  value_color = NULL,  
  value_size = NULL,  
  value_weight = NULL,  
  accent_color = NULL,  
  play_button_background = NULL,  
  play_button_color = NULL,  
  play_button_border_color = NULL,  
  track_color = NULL,  
  active_color = NULL,  
  track_height = NULL,  
  thumb_color = NULL,  
  thumb_border_color = NULL,  
  thumb_size = NULL,  
  histogram_height = NULL,  
  histogram_bar_color = NULL,  
  histogram_active_opacity = NULL,  
  histogram_inactive_opacity = NULL  
)
```

Arguments

preset	Optional base theme: "light", "dark", or "auto". "auto" currently resolves to the light preset.
width	Slider container width in pixels.
background_color	CSS color for the control background.
background_opacity	Numeric in [0, 1] for background opacity.

text_color	CSS color for general text.
border_color	CSS color for the container border.
border_width	Border width in pixels.
border_radius	Corner radius in pixels.
font_family	CSS font-family string.
font_size	Font size in pixels.
font_weight	CSS font-weight.
padding	Internal padding as a CSS size string or numeric pixels.
shadow	Logical; whether to draw a drop shadow.
shadow_color	CSS color of the drop shadow.
shadow_size	Blur radius of the drop shadow in pixels.
title_color, title_size, title_weight	Styling for the optional title.
value_color, value_size, value_weight	Styling for the current value label.
accent_color	Main accent color. Used as the default active track, thumb, play hover, and histogram color.
play_button_background, play_button_color, play_button_border_color	Styling for the play/pause button.
track_color, active_color	CSS colors for the inactive and active track.
track_height	Track height in pixels.
thumb_color, thumb_border_color	CSS colors for the slider thumb.
thumb_size	Thumb diameter in pixels.
histogram_height	Histogram height in pixels.
histogram_bar_color	Histogram bar color.
histogram_active_opacity, histogram_inactive_opacity	Bar opacity for selected and unselected histogram bins.

Value

A list of class `mapgl_slider_style`.

Examples

```
## Not run:
slider_style("dark", accent_color = "#9fd3ff")

slider_style(
  background_color = "rgba(20, 30, 48, 0.92)",
  text_color = "#f8fafc",
```

```
    active_color = "#7dd3fc",  
    thumb_size = 18  
)  
  
## End(Not run)
```

step_classification	<i>Step expressions with automatic classification</i>
---------------------	---

Description

These functions create step expressions using different classification methods, similar to choropleth mapping in GIS software. They automatically calculate break points and generate appropriate step expressions for styling map layers.

Usage

```
step_equal_interval(  
  data = NULL,  
  column,  
  data_values = NULL,  
  n = 5,  
  palette = NULL,  
  colors = NULL,  
  na_color = "grey"  
)
```

```
step_quantile(  
  data = NULL,  
  column,  
  data_values = NULL,  
  n = 5,  
  palette = NULL,  
  colors = NULL,  
  na_color = "grey"  
)
```

```
step_jenks(  
  data = NULL,  
  column,  
  data_values = NULL,  
  n = 5,  
  palette = NULL,  
  colors = NULL,  
  na_color = "grey"  
)
```

Arguments

<code>data</code>	A data frame or sf object containing the data. If provided, <code>data_values</code> will be extracted from <code>data[[column]]</code> . Either <code>data</code> or <code>data_values</code> must be provided.
<code>column</code>	The name of the column to use for the step expression.
<code>data_values</code>	A numeric vector of the actual data values used to calculate breaks. If NULL and <code>data</code> is provided, will be extracted from <code>data[[column]]</code> .
<code>n</code>	The number of classes/intervals to create. Defaults to 5.
<code>palette</code>	A function that takes <code>n</code> and returns a character vector of colors. If NULL and <code>colors</code> is also NULL, defaults to <code>viridisLite::viridis</code> .
<code>colors</code>	A character vector of colors to use. Must have exactly <code>n</code> colors for step classification functions. Either <code>palette</code> or <code>colors</code> should be provided, but not both.
<code>na_color</code>	The color to use for missing values. Defaults to "grey".

Details

step_equal_interval() Creates equal interval breaks by dividing the data range into equal parts

step_quantile() Creates quantile breaks ensuring approximately equal numbers of observations in each class

step_jenks() Creates Jenks natural breaks using Fisher-Jenks optimization to minimize within-class variance

Value

A list of class "mapgl_classification" containing the step expression and metadata.

See Also

[interpolate_palette\(\)](#) for continuous color scales

Examples

```
## Not run:
# Texas county income data
library(tidycensus)
tx <- get_acs(geography = "county", variables = "B19013_001",
             state = "TX", geometry = TRUE)

# Using palette function (recommended)
eq_class <- step_equal_interval(data = tx, column = "estimate", n = 5,
                              palette = viridisLite::plasma)

