Package 'mapgl'

July 17, 2025

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

Version 0.3.2

Date 2025-07-17

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

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

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

Suggests mapboxapi, usethis, leaflet

NeedsCompilation no

Author Kyle Walker [aut, cre]

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

Repository CRAN

Date/Publication 2025-07-17 21:30:02 UTC

Contents

`add_circle_layer` *Add a circle layer to a Mapbox GL map*

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_opacity = 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,
  hover_options = NULL,
  before_id = NULL,
  filter = NULL,
  cluster_options = NULL
)
```

Arguments

`map`	A map object created by the `mapboxgl` function.
`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_opacity The opacity at which the circle will be drawn.

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 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.

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 `cluster_options()` function.

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, ...)
```

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.
`...`	Additional arguments passed to the JavaScript side.

Value

The modified map object with the custom control added.

Examples

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

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"
    )

## End(Not run)
```

`add_draw_control` *Add a draw control to a map*

Description

Add a draw control to a map

Usage

```
add_draw_control(
  map,
  position = "top-left",
  freehand = FALSE,
  simplify_freehand = FALSE,
  orientation = "vertical",
  source = 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",
  ...
)
```

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.
`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.

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".
...	Additional named arguments. See https://github.com/mapbox/mapbox-gl-draw/blob/main/docs/API.md#options for a list of options.

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
```

```
    )

## 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_base = NULL,
  fill_extrusion_color = NULL,
  fill_extrusion_height = NULL,
  fill_extrusion_opacity = NULL,
  fill_extrusion_pattern = NULL,
  fill_extrusion_translate = NULL,
  fill_extrusion_translate_anchor = "map",
  visibility = "visible",
  slot = NULL,
  min_zoom = NULL,
  max_zoom = NULL,
  popup = NULL,
  tooltip = NULL,
  hover_options = NULL,
  before_id = NULL,
  filter = NULL
)
```

Arguments

map	A map object created by the mapboxgl function.
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_base	The base height of the fill extrusion.
fill_extrusion_color	The color of the fill extrusion.

`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`.

`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.

`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 *Add a fill layer to a map*

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_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,
  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_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	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.
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_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 `mapboxgl` or `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".
`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 MAPTILER_API_KEY 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 *Add a globe control to a map*

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 *Add a Globe Minimap to a map*

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

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_heatmap_layer *Add a heatmap layer to a Mapbox GL map*

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 mapboxgl function.
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).
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.

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

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 *Add a layer to a map from a source*

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
)
```

Arguments

map	A map object created by the `mapboxgl()` or `maplibre()` 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 `type` and `url` 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	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.
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 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 *Add a layers control to the map*

Description

Add a layers control to the map

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
)
```

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	A vector of layer IDs to be included in the control. 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	Custom top margin in pixels, allowing for fine control over control positioning to avoid overlaps. 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.

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)

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"
    )

# Avoid collision with other controls using margin parameters
maplibre() |>
    add_navigation_control(position = "top-right") |>
    add_layers_control(
        position = "top-right",
        margin_top = 110
    )

## End(Not run)
```

`add_line_layer` *Add a line layer to a map*

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_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_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,
  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_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_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` 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.

`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 `mapboxgl` or `maplibre` functions.
`data`	A length-2 numeric vector of coordinates, a list of length-2 numeric vectors, or an `sf` 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 `sf` 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 sf 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 *Add a PMTiles source to a Mapbox GL or Maplibre GL map*

Description

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

Usage

```
add_pmtiles_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 PMTiles archive.
...	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` *Add a raster layer to a Mapbox GL map*

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_contrast = 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

`map`	A map object created by the `mapboxgl` function.
`id`	A unique ID for the layer.
`source`	The ID of the source.
`source_layer`	The source layer (for vector sources).
`raster_brightness_max`	The maximum brightness of the image.
`raster_brightness_min`	The minimum brightness of the image.
`raster_contrast`	Increase or reduce the brightness of the image.
`raster_fade_duration`	The duration of the fade-in/fade-out effect.
`raster_hue_rotate`	Rotates hues around the color wheel.

raster_opacity	The opacity at which the raster will be drawn.
raster_resampling	The resampling/interpolation method to use for overscaling.
raster_saturation	Increase or reduce the saturation of the image.
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).

