Package 'geolibre'

September 3, 2026

Title Interactive GIS with 'GeoLibre'

Version 0.2.0

Description Embeds the full 'GeoLibre' geographic information system in 'R Markdown', 'Quarto', 'Shiny', and the 'RStudio' Viewer. Create maps from 'GeoJSON' and 'sf' objects, markers, heatmaps, and tabular coordinates; add 'Cloud Optimized GeoTIFF', 'XYZ', 'WMS', 'WMTS', 'WFS', 'PMTiles', vector tile, '3D Tiles', and video sources; classify choropleths, arrange layers, and add legends, colorbars, and split-map comparisons. Control the camera, export standalone 'HTML', and read and write '.geolibre.json' project files. The underlying application is described in Wu (2026) <doi:10.5281/zenodo.20785400>.

License MIT + file LICENSE

URL https://r.geolibre.app, https://github.com/opengeos/geolibre-r, https://geolibre.app

BugReports https://github.com/opengeos/geolibre-r/issues

Depends R (>= 4.1.0)

Imports htmlwidgets, jsonlite, utils

Suggests sf, shiny, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Qiusheng Wu [aut, cre], Open Geospatial Solutions [cph]

Maintainer Qiusheng Wu <giswqs@gmail.com>

Repository CRAN

Date/Publication 2026-09-03 11:30:02 UTC

Contents

`add_3d_tiles` *Add a 3D Tiles layer*

Description

Add a 3D Tiles layer

Usage

```
add_3d_tiles(
  map,
  url,
  name = "3D Tiles",
  altitude_offset = 0,
  request_headers = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`url`	URL of the 3D Tiles `tileset.json`.
`name`	Layer name.
`altitude_offset`	Vertical offset applied to the tileset, in meters.
`request_headers`	Optional named list of request headers. These are stored in the project, so avoid persisting secrets; `save_project()` strips them unless `keep_credentials = TRUE`.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_3d_tiles("https://example.com/tileset.json", altitude_offset = 20)
stopifnot(map$x$project$layers[[1]]$type == "3d-tiles")
```

add_choropleth *Add a data-driven choropleth layer*

Description

Classifies `column` into `class_count` numeric ranges and colors each range from `colormap`, producing the same graduated symbology the application's Style panel builds from the interface.

Usage

```
add_choropleth(
  map,
  data,
  column,
  name = "Choropleth",
  class_count = 5,
  colormap = "viridis",
  scheme = c("equal-interval", "quantile"),
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`data`	GeoJSON as a parsed list (a `FeatureCollection`, `Feature`, or bare geometry), a JSON string, a file path, an HTTP(S) URL, or an `sf` object. A URL or file is read and inlined into the project, up to a 50 MB limit; for larger datasets prefer `add_vector()`, which lets the browser stream the source.
`column`	Name of the numeric feature property to classify.
`name`	Layer name.
`class_count`	Number of classes, clamped to between 2 and 12.
`colormap`	A color ramp name from `color_ramp_names()`.
`scheme`	Classification scheme, `"equal-interval"` or `"quantile"`.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

See Also

`classify_layer()` to symbolize a layer that is already on the map.

Examples

```
counties <- list(
  type = "FeatureCollection",
  features = list(
    list(
      type = "Feature", properties = list(pop = 100),
      geometry = list(type = "Point", coordinates = c(-77, 39))
    ),
    list(
      type = "Feature", properties = list(pop = 900),
      geometry = list(type = "Point", coordinates = c(-76, 40))
    )
  )
)
map <- geolibre() |>
  add_choropleth(counties, column = "pop", colormap = "blues", class_count = 3)
stopifnot(map$x$project$layers[[1]]$style$vectorStyleMode == "graduated")
```

---

add_circle_markers *Add circle markers*

Description

`add_markers()` with the circle radius surfaced as a named argument.

Usage

```
add_circle_markers(
  map,
  points,
  name = "Circle Markers",
  radius = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
points	Points as a two-column matrix or a data frame of coordinates, a list of c(longitude, latitude) pairs or named list(lng = , lat = , ...) entries, a point GeoJSON source, or an sf object of points.
name	Layer name.
radius	Optional circle radius in pixels.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_circle_markers(list(c(-77, 39)), radius = 12, fillColor = "#16a34a")
```

---

add_cog *Add a Cloud Optimized GeoTIFF layer*

Description

Add a Cloud Optimized GeoTIFF layer

Usage

```
add_cog(
  map,
  url,
  name = "COG",
  bands = NULL,
  colormap = NULL,
  rescale = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
url	Public HTTP(S) URL of a Cloud Optimized GeoTIFF or GeoTIFF.
name	Layer name.
bands	Optional one-based band indices. Three or more bands render as RGB; one renders as a single band, which colormap then colors.
colormap	Optional GeoLibre colormap name for single-band rendering.
rescale	Optional list of numeric c(min, max) ranges, one per rendered band. A single c(min, max) pair is accepted for the one-band case.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

See Also

add_raster(), add_tile_layer(), add_colorbar()

Examples

```
map <- geolibre() |>
  add_cog("https://example.com/image.tif", bands = c(1, 2, 3))
stopifnot(map$x$project$layers[[1]]$type == "cog")
```

---

add_colorbar *Add a colorbar to the map*

Description

Renders a gradient with minimum and maximum ticks, from either a named color ramp or an explicit list of CSS colors. Each call adds another colorbar.

Usage

```
add_colorbar(
  map,
  colormap = "viridis",
  vmin = 0,
  vmax = 1,
  label = "",
  units = "",
  colors = NULL,
  orientation = c("vertical", "horizontal"),
  position = c("bottom-right", "bottom-left", "top-left", "top-right")
)
```

Arguments

map	A GeoLibre widget.
colormap	A color ramp name from color_ramp_names(). Ignored when colors is supplied.
vmin	Value at the low end of the colorbar.
vmax	Value at the high end of the colorbar.
label	Title shown alongside the colorbar.
units	Units suffix shown with the values.
colors	Optional character vector of CSS colors defining a custom gradient, used instead of colormap.
orientation	"vertical" or "horizontal".
position	Corner for the colorbar: "top-left", "top-right", "bottom-left", or "bottom-right".

Value

The modified widget.

See Also

add_legend() for categorical classes

Examples

```
map <- geolibre() |>
  add_raster("https://example.com/dem.tif", bands = 1, colormap = "terrain") |>
  add_colorbar(
    colormap = "terrain", vmin = 0, vmax = 3000,
    label = "Elevation", units = "m"
  )
```

add_colormap *Add a colorbar from a named color ramp*

Description

`add_colorbar()` with `colormap` in the leading position, for parity with the Python API.