# Or with piping
eq_class <- tx |> step_equal_interval("estimate", n = 5)

# Using specific colors
qt_class <- step_quantile(data = tx, column = "estimate", n = 3,
                        colors = c("red", "yellow", "blue"))

# Jenks natural breaks with default viridis
```

```

jk_class <- step_jenks(data = tx, column = "estimate", n = 5)

# Use in a map with formatted legend
maplibre() |>
  add_fill_layer(source = tx, fill_color = eq_class$expression) |>
  add_legend(
    legend_title = "Median Income",
    values = get_legend_labels(eq_class, format = "currency"),
    colors = get_legend_colors(eq_class),
    type = "categorical"
  )

# Compare different methods
print(eq_class, format = "currency")
print(qt_class, format = "compact", prefix = "$")

## End(Not run)

```

step_expr

Create a step expression

Description

This function generates a step expression that can be used in your styles.

Usage

```
step_expr(column = NULL, property = NULL, base, values, stops, na_color = NULL)
```

Arguments

column	The name of the column to use for the step expression. If specified, property should be NULL.
property	The name of the property to use for the step expression. If specified, column should be NULL.
base	The base value to use for the step expression.
values	A numeric vector of values at which steps occur.
stops	A vector of corresponding stops (colors, sizes, etc.) for the steps.
na_color	The color to use for missing values. Mapbox GL JS defaults to black if this is not supplied.

Value

A list representing the step expression.

Examples

```
step_expr(
  column = "value",
  base = "#ffffff",
  values = c(1000, 5000, 10000),
  stops = c("#ff0000", "#00ff00", "#0000ff")
)
```

story_leaflet

Create a scrollytelling story map with Leaflet

Description

Create a scrollytelling story map with Leaflet

Usage

```
story_leaflet(
  map_id,
  sections,
  root_margin = "-20% 0px -20% 0px",
  threshold = 0,
  styles = NULL,
  bg_color = "rgba(255,255,255,0.9)",
  text_color = "#34495e",
  font_family = NULL
)
```

Arguments

map_id	The ID of your mapboxgl, maplibre, or leaflet output defined in the server, e.g. "map"
sections	A named list of story_section objects. Names will correspond to map events defined within the server using on_section().
root_margin	The margin around the viewport for triggering sections by the intersection observer. Should be specified as a string, e.g. "-20% 0px -20% 0px".
threshold	A number that indicates the visibility ratio for a story ' panel to be used to trigger a section; should be a number between 0 and 1. Defaults to 0, meaning that the section is triggered as soon as the first pixel is visible.
styles	Optional custom CSS styles. Should be specified as a character string within shiny::tags\$style().
bg_color	Default background color for all sections
text_color	Default text color for all sections
font_family	Default font family for all sections

story_map

Create a scrollytelling story map

Description

Create a scrollytelling story map

Usage

```
story_map(
  map_id,
  sections,
  map_type = c("mapboxgl", "maplibre", "leaflet"),
  root_margin = "-20% 0px -20% 0px",
  threshold = 0,
  styles = NULL,
  bg_color = "rgba(255,255,255,0.9)",
  text_color = "#34495e",
  font_family = NULL
)
```

Arguments

map_id	The ID of your mapboxgl, maplibre, or leaflet output defined in the server, e.g. "map"
sections	A named list of story_section objects. Names will correspond to map events defined within the server using on_section().
map_type	One of "mapboxgl", "maplibre", or "leaflet". This will use either mapboxglOutput(), maplibreOutput(), or leafletOutput() respectively, and must correspond to the appropriate render*() function used in the server.
root_margin	The margin around the viewport for triggering sections by the intersection observer. Should be specified as a string, e.g. "-20% 0px -20% 0px".
threshold	A number that indicates the visibility ratio for a story ' panel to be used to trigger a section; should be a number between 0 and 1. Defaults to 0, meaning that the section is triggered as soon as the first pixel is visible.
styles	Optional custom CSS styles. Should be specified as a character string within shiny::tags\$style().
bg_color	Default background color for all sections
text_color	Default text color for all sections
font_family	Default font family for all sections

story_maplibre

Create a scrollytelling story map with MapLibre

Description

Create a scrollytelling story map with MapLibre

Usage

```
story_maplibre(
  map_id,
  sections,
  root_margin = "-20% 0px -20% 0px",
  threshold = 0,
  styles = NULL,
  bg_color = "rgba(255,255,255,0.9)",
  text_color = "#34495e",
  font_family = NULL
)
```

Arguments

map_id	The ID of your mapboxgl, maplibre, or leaflet output defined in the server, e.g. "map"
sections	A named list of story_section objects. Names will correspond to map events defined within the server using on_section().
root_margin	The margin around the viewport for triggering sections by the intersection observer. Should be specified as a string, e.g. "-20% 0px -20% 0px".
threshold	A number that indicates the visibility ratio for a story ' panel to be used to trigger a section; should be a number between 0 and 1. Defaults to 0, meaning that the section is triggered as soon as the first pixel is visible.
styles	Optional custom CSS styles. Should be specified as a character string within shiny::tags\$style().
bg_color	Default background color for all sections
text_color	Default text color for all sections
font_family	Default font family for all sections

story_section	Create a story section for story maps
---------------	---------------------------------------

Description

Create a story section for story maps

Usage

