Package: tmap (via r-universe)

June 3, 2026

Title Thematic Maps

Version 4.3.0.9000

Description Thematic maps are geographical maps in which spatial data distributions are visualized. This package offers a flexible, layer-based, and easy to use approach to create thematic maps, such as choropleths and bubble maps.

License GPL-3

URL https://github.com/r-tmap/tmap, https://r-tmap.github.io/tmap/

BugReports https://github.com/r-tmap/tmap/issues

Depends R (>= 4.1)

Imports classInt (>= 0.4-3), cli, cols4all (>= 0.10), data.table, grid, htmltools, htmlwidgets, base64enc, leafem (>= 0.2.4), leafgl, leaflegend, leaflet (>= 2.0.2), leafsync, maptiles, methods, rlang, sf (>= 0.9-3), stars (>= 0.4-2), stats, s2, tmaptools (>= 3.1), units (>= 0.6-1), servr

Suggests av, transformr, colorspace, ggplot2, gifski, knitr, shiny, terra, testthat (>= 3.2.0), widgetframe, lobstr, rsvg, rstudioapi

Config/Needs/check Nowosad/spDataLarge, lwgeom, r-tmap/tmaptools, r-tmap/tmap

Config/Needs/coverage Nowosad/spDataLarge, lwgeom, r-tmap/tmaptools, r-tmap/tmap

Config/Needs/website bookdown, rmarkdown, sfnetworks, r-tmap/tmaptools, r-tmap/tmap.glyphs, r-tmap/tmap.networks, walkerke/mapgl, r-tmap/tmap.mapgl, geofacet, sits, r-tmap/tmap.cartogram, mapsf, mapview

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev zlib1g-dev

Repository https://cynkra.r-universe.dev

Date/Publication 2026-06-03 21:31:41 UTC

RemoteUrl https://github.com/r-tmap/tmap

RemoteRef HEAD

RemoteSha d3b83763b8f6847f2e6b968475a1ab16fcb3d0c0

Contents

tmap-package *Thematic Map Visualization*

Description

Thematic maps are geographical maps in which spatial data distributions are visualized. This package offers a flexible, layer-based, and easy to use approach to create thematic maps, such as choropleths and bubble maps. It is based on the grammar of graphics, and resembles the syntax of ggplot2.

Author(s)

Martijn Tennekes <mtennekes@gmail.com>

References

Tennekes, M., 2018, tmap: Thematic Maps in R, Journal of Statistical Software, 84(6), 1-39, doi:10.18637/jss.v084.i06

See Also

Main documentation

.tmap_providers *Get basemap tile providers*

Description

Get basemap tile providers and their credits (attribution text). `tmap_providers()` returns a list or vector of provider names or credits. `tmap_provider_credits()` returns the attribution text for a specific provider.

Usage

```
.tmap_providers

tmap_provider_credits(provider)

tmap_providers(mode, credits = FALSE, as.list = credits)
```

Arguments

provider	provider name
mode	mode. If not specified the current active mode is used.
credits	If TRUE the credit (attribution) text is returned. If FALSE (default) the provider name.
as.list	Should the output be returned as a list where names are provider names? By default TRUE when credits is also TRUE.

Details

.tmap_providers is an environment populated with all available provider names as named entries. Its primary purpose is to enable autocomplete in IDEs such as RStudio: typing .tmap_providers$ in the console or a script triggers a dropdown list of all available providers, making it easy to discover and select provider names without consulting the documentation. It is not intended to be called as a function.

Value

tmap_providers() returns a list or vector (see as.list) of provider names or credits. tmap_provider_credits() returns the attribution text for the specified provider. .tmap_providers is an environment; see Details.

Examples

```
# List all providers for the current mode
tmap_providers()

# Use IDE autocomplete to discover providers interactively:
# type .tmap_providers$ in the RStudio console
```

---

land *Spatial data of global land cover*

Description

Spatial data of global land cover, percent tree cover, and elevation of class stars. Two attributes in this object relates to global land cover. The cover layer classifies the status of land cover of the whole globe into 20 categories, while the cover_cls layer uses 8 simplified categories. Percent Tree Cover (trees) represents the density of trees on the ground, and the last attribute represents elevation.

Usage

```
land
```

Format

An object of class stars with 1080 rows and 540 columns.