Usage

```
add_colormap(map, colormap = "viridis", vmin = 0, vmax = 1, label = "", ...)
```

Arguments

`map`	A GeoLibre widget.
`colormap`	A color ramp name from `color_ramp_names()`. Ignored when `colors` is supplied.
`vmin`	Value at the low end of the colorbar.
`vmax`	Value at the high end of the colorbar.
`label`	Title shown alongside the colorbar.
`...`	Forwarded to `add_colorbar()`, for example `units`, `orientation`, or `position`.

Value

The modified widget.

Examples

```
geolibre() |> add_colormap("plasma", vmin = 0, vmax = 100, label = "Index")
```

add_csv *Add a CSV of point coordinates*

Description

`add_xy_data()` with a CSV-oriented default layer name.

Usage

```
add_csv(
  map,
  data,
  x = "longitude",
  y = "latitude",
  name = "CSV",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
data	A data frame, a CSV file path, a CSV URL, CSV text, or a list of row lists.
x	Name of the longitude column.
y	Name of the latitude column.
name	Layer name.
style	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
text <- "name,longitude,latitude\nDC,-77.0369,38.9072"
map <- geolibre() |> add_csv(text)
```

---

add_data *Add data, optionally symbolized by a column*

Description

With `column` supplied this is `add_choropleth()`; without it, `add_geojson()`. Provided for parity with the Python API.

Usage

```
add_data(map, data, column = NULL, name = "Data", ...)
```

Arguments

map	A GeoLibre widget.
data	GeoJSON as a parsed list (a FeatureCollection, Feature, or bare geometry), a JSON string, a file path, an HTTP(S) URL, or an sf object. A URL or file is read and inlined into the project, up to a 50 MB limit; for larger datasets prefer add_vector(), which lets the browser stream the source.
column	Optional numeric property to drive graduated symbology.
name	Layer name.
...	Forwarded to add_choropleth() or add_geojson().

Value

The modified widget.

Examples

```
point <- list(type = "Point", coordinates = c(-77, 39))
map <- geolibre() |> add_data(point, name = "Point")
```

---

add_flatgeobuf *Add a FlatGeobuf layer*

Description

Add a FlatGeobuf layer

Usage

```
add_flatgeobuf(map, data, name = "FlatGeobuf", ...)
```

Arguments

map	A GeoLibre widget.
data	A dataset URL, a local file path, or an sf object.
name	Layer name.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
geolibre() |> add_flatgeobuf("https://example.com/data.fgb")
```

`add_geojson` *Add GeoJSON to a GeoLibre map*

Description

Add GeoJSON to a GeoLibre map

Usage

```
add_geojson(
  map,
  data,
  name = "GeoJSON",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`data`	GeoJSON as a parsed list (a `FeatureCollection`, `Feature`, or bare geometry), a JSON string, a file path, an HTTP(S) URL, or an `sf` object. A URL or file is read and inlined into the project, up to a 50 MB limit; for larger datasets prefer `add_vector()`, which lets the browser stream the source.
`name`	Layer name.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

See Also

`add_sf()`, `add_choropleth()`, `add_vector()`

Examples

```
point <- list(
  type = "Feature",
  properties = list(name = "Washington, DC"),
  geometry = list(type = "Point", coordinates = c(-77.0369, 38.9072))
)
map <- geolibre() |> add_geojson(point, name = "Places", fillColor = "#dc2626")
stopifnot(length(map$x$project$layers) == 1L)
```

---

add_geoparquet *Add a GeoParquet layer*

Description

Add a GeoParquet layer

Usage

```
add_geoparquet(map, data, name = "GeoParquet", ...)
```

Arguments

map	A GeoLibre widget.
data	A dataset URL, a local file path, or an sf object.
name	Layer name.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
geolibre() |> add_geoparquet("https://example.com/data.parquet")
```

add_gpkg *Add a GeoPackage layer*

Description

Add a GeoPackage layer

Usage

```
add_gpkg(map, data, name = "GeoPackage", layer = NULL, ...)
```

Arguments

map	A GeoLibre widget.
data	A dataset URL, a local file path, or an sf object.
name	Layer name.
layer	Optional table name inside the GeoPackage.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
geolibre() |> add_gpkg("https://example.com/data.gpkg", layer = "parcels")
```

add_heatmap *Add a point density heatmap*

Description

Add a point density heatmap

Usage

```
add_heatmap(
  map,
  points,
  name = "Heatmap",
  radius = 30,
  intensity = 1,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
points	Points as a two-column matrix or a data frame of coordinates, a list of `c(longitude, latitude)` pairs or named `list(lng = , lat = , ...)` entries, a point GeoJSON source, or an `sf` object of points.
name	Layer name.
radius	Heatmap kernel radius in pixels.
intensity	Heatmap intensity multiplier.
style	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_heatmap(list(c(-77, 39), c(-77.05, 39.02)), radius = 40)
stopifnot(map$x$project$layers[[1]]$style$pointRenderer == "heatmap")
```

add_kml *Add a KML or KMZ layer*

Description

Add a KML or KMZ layer

Usage

```
add_kml(map, data, name = "KML", ...)
```

Arguments

map	A GeoLibre widget.
data	A dataset URL, a local file path, or an sf object.
name	Layer name.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
geolibre() |> add_kml("https://example.com/places.kml")
```

add_legend *Add a legend to the map*

Description

Supply the legend entries exactly one of three ways: a built-in preset (builtin), a named vector or list of label-to-color pairs (legend), or parallel labels and colors vectors. Each call adds another legend, so a map can carry several at once.

Usage

```
add_legend(
  map,
  title = NULL,
  legend = NULL,
  labels = NULL,
  colors = NULL,
  builtin = NULL,
  position = c("bottom-left", "bottom-right", "top-left", "top-right"),
  shape = c("square", "circle", "line")
)
```

Arguments

map	A GeoLibre widget.
title	Legend title. Defaults to "Legend", or the preset's own title when builtin is supplied without one.
legend	A named vector or list mapping label to CSS color. Order is preserved.
labels	Item labels, paired position-wise with colors.
colors	Item CSS colors, paired position-wise with labels.
builtin	A built-in preset name from builtin_legend_names().
position	Corner for the legend: "top-left", "top-right", "bottom-left", or "bottom-right".
shape	Swatch shape for every item: "square", "circle", or "line".

Value

The modified widget.

See Also

add_colorbar() for continuous rasters, builtin_legend_names()

Examples

```
map <- geolibre() |>
  add_legend(
    "Land cover",
    legend = c(Water = "#466b9f", Forest = "#1c5f2c"),
    position = "bottom-left"
  )

# A built-in preset carries its own title and colors.
geolibre() |> add_legend(builtin = "nlcd")
```

add_marker *Add a single point marker*

Description

Add a single point marker

Usage

```
add_marker(
  map,
  lng,
  lat,
  name = "Marker",
  properties = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
lng	Marker longitude.
lat	Marker latitude.
name	Layer name.
properties	Optional named list of feature properties, shown when the point is clicked.
style	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77.0369, 38.9072, name = "DC", circleRadius = 10)
```

add_markers *Add point markers*

Description

Add point markers

Usage

```
add_markers(
  map,
  points,
  name = "Markers",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
points	Points as a two-column matrix or a data frame of coordinates, a list of `c(longitude, latitude)` pairs or named `list(lng = , lat = , ...)` entries, a point GeoJSON source, or an `sf` object of points.
name	Layer name.
style	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

See Also

`add_circle_markers()`, `add_marker_cluster()`, `add_heatmap()`

Examples

```
map <- geolibre() |>
  add_markers(list(c(-77.0369, 38.9072), c(-74.006, 40.7128)), name = "Cities")
stopifnot(length(map$x$project$layers[[1]]$geojson$features) == 2L)
```

`add_marker_cluster` *Add clustered point markers*

Description

Builds a point layer with the cluster renderer enabled, so nearby points collapse into count bubbles that split apart as the map zooms in.