```
story_section(  
  title,  
  content,  
  position = c("left", "center", "right"),  
  width = 400,  
  bg_color = NULL,  
  text_color = NULL,  
  font_family = NULL  
)
```

Arguments

title	Section title
content	Section content - can be text, HTML, or Shiny outputs
position	Position of text block ("left", "center", "right")
width	Width of text block in pixels (default: 400)
bg_color	Background color (with alpha) for text block
text_color	Text color
font_family	Font family for the section

terradraw_options	Configure the Terra Draw provider for the draw control
-------------------	--

Description

This helper builds the options argument for `add_draw_control()` with `provider = "terra-draw"`. All arguments are optional; the defaults give a fully editable select mode without snapping.

Usage

```
terrdraw_options(
  drag_features = TRUE,
  rotate_features = FALSE,
  scale_features = FALSE,
  drag_vertices = TRUE,
  delete_vertices = TRUE,
  midpoints = TRUE,
  resize = NULL,
  snap_to_coordinates = FALSE,
  snap_to_lines = FALSE,
  editable_while_drawing = FALSE,
  show_coordinate_points = FALSE,
  drag_interaction = NULL,
  pointer_distance = NULL,
  keep_mode_active = FALSE,
  modes = NULL
)
```

Arguments

<code>drag_features</code>	Logical, whether selected features can be moved by dragging. Default is TRUE.
<code>rotate_features</code>	Logical, whether selected features can be rotated by holding Control+R while dragging the feature in select mode. This uses the Control key on macOS as well as Windows and Linux. Default is FALSE.
<code>scale_features</code>	Logical, whether selected features can be scaled by holding Control+S while dragging the feature in select mode. This uses the Control key on macOS as well as Windows and Linux. Default is FALSE.
<code>drag_vertices</code>	Logical, whether individual vertices of a selected feature can be dragged. Default is TRUE.
<code>delete_vertices</code>	Logical, whether individual vertices of a selected feature can be deleted. Default is TRUE.
<code>midpoints</code>	Logical, whether midpoints are shown between vertices so new vertices can be inserted. Default is TRUE. Ignored (treated as FALSE) when <code>resize</code> is set, as Terra Draw does not support both.
<code>resize</code>	How selected features may be resized by dragging their selection points: one of "center", "opposite", "center-fixed", or "opposite-fixed". Default NULL disables resizing.
<code>snap_to_coordinates</code>	Logical, whether drawing and editing snap to existing feature coordinates. Default is FALSE.
<code>snap_to_lines</code>	Logical, whether drawing and editing snap to existing feature lines. Default is FALSE.

editable_while_drawing	Logical, whether features can be edited (vertices dragged) while still being drawn, for the point, linestring, and polygon modes. Default is FALSE.
show_coordinate_points	Logical, whether coordinate points are displayed on linestring and polygon features while drawing. Default is FALSE.
drag_interaction	How rectangle and circle drawing works: one of "click-move" (click, move, click), "click-drag", or "click-move-or-drag". Default NULL uses the Terra Draw default.
pointer_distance	Pointer tolerance in pixels for drawing interactions. Default NULL uses the Terra Draw default.
keep_mode_active	Logical. By default (FALSE) the toolbar returns to the select tool after a shape is finished, matching the mapbox-gl-draw workflow; set to TRUE to stay in the active drawing mode instead.
modes	Advanced escape hatch: a named list keyed by mode name whose entries are raw Terra Draw mode constructor options (in Terra Draw's own camelCase naming, e.g. <code>list(polygon = list(pointerDistance = 30))</code>). These are merged over what mapgl generates, with styles entries merged key by key. modeName overrides are not allowed.

Details

Rotation and scaling use keyboard-modified feature dragging rather than visible handles. First select a feature and ensure the map has keyboard focus. Hold Control+R while dragging to rotate, or Control+S while dragging to scale. Release the pointer to finish the transformation. The resize option is separate and provides visible selection points that can be dragged to resize a feature.

Value

A list of class "mapgl_terradraw_options" for use as the options argument of `add_draw_control()`.

Examples

```
## Not run:
library(mapgl)

maplibre() |>
  add_draw_control(
    provider = "terra-draw",
    options = terradraw_options(
      snap_to_coordinates = TRUE,
      rotate_features = TRUE,
      scale_features = TRUE
    )
  )