Details

Important: publication of these maps is only allowed when cited to Tateishi et al. (2014), and when "Geospatial Information Authority of Japan, Chiba University and collaborating organizations." is shown.

References

Production of Global Land Cover Data - GLCNMO2008, Tateishi, R., Thanh Hoan, N., Kobayashi, T., Alsaaideh, B., Tana, G., Xuan Phong, D. (2014), Journal of Geography and Geology, 6 (3).

`metro` *Spatial data of metropolitan areas*

Description

`metro` includes a population time series from 1950 to (forecasted) 2030. All metro areas with over 1 million inhabitants in 2010 are included.

Usage

```
metro
```

Format

An object of class `sf` (inherits from `data.frame`) with 436 rows and 13 columns.

Source

https://population.un.org/wup/

References

United Nations, Department of Economic and Social Affairs, Population Division (2014). World Urbanization Prospects: The 2014 Revision, CD-ROM Edition.

NLD_prov *Netherlands datasets*

Description

Datasets of the Netherlands for 2022 at three levels: `NLD_prov` (12) provinces, `NLD_muni` (345) municipalities and `NLD_dist` (3340) districts , all class `sf`

Usage

```
NLD_prov

NLD_muni

NLD_dist
```

Details

The data variables for `NLD_muni` and `NLD_dist` are identical:

Variable	Description
code	Code. Format is "GMaaaa" (municipality/'**gem**eente') and "WKaaaabb" (district/**wij**k). Here, "aaaa"
name	Name.
province	Province name.
area	Total area in km2. This area corresponds to the area of the polygons (including inland waters, exclud
urbanity	Level of urbanity. Five classes, determined by the number of addresses per km2 (break values are 25
population	The total population count at 2022-01-01.
pop_0_14	Percentage (rounded) of people between 0 and 15.
pop_15_24	Percentage (rounded) of people between 15 and 25.
pop_25_44	Percentage (rounded) of people between 25 and 45.
pop_45_64	Percentage (rounded) of people between 45 and 65.
pop_65plus	Percentage (rounded) of people of 65 and older.
dwelling_total	Number of dwellings.
dwelling_value	Average dwelling value (Dutch: WOZ-value).
dwelling_ownership	Percentage of dwellings owned by the residents.
employment_rate	Share of the employed population within the total population from 15 to 75 years old.
income_low	Percentage of individuals in private households belonging to the lowest 40% of personal income nati
income_high	Percentage of individuals in private households belonging to the highest 20% of personal income nat
edu_appl_sci	Percentage of people aged 15 to 75 with a university of applied sciences (Dutch: HBO) or university

See source for detailed information about the variables.

This dataset, created Noveber 2024, is an update from the datasets `NLD_muni` and `NLD_prov` used in tmap <= 3, which has been created around 2016. Note that the number of municipalities have been reduced (due to mergings). All old variables are included, except for variables related to ethnicity. Many new variable have been added, and moreover, district (Dutch: wijk) level data have added: `NLD_dist`.

The CRS (coordinate reference system) used is the Rijksdriekhoekstelsel New, EPSG 28992. Coordinates have been rounded to meters to reduce file size.

Source

https://www.cbs.nl/nl-nl/maatwerk/2024/11/kerncijfers-wijken-en-buurten-2022

References

Statistics Netherlands (2024), The Hague/Heerlen, Netherlands, https://www.cbs.nl/.

`print.tmap` *Draw thematic map*

Description

Draw thematic map

Usage

```
## S3 method for class 'tmap'
print(
  x,
  return.asp = FALSE,
  show = TRUE,
  vp = NULL,
  knit = FALSE,
  options = NULL,
  in.shiny = FALSE,
  proxy = FALSE,
  ...
)

## S3 method for class 'tmap'
knit_print(x, ..., options = NULL)
```

Arguments

`x`	tmap object.
`return.asp`	should the aspect ratio be returned?
`show`	show the map
`vp`	viewport (for `"plot"` mode)
`knit`	A logical, should knit?
`options`	A vector of options
`in.shiny`	A logical, is the map drawn in **shiny**?

proxy	A logical, if in.shiny, is tmapProxy() used?
...	passed on internally (for developers: in "view" mode, the proxy leaflet object is passed to tmapLeafletInit).