Usage

```
add_marker_cluster(
  map,
  points,
  name = "Marker Cluster",
  cluster_radius = 50,
  cluster_max_zoom = 14,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`points`	Points as a two-column matrix or a data frame of coordinates, a list of `c(longitude, latitude)` pairs or named `list(lng = , lat = , ...)` entries, a point GeoJSON source, or an `sf` object of points.
`name`	Layer name.
`cluster_radius`	Cluster radius in pixels.
`cluster_max_zoom`	Zoom level beyond which points are no longer clustered.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker_cluster(list(c(-77, 39), c(-77.1, 39.1)), cluster_radius = 60)
stopifnot(map$x$project$layers[[1]]$style$pointRenderer == "cluster")
```

`add_pmtiles` *Add a PMTiles layer*

Description

The application registers the `pmtiles://` protocol itself, so a plain `https://` URL to the archive is what this expects.

Usage

```
add_pmtiles(
  map,
  url,
  name = "PMTiles",
  tile_type = c("vector", "raster"),
  source_layers = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`url`	URL of the `.pmtiles` archive.
`name`	Layer name.
`tile_type`	`"vector"` or `"raster"`.
`source_layers`	Vector source layer names to render. Vector tiles only.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_pmtiles("https://example.com/data.pmtiles", source_layers = "buildings")
stopifnot(map$x$project$layers[[1]]$type == "pmtiles")
```

`add_raster` *Add a remote raster to a GeoLibre map*

Description

`add_cog()` with a generic default layer name.

Usage

```
add_raster(
  map,
  url,
  name = "Raster",
  bands = NULL,
  colormap = NULL,
  rescale = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`url`	Public HTTP(S) URL of a Cloud Optimized GeoTIFF or GeoTIFF.
`name`	Layer name.
`bands`	Optional one-based band indices. Three or more bands render as RGB; one renders as a single band, which `colormap` then colors.
`colormap`	Optional GeoLibre colormap name for single-band rendering.
`rescale`	Optional list of numeric `c(min, max)` ranges, one per rendered band. A single `c(min, max)` pair is accepted for the one-band case.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_raster("https://example.com/image.tif", bands = c(1, 2, 3))
stopifnot(map$x$project$layers[[1]]$type == "cog")
```

---

add_sf *Add an* sf *object to a GeoLibre map*

Description

The object is transformed to EPSG:4326 before serialization. An object with no CRS is taken to be in longitude/latitude order already, since that is what GeoJSON means.

Usage

```
add_sf(
  map,
  data,
  name = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
data	An sf, sfc, or sfg object.
name	Layer name. Defaults to the expression passed as data.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  point <- sf::st_sf(
    name = "Washington, DC",
    geometry = sf::st_sfc(sf::st_point(c(-77.0369, 38.9072)), crs = 4326)
  )
  map <- geolibre() |> add_sf(point, name = "Places")
}
```

---

add_shp *Add a Shapefile layer*

Description

Add a Shapefile layer

Usage

```
add_shp(map, data, name = "Shapefile", ...)
```

Arguments

map	A GeoLibre widget.
data	A zipped Shapefile URL, or a local .shp path read with sf.
name	Layer name.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
geolibre() |> add_shp("https://example.com/data.zip")
```

`add_tile_layer` *Add an XYZ raster tile layer*

Description

Add an XYZ raster tile layer

Usage

```
add_tile_layer(
  map,
  url,
  name = "Tile Layer",
  tile_size = 256,
  attribution = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`url`	An XYZ tile URL template containing `{z}`, `{x}`, and `{y}`.
`name`	Layer name.
`tile_size`	Tile size in pixels, typically 256.
`attribution`	Optional attribution string.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_tile_layer(
    "https://tile.openstreetmap.org/{z}/{x}/{y}.png",
    name = "OpenStreetMap",
    attribution = "OpenStreetMap contributors"
  )
stopifnot(map$x$project$layers[[1]]$type == "xyz")
```

`add_vector` *Add a vector dataset from a URL or local file*

Description

A remote URL is handed to the application's in-browser vector control, so any format it reads (GeoParquet, FlatGeobuf, zipped Shapefile, GeoPackage, GeoJSON, KML, ...) streams without being inlined in the project. A local file is read with `sf` and inlined as GeoJSON, since the browser cannot reach a file on this machine.

Usage

```
add_vector(
  map,
  data,
  name = "Vector",
  render_mode = c("geojson", "tiles"),
  data_format = NULL,
  source_layer = NULL,
  picker = NULL,
  ingest_mode = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`data`	A dataset URL, a local file path, or an `sf` object.
`name`	Layer name.
`render_mode`	`"geojson"` to load into a GeoJSON source, or `"tiles"` to stream as vector tiles. Remote URLs only.
`data_format`	Optional format hint for remote URLs, for example `"parquet"` or `"flatgeobuf"`. The control auto-detects when omitted.

source_layer	Optional layer or table name inside a multi-layer container such as a GeoPackage.
picker	Optional toggle for the control's feature-inspection popup.
ingest_mode	Optional ingest strategy, "table" or "stream".
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_vector("https://example.com/data.parquet", name = "Parcels")
stopifnot(map$x$project$layers[[1]]$metadata$sourceKind == "maplibre-gl-vector")
```

---

add_vector_tiles *Add a vector tile layer from a TileJSON endpoint*

Description

Add a vector tile layer from a TileJSON endpoint

Usage

```
add_vector_tiles(
  map,
  url,
  name = "Vector Tiles",
  source_layers = NULL,
  source_layer = NULL,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
url	TileJSON endpoint for the vector tileset.
name	Layer name.
source_layers	Source layer names to render, for multi-layer tilesets.
source_layer	A single source layer name, for the common single-layer case. Ignored when source_layers is supplied.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_vector_tiles("https://example.com/tiles.json", source_layer = "roads")
stopifnot(map$x$project$layers[[1]]$type == "vector-tiles")
```

---

add_video *Add a georeferenced video layer*

Description

Add a georeferenced video layer

Usage