# In select mode, select a feature and hold Control+R or Control+S while
# dragging it to rotate or scale, respectively.
```

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

tooltip_style	<i>Style a tooltip or popup</i>
---------------	---------------------------------

Description

Builds a styling specification for map tooltips and popups, mirroring [legend_style\(\)](#). Pass the result – or simply a preset name like "dark" – to the tooltip_style / popup_style argument of a layer function or [add_flowmap\(\)](#). Start from an optional light/dark preset and override any individual properties on top of it. With no styling argument supplied, tooltips keep their default (unstyled) appearance.

Usage

```
tooltip_style(
  preset = NULL,
  background_color = NULL,
  background_opacity = NULL,
  text_color = NULL,
  border_color = NULL,
  border_width = NULL,
  border_radius = NULL,
  font_family = NULL,
  font_size = NULL,
  font_weight = NULL,
  padding = NULL,
  max_width = NULL,
  shadow = NULL,
  shadow_color = NULL,
  shadow_size = NULL,
  position = NULL,
  offset = NULL
)
```

```
popup_style(
  preset = NULL,
  background_color = NULL,
  background_opacity = NULL,
  text_color = NULL,
  border_color = NULL,
  border_width = NULL,
  border_radius = NULL,
  font_family = NULL,
  font_size = NULL,
  font_weight = NULL,
```

```
padding = NULL,
max_width = NULL,
shadow = NULL,
shadow_color = NULL,
shadow_size = NULL,
position = NULL,
offset = NULL
)
```

Arguments

preset	Optional base theme: "light", "dark", or "auto" (resolve to light or dark from the flowmap dark mode; ordinary layers treat "auto" as light). Individual arguments below override the preset.
background_color	CSS color for the tooltip/popup background.
background_opacity	Numeric in [0, 1] for the background opacity.
text_color	CSS color for the text.
border_color	CSS color for the border.
border_width	Border width in pixels.
border_radius	Corner radius in pixels.
font_family	CSS font-family string.
font_size	Font size in pixels.
font_weight	CSS font-weight.
padding	Internal padding in pixels.
max_width	Maximum width as a CSS size (e.g. "320px").
shadow	Logical; whether to draw a drop shadow.
shadow_color	CSS color of the drop shadow.
shadow_size	Blur radius of the drop shadow in pixels.
position	For flowmap tooltips/popups, "floating" (cursor-following) or "anchored" (pinned to the feature). Ignored by ordinary layers.
offset	For floating flowmap tooltips, a numeric pixel offset or a length-2 numeric c(x, y).

Value

A list of class `mapgl_tooltip_style`.

Examples

```
## Not run:
# A dark preset, tweaked
tooltip_style("dark", border_radius = 8, font_size = 13)
```

```
# Fully custom
tooltip_style(
  background_color = "rgba(20,30,48,0.95)",
  text_color = "#e6edf3",
  border_radius = 6
)

## End(Not run)
```

turf_area	<i>Calculate area of geometries</i>
-----------	-------------------------------------

Description

This function calculates the area of polygons in a layer or sf object. Note: This function only works with proxy objects as it returns a numeric value to R.

Usage

```
turf_area(proxy, layer_id = NULL, data = NULL, input_id = "turf_area_result")
```

Arguments

proxy	A mapboxgl_proxy or maplibre_proxy object.
layer_id	The ID of the layer or source containing the polygons (mutually exclusive with data).
data	An sf object containing polygons (mutually exclusive with layer_id).
input_id	Character string specifying the Shiny input ID suffix for storing the area result. Default is "turf_area_result". Result will be available as input[[paste0(map_id, "_turf_", input_id)]].

Value

The proxy object for method chaining.

turf_buffer	<i>Turf.js Geospatial Operations for mapgl</i>
-------------	--

Description

This module provides client-side geospatial operations using the turf.js library. All operations work with both mapboxgl and maplibre proxies. Create a buffer around geometries

Usage

```
turf_buffer(
  map,
  layer_id = NULL,
  data = NULL,
  coordinates = NULL,
  radius,
  units = "meters",
  source_id,
  input_id = NULL
)
```

Arguments

map	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
layer_id	The ID of a layer or source to buffer (mutually exclusive with data and coordinates).
data	An sf object to buffer (mutually exclusive with layer_id and coordinates).
coordinates	A numeric vector of length 2 with lng/lat coordinates to create a point and buffer (mutually exclusive with layer_id and data).
radius	The buffer distance.
units	The units for the buffer distance. One of "meters", "kilometers", "miles", "feet", "inches", "yards", "centimeters", "millimeters", "degrees", "radians".
source_id	The ID for the new source containing the buffered results. Required.
input_id	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as input[[paste0(map_id, "_turf_", input_id)]].

Details

This function creates a buffer around geometries at a specified distance. The operation is performed client-side using turf.js. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, `add_line_layer()`, etc.

Value

The map or proxy object for method chaining.

Examples

