

Package: ggplot2qgis (via r-universe)

October 5, 2026

Title Export 'ggplot2' as 'QGIS' Project

Version 0.0.0.9000

Description Convert a 'ggplot2'-based map plot (e.g. 'geom_sf()') or a 'tmap' map into a 'QGIS' project ('.qgs') file.

License GPL-2 + file LICENSE

URL <https://github.com/yutannihilation/ggplot2qgis>,
<https://yutannihilation.github.io/ggplot2qgis/>

BugReports <https://github.com/yutannihilation/ggplot2qgis/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports ggplot2 (>= 4.0.0), grDevices, isoband, rlang, sf (>= 1.0.0), tools

Suggests maptiles, scales, terra (>= 1.7-3), testthat (>= 3.0.0), tidyterra (>= 1.2.0), tmap (>= 4.4), withr, xml2

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository <https://yutannihilation.r-universe.dev>

Date/Publication 2026-09-05 09:42:25 UTC

RemoteUrl <https://github.com/yutannihilation/ggplot2qgis>

RemoteRef HEAD

RemoteSha 3cdf87aca2e55cfa889c3cbc079e53ea28c64557

Contents

write_qgs.tmap	2
Index	11

write_qgs.tmap

Write a ggplot2 or tmap map as a QGIS project

Description

Converts a ggplot2 plot whose layers are drawn from sf objects (or, for `ggplot2::geom_point()`, `ggplot2::geom_path()`, `ggplot2::geom_line()` and `ggplot2::geom_polygon()`, from plain data frames, or from a SpatRaster via `tidyterra::geom_spatraster()`, `tidyterra::geom_spatraster_rgb()`, `tidyterra::geom_spatraster_contour()`, `tidyterra::geom_spatraster_contour_text()` or `tidyterra::geom_spatraster_contour_filled()`) into a QGIS project (.qgs) file. A SpatVector layer, drawn with `tidyterra::geom_spatvector()` or with `geom_sf()` itself, counts as an sf layer (see *SpatVector layers*). The data of each layer is saved as a GeoPackage (a GeoTIFF for raster layers) under `<path minus extension>_data/`, and the layer is styled after the plot's trained color scale:

Usage

```
## S3 method for class 'tmap'
write_qgs(
  plot,
  path,
  use_plot_crs = TRUE,
  gradient_style = c("graduated", "continuous"),
  overwrite = FALSE,
  layer_names = NULL,
  basemap = NULL,
  create_na_layer = TRUE,
  ...
)

write_qgs(plot, path, ...)

## S3 method for class 'ggplot'
write_qgs(
  plot,
  path,
  use_plot_crs = FALSE,
  gradient_style = c("graduated", "continuous"),
  overwrite = FALSE,
  layer_names = NULL,
  basemap = NULL,
  ...
)
```

Arguments

`plot` A ggplot object whose layers are backed by sf data (or SpatVector data, see *SpatVector layers*), one of the supported data.frame geoms (see *Data frame lay-*