```
add_video(
  map,
  urls,
  coordinates,
  name = "Video",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
urls	One video URL, or several as format fallbacks such as MP4 then WebM. URLs must be https://, since the browser's media policy blocks plain HTTP.
coordinates	Four c(longitude, latitude) corners in top-left, top-right, bottom-right, bottom-left order, given as a list of pairs or a four-row matrix.
name	Layer name.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_video(
    "https://example.com/clip.mp4",
    coordinates = list(
      c(-77.1, 39.0), c(-77.0, 39.0), c(-77.0, 38.9), c(-77.1, 38.9)
    )
  )
stopifnot(map$x$project$layers[[1]]$type == "video")
```

---

add_wfs *Add a WFS layer*

Description

The GetFeature response is fetched and inlined into the project, so the endpoint must be able to return GeoJSON. This function contacts the service when called.

Usage

```
add_wfs(
  map,
  endpoint,
  type_name,
  name = "WFS Layer",
```

```
  version = "2.0.0",
  output_format = "application/json",
  srs_name = "EPSG:4326",
  max_features = 1000,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

map	A GeoLibre widget.
endpoint	WFS service endpoint.
type_name	WFS feature type name, for example "topp:states".
name	Layer name.
version	WFS protocol version, for example "2.0.0" or "1.1.0".
output_format	Requested output format; must yield GeoJSON.
srs_name	Spatial reference of the response.
max_features	Cap on the number of returned features, since the response is inlined. Pass NULL to request every feature.
style	Named list of GeoLibre style overrides such as fillColor, strokeColor, and strokeWidth.
visible	Whether the layer is initially visible.
opacity	Layer opacity from zero to one.
...	Additional style overrides given as named arguments, merged into style. add_geojson(map, data, fillColor = "red") and add_geojson(map, data, style = list(fillColor = "red")) are equivalent.

Value

The modified widget.

Examples

```
## Not run:
geolibre() |>
  add_wfs("https://ahocevar.com/geoserver/wfs", type_name = "topp:states")

## End(Not run)
```

add_wms *Add a WMS layer*

Description

The layer is rendered as tiled raster from WMS `GetMap` requests, built exactly as the application's Add Data dialog builds them.

Usage

```
add_wms(
  map,
  endpoint,
  layers,
  name = "WMS Layer",
  styles = "",
  image_format = "image/png",
  transparent = TRUE,
  tile_size = 256,
  version = "1.1.1",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`endpoint`	WMS service endpoint, the `GetMap` base URL.
`layers`	Comma-separated WMS layer name(s).
`name`	Layer name.
`styles`	Comma-separated WMS style name(s); empty for the server default.
`image_format`	WMS image format, for example `"image/png"`.
`transparent`	Whether to request transparent tiles.
`tile_size`	Tile size in pixels.
`version`	WMS protocol version, `"1.1.1"` or `"1.3.0"`. Version 1.3.0 sends `CRS` instead of `SRS`; some servers accept only one version.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_wms(
    "https://example.com/geoserver/wms",
    layers = "topp:states",
    name = "States"
  )
stopifnot(map$x$project$layers[[1]]$type == "wms")
```

`add_wmts` *Add a WMTS layer*

Description

Add a WMTS layer

Usage

```
add_wmts(
  map,
  url,
  name = "WMTS Layer",
  tile_size = 256,
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`url`	A WMTS tile URL template in WMTS REST `{z}/{y}/{x}` order, with the row before the column. This differs from the `{z}/{x}/{y}` templates `add_tile_layer()` expects.
`name`	Layer name.
`tile_size`	Tile size in pixels.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_wmts("https://example.com/wmts/layer/{z}/{y}/{x}.png", name = "Imagery")
stopifnot(map$x$project$layers[[1]]$type == "wmts")
```

`add_xy_data` *Add points from tabular longitude and latitude columns*

Description

Add points from tabular longitude and latitude columns

Usage

```
add_xy_data(
  map,
  data,
  x = "longitude",
  y = "latitude",
  name = "XY Data",
  style = list(),
  visible = TRUE,
  opacity = 1,
  ...
)
```

Arguments

`map`	A GeoLibre widget.
`data`	A data frame, a CSV file path, a CSV URL, CSV text, or a list of row lists.
`x`	Name of the longitude column.
`y`	Name of the latitude column.
`name`	Layer name.
`style`	Named list of GeoLibre style overrides such as `fillColor`, `strokeColor`, and `strokeWidth`.
`visible`	Whether the layer is initially visible.
`opacity`	Layer opacity from zero to one.
`...`	Additional style overrides given as named arguments, merged into `style`. `add_geojson(map, data, fillColor = "red")` and `add_geojson(map, data, style = list(fillColor = "red"))` are equivalent.

Value

The modified widget.

Examples

```
cities <- data.frame(
  name = c("Washington", "New York"),
  longitude = c(-77.0369, -74.006),
  latitude = c(38.9072, 40.7128)
)
map <- geolibre() |> add_xy_data(cities, name = "Cities")
stopifnot(length(map$x$project$layers[[1]]$geojson$features) == 2L)
```

---

basemaps *Named GeoLibre basemaps*

Description

Lists the vector basemap styles that `set_basemap()` and `geolibre()` accept by name. Raster tile basemaps such as OpenStreetMap are added as layers with `add_tile_layer()` instead.

Usage

```
basemaps()
```

Value

A named character vector mapping basemap name to MapLibre style URL.

Examples

```
basemaps()
names(basemaps())
```

---

builtin_legend_names *Built-in legend preset names*

Description

The preset names accepted by the `builtin` argument of `add_legend()`.

Usage

```
builtin_legend_names()
```

Value

A character vector of preset names.

Examples

```
builtin_legend_names()
```

`classify_layer` *Symbolize an existing layer as a choropleth*

Description

Symbolize an existing layer as a choropleth

Usage

```
classify_layer(
  map,
  layer,
  column,
  class_count = 5,
  colormap = "viridis",
  scheme = c("equal-interval", "quantile")
)
```

Arguments

`map`	A GeoLibre widget.
`layer`	A layer id or layer name. The layer must carry inlined GeoJSON.
`column`	Name of the numeric feature property to classify.
`class_count`	Number of classes, clamped to between 2 and 12.
`colormap`	A color ramp name from `color_ramp_names()`.
`scheme`	Classification scheme, `"equal-interval"` or `"quantile"`.

Value

The modified widget.

See Also

`add_choropleth()` to add and symbolize in one call.

Examples

```
counties <- list(
  type = "FeatureCollection",
  features = list(
    list(
      type = "Feature", properties = list(pop = 10),
      geometry = list(type = "Point", coordinates = c(-77, 39))
    ),
    list(
      type = "Feature", properties = list(pop = 90),
      geometry = list(type = "Point", coordinates = c(-76, 40))
    )
  )
)
map <- geolibre() |>
  add_geojson(counties, name = "Counties") |>
  classify_layer("Counties", "pop", colormap = "reds")
stopifnot(map$x$project$layers[[1]]$style$vectorStyleProperty == "pop")
```

---

clear_layers *Remove every layer*

Description

Remove every layer

Usage

```
clear_layers(map)
```

Arguments

map	A GeoLibre widget.