```
## Not run:
# Buffer existing layer
map |>
  turf_buffer(layer_id = "points", radius = 1000, units = "meters",
             source_id = "point_buffers") |>
  add_fill_layer(id = "buffers", source = "point_buffers", fill_color = "blue")
```

```
# Buffer sf object
map |>
  turf_buffer(data = sf_points, radius = 0.5, units = "miles",
             source_id = "buffers") |>
  add_fill_layer(id = "buffer_layer", source = "buffers")

# Buffer coordinates (great for hover events)
maplibre_proxy("map") |>
  turf_buffer(coordinates = c(-122.4, 37.7), radius = 500, units = "meters",
             source_id = "hover_buffer")

## End(Not run)
```

turf_center_of_mass	<i>Calculate center of mass</i>
---------------------	---------------------------------

Description

This function calculates the center of mass (geometric centroid) for each feature. Uses `turf.centerOfMass` which provides more accurate centroids than `turf.centroid`, matching the behavior of `sf::st_centroid()` and PostGIS `ST_Centroid`. The result is added as a source to the map, which can then be styled using `add_circle_layer()`, etc.

Usage

```
turf_center_of_mass(
  map,
  layer_id = NULL,
  data = NULL,
  coordinates = NULL,
  source_id,
  input_id = NULL
)
```

Arguments

<code>map</code>	A <code>mapboxgl</code> , <code>maplibre</code> , <code>mapboxgl_proxy</code> , or <code>maplibre_proxy</code> object.
<code>layer_id</code>	The ID of a layer or source (mutually exclusive with <code>data</code> and <code>coordinates</code>).
<code>data</code>	An <code>sf</code> object (mutually exclusive with <code>layer_id</code> and <code>coordinates</code>).
<code>coordinates</code>	A list of coordinate pairs <code>list(c(lng,lat), c(lng,lat), ...)</code> for multiple points (mutually exclusive with <code>layer_id</code> and <code>data</code>).
<code>source_id</code>	The ID for the new source containing the center of mass points. Required.
<code>input_id</code>	Optional. Character string specifying the Shiny input ID suffix for storing results. If <code>NULL</code> (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_centroid	<i>Calculate centroid of geometries</i>
---------------	---

Description

This function calculates the centroid of geometries in a layer or sf object. The result is added as a source to the map, which can then be styled using `add_circle_layer()`, etc.

Usage

```
turf_centroid(  
  map,  
  layer_id = NULL,  
  data = NULL,  
  coordinates = NULL,  
  source_id,  
  input_id = NULL  
)
```

Arguments

map	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
layer_id	The ID of a layer or source containing geometries (mutually exclusive with data and coordinates).
data	An sf object containing geometries (mutually exclusive with layer_id and coordinates).
coordinates	A list of coordinate pairs <code>list(c(lng,lat), c(lng,lat), ...)</code> for multiple points (mutually exclusive with layer_id and data).
source_id	The ID for the new source containing the centroid. Required.
input_id	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_concave_hull *Create concave hull*

Description

This function creates a concave hull around a set of points. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_concave_hull(
  map,
  layer_id = NULL,
  data = NULL,
  coordinates = NULL,
  max_edge = NULL,
  units = "kilometers",
  source_id,
  input_id = NULL
)
```

Arguments

map	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
layer_id	The ID of a layer or source containing points (mutually exclusive with data and coordinates).
data	An sf object containing points (mutually exclusive with layer_id and coordinates).
coordinates	A list of coordinate pairs <code>list(c(lng,lat), c(lng,lat), ...)</code> for multiple points (mutually exclusive with layer_id and data).
max_edge	The maximum edge length for the concave hull. If NULL (default), an optimal value is calculated automatically.
units	The units for max_edge. One of "meters", "kilometers", "miles", etc.
source_id	The ID for the new source containing the concave hull. Required.
input_id	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Details

If `max_edge` is too small and no concave hull can be computed, the function will automatically calculate an optimal `max_edge` value based on point distances. If that fails, it falls back to a convex hull to ensure a result is always returned.

Value

The map or proxy object for method chaining.

turf_convex_hull	<i>Create convex hull</i>
------------------	---------------------------

Description

This function creates a convex hull around a set of points. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_convex_hull(
  map,
  layer_id = NULL,
  data = NULL,
  coordinates = NULL,
  source_id,
  input_id = NULL
)
```

Arguments

map	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
layer_id	The ID of a layer or source containing points (mutually exclusive with data and coordinates).
data	An sf object containing points (mutually exclusive with layer_id and coordinates).
coordinates	A list of coordinate pairs <code>list(c(lng,lat), c(lng,lat), ...)</code> for multiple points (mutually exclusive with layer_id and data).
source_id	The ID for the new source containing the convex hull. Required.
input_id	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_difference	<i>Find difference between two geometries</i>
-----------------	---

Description

This function subtracts the second geometry from the first. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_difference(
  map,
  layer_id = NULL,
  layer_id_2 = NULL,
  data = NULL,
  data_2 = NULL,
  source_id,
  input_id = NULL
)
```