	<code>ers</code>), <code>tidyterra::geom_spatraster()</code>, <code>tidyterra::geom_spatraster_rgb()</code>, <code>tidyterra::geom_spatraster_contour()</code>, <code>tidyterra::geom_spatraster_contour_text()</code> or <code>tidyterra::geom_spatraster_contour_filled()</code> (see <i>SpatRaster layers</i>), or a <code>tmap</code> object (see <i>tmap plots</i> and <i>tmap raster layers</i>).
<code>path</code>	Path of the <code>.qgs</code> file to write. Tilde paths (e.g. <code>~/x.qgs</code>) are expanded.
<code>use_plot_crs</code>	If TRUE, the project (map canvas) CRS is the plot's display CRS: for a <code>ggplot</code>, resolved the way <code>ggplot2::coord_sf()</code> does (its <code>crs</code> argument if specified, otherwise the CRS of the first layer that defines one); for a <code>tmap</code> object, <code>tmap</code>'s own display CRS (see <i>tmap plots</i>). If FALSE, the project CRS is EPSG:3857 (Web Mercator). The default is FALSE for <code>ggplot</code> plots and TRUE for <code>tmap</code> objects. Either way the layers keep the CRS of their data; QGIS reprojects them on the fly.
<code>gradient_style</code>	How a continuous fill/colour scale is rendered: • "graduated" (the default): a graduated renderer with 25 equal-interval classes. The gradient is slightly banded, but the legend shows the classes with their value ranges. • "continuous": the exact <code>ggplot2</code> look. The color is interpolated per feature by a data-defined expression on the symbol color (<code>ramp_color(create_ramp(...), ...)</code>). Caveats: QGIS cannot display a color ramp in the legend for a data-defined color, so the legend is a single swatch without any value labels, and the gradient is only discoverable in the layer styling panel behind the data-defined override of the symbol color, not in the renderer dropdown. Binned scales are unaffected: their bins are exact in a graduated renderer, so there is nothing to trade off. Requesting "continuous" for a layer with a binned scale keeps the bins, with a warning. Raster layers are unaffected too: their shader is exact <i>and</i> legend-friendly, so there is no trade-off to make.
<code>overwrite</code>	If FALSE (the default), writing to a path that already exists is an error. Set to TRUE to overwrite it.
<code>layer_names</code>	Names for the layers, used for the GeoPackage (or GeoTIFF) files and in the QGIS layer tree: a character vector with one name per layer, bottom-most first (layers that are skipped because they draw nothing — see <i>SpatRaster layers</i> — do not count). <code>/</code>, <code>\</code>, <code> </code>, <code>:</code>, <code>*</code>, <code>?</code>, <code>"</code>, <code><</code>, <code>></code> and control characters cannot be used (the name becomes a file name). If NULL (the default), each layer is named after the first of these that applies: • the layer's own name (e.g. <code>geom_sf(name = "counties")</code>), • the variable its data came from (e.g. <code>nc</code> for <code>geom_sf(data = nc)</code>, or for <code>ggplot(nc)</code> when the variable is unambiguous), • the geom (e.g. <code>geom_sf</code>), with a numbered suffix (<code>nc_2</code>) on collision.
<code>basemap</code>	An XYZ tile layer to add below the vector layers, or NULL (the default) for none. Either a predefined key or an arbitrary XYZ URL template (a string containing the <code>{z}</code>, <code>{x}</code> and <code>{y}</code> placeholders, e.g. <code>"https://tile.openstreetmap.org/{z}/{x}/{y}.png"</code>). The predefined keys are: • "osm": OpenStreetMap.

- "gsi_standard": the GSI standard map (<https://cyberjapandata.gsi.go.jp/xyz/std/{z}/{x}/{y}.png>).
- "gsi_pale": the GSI pale map (<https://cyberjapandata.gsi.go.jp/xyz/pale/{z}/{x}/{y}.png>).

XYZ tiles are in EPSG:3857; QGIS reprojects them to the project CRS on the fly.

create_na_layer

tmap vector layers only. If TRUE (the default), features whose mapped value is missing become a separate "<layer> (missing value)" layer in tmap's value.na color (see *tmap plots*). If FALSE, they are not drawn. Raster layers express missing cells exactly and ignore this (see *tmap raster layers*).

... Passed on to the methods.

Details

- a continuous fill/colour scale becomes a graduated renderer with fine-grained equal-interval classes (or a continuously interpolated color, see *gradient_style*),
- a binned one (e.g. `ggplot2::scale_fill_steps()`) becomes a graduated renderer with one class per bin, using the scale's exact bin boundaries and colors,
- a discrete one becomes a categorized renderer,
- a layer with no fill/colour mapping becomes a single symbol with the color ggplot2 would have used.

Following ggplot2's semantics for polygons, fill is the interior and colour is the border: a colour scale on a polygon layer colors the outlines while the interior keeps the constant fill. Constant outline colors and widths are taken from the plot as well, as are the marker constants of a point layer (see *Point symbols*) and alpha (see *Opacity*). Mapping both fill and colour on the same layer is not supported.

A constant fill/colour of NA renders as "not drawn", the way ggplot2 draws it: an unfilled polygon keeps its border, a colour = NA one loses it, and a layer left with nothing to draw at all (a polygon with neither color, a line without its own) is an error.