Value

The modified widget.

Examples

```
map <- geolibre() |> add_marker(-77, 39) |> clear_layers()
stopifnot(length(map$x$project$layers) == 0L)
```

color_ramp_names *Available color ramp names*

Description

The ramp names accepted by `add_choropleth()`, `classify_layer()`, and `add_colorbar()`.

Usage

```
color_ramp_names()
```

Value

A character vector of ramp names.

Examples

```
color_ramp_names()
```

column_values *Read one feature property across a layer*

Description

Read one feature property across a layer

Usage

```
column_values(x, layer, column)
```

Arguments

`x`	A GeoLibre widget or a project list.
`layer`	A layer id or layer name.
`column`	The feature property name.

Value

A list of the raw values, one per feature, with `NULL` where the property is absent.

Examples

```
point <- list(
  type = "Feature", properties = list(pop = 700000),
  geometry = list(type = "Point", coordinates = c(-77, 39))
)
map <- geolibre() |> add_geojson(point, name = "Places")
column_values(map, "Places", "pop")
```

describe_project *Summarize a GeoLibre project*

Description

Reports the camera, basemap, layers, and active map controls. URLs come back with their credentials stripped, since several basemap providers put an API key in the style URL itself.

Usage

```
describe_project(project)
```

Arguments

project	A GeoLibre widget or a project list.

Value

A list with the project `name`, `version`, `mapView`, `basemapStyleUrl`, `layerCount`, a `layers` data frame, and the names of the active `mapControls`.

See Also

`get_layers()`

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  add_legend(legend = c(Pin = "#3b82f6"))
summary <- describe_project(map)
summary$layerCount
summary$mapControls
```

duplicate_layer *Duplicate a layer*

Description

The copy is appended to the top of the draw order, where a newly added layer lands. Use `move_layer()` to put it elsewhere.

Usage

```
duplicate_layer(map, layer, name = NULL)
```

Arguments

map	A GeoLibre widget.
layer	A layer id or layer name.
name	Name for the copy. Defaults to the source name followed by " copy".

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  duplicate_layer("Pin")
stopifnot(layer_names(map)[[2]] == "Pin copy")
```

---

find_layer_index *Position of a layer in the draw order*

Description

Position of a layer in the draw order

Usage

```
find_layer_index(x, name)
```

Arguments

x	A GeoLibre widget or a project list.
name	A layer id or layer name.

Value

The one-based index of the first matching layer, or -1 when none matches. Unlike the functions that modify a layer, a name several layers share resolves to the first of them rather than raising.

Examples

```
map <- geolibre() |> add_marker(-77, 39, name = "Pin")
find_layer_index(map, "Pin")
find_layer_index(map, "Missing")
```

fit_bounds *Frame a bounding box*

Description

A saved project records a center and zoom rather than a box to fit: the application applies `mapView$center` and `mapView$zoom` verbatim when it opens a project. So the box is resolved to a camera here, using an assumed viewport, and recorded alongside it for reference. The result is approximate by construction; expect the application's own "zoom to layer" to land within roughly half a zoom level.

Usage

```
fit_bounds(map, bbox, padding = 40)
```

Arguments

`map`	A GeoLibre widget.
`bbox`	`c(west, south, east, north)` bounds. Per RFC 7946, a west greater than the east means the box crosses the antimeridian, and is framed as such.
`padding`	Pixels of margin to leave around the box.

Value

The modified widget.

Examples

```
map <- geolibre() |> fit_bounds(c(-125, 24, -66, 50))
round(map$x$project$mapView$zoom, 1)
```

geolibre *Create a GeoLibre widget*

Description

Creates an `htmlwidget` that embeds the GeoLibre geographic information system. The widget carries a `.geolibre.json` project, which the `add_*()`, `set_*()`, and control functions build up with the pipe.

Usage

```
geolibre(
  project = NULL,
  center = NULL,
  zoom = NULL,
  basemap = NULL,
  name = "Untitled Project",
  width = NULL,
  height = NULL,
  app_url = getOption("geolibre.app_url", "https://web.geolibre.app/"),
  layout = c("embed", "full", "maponly"),
  theme = c("light", "dark"),
  map_only = FALSE,
  elementId = NULL,
  panels = c("expanded", "collapsed", "hidden")
)
```

Arguments

project	A GeoLibre project list, a path to a .geolibre.json file, or NULL for a new project.
center	Optional initial c(longitude, latitude) map center.
zoom	Optional initial zoom level.
basemap	Optional basemap name from basemaps() or a MapLibre style JSON URL. Ignored when project is supplied, which carries its own.
name	Project name recorded in the project file.
width, height	Widget dimensions passed to htmlwidgets::createWidget().
app_url	URL of a GeoLibre web deployment. It must support the ?embed=1 project bridge.
layout	Application chrome to show: "embed" (compact controls), "full" (the complete desktop interface), or "maponly" (map only).
theme	Application theme, "light" or "dark".
map_only	Deprecated. TRUE is equivalent to layout = "maponly".
elementId	Optional widget element ID.
panels	Initial side-panel state: "expanded", "collapsed" (icon rails remain available), or "hidden".

Details

The default hosted application requires internet access when the widget is displayed. Package installation, project construction, and file operations do not contact it. Set app_url or the geolibre.app_url option to use a self-hosted deployment.

Value

An htmlwidget that can be modified with add_*() functions.

See Also

add_geojson(), set_view(), save_project()

Examples

```
map <- geolibre(center = c(-77.0369, 38.9072), zoom = 10, layout = "maponly")
stopifnot(inherits(map, "geolibre"))

# A dark-themed map with the full application interface.
geolibre(basemap = "dark", layout = "full")
```

---

geolibreOutput *Shiny bindings for GeoLibre*

Description

Shiny bindings for GeoLibre

Usage

```
geolibreOutput(outputId, width = "100%", height = "700px")

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

Arguments

outputId	Output variable to read from.
width, height	Widget dimensions.
expr	An expression that generates a GeoLibre widget.
env	Environment in which to evaluate expr.
quoted	Whether expr is quoted.

Value

geolibreOutput() returns a Shiny output element; renderGeolibre() returns a render function for it.

Shiny inputs

A rendered widget reports back through four inputs, where id is the outputId:

- input$id_project — the full project each time the user edits the map.
- input$id_error — the message from a project the application rejected.
- input$id_result — the reply to a proxy command; a list with requestId, method, ok, and then value or error.
- input$id_event — user interaction; a list with event, one of "click", "selection-change", or "layer-change", and its payload.

Examples