Arguments

<code>map</code>	A <code>mapboxgl</code> , <code>maplibre</code> , <code>mapboxgl_proxy</code> , or <code>maplibre_proxy</code> object.
<code>layer_id</code>	The ID of the first layer or source (geometry to subtract from, mutually exclusive with <code>data</code>).
<code>layer_id_2</code>	The ID of the second layer or source (geometry to subtract, mutually exclusive with <code>data_2</code>).
<code>data</code>	An <code>sf</code> object for the first geometry (mutually exclusive with <code>layer_id</code>).
<code>data_2</code>	An <code>sf</code> object for the second geometry (mutually exclusive with <code>layer_id_2</code>).
<code>source_id</code>	The ID for the new source containing the difference result. Required.
<code>input_id</code>	Optional. Character string specifying the Shiny input ID suffix for storing results. If <code>NULL</code> (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_distance	<i>Calculate distance between two features</i>
---------------	--

Description

This function calculates the distance between the first features of two layers or coordinates. Note: This function only works with proxy objects as it returns a numeric value to R.

Usage

```
turf_distance(  
  proxy,  
  layer_id = NULL,  
  layer_id_2 = NULL,  
  data = NULL,  
  coordinates = NULL,  
  coordinates_2 = NULL,  
  units = "kilometers",  
  input_id = "turf_distance_result"  
)
```

Arguments

proxy	A mapboxgl_proxy or maplibre_proxy object.
layer_id	The ID of the first layer or source (mutually exclusive with data and coordinates).
layer_id_2	The ID of the second layer or source (required if layer_id is used).
data	An sf object for the first geometry (mutually exclusive with layer_id and coordinates).
coordinates	A numeric vector of length 2 with lng/lat coordinates for the first point (mutually exclusive with layer_id and data).
coordinates_2	A numeric vector of length 2 with lng/lat coordinates for the second point (required if coordinates is used).
units	The units for the distance calculation. One of "meters", "kilometers", "miles", etc.
input_id	Character string specifying the Shiny input ID suffix for storing the distance result. Default is "turf_distance_result". Result will be available as input[[paste0(map_id, "_turf_", input_id)]].

Value

The proxy object for method chaining.

turf_filter

*Spatial filter features by predicate***Description**

This function filters features from the first layer based on their spatial relationship with features in the second layer using various spatial predicates.

Usage

```
turf_filter(
  map,
  layer_id = NULL,
  filter_layer_id = NULL,
  data = NULL,
  filter_data = NULL,
  predicate = c("intersects", "within", "contains", "crosses", "disjoint"),
  source_id,
  input_id = NULL
)
```

Arguments

<code>map</code>	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
<code>layer_id</code>	The ID of the layer or source to filter (mutually exclusive with data).
<code>filter_layer_id</code>	The ID of the layer or source to filter against (mutually exclusive with filter_data).
<code>data</code>	An sf object containing features to filter (mutually exclusive with layer_id).
<code>filter_data</code>	An sf object containing the filter geometry (mutually exclusive with filter_layer_id).
<code>predicate</code>	The spatial relationship to test. One of: "intersects", "within", "contains", "crosses", "disjoint".
<code>source_id</code>	The ID for the new source containing the filtered results. Required.
<code>input_id</code>	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_intersect	<i>Find intersection of two geometries</i>
----------------	--

Description

This function finds the intersection between geometries in two layers or sf objects. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_intersect(
  map,
  layer_id = NULL,
  layer_id_2 = NULL,
  data = NULL,
  data_2 = NULL,
  source_id,
  input_id = NULL
)
```

Arguments

<code>map</code>	A <code>mapboxgl</code> , <code>maplibre</code> , <code>mapboxgl_proxy</code> , or <code>maplibre_proxy</code> object.
<code>layer_id</code>	The ID of the first layer or source (mutually exclusive with <code>data</code>).
<code>layer_id_2</code>	The ID of the second layer or source (mutually exclusive with <code>data_2</code>).
<code>data</code>	An sf object for the first geometry (mutually exclusive with <code>layer_id</code>).
<code>data_2</code>	An sf object for the second geometry (mutually exclusive with <code>layer_id_2</code>).
<code>source_id</code>	The ID for the new source containing the intersection result. Required.
<code>input_id</code>	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_union	<i>Union geometries</i>
------------	-------------------------

Description

This function unions all polygons in a layer into a single geometry. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_union(map, layer_id = NULL, data = NULL, source_id, input_id = NULL)
```

Arguments

<code>map</code>	A <code>mapboxgl</code> , <code>maplibre</code> , <code>mapboxgl_proxy</code> , or <code>maplibre_proxy</code> object.
<code>layer_id</code>	The ID of a layer or source to union (mutually exclusive with <code>data</code>).
<code>data</code>	An <code>sf</code> object to union (mutually exclusive with <code>layer_id</code>).
<code>source_id</code>	The ID for the new source containing the union result. Required.
<code>input_id</code>	Optional. Character string specifying the Shiny input ID suffix for storing results. If <code>NULL</code> (default), no input is registered. For proxy operations, the result will be available as <code>input[[paste0(map_id, "_turf_", input_id)]]</code> .