---

qtm *Quick thematic map plot*

Description

Draw a thematic map quickly. This function is a convenient wrapper of the main plotting method of stacking tmap-elements. Without arguments or with a search term, this functions draws an interactive map.

Usage

```
qtm(
  shp = NULL,
  fill = tmap::tm_const(),
  col = tmap::tm_const(),
  size = tmap::tm_const(),
  shape = tmap::tm_const(),
  lwd = tmap::tm_const(),
  lty = tmap::tm_const(),
  fill_alpha = tmap::tm_const(),
  col_alpha = tmap::tm_const(),
  text = tmap::tm_const(),
  text_col = tmap::tm_const(),
  text_size = tmap::tm_const(),
  by = NULL,
  scale = NULL,
  title = NULL,
  crs = NULL,
  bbox = NULL,
  basemaps = NA,
  overlays = NA,
  zindex = NA,
  group = NA,
  group.control = "check",
  style = NULL,
  format = NULL,
  ...
)
```

Arguments

shp	One of:

- shape object, which is an object from a class defined by the sf or stars package. Objects from the packages sp and raster are also supported, but discouraged.
- Not specified, i.e. qtm() is executed. In this case a plain interactive map is shown.
- An OpenStreetMap search string, e.g. qtm("Amsterdam"). In this case a plain interactive map is shown positioned according to the results of the search query (from OpenStreetMap nominatim)

fill, col, size, shape, lwd, lty, fill_alpha, col_alpha
: Visual variables.

text, text_col, text_size
: Visual variables.

by
: data variable name by which the data is split, or a vector of two variable names to split the data by two variables (where the first is used for the rows and the second for the columns). See also tm_facets().

scale
: numeric value that serves as the global scale parameter. All font sizes, symbol sizes, border widths, and line widths are controlled by this value. The parameters symbols.size, text.size, and lines.lwd can be scaled separately with respectively symbols.scale, text.scale, and lines.scale. See also ....

title
: main title. For legend titles, use X.legend, where X is the layer name (see ...).

crs
: Either a crs object or a character value (PROJ.4 character string). By default, the projection is used that is defined in the shp object itself.

bbox
: bounding box. Argument passed on to tm_shape()

basemaps
: name(s) of the provider or an URL of a tiled basemap. It is a shortcut to tm_basemap(). Set to NULL to disable basemaps. By default, it is set to the tmap option basemaps.

overlays
: name(s) of the provider or an URL of a tiled overlay map. It is a shortcut to tm_tiles().

zindex
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

group
: group

group.control
: group.control

style
: Layout options (see tm_layout()) that define the style. See tmap_style() for details.

format
: Deprecated, see tm_format() for alternatives

...
: arguments associated with the visual variables are passed on to the layer functions tm_polygons(), tm_lines(), tm_symbols(), and tm_text(). For instance, fill.scale is the scale specifications of the fill color of polygons (see tm_polygons()).

Details

The first argument is a shape object (normally specified by tm_shape()). The next arguments, from fill to raster, are the aesthetics from the main layers. The remaining arguments are related to the

map layout. Any argument from any main layer function, such as `tm_polygons()`, can be specified (see ...). It is also possible to stack `tmap-element`s on a `qtm` plot. See examples.

By default, a scale bar is shown. This option can be set with `tmap_options()` (argument `qtm.scalebar`). A minimap is shown by default when `qtm` is called without arguments of with a search term. This option can be set with `tmap_options()` (argument `qtm.minimap`).

zindex and pane names:

In view mode, each layer is rendered in a Leaflet pane named `"tmap{zindex}"` (e.g., `"tmap401"`, `"tmap402"`), with base tile layers placed in the standard `"tile"` pane.

Value

A `tmap-element`

References

Tennekes, M., 2018, tmap: Thematic Maps in R, Journal of Statistical Software, 84(6), 1-39, doi:10.18637/jss.v084.i06

Examples

```
data(World, World_rivers, metro)

# just the map
qtm(World)

# choropleth
qtm(World, fill = "economy", style = "cobalt", crs = "+proj=eck4")

qtm(World, col = NULL) +
qtm(metro, size = "pop2010",
 size.legend = tm_legend("Metropolitan Areas"))

# dot map
## Not run:
current.mode <- tmap_mode("view")
qtm(metro, bbox = "China")
tmap_mode(current.mode) # restore mode

## End(Not run)

## Not run:
# without arguments, a plain interactive map is shown (the mode is set to view)
qtm()

# search query for OpenStreetMap nominatim
qtm("Amsterdam")

## End(Not run)
```

renderTmap *Wrapper functions for using* **tmap** *in* **shiny**

Description

- `tmapOutput()` creates a UI element
- `renderTmap()` renders a tmap map
- `tmapProxy()` updates a tmap map in `view` mode

Adding layers is as usual via the map layer functions like `tm_polygons()`. Removing layers can be done, removing with `tm_remove_layer()`.