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

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 *Add a reset control to a map*

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` *Add a scale control to a map*

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_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` *Add a symbol layer to a map*

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_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_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,
  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-ma

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_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_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.

`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 `cluster_options()` function.

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_vector_source` *Add a vector tile source to a Mapbox GL or Maplibre GL map*

Description

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

Usage

```
add_vector_source(map, id, url, 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.
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 *Add a video source to a Mapbox GL or Maplibre GL map*

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 *Add a visualization layer to an existing map*

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 mapboxgl_view(), maplibre_view(), mapboxgl(), or maplibre()
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 step_expr() instead of interpolate().
palette	Color palette function that takes n and returns a character vector of colors. Defaults to viridisLite::viridis.
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)
```

---

carto_style *Get CARTO Style URL*

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` *Extract information from classification and continuous scale objects*

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 all controls from a Mapbox GL or Maplibre GL map in a Shiny app*

Description

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

Usage

```
clear_controls(map)
```

Arguments

map	A map object created by the `mapboxgl` or `maplibre` function.

Value

The modified map object with all controls removed.

`clear_drawn_features` *Clear all drawn features from a map*

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 `mapboxgl_proxy` or `maplibre_proxy` 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 `mapboxgl_proxy` or `maplibre_proxy`.
`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` *Clear legends from a map*

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 `mapboxgl_proxy()` or `maplibre_proxy()`.
`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 *Clear markers from a map in a Shiny session*

Description

Clear markers from a map in a Shiny session

Usage

```
clear_markers(map)
```

Arguments

map	A map object created by the mapboxgl_proxy or maplibre_proxy function.

Value

The modified map object with the markers cleared.

`cluster_options` *Prepare cluster options for circle layers*

Description

This function creates a list of options for clustering circle layers.

Usage

```
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 = NULL,
  circle_opacity = NULL,
  circle_stroke_color = NULL,
  circle_stroke_opacity = NULL,
  circle_stroke_width = NULL,
  text_color = "black"
)
```

Arguments

`max_zoom`	The maximum zoom level at which to cluster points.
`cluster_radius`	The radius of each cluster when clustering points.
`color_stops`	A vector of colors for the circle color step expression.
`radius_stops`	A vector of radii for the circle radius step expression.
`count_stops`	A vector of point counts for both color and radius step expressions.
`circle_blur`	Amount to blur the circle.
`circle_opacity`	The opacity of the circle.
`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.
`text_color`	The color to use for labels on the cluster circles.

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
)
```

---

compare *Create a Compare widget*

Description

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

Usage

```
compare(
  map1,
  map2,
  width = "100%",
  height = NULL,
  elementId = NULL,
  mousemove = FALSE,
  orientation = "vertical",
  mode = "swipe",
  swiper_color = NULL
)
```

Arguments

map1	A `mapboxgl` or `maplibre` object representing the first map.
map2	A `mapboxgl` or `maplibre` object representing the second map.
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.
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".

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
- mode="sync" - Places the maps next to each other with synchronized navigation

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

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")

# 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"))
```

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 `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 easing to the view.

Value

The updated map object.

enable_shiny_hover *Enable hover events for Shiny applications*

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

map	A maplibre or mapboxgl widget object.
coordinates	Logical. If TRUE, provides general mouse coordinates via _hover input. Defaults to TRUE.
features	Logical. If TRUE, provides feature information via _feature_hover input when hovering over map features. Defaults to TRUE.
layer_id	Character. If provided, only features from the specified layer will be included in the _feature_hover 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)
```

`fit_bounds` *Fit the map to a bounding box*

Description

Fit the map to a bounding box

Usage

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

Arguments

`map`	A map object created by the `mapboxgl` or `maplibre` function or a proxy object.
`bbox`	A bounding box specified as a numeric vector of length 4 (minLng, minLat, maxLng, maxLat), or an sf 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 FALSE.
`...`	Additional named arguments for fitting the bounds.

Value

The updated map object.

`fly_to` *Fly to a given view*

Description

Fly to a given view

Usage

```
fly_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 flying to the view.