Value

The map or proxy object for method chaining.

turf_voronoi	<i>Create Voronoi diagram</i>
--------------	-------------------------------

Description

This function creates a Voronoi diagram from a set of points. The result is added as a source to the map, which can then be styled using `add_fill_layer()`, etc.

Usage

```
turf_voronoi(
  map,
  layer_id = NULL,
  data = NULL,
  coordinates = NULL,
  bbox = NULL,
  property = NULL,
  source_id,
  input_id = NULL
)
```

Arguments

map	A mapboxgl, maplibre, mapboxgl_proxy, or maplibre_proxy object.
layer_id	The ID of a layer or source containing points (mutually exclusive with data and coordinates).
data	An sf object containing points (mutually exclusive with layer_id and coordinates).
coordinates	A list of coordinate pairs list(c(lng,lat), c(lng,lat), ...) for multiple points (mutually exclusive with layer_id and data).
bbox	Optional. Can be: (1) A numeric vector of length 4, in format 'c(xmin, ymin, xmax, ymax)', (2) An sf object to extract bbox from, or (3) A layer_id string to extract bbox from and clip results to.
property	Optional. Character string specifying a column name from the input points to transfer to the Voronoi polygons using spatial collection.
source_id	The ID for the new source containing the Voronoi diagram. Required.
input_id	Optional. Character string specifying the Shiny input ID suffix for storing results. If NULL (default), no input is registered. For proxy operations, the result will be available as input[[paste0(map_id, "_turf_", input_id)]].

Value

The map or proxy object for method chaining.

update_slider_control *Update a slider control from Shiny*

Description

Proxy-only companion to [add_slider_control\(\)](#). Moves the slider to a specific value, starts or stops playback, or adjusts playback speed without recreating the control.

Usage

```
update_slider_control(
  proxy,
  value = NULL,
  playing = NULL,
  animation_duration = NULL
)
```

Arguments

proxy	A proxy object from mapboxgl_proxy() or maplibre_proxy() .
value	Numeric; one of the values the slider was created with. The slider snaps to the closest value if an exact match is not found.
playing	Logical; TRUE starts playback, FALSE stops.
animation_duration	Milliseconds per step when playing.

Value

The proxy object, invisibly.

Index

- * **datasets**
 - bixi_flows, [78](#)
 - bixi_locations, [79](#)
- add_bivariate_legend, [5](#)
- add_categorical_legend (map_legends), [122](#)
- add_categorical_legend(), [88](#)
- add_circle_layer, [6](#)
- add_circle_layer(), [146](#)
- add_continuous_legend (map_legends), [122](#)
- add_control, [10](#)
- add_coordinates_control, [11](#)
- add_draw_control, [12](#)
- add_draw_control(), [70](#), [95](#), [167](#), [169](#)
- add_features_to_draw, [17](#)
- add_features_to_draw(), [15](#)
- add_fill_extrusion_layer, [18](#)
- add_fill_layer, [21](#)
- add_flowmap, [24](#)
- add_flowmap(), [102](#), [148](#), [170](#)
- add_fullscreen_control, [28](#)
- add_geocoder_control, [28](#)
- add_geolocate_control, [30](#)
- add_globe_control, [31](#)
- add_globe_minimap, [32](#)
- add_h3j_source, [33](#)
- add_h3j_source(), [34](#)
- add_h3t_source, [34](#)
- add_heatmap_layer, [35](#)
- add_image, [37](#)
- add_image(), [76](#)
- add_image_source, [38](#)
- add_layer, [39](#)
- add_layers_control, [41](#)
- add_layers_control(), [62](#)
- add_legend (map_legends), [122](#)
- add_legend(), [62](#)
- add_line_layer, [44](#)
- add_markers, [47](#)
- add_navigation_control, [49](#)
- add_pmtils_source, [50](#)
- add_raster_dem_source, [51](#)
- add_raster_layer, [52](#)
- add_raster_source, [54](#)
- add_reset_control, [55](#)
- add_scale_control, [56](#)
- add_screenshot_control, [57](#)
- add_screenshot_control(), [144](#)
- add_slider_control, [58](#)
- add_slider_control(), [75](#), [158](#), [183](#)
- add_source, [63](#)
- add_symbol_layer, [64](#)
- add_symbol_layer(), [87](#), [146](#)
- add_terradraw_control, [70](#)
- add_terradraw_control(), [15](#)
- add_vector_source, [72](#)
- add_video_source, [73](#)
- add_view, [74](#)
- as_time_property, [75](#)
- as_time_property(), [60](#), [62](#)
- basemap_style, [76](#)
- basemap_style(), [37](#)
- bivariate_palettes, [77](#)
- bivariate_scale, [77](#)
- bixi_flows, [78](#), [80](#)
- bixi_locations, [79](#), [79](#)
- carto_style, [80](#)
- case_expr (conditional_expressions), [93](#)
- classification_helpers, [80](#)
- clear_controls, [82](#)
- clear_drawn_features, [83](#)
- clear_drawn_features(), [15](#)
- clear_layer, [84](#)
- clear_layer(), [8](#), [69](#)
- clear_legend, [85](#)
- clear_markers, [86](#)
- cluster_options, [86](#)