```
if (requireNamespace("shiny", quietly = TRUE)) {
  output <- geolibreOutput("map", height = "500px")
  stopifnot(inherits(output, "shiny.tag.list"))
}
```

`geolibre_command` *Send a command to a live GeoLibre map*

Description

The escape hatch behind the `geolibre_*()` command functions: it forwards any method the application's scripting bridge implements. The reply arrives asynchronously on the `input$id_result` input described in `geolibreOutput()`.

Usage

```
geolibre_command(proxy, method, params = list(), request_id = NULL)
```

Arguments

`proxy`	A GeoLibre proxy created by `geolibre_proxy()`.
`method`	The scripting method name, for example `"flyTo"` or `"toImage"`.
`params`	Named list of parameters for the method.
`request_id`	Optional id echoed back with the reply, so several in-flight commands can be told apart. Generated when omitted.

Value

The proxy, invisibly.

See Also

`geolibre_fly_to()`, `geolibre_get_view()`, `geolibre_run_algorithm()`

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_command(geolibre_proxy("map"), "zoomToLayer", list(layerId = "abc"))
}
```

geolibre_fit_bounds *Fit a live GeoLibre map to a bounding box*

Description

Fit a live GeoLibre map to a bounding box

Usage

```
geolibre_fit_bounds(proxy, bbox, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
bbox	c(west, south, east, north) bounds.
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_fit_bounds(geolibre_proxy("map"), c(-125, 24, -66, 50))
}
```

geolibre_fly_to *Animate the camera of a live GeoLibre map*

Description

Unlike set_view(), which records the camera a project opens at, this animates the map that is already on screen. Omitted fields keep their current value.

Usage

```
geolibre_fly_to(
  proxy,
  center = NULL,
  zoom = NULL,
  bearing = NULL,
  pitch = NULL,
  duration = NULL,
  request_id = NULL
)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
center	Optional c(longitude, latitude) target.
zoom	Optional target zoom level.
bearing	Optional target bearing in degrees.
pitch	Optional target pitch in degrees.
duration	Optional animation duration in milliseconds.
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_fly_to(geolibre_proxy("map"), center = c(-77.0369, 38.9072), zoom = 12)
}
```

---

geolibre_get_view *Read the camera of a live GeoLibre map*

Description

The reply arrives on input$id_result with a value holding center, zoom, bearing, pitch, and the current bbox.

Usage

```
geolibre_get_view(proxy, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_get_view(geolibre_proxy("map"))
}
```

geolibre_identify *Identify features at a point on a live GeoLibre map*

Description

Identify features at a point on a live GeoLibre map

Usage

```
geolibre_identify(proxy, lnglat, layer_id = NULL, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
lnglat	c(longitude, latitude) of the point to query.
layer_id	Optional layer id to restrict the query to.
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_identify(geolibre_proxy("map"), c(-77.0369, 38.9072))
}
```

geolibre_layer_features *Read features from a live GeoLibre map*

Description

geolibre_layer_features() returns one layer's features, geolibre_selected_features() the current selection, and geolibre_drawn_features() whatever the user drew with the map's editing tools. Each reply arrives on input$id_result.

Usage

```
geolibre_layer_features(proxy, layer_id, request_id = NULL)

geolibre_selected_features(proxy, request_id = NULL)

geolibre_drawn_features(proxy, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
layer_id	The layer's id, as reported by get_layers().
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_drawn_features(geolibre_proxy("map"))
}
```

---

geolibre_proxy *Create a GeoLibre Shiny proxy*

Description

A proxy addresses a widget that is already on screen, so a Shiny app can replace its project with update_geolibre() or drive the live map with the geolibre_*() command functions, without re-rendering the widget.

Usage

```
geolibre_proxy(outputId, session = NULL)
```

Arguments

outputId	ID of an existing GeoLibre widget.
session	A Shiny session. Defaults to the current reactive domain.

Value

A geolibre_proxy object.

See Also

update_geolibre(), geolibre_fly_to(), geolibre_command()

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  proxy <- geolibre_proxy("map")
}
```

geolibre_run_algorithm

Run a processing algorithm on a live GeoLibre map

Description

`geolibre_list_algorithms()` reports the available tools with their parameters; `geolibre_run_algorithm()` runs one in the browser. Results arrive on `input$id_result`, and any output layers are added to the map.

Usage

```
geolibre_run_algorithm(proxy, algorithm, params = list(), request_id = NULL)

geolibre_list_algorithms(proxy, request_id = NULL)
```

Arguments

`proxy`	A GeoLibre proxy created by `geolibre_proxy()`.
`algorithm`	The algorithm id, as reported by `geolibre_list_algorithms()`.
`params`	Named list of algorithm parameters.
`request_id`	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  proxy <- geolibre_proxy("map")
  geolibre_list_algorithms(proxy)
  geolibre_run_algorithm(proxy, "buffer", list(layerId = "abc", distance = 500))
}
```

geolibre_to_image

Capture a live GeoLibre map as a PNG

Description

The reply arrives on `input$id_result` with a `value` holding a `...` URL. Strip the prefix up to the comma and pass the rest to `jsonlite::base64_dec()` to recover the PNG bytes.

Usage

```
geolibre_to_image(proxy, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_to_image(geolibre_proxy("map"))
}
```

---

geolibre_zoom_to_layer

Zoom a live GeoLibre map to a layer's extent

Description

Zoom a live GeoLibre map to a layer's extent

Usage

```
geolibre_zoom_to_layer(proxy, layer_id, request_id = NULL)
```

Arguments

proxy	A GeoLibre proxy created by geolibre_proxy().
layer_id	The layer's id, as reported by get_layers().
request_id	Optional id echoed back with the reply.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  geolibre_zoom_to_layer(geolibre_proxy("map"), "layer-id")
}
```

get_color_ramp *Anchor colors of a color ramp*

Description

Anchor colors of a color ramp

Usage

```
get_color_ramp(name = "viridis")
```

Arguments

name	A ramp name from color_ramp_names(). An unknown name falls back to "viridis", matching the application's own lookup.

Value

A character vector of #rrggbb colors.

Examples

```
get_color_ramp("viridis")
get_color_ramp("blues")
```

get_layer *Read one layer's full definition*

Description

Read one layer's full definition

Usage

```
get_layer(x, layer)
```

Arguments

x	A GeoLibre widget or a project list.
layer	A layer id or layer name.

Value

The layer as a list, with credential-bearing fields stripped.

Examples

```
map <- geolibre() |> add_marker(-77, 39, name = "Pin")
get_layer(map, "Pin")$type
```

`get_layers` *Summarize a project's layers*

Description

Summarize a project's layers

Usage

```
get_layers(x)
```

Arguments

`x`	A GeoLibre widget or a project list.

Value

A data frame with one row per layer, in draw order, holding its `id`, `name`, `type`, `visible`, `opacity`, `source` URL (credentials stripped), and inlined `features` count.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  add_raster("https://example.com/image.tif", name = "Image")
get_layers(map)
```

`get_project` *Read a project out of a widget*

Description

Read a project out of a widget

Usage

```
get_project(map, keep_credentials = FALSE)
```

Arguments

`map`	A GeoLibre widget or project list.
`keep_credentials`	Keep credential-bearing configuration. Defaults to `FALSE`, so the returned project is safe to print or serialize.