Only a bare column name is supported for the fill/colour aesthetics; a constant or a computed expression (e.g. `aes(fill = AREA * 2)`) is an error.

Value

path, invisibly.

Line types

A constant linetype (tmap: lty) is carried over to the symbol's stroke — the body of a line, the border of a polygon, the ring around a point marker. "solid", "dashed", "dotted" and "dotdash" (1-4 in numeric form) map to the matching QGIS pen styles. "longdash", "twodash" and the hex on/off patterns (e.g. "1343") have no QGIS preset: on a line layer they become the equivalent custom dash pattern (scaled by the line width, like in R), while polygon borders and marker rings, which only support the preset pen styles, get the nearest preset instead. "blank" (0) turns a polygon border or marker ring off; a blank line layer would draw nothing and is an error. A linetype that

varies by feature (a mapped `linetype/lty`) cannot be represented by these renderers: the symbols are drawn with solid lines, with a warning.

Point symbols

A point layer's shape becomes the QGIS marker of the same outline: pch 0/15/22 a square, 1/16/19/20/21 a circle, 5/18/23 a diamond, 2/17/24 an `equilateral_triangle` (6/25 the same, rotated 180 degrees), 3 a cross and 4 a cross2; ggplot2's shape names ("circle filled", ...) are those same symbols. The composite symbols (pch 7-14), R's thin asterisk (8), a single-character shape (which draws a text glyph) and tmap's grob shapes have no QGIS counterpart and are errors rather than a silently different marker.

How R colors the shape decides where the layer's colors go: pch 0-6 are stroke-only, so the marker gets a transparent interior and its colour; pch 15-20 are filled with a single color (ggplot2's colour, tmap's fill, see *tmap symbol constants*) and get no distinct border; pch 21-25 fill with fill and stroke with colour. Mapping a color to the slot the shape does not draw is an error.

The marker's size is the symbol's extent, which each package computes its own way: ggplot2 spans size millimeters plus two thirds of stroke, tmap a multiple of the text line height (see *tmap symbol constants*). QGIS is then given the width the shape's ink actually spans — three quarters of that extent for a circle — so a default ggplot2 point (size = 1.5, stroke = 0.5, pch 19) is a 1.38 mm circle. Its ring is stroke wide in R's line-width units, i.e. 0.25 mm by default; linewidth, which the point geoms do not have, plays no part.

A size, shape or stroke that varies by feature (a mapped aesthetic) is drawn with the first feature's value, with a warning: the QGIS renderers vary the color and nothing else.

Opacity

A constant alpha (tmap: fill_alpha and col_alpha) becomes the alpha component of the colors it applies to, rather than QGIS's symbol-wide opacity — which could not render a translucent fill under an opaque border. Which colors it applies to is what the plotting package draws with it: ggplot2 gives a polygon's interior its alpha and leaves the border opaque, applies it to a line's own color, and to both colors of a point marker; tmap's two alphas belong to fill and col respectively. Like the other constants, an alpha that varies by feature is dropped down to the first feature's value, with a warning.

An alpha of 0 means "not drawn", like an NA color, and a layer with nothing left to draw is an error. The classes of a graduated or categorized renderer carry the alpha too, as does the data-defined color expression of `gradient_style = "continuous"`, but the renderer's [source] color ramp stays opaque: it is the palette rather than the rendering opacity, so re-classifying the layer in QGIS drops the transparency. An alpha a color carries itself (colour = "#FF000080") is not carried over — only its RGB part is.

Text and labels

A `ggplot2::geom_sf_text()` or `ggplot2::geom_sf_label()` layer becomes a separate labels-only QGIS layer: its features are not drawn (a null-symbol renderer) and the label aesthetic — a bare column name, like fill/colour — becomes the labeled field, rendered by QGIS's own labeling engine. `ggplot2::geom_text()` and `ggplot2::geom_label()` work the same way for plain data frames, under the *Data frame layers* rules (the plot must use `ggplot2::coord_sf()`, x/y must be bare untransformed columns, identity stat and position — so nudging is not supported).