Value

The updated map object.

`get_column` *Get column or property for use in mapping*

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` *Get drawn features from the map*

Description

Get drawn features from the map

Usage

```
get_drawn_features(map)
```

Arguments

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

Value

An sf object containing the drawn features.

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` *Create an interpolation expression with automatic palette and break calculation*

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"
)
```

Arguments

`data`	A data frame or sf object containing the data. If provided, data_values will be extracted from `data[[column]]`. Either data or data_values must be provided.
`column`	The name of the column to use for the interpolation.
`data_values`	A numeric vector of the actual data values used to calculate breaks. If NULL and data is provided, will be extracted from `data[[column]]`.
`method`	The method for calculating breaks. Options are "equal" (equal intervals), "quantile" (quantile breaks), or "jenks" (Jenks natural breaks). Defaults to "equal".
`n`	The number of break points to create. Defaults to 5.

palette	A function that takes n and returns a character vector of colors. If NULL and colors is also NULL, defaults to `viridisLite::viridis`.
colors	A character vector of colors to use. If provided, these colors will be interpolated to match the number of breaks if needed. Either palette or colors should be provided, but not both.
na_color	The color to use for missing values. Defaults to "grey".

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 *Jump to a given view*

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 *Create custom styling for map legends*

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

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

style	The Mapbox style to use.
center	A numeric vector of length 2 specifying the initial center of the map.
zoom	The initial zoom level of the map.
bearing	The initial bearing (rotation) of the map, in degrees.
pitch	The initial pitch (tilt) of the map, in degrees.
projection	The map projection to use (e.g., "mercator", "globe").
parallels	A vector of two numbers representing the standard parallels of the projection. Only available when the projection is "albers" or "lambertConformalConic".
access_token	Your Mapbox access token.
bounds	An sf object or bounding box to fit the map to.
width	The width of the output htmlwidget.
height	The height of the output htmlwidget.
...	Additional named parameters to be passed to the Mapbox GL map.

Value

An HTML widget for a Mapbox map.

Examples

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

## 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 side to target in the compare widget, either "before" or "after".

Value

A proxy object for the Mapbox GL Compare widget.

mapboxgl_proxy *Create a proxy object for a Mapbox GL map in Shiny*

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 *Quick visualization of geometries with Mapbox GL*

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",
  ...
)
```

Arguments

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 step_expr() instead of interpolate().
palette	Color palette function that takes n and returns a character vector of colors. Defaults to viridisLite::viridis.
style	The Mapbox style to use. Defaults to mapbox_style("light").
layer_id	The layer ID to use for the visualization. Defaults to "quickview".
legend	Logical, whether to add a legend when a column is specified. Defaults to TRUE.
legend_position	The position of the legend on the map. Defaults to "top-left".
...	Additional arguments passed to mapboxgl()

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 *Get Mapbox Style URL*

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 *Initialize a Maplibre GL Map*

Description

Initialize a Maplibre GL Map

Usage

```
maplibre(
  style = carto_style("voyager"),
  center = c(0, 0),
  zoom = 0,
  bearing = 0,
  pitch = 0,
  bounds = NULL,
  width = "100%",
  height = NULL,
  ...
)
```

Arguments

style	The style JSON to use.
center	A numeric vector of length 2 specifying the initial center of the map.
zoom	The initial zoom level of the map.
bearing	The initial bearing (rotation) of the map, in degrees.
pitch	The initial pitch (tilt) of the map, in degrees.
bounds	An sf object or bounding box to fit the map to.
width	The width of the output htmlwidget.
height	The height of the output htmlwidget.
...	Additional named parameters to be passed to the Mapbox GL map.

Value

An HTML widget for a Mapbox map.