- cluster_options(), 8, 69
- coalesce_expr
 - (conditional_expressions), 93
- compare, 89
- concat, 92, 93
- conditional_expressions, 93
- draw_attribute, 95
- draw_attribute(), 14, 71
- ease_to, 96
- enable_shiny_hover, 97
- esri_open_style, 98
- esri_style, 99
- fit_bounds, 101
- flowmap_color_schemes, 102
- flowmap_color_schemes(), 25
- fly_to, 102
- get_breaks(classification_helpers), 80
- get_column, 93, 103
- get_drawn_features, 103
- get_drawn_features(), 15
- get_legend_colors
 - (classification_helpers), 80
- get_legend_labels
 - (classification_helpers), 80
- get_queried_features, 104
- has_column(conditional_expressions), 93
- html_escape_expr
 - (conditional_expressions), 93
- if_else_expr(conditional_expressions), 93
- interpolate, 93, 105
- interpolate(), 61
- interpolate_palette, 106
- interpolate_palette(), 25, 148, 162
- is_blank(conditional_expressions), 93
- jump_to, 108
- knitr::include_graphics(), 138, 139
- legend_style, 108, 129
- legend_style(), 170
- map_legends, 122
- mapbox_style, 116
- mapboxgl, 111
- mapboxgl(), 25, 37, 59, 76, 138, 144, 148
- mapboxgl_compare_proxy, 113
- mapboxgl_proxy, 114
- mapboxgl_proxy(), 59, 148, 183
- mapboxgl_view, 114
- mapboxglCompareOutput, 112
- mapboxglOutput, 113
- maplibre, 99, 100, 116
- maplibre(), 25, 37, 59, 76, 138, 144, 148
- maplibre_compare_proxy, 119
- maplibre_proxy, 120
- maplibre_proxy(), 59, 148, 183
- maplibre_view, 120
- maplibreCompareOutput, 118
- maplibreOutput, 119
- maptiler_style, 122
- match_expr, 93, 132
- match_expr(), 61, 87
- move_layer, 133
- number_format, 93, 134
- on_section, 136
- openfreemap_style, 136
- palette_to_lut, 137
- popup_style(tooltip_style), 170
- popup_style(), 151
- print.mapgl_classification
 - (classification_helpers), 80
- print.mapgl_continuous_scale
 - (classification_helpers), 80
- print_map, 138
- query_rendered_features, 139
- renderMapboxgl, 142
- renderMapboxglCompare, 142
- renderMaplibre, 143
- renderMaplibreCompare, 143
- save_map, 144
- set_config_property, 145
- set_filter, 146
- set_filter(), 62
- set_flowmap_filter, 147
- set_flowmap_filter(), 148
- set_flowmap_settings, 148

set_fog, 149
set_layout_property, 149
set_paint_property, 150
set_popup, 151
set_projection, 151
set_rain, 152
set_snow, 153
set_source, 155
set_source(), 8, 69, 147
set_style, 155
set_terrain, 156
set_tooltip, 157
set_view, 158
slider_style, 158
slider_style(), 61
step_classification, 161
step_equal_interval
 (step_classification), 161
step_expr, 93, 163
step_jenks (step_classification), 161
step_quantile (step_classification), 161
story_leaflet, 164
story_map, 165
story_maplibre, 166
story_section, 167

terra draw_options, 167
terra draw_options(), 14, 70, 71
tooltip_style, 170
tooltip_style(), 8, 20, 23, 27, 40, 46, 69,
 151, 157
turf_area, 172
turf_buffer, 172
turf_center_of_mass, 174
turf_centroid, 175
turf_concave_hull, 176
turf_convex_hull, 177
turf_difference, 178
turf_distance, 179
turf_filter, 180
turf_intersect, 181
turf_union, 182
turf_voronoi, 182

update_slider_control, 183
update_slider_control(), 62