The carried-over styles are the text size (in the layer's `size.unit`), the font family and the text color and, for the label geoms, the background fill color (`geom_label()`'s `fill`; drawn as a plain rectangle, `fill = NA` disables it). R's default sans-serif family ("`"` or "`sans`") is a device alias rather than a font name, so it leaves QGIS its own default font. Everything else (`fontface`, rounded corners, `hjust/vjust`, `alpha`, ...) keeps the QGIS labeling defaults, and the label placement is QGIS's: point labels are drawn over the point, line labels along and *on* the line (rather than QGIS's default of above it, since `ggplot2` centers its text on the geometry), polygon labels around the centroid.

A layer whose features are drawn under its own labels — that is `geom_spatraster_contour_text()`, see *SpatRaster layers* — masks them, so the text is not overdrawn by the line it labels. The labels-only layers of the text/label geoms have nothing to mask.

Data frame layers

A `geom_point()`, `geom_path()`, `geom_line()` or `geom_polygon()` layer drawn from a plain data frame is converted to an sf layer: one point per row, or one linestring/polygon per group (`ggplot2`'s grouping — an explicit group aesthetic or the interaction of the discrete aesthetics; `geom_line()` orders each line by x like `ggplot2` does, and polygon rings are closed). The plot must use `ggplot2::coord_sf()`, and the x/y values are taken to be coordinates in the panel CRS: `coord_sf()`'s `crs` argument if given, otherwise the CRS of the first sf layer (`coord_sf(default_crs = )` is not supported). Like `fill/colour`, the x/y aesthetics must be bare column names, and the layer must use the identity stat and position.

A point layer keeps every column of the data frame as attributes; a line/polygon layer keeps the columns that are constant within every group, one feature per group (a mapped `fill/colour` column must be constant within each group). `geom_polygon()`'s default `colour` is NA, so such a layer is written without a border. A `geom_point()` layer's size, shape, stroke and alpha are carried over like a `geom_sf()` point layer's (see *Point symbols* and *Opacity*).

The project opens zoomed to the plot's displayed range (the panel range, including the default expansion and any `ggplot2::coord_sf()` `xlim/ylim`), reprojected to the project CRS, rather than the whole world.

SpatRaster layers

A `tidyterra::geom_spatraster()` layer becomes a QGIS raster layer: the SpatRaster is written as a GeoTIFF (`<layer name>.tif`, named after the band by default) next to the GeoPackages, and rendered by a single-band pseudocolor renderer reproducing the plot's continuous `fill` scale as an interpolated color ramp. Unlike for vector layers the ramp is exact *and* the legend shows it as a continuous ramp, so the `gradient_style` option does not apply to raster layers. Cells with missing values are transparent (the GeoTIFF's nodata value).

A `tidyterra::geom_spatraster_rgb()` layer becomes a QGIS raster layer with a multiband (true color) renderer. The written GeoTIFF holds the three bands in red-green-blue order, with the layer's r/g/b band selection, `zlim/stretch` rescaling and the constant alpha (the renderer's opacity) carried over. A `max_col_value` other than 255 becomes a linear contrast stretch from 0..`max_col_value`, matching how `tidyterra` scales the channel values.

What is written is the data the plot draws: a raster larger than the geom's `maxcell` argument (500,000 cells by default) has already been downsampled by `tidyterra` when the layer was created and the original is not recoverable from the plot object — so the GeoTIFF stays small, but is not the full-resolution source. For `geom_spatraster()`, only a single-band SpatRaster with a

continuous (not binned) fill scale and the default fill mapping (the band value) is supported for now; tidyterra's color tables (`scale_fill_coltab()`) are not.

The contour geoms are the exception among the SpatRaster geoms: they draw vector shapes, so they become ordinary GeoPackage-backed layers. `tidyterra::geom_spatraster_contour()` becomes a LineString layer with one feature per contour line (per level and per piece), and `tidyterra::geom_spatraster_contour_filled()` becomes a Polygon layer with one MULTIPOLYGON feature per band, the holes punched by the bands above it kept as holes. Both keep the band name as a `lyr` attribute and the contour value as a `level` one — a number for the lines, the band's label (e.g. "(70, 80]") for the filled bands. The shapes are the ones the layer's stat computed, so the geom's breaks/bins/binwidth are reproduced as they are — and, since the stat reprojects the raster to the plot's CRS before contouring, the layer's CRS is the plot's, not the raster's.