Usage

```
renderTmap(
  expr,
  env = parent.frame(),
  quoted = FALSE,
  execOnResize = TRUE,
  mode = NA
)

tmapOutput(outputId, width = "100%", height = 400, mode = NA)

tmapProxy(mapId, session = shiny::getDefaultReactiveDomain(), x, mode = NA)

tm_remove_layer(zindex)

renderTmapGS(x, ...)

tmapOutputGS(x, ...)

tmapProxyGS(x, ...)
```

Arguments

`expr`	A tmap object. A tmap object is created with `qtm()` or by stacking `tmap-element`s.
`env`	The environment in which to evaluate expr
`quoted`	Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable
`execOnResize`	If `TRUE` (default), when the plot is resized, the map is regenerated. When set to `FALSE` the map is rescaled: the aspect ratio is kept, but the layout will be less desirable.
`mode`	tmap mode, see `tmap_mode()` If not defined, the current mode is used
`outputId`	Output variable to read from

width, height	the width and height of the map
mapId	single-element character vector indicating the output ID of the map to modify (if invoked from a Shiny module, the namespace will be added automatically)
session	the Shiny session object to which the map belongs; usually the default value will suffice
x	the tmap object that specifies the added and removed layers.
zindex	The stacking number of the layer to be removed. It is recommended to specify the zindex for this layer when creating the map (inside renderTmap()).
...	passed on to the mode-specific methods

Details

Two features from tmap are not (yet) supported in Shiny: small multiples (facets) and colored backgrounds (argument bg.color of tm_layout()). Workarounds for small multiples: create multiple independent maps or specify as.layers = TRUE in tm_facets().

Examples

```
if (interactive() && require("shiny")) {

data(World)
world_vars <- setdiff(names(World), c("iso_a3", "name", "sovereignt", "geometry"))

current.mode <- tmap_mode("plot")

shinyApp(
 ui = fluidPage(
  tmapOutput("map", height = "600px"),
  selectInput("var", "Variable", world_vars)
 ),
 server <- function(input, output, session) {
  output$map <- renderTmap({
   tm_shape(World) +
    tm_polygons(input$var, zindex = 401)
  })
 }
)

tmap_mode("view")

shinyApp(
 ui = fluidPage(
  tmapOutput("map", height = "600px"),
  selectInput("var", "Variable", world_vars)
 ),
 server <- function(input, output, session) {
  output$map <- renderTmap({
   tm_shape(World, id = "iso_a3") +
    tm_polygons(fill = world_vars[1], zindex = 401)
  })
```

```
  observe({
   var <- input$var
   tmapProxy("map", session, {
    tm_remove_layer(401) +
     tm_shape(World, id = "iso_a3") +
     tm_polygons(fill = var, zindex = 401)
   })
  })
 },options = list(launch.browser=TRUE)
)

tmap_mode(current.mode)
}
```

---

theme_ps *ggplot2 theme for proportional symbols*

Description

ggplot2 theme for proportional symbols. By default, this theme only shows the plotting area, so without titles, axes, and legend.

Usage

```
theme_ps(
  base_size = 12,
  base_family = "",
  plot.axes = FALSE,
  plot.legend = FALSE
)
```

Arguments

base_size	base size
base_family	base family
plot.axes	should the axes be shown?
plot.legend	should the legend(s) be shown?

`tm_add_legend` *Map component: manual legend*

Description

Map component that adds a manual legend.