Value

The project as a list.

Examples

```
map <- geolibre() |> add_marker(-77, 39, name = "Pin")
project <- get_project(map)
project$layers[[1]]$name
```

`interpolate_ramp_colors` *Sample a color ramp into evenly spaced colors*

Description

Mirrors the application's ramp interpolation, so a palette generated here matches the one the Style panel would compute.

Usage

```
interpolate_ramp_colors(name = "viridis", count = 5)
```

Arguments

`name`	A ramp name from `color_ramp_names()`.
`count`	Number of colors to produce.

Value

A character vector of `count` `#rrggbb` colors.

Examples

```
interpolate_ramp_colors("viridis", 5)
```

layer_names *Layer names in draw order*

Description

Layer names in draw order

Usage

```
layer_names(x)
```

Arguments

x	A GeoLibre widget or a project list.

Value

A character vector of layer names, bottom layer first.

Examples

```
map <- geolibre() |> add_marker(-77, 39, name = "Pin")
layer_names(map)
```

layer_properties *Sample a layer's feature properties*

Description

Lets you discover what an inlined vector layer can be styled or filtered by without reading every feature back.

Usage

```
layer_properties(x, layer)
```

Arguments

x	A GeoLibre widget or a project list.
layer	A layer id or layer name.

Value

A named list mapping each property name to up to 25 distinct sample values, in first-seen order.

Examples

```
point <- list(
  type = "Feature", properties = list(name = "DC", pop = 700000),
  geometry = list(type = "Point", coordinates = c(-77, 39))
)
map <- geolibre() |> add_geojson(point, name = "Places")
layer_properties(map, "Places")
```

---

load_project *Read a GeoLibre project*

Description

Read a GeoLibre project

Usage

```
load_project(source)
```

Arguments

source	Path to a `.geolibre.json` file or a JSON string.

Value

A project list.

See Also

save_project(), describe_project()

Examples

```
project <- load_project('{"version":"0.2.0","name":"Example","mapView":{}}')
stopifnot(project$name == "Example")

# A saved project can be reopened as a widget.
path <- tempfile(fileext = ".geolibre.json")
save_project(geolibre() |> add_marker(-77, 39), path)
map <- geolibre(load_project(path))
```

move_layer *Move a layer in the draw order*

Description

Move a layer in the draw order

Usage

```
move_layer(map, layer, index)
```

Arguments

map	A GeoLibre widget.
layer	A layer id or layer name.
index	One-based destination position, counted from the bottom of the draw order. Negative values count from the top, so -1 moves the layer to the very top. Out-of-range values are clamped.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Bottom") |>
  add_marker(-76, 40, name = "Top") |>
  move_layer("Top", 1)
stopifnot(layer_names(map)[[1]] == "Top")
```

redact_credentials *Strip credentials from a whole project*

Description

Returns a project safe to publish, export, or hand to someone else: layer request headers and signed URLs, basemap style keys, geocoding keys, stored environment variables, and third-party plugin settings are all removed. The first-party map controls (legend, colorbar, swipe) are kept, since a project needs them to render as it was built.

Usage

```
redact_credentials(project)
```

Arguments

project	A GeoLibre widget or a project list.

Details

save_project() applies this by default.

Value

The project list with credentials removed.

Examples

```
map <- geolibre() |> add_marker(-77, 39)
safe <- redact_credentials(map)
safe$name
```

---

redact_layer *Strip credentials from one layer*

Description

Strip credentials from one layer

Usage

```
redact_layer(layer)
```

Arguments

layer	A layer list, as returned inside a project's layers.

Value

The layer with its credential-bearing configuration removed.

Examples

```
map <- geolibre() |>
  add_3d_tiles(
    "https://example.com/tileset.json",
    request_headers = list(Authorization = "Bearer secret")
  )
layer <- redact_layer(map$x$project$layers[[1]])
is.null(layer$source$requestHeaders)
```

redact_url *Strip credentials from a URL*

Description

Removes any `user:password@` prefix and any query parameter whose name marks it as a credential, such as `api_key`, `access_token`, or an Azure shared-access signature. The rest of the URL is left byte-for-byte intact.

Usage

```
redact_url(url)
```

Arguments

`url`	A URL string.

Value

The URL with its credentials removed.

Examples

```
redact_url("https://tiles.example.com/style.json?api_key=secret&lang=en")
redact_url("https://user:pw@example.com/data.tif")
```

remove_layer *Remove a layer*

Description

Remove a layer

Usage

```
remove_layer(map, layer)
```

Arguments

`map`	A GeoLibre widget.
`layer`	A layer id or layer name.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  remove_layer("Pin")
stopifnot(length(map$x$project$layers) == 0L)
```

`rename_layer` *Rename a layer*

Description

Rename a layer

Usage

```
rename_layer(map, layer, name)
```

Arguments

`map`	A GeoLibre widget.
`layer`	A layer id or layer name.
`name`	The new display name. Surrounding whitespace is stripped.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  rename_layer("Pin", "Capital")
stopifnot(layer_names(map) == "Capital")
```

`save_project` *Save a GeoLibre project*

Description

Save a GeoLibre project

Usage

```
save_project(map, path, keep_credentials = FALSE)
```

Arguments

map	A GeoLibre widget or project list.
path	Output path, conventionally ending in .geolibre.json.
keep_credentials	Keep credential-bearing configuration such as layer request headers and signed URLs. Defaults to FALSE, so a saved project is safe to commit or share. See redact_credentials().

Value

path, invisibly.

See Also

load_project(), redact_credentials()

Examples

```
path <- tempfile(fileext = ".geolibre.json")
save_project(geolibre(), path)
project <- load_project(path)
stopifnot(project$name == "Untitled Project")
```

---

set_basemap *Set the background basemap*

Description

Set the background basemap

Usage

```
set_basemap(map, basemap)

add_basemap(map, basemap)
```

Arguments

map	A GeoLibre widget.
basemap	A basemap name from basemaps() or a MapLibre style JSON URL.

Value

The modified widget.

See Also

basemaps(), add_tile_layer() for raster basemaps such as OpenStreetMap.

Examples

```
map <- geolibre() |> set_basemap("dark")
map$x$project$basemapStyleUrl
```

---

set_bearing *Set the camera bearing*

Description

Set the camera bearing

Usage

```
set_bearing(map, bearing)
```

Arguments

map	A GeoLibre widget.
bearing	Clockwise rotation in degrees.

Value

The modified widget.

Examples

```
map <- geolibre() |> set_bearing(45)
stopifnot(map$x$project$mapView$bearing == 45)
```

---

set_center *Center the map*

Description

Center the map

Usage

```
set_center(map, lng, lat, zoom = NULL)
```

Arguments

map	A GeoLibre widget.
lng	Longitude of the new center.
lat	Latitude of the new center.
zoom	Optional zoom level.

Value

The modified widget.

Examples

```
map <- geolibre() |> set_center(-77.0369, 38.9072, zoom = 11)
stopifnot(map$x$project$mapView$zoom == 11)
```

`set_layer_opacity` *Set a layer's opacity*

Description

Set a layer's opacity

Usage

```
set_layer_opacity(map, layer, opacity)
```

Arguments

`map`	A GeoLibre widget.
`layer`	A layer id or layer name.
`opacity`	Opacity from zero to one.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  set_layer_opacity("Pin", 0.4)
stopifnot(map$x$project$layers[[1]]$opacity == 0.4)
```

set_layer_style *Restyle a layer*

Description

Merges style overrides into a layer's existing style. Keys not mentioned keep their current values.