`tidyterra::geom_spatraster_contour_text()` draws the same isolines with their value written along them, so it becomes that same LineString layer with QGIS labeling enabled (see *Text and labels*). The text is written as a `label` attribute — the `label` aesthetic (the contour value by default) run through the geom's `label_format`, so a custom format or a vector of labels is reproduced as it is; `label_format = NULL`, which tells the geom to place no labels, leaves the layer unlabeled. The labels mask the lines they are written into, which is how the gap `ggplot2` breaks in each line under its label is reproduced.

On a `geom_spatraster_contour()` layer, `colour` may be mapped to `after_stat(level)`, which becomes a renderer on the `level` attribute under the usual `gradient_style` rules. A `geom_spatraster_contour_filled()` layer always varies its `fill` (the stat maps it to `after_stat(level)`), which becomes a categorized renderer, one category per level of the fill scale. `level` is the only computed value written, so any other `colour/fill` expression is an error, as is a mapping the geom does not draw (`fill` on the lines and on the text, `colour` on the bands). On the text geom, `colour` colors the labels as well as the lines, and a QGIS labeling carries a single text color: the lines follow the scale and every label is drawn in the first feature's color, with a warning (silently when the layer has no labels to begin with). The constant `colour`, `linewidth`, `linetype` and `alpha` are carried over like any other line or polygon layer's. A multi-band SpatRaster is not supported.

The layers are named "<band>_contour", "<band>_contour_text" and "<band>_contour_filled" by default, so overlaying the contours on the raster itself gives distinguishable layers.

`geom_spatraster()` appends an invisible helper layer (a single empty point carrying the raster's CRS to `coord_sf()`). Such a layer draws nothing, so it is not written to the project: in general, an `sf` layer whose geometries are all empty is skipped and does not count for `layer_names`.

SpatVector layers

`tidyterra::geom_spatvector()`, `tidyterra::geom_spatvector_text()` and `tidyterra::geom_spatvector_label()` are wrappers of `ggplot2::geom_sf()`, `ggplot2::geom_sf_text()` and `ggplot2::geom_sf_label()`, and tidyterra registers a `fortify()` method that turns a SpatVector into an `sf` object when the plot or the layer is created. Such a layer is therefore an ordinary `sf` layer here, converted by all the rules above, and so is a plain `geom_sf()` layer given a SpatVector as its data. `terra::vect()` typically produces a geometry column mixing the single and MULTI variants of one type (e.g. POLYGON and MULTIPOLYGON), which is cast to the MULTI variant since a GeoPackage table holds a single geometry type.

One consequence of the wrapping shows up in the derived layer names: `ggplot2` records the *wrapped* `geom_sf()` call as the layer's constructor, so the name comes from the variable the SpatVector is bound to rather than from the call. When there is no such variable (e.g. `geom_spatvector(data =`

`terra::vect(path)))`, the geom fallback names the layer `geom_sf`, not `geom_spatvector` — the wrapper’s own name is not recorded anywhere in the layer. Pass `layer_names` for full control.

tmap plots

A tmap (≥ 4.4) object is converted the same way: `tm_polygons()/tm_fill()/tm_borders()`, `tm_lines()`, and `tm_symbols()/tm_dots()/tm_bubbles()/tm_squares()` (on point data) are supported, including `tmap::qtm()` maps, as is `tmap::tm_raster()` on a raster shape (see *tmap raster layers*). The color scales are reproduced from tmap’s own trained scales:

- `tmap::tm_scale_intervals()` (any classification style) becomes a graduated renderer with tmap’s exact break boundaries and colors (zero-width bins from tied breaks are collapsed; the continuous-style legend variants, `label.style`, are not supported),
- `tmap::tm_scale_categorical()` and `tmap::tm_scale_ordinal()` become a categorized renderer keyed by the raw data values (missing values become the "all other values" category); tmap’s formatted legend labels are not carried over,
- `tmap::tm_scale_continuous()` (including the transformed variants, approximated with piecewise-linear color stops) becomes a graduated renderer with 25 equal-interval classes, or an exact continuous gradient with `gradient_style = "continuous"`.