Usage

```
tm_add_legend(
  ...,
  labels = "",
  type = "symbols",
  title = "",
  orientation = NULL,
  position = NULL,
  group_id = NA_character_,
  group = NA,
  group.control = "check",
  z = NA_integer_
)
```

Arguments

`...`	visual variables and arguments passed on to `tm_legend()`. By default, the argument `type` is set to `"symbols"`, which means that the supported visual variables are: `"fill"`, `"col"`, `"shape"`, `"size"`, `"fill_alpha"`, `"col_alpha"`, `"lty"`, `"lwd"`, `"linejoin"`, and `"lineend"`. The number of legend items will be equal to the maximum number of specific values (and specified labels.)
`labels`	labels by default `""` (so omitted)
`type`	the layer type from which the visual variables (see `...`) are taken. Options: `"symbols"` (default), `"lines"`, `"polygons"`, and `"text"`.
`title`	The title of the legend.
`orientation`	The orientation of the legend.
`position`	The position of the legend. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details
`group_id`	Component group id name. All components (e.g. legends, titles, etc) with the same `group_id` will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in `tm_components()` where its argument `id` should correspond to `group_id`.
`group`	Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

group.control	In view mode, the group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).
z	z index, e.g. the place of the component relative to the other componets

Examples

```
## Not run:
tm_shape(NLD_muni) +
 tm_borders() +
 tm_basemap("OpenStreetMap") +
 tm_add_legend(labels = c("Motorway", "Primary road", "Secondary road", "Railway"),
      col = c("#E892A1", "#FCD6A4", "#F8FABF", "#707070"),
      lty = c("solid", "solid", "solid", "dotted"),
      lwd = 3,
      type = "lines",
      bg.color = "grey92",
      bg.alpha = 1)

## End(Not run)
```

---

tm_animate_fast *Specify an animation*

Description

Specify an animation from a tmap plot. This is similar to creating facets with tm_facets(). The animation subsequently can be exported to a gif or video file (e.g. mp4) with tmap_animation(). If the tmap plot with tm_animate() is printed, the animation will be previewed. The default tm_animate() will show the individual frames slowly (frame per seconds (fps) set to 2) whereas tm_animate_fast() will show them like a movie (with a fps set to 24).

Usage

```
tm_animate_fast(
  frames = "VARS__",
  nframes = 60L,
  fps = 24L,
  play = c("loop", "pingpong", "once"),
  dpr = 2,
  ...
)

tm_animate(
  frames = "VARS__",
  nframes = 60L,
  fps = 2L,
```

```
  play = c("loop", "pingpong", "once"),
  dpr = 2,
  ...
)
```

Arguments

frames	group by variable used to create the animation frames. This is similar to the by argument of tm_facets_wrap(). Instead of showing facets next to each other, they are shown as animation frames. However, under the hood frames will be used to specify pages of tm_facets(). This makes it possible to create an animation of regular facets.
nframes	number of animation frames. So far, this only applied experimentally in transition map variables. See the extension package tmap.cartogram.
fps	frames per second. Default: 30 for tm_facets_animate and 2 for tm_facets_animate_slow.
play	how should the animation be played? One of "loop" (default), "pingpong", and "once", where "loop" means that the animation will loop indefinitely, "pingpong" means that it will play forward and then backward, and "once" means that it will play only once.
dpr	device pixel ratio. The ratio between the physical pixel density of a device and its logical pixel density.
...	passed on to tm_facets(). Note that for animated facets, by can be specified to create animated facet wraps, and rows and cols to created animated facet grids.

Note

In older versions (< 4.1) tm_facets() with page specification was used to create animations frames and tmap_animation() to create the animation itself using inputs like the frame rate specification. As of version 4.2, the whole animation, including frame rate, is specified in tm_animate(). The animation can still be saved via tmap_animation().

See Also

tm_facets() which is the core function, and tmap_animation() used to save the animation

Examples

```
if (interactive()) {
 tm_shape(NLD_prov) +
  tm_polygons("yellow") +
  tm_animate(frames = "name")

 tm_shape(metro) +
  tm_symbols(size = paste0("pop", seq(1950, 2030, by=10)),
       size.free = FALSE,
       size.legend = tm_legend("Population")) +
  tm_layout(panel.labels = seq(1970, 2030, by=10)) +
  tm_animate()
}
```

tm_basemap *Map layer: basemap / overlay tiles*

Description

Map layer that draws tiles from a tile server. `tm_basemap()` draws the tile layer as basemap, i.e. as bottom layer. In contrast, `tm_tiles()` draws the tile layer as overlay layer, where the stacking order corresponds with the order in which this layer is called, just like other map layers.