Examples

```
## Not run:
maplibre()

## 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 side to target in the compare widget, either "before" or "after".

Value

A proxy object for the Maplibre GL Compare widget.

maplibre_proxy *Create a proxy object for a Maplibre GL map in Shiny*

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 *Quick visualization of geometries with MapLibre GL*

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 = carto_style("positron"),
  layer_id = "quickview",
  legend = TRUE,
  legend_position = "top-left",
  ...
)
```

Arguments

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 step_expr() instead of interpolate().
palette	Color palette function that takes n and returns a character vector of colors. Defaults to viridisLite::viridis.
style	The MapLibre style to use. Defaults to carto_style("positron").
layer_id	The layer ID to use for the visualization. Defaults to "quickview".
legend	Logical, whether to add a legend when a column is specified. Defaults to TRUE.
legend_position	The position of the legend on the map. Defaults to "top-left".
...	Additional arguments passed to maplibre()

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 *Get MapTiler Style URL*

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 *Add legends to Mapbox GL and MapLibre GL maps*

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,
  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
)

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
)

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
)
```

Arguments

map	A map object created by the mapboxgl or maplibre 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 add_legend only).
circular_patches	(Deprecated) Logical, whether to use circular patches in the legend. Use patch_shape = "circle" instead.
patch_shape	Character or sf 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 sf 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 that this legend is associated with. If provided, the legend will be shown/hidden when the layer visibility is toggled.
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 legend_style() or a list of style options.

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
)


## End(Not run)
```

`match_expr` *Create a match expression*

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 an existing Mapbox GL or Maplibre GL map using a proxy object.

Usage

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

Arguments

proxy	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 *immediately behind* 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 proxy object.

number_format *Create a number formatting expression*

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 *Observe events on story map section transitions*

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

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

---

query_rendered_features *Query rendered features on a map in a Shiny session*

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

proxy	A MapboxGL or Maplibre proxy object, defined with `mapboxgl_proxy()`, `maplibre_proxy()`, `mapboxgl_compare_proxy()`, or `maplibre_compare_proxy()`
geometry	The geometry to query. Can be: • `NULL` (default): Query the entire viewport • A length-2 vector `c(x, y)`: Query at a single point in pixel coordinates • A length-4 vector `c(xmin, ymin, xmax, ymax)`: Query within a bounding box in pixel coordinates
layer_id	A character vector of layer names to include in the query. Can be a single layer name or multiple layer names. If `NULL` (default), all layers are queried.
filter	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 `set_filter()` separately, you must pass the same filter here to get aligned results.
callback	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 *Render a Mapbox GL output element in Shiny*

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 *Render a Mapbox GL Compare output element in Shiny*

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 *Render a Maplibre GL output element in Shiny*

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 *Render a Maplibre GL Compare output element in Shiny*

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

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 mapboxgl function or a proxy object defined with mapboxgl_proxy().
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 *Set a filter on a map layer*

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 mapboxgl or maplibre 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.

set_fog *Set fog on a Mapbox GL map*

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 *Set a layout property on a map layer*

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 *Set a paint property on a map layer*

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 *Set popup on a map layer*

Description

Set popup on a map layer

Usage

```
set_popup(map, layer_id = NULL, popup, 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.
popup	The name of the popup property or an expression to set.
layer	Deprecated. Use layer_id instead.

Value

The updated map object.

set_projection *Set Projection for a Mapbox/Maplibre Map*

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 `mapboxgl` 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 c(0, 80).
`droplet_size`	A numeric vector of length 2 controlling the rain droplet size (x - normal to direction, y - along direction). Default is c(2.6, 18.2).
`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 mapboxgl 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 c(0, 50).
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` *Set source of a map layer*

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 `mapboxgl` or `maplibre` 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 `layer_id` instead.

Value

The updated map object.

`set_style` *Update the style of a map*

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 *Set terrain properties on a map*

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)
```

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	The name of the tooltip to set.
layer	Deprecated. Use layer_id instead.

Value

The updated map object.

set_view *Set the map center and zoom level*

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.

step_classification *Step expressions with automatic classification*

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

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

Index