A layer maps either `fill` or `col` to a data column (not both). The constants tmap computed for the layer — `lwd`, `lty` (see *Line types*), `size` and `shape` (see *Point symbols* and *tmap symbol constants*), `fill_alpha` and `col_alpha` (see *Opacity*) — are carried over; a constant that varies between features is dropped down to its first value, with a warning. Layers sharing one `tmap::tm_shape()` share one GeoPackage: the data is written once and every layer of the shape references the same table.

QGIS’s graduated renderers have no missing-value class, so for an intervals/continuous scale the features whose value is missing go to a separate layer named "`<layer> (missing value)`", drawn in tmap’s `value.na` color directly below its parent layer (the same GeoPackage table, filtered with the QGIS provider filter "`column IS NULL`"). Set `create_na_layer = FALSE` to leave the missing features undrawn instead. Categorical scales render missing values within their own layer (the "all other values" category), so they never get the extra layer. `tmap::tm_basemap()` layers become XYZ tile layers (overriding the `basemap` argument): a URL template is used as is, a provider name is resolved via `maptiles::get_providers()`, and with several basemaps only the first one is checked (visible) in the layer tree. Facets, `tm_text()` and the other scale types are errors.

Map decorations are dropped: `tmap::tm_graticules()/tmap::tm_grid()`, and the components tmap draws around the map (`tmap::tm_compass()`, `tmap::tm_scalebar()`, `tmap::tm_title()`, `tmap::tm_credits()`, ...). They carry no data of their own, and QGIS offers its own equivalents. `tmap::tm_tiles()` is *not* dropped but an error: it draws content that would silently go missing.

A caveat on the bin edges: tmap’s interval bins are left-closed ($[a, b)$) while every QGIS classed renderer is right-closed ($(a, b]$), so a feature or cell whose value falls exactly on an interior break renders one class lower in QGIS than in tmap.

The project CRS defaults to the tmap display CRS (`use_plot_crs = TRUE` for tmap objects): `[tmap::tm_crs()]` or the main shape’s CRS — or EPSG:3857 when the map has basemaps, which is how tmap itself resolves it — so the project opens looking like the tmap plot, zoomed to the main shape’s bounding box. `use_plot_crs = FALSE` forces EPSG:3857.

The conversion relies on tmap internals that are not part of its public API, so a tmap version older than 4.4 is rejected.

tmap symbol constants

What *Point symbols* leaves to each package is the symbol's extent: tmap's size is a multiple of the text line height, which is 5.08 mm on an R device with the default 12 pt font, so a default `tm_symbols()` marker is a 3.81 mm circle (three quarters of it) and a `tm_dots()` one 1.143 mm. A device opened with a different `pointsize` would draw tmap's symbols at a different physical size; the conversion assumes the default. `[tmap::tm_layout()]`'s scale is already part of the size tmap computes, so it carries over.

tmap moves `col` into `fill` before drawing a pch 15-20 symbol (R fills those with `col`), so the single color of such a marker is tmap's `fill` — which is also why mapping `col` on one is an error.

tmap raster layers

`tmap::tm_raster()` on a raster shape (a stars object, a `SpatRaster` or a `RasterLayer` passed to `tmap::tm_shape()`) becomes a QGIS raster layer, written as a single-band GeoTIFF next to the project. The renderer comes from the same trained scale as for vector layers:

- `tmap::tm_scale_intervals()` (the default for a numeric variable) becomes a `singlebandpseudocolor` renderer with a DISCRETE color-ramp shader holding tmap's exact breaks, colors and bin labels,
- `tmap::tm_scale_categorical()` and `tmap::tm_scale_ordinal()` become a `paletted` renderer with one entry per value. A factor variable is written as its integer codes and labeled with the level names; values that no cell has are dropped,
- `tmap::tm_scale_continuous()` becomes a `singlebandpseudocolor` renderer with an INTERPOLATED shader, which reproduces the gradient exactly and shows a continuous legend ramp — so `gradient_style` does not apply to raster layers.