Usage

```
set_layer_style(map, layer, style = list(), ...)
```

Arguments

map	A GeoLibre widget.
layer	A layer id or layer name.
style	Named list of style keys to set.
...	Additional style overrides given as named arguments.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  set_layer_style("Pin", fillColor = "#f59e0b", circleRadius = 9)
stopifnot(map$x$project$layers[[1]]$style$fillColor == "#f59e0b")
```

set_layer_visibility *Show or hide a layer*

Description

Show or hide a layer

Usage

```
set_layer_visibility(map, layer, visible = TRUE)

show_layer(map, layer)

hide_layer(map, layer)
```

Arguments

map	A GeoLibre widget.
layer	A layer id or layer name.
visible	TRUE to show the layer, FALSE to hide it.

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_marker(-77, 39, name = "Pin") |>
  set_layer_visibility("Pin", FALSE)
stopifnot(isFALSE(map$x$project$layers[[1]]$visible))
```

---

set_pitch *Set the camera pitch*

Description

Set the camera pitch

Usage

```
set_pitch(map, pitch)
```

Arguments

map	A GeoLibre widget.
pitch	Tilt in degrees, clamped to between 0 and 85.

Value

The modified widget.

Examples

```
map <- geolibre() |> set_pitch(60)
stopifnot(map$x$project$mapView$pitch == 60)
```

set_project_name *Set the project name*

Description

Set the project name

Usage

```
set_project_name(map, name)
```

Arguments

map	A GeoLibre widget.
name	The project's display name, as shown in the application and stored in the project file.

Value

The modified widget.

Examples

```
map <- geolibre() |> set_project_name("Chesapeake Bay")
map$x$project$name
```

set_view *Set the GeoLibre camera*

Description

Set the GeoLibre camera

Usage

```
set_view(
  map,
  center = NULL,
  zoom = NULL,
  bearing = NULL,
  pitch = NULL,
  bbox = NULL
)
```

Arguments

map	A GeoLibre widget.
center	Optional c(longitude, latitude) pair.
zoom	Optional zoom level, clamped to between 0 and 24.
bearing	Optional clockwise rotation in degrees.
pitch	Optional tilt in degrees, clamped to between 0 and 85.
bbox	Optional c(west, south, east, north) bounds. When supplied it takes precedence over center and zoom, and is resolved to a center and zoom by fit_bounds().

Value

The modified widget.

See Also

fit_bounds(), set_center(), set_zoom()

Examples

```
map <- geolibre() |>
  set_view(center = c(-77.0369, 38.9072), zoom = 10, pitch = 30)
stopifnot(map$x$project$mapView$zoom == 10)
```

---

set_zoom *Set the map zoom*

Description

Set the map zoom

Usage

```
set_zoom(map, zoom)
```

Arguments

map	A GeoLibre widget.
zoom	Zoom level, clamped to between 0 and 24.

Value

The modified widget.

Examples

```
map <- geolibre() |> set_zoom(6)
stopifnot(map$x$project$mapView$zoom == 6)
```

split_map *Add a split-map comparison slider*

Description

Enables the Layer Swipe control, which clips one set of layers to one side of a draggable slider and another set to the other, for before-and-after comparisons.

Usage

```
split_map(
  map,
  left_layers = NULL,
  right_layers = NULL,
  orientation = c("vertical", "horizontal"),
  position = 50,
  control_position = c("top-left", "top-right", "bottom-left", "bottom-right")
)
```

Arguments

map	A GeoLibre widget.
left_layers	Layer ids or names shown on the left or top of the slider. The string "__basemap__" selects the basemap.
right_layers	Layer ids or names shown on the right or bottom.
orientation	"vertical" to move the slider left and right, or "horizontal" to move it up and down.
position	Initial slider position as a percentage from 0 to 100.
control_position	Corner for the swipe panel: "top-left", "top-right", "bottom-left", or "bottom-right".

Value

The modified widget.

Examples

```
map <- geolibre() |>
  add_raster("https://example.com/before.tif", name = "Before") |>
  add_raster("https://example.com/after.tif", name = "After") |>
  split_map("Before", "After")
stopifnot(length(map$x$project$plugins$settings) == 1L)
```

to_html *Export a map as a standalone HTML page*

Description

The page embeds the GeoLibre application in an `iframe` and injects the project into it over the same `postMessage` bridge the widget uses, so it renders the map as configured. Unlike the widget it needs no running R session; by default it loads the hosted application over the network so the file stays portable.

Usage

```
to_html(
  map,
  path = NULL,
  title = "GeoLibre Map",
  width = "100%",
  height = "800px",
  app_url = getOption("geolibre.app_url", "https://web.geolibre.app/")
)
```

Arguments

`map`	A GeoLibre widget or project list.
`path`	Optional output path. When supplied the page is written there and `path` is returned invisibly; otherwise the HTML is returned as a string.
`title`	The exported page's `title`.
`width`	CSS width of the embedded map, for example `"100%"` or `"800px"`.
`height`	CSS height of the embedded map.
`app_url`	Base URL of the GeoLibre application to embed. Defaults to the hosted viewer so the export stays portable; pass a self-hosted deployment URL to pin a specific version.

Details

Credentials are stripped from the inlined project on the way out, as `save_project()` does.

Value

The HTML string, or `path` invisibly when `path` is supplied.

See Also

`save_project()` to write the project itself

Examples

```
map <- geolibre() |> add_marker(-77.0369, 38.9072, name = "DC")
html <- to_html(map, title = "Washington, DC")
substr(html, 1, 15)

path <- tempfile(fileext = ".html")
to_html(map, path)
```

`update_geolibre` *Replace the project displayed by a GeoLibre Shiny widget*

Description

Replace the project displayed by a GeoLibre Shiny widget

Usage

```
update_geolibre(proxy, map)
```

Arguments

`proxy`	A GeoLibre proxy created by `geolibre_proxy()`.
`map`	A GeoLibre widget or project list.

Value

The proxy, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  update_geolibre(geolibre_proxy("map"), geolibre())
}
```

Index