What is written is the grid tmap *draws*: tmap downsamples a raster beyond `tmap_options(raster.max_cells =)` and reprojects it to the display CRS before the scales are trained, and the original is not recoverable from the plot object.

Missing cells become the GeoTIFF's nodata value and are painted in tmap's `value.na` color through the renderer's `nodataColor` — exactly, so raster layers never get the separate "(missing value)" layer that vector layers do and `create_na_layer` does not apply to them. A constant `col_alpha` becomes the layer opacity; a per-cell one is an error. Unlike vector layers, raster layers of one `tmap::tm_shape()` do *not* share a file: each writes its own single-band GeoTIFF, since two variables can differ in value type and nodata value.

`[tmap::tm_rgb()]/[tmap::tm_rgba()]`, `[tmap::tm_scale_discrete()]`, and curvilinear, rotated or irregularly spaced grids are errors.

```
[a, b)) while every QGIS classed renderer is right-closed ((a, b]: R:a,%20b)%60)%20while%20every%20QGIS%20
closed%0A(%60(a,%20b [tmap::tm_crs(): R:tmap::tm_crs() [tmap::tm_layout(): R:tmap::tm_layout()
tmap::tm_raster(): R:tmap::tm_raster() tmap::tm_shape(): R:tmap::tm_shape() tmap::tm_scale_intervals():
R:tmap::tm_scale_intervals() tmap::tm_scale_categorical(): R:tmap::tm_scale_categorical()
tmap::tm_scale_ordinal(): R:tmap::tm_scale_ordinal() tmap::tm_scale_continuous(): R:tmap::tm_scale_continuous()
tmap::tm_shape(): R:tmap::tm_shape() [tmap::tm_rgb(): R:tmap::tm_rgb() [tmap::tm_rgba():
R:tmap::tm_rgba() [tmap::tm_scale_discrete(): R:tmap::tm_scale_discrete()
```

Examples

```
library(ggplot2)

nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
p <- ggplot(nc) +
  geom_sf(aes(fill = AREA))

write_qgs(p, tempfile(fileext = ".qgs"))

# tmap objects work the same way
if (requireNamespace("tmap", quietly = TRUE)) {
  x <- tmap::tm_shape(nc) + tmap::tm_polygons(fill = "AREA")
  write_qgs(x, tempfile(fileext = ".qgs"))
}
```

Index

`ggplot2::coord_sf()`, 3, 5, 6
`ggplot2::geom_label()`, 5
`ggplot2::geom_line()`, 2
`ggplot2::geom_path()`, 2
`ggplot2::geom_point()`, 2
`ggplot2::geom_polygon()`, 2
`ggplot2::geom_sf()`, 7
`ggplot2::geom_sf_label()`, 5, 7
`ggplot2::geom_sf_text()`, 5, 7
`ggplot2::geom_text()`, 5
`ggplot2::scale_fill_steps()`, 4

`maptiles::get_providers()`, 8

`tidyterra::geom_spatraster()`, 2, 3, 6
`tidyterra::geom_spatraster_contour()`,
2, 3, 7
`tidyterra::geom_spatraster_contour_filled()`,
2, 3, 7
`tidyterra::geom_spatraster_contour_text()`,
2, 3, 7
`tidyterra::geom_spatraster_rgb()`, 2, 3,
6
`tidyterra::geom_spatvector()`, 2, 7
`tidyterra::geom_spatvector_label()`, 7
`tidyterra::geom_spatvector_text()`, 7
`tmap::qtm()`, 8
`tmap::tm_basemap()`, 8
`tmap::tm_compass()`, 8
`tmap::tm_credits()`, 8
`tmap::tm_graticules()`, 8
`tmap::tm_grid()`, 8
`tmap::tm_raster()`, 8, 9
`tmap::tm_scale_categorical()`, 8, 9
`tmap::tm_scale_continuous()`, 8, 9
`tmap::tm_scale_intervals()`, 8, 9
`tmap::tm_scale_ordinal()`, 8, 9
`tmap::tm_scalebar()`, 8
`tmap::tm_shape()`, 8, 9
`tmap::tm_tiles()`, 8

`tmap::tm_title()`, 8

`write_qgs(write_qgs.tmap)`, 2
`write_qgs.tmap`, 2