Usage

```
tm_basemap(
  server = NA,
  alpha = NULL,
  zoom = NULL,
  api = NULL,
  max.native.zoom = 17,
  sub = "abc",
  zindex = 0,
  group = NA,
  group.control = "radio"
)

tm_tiles(
  server = NA,
  alpha = NULL,
  zoom = NULL,
  max.native.zoom = 17,
  sub = "abc",
  zindex = NA,
  group = NA,
  group.control = "check"
)
```

Arguments

server	Name of the provider or an URL. Or a vector of multiple values. The list of available providers can be obtained with `providers` (tip: in RStudio, type `leaflet::providers$` to see the options). See https://leaflet-extras.github.io/leaflet-providers/preview/ for a preview of those. When a URL is provided, it should be in template format, e.g. `"https://{s}.tile.openstreetmap.org/{z}/{` Use `NULL` in `tm_basemap()` to disable basemaps. It can be a named vector. In that case these names will be used a group names, as alternative to the argument `group`.
alpha	Transparency level
zoom	Zoom level (only used in plot mode)

api	API key. Needed for Stadia and Thunderforest maps in plot mode. See details
max.native.zoom	Maximum native zoom level (only used in view mode). The minimum and maximum zoom levels are determined in tm_view().
sub	subdomain of the tile server. Only used when server is a url template. The default is "abc" which works for most tile servers.
zindex	Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.
group	Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see group.control)
group.control	In view mode, the group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).

Details

API keys. For Stadia and Thunderforest maps, an API key is required. This can be set via the argument api. Alternatively they can be stored in environment variables "STADIA_MAPS" and THUNDERFOREST_MAPS with Sys.setenv

In view mode, each layer is rendered in a Leaflet pane named "tmap{zindex}" (e.g., "tmap401", "tmap402"), with base tile layers placed in the standard "tile" pane.

See Also

Basemap examples

Examples

```
## Not run:
if (requireNamespace("maptiles")) {
 # view mode
 current_mode = tmap_mode("view")
 tm_basemap("Stadia.StamenWatercolor") +
  tm_shape(World) +
  tm_polygons(
    "HPI",
    fill.scale = tm_scale(values = "reds"),
    fill_alpha.scale = 0.5)

 tm_shape(World, crs = "+proj=eqearth") +
  tm_polygons(
   "HPI",
   fill.scale = tm_scale(values = "reds"),
   fill_alpha.scale = 0.5) +
 tm_basemap(NULL)

 # plot mode:
```

```
 tmap_mode("plot")
 tm_basemap() +
  tm_shape(World) +
  tm_polygons("HPI")

 tm_basemap("OpenTopoMap") +
  tm_shape(World) +
  tm_polygons(fill = NA, col = "black")

 tm_basemap("CartoDB.PositronNoLabels") +
 tm_shape(NLD_prov, crs = 4236) +
  tm_borders() +
  tm_facets_wrap("name") +
  tm_tiles("CartoDB.PositronOnlyLabels")

 # restore mode
 tmap_mode(current_mode)
}

## End(Not run)
```

---

tm_chart *Legend charts*

Description

Legend charts are small charts that are added to the map, usually in addition to legends.

Usage

```
tm_chart_histogram(
  breaks,
  plot.axis.x,
  plot.axis.y,
  extra.ggplot2,
  position,
  group_id,
  width,
  height,
  stack,
  z,
  ...
)

tm_chart_bar(
  plot.axis.x,
  plot.axis.y,
  extra.ggplot2,
  position,
```

```
  group_id,
  width,
  height,
  stack,
  z,
  ...
)

tm_chart_donut(position, group_id, width, height, stack, z, ...)

tm_chart_violin(position, group_id, width, height, stack, z, ...)

tm_chart_box(position, group_id, width, height, stack, z, ...)

tm_chart_none()

tm_chart_heatmap(position, group_id, width, height, stack, z, ...)
```

Arguments

`breaks`	The breaks of the bins (for histograms)
`plot.axis.x, plot.axis.y`	Should the x axis and y axis be plot?
`extra.ggplot2`	Extra ggplot2 code
`position`	The position specification of the component: an object created with `tm_pos_in()` or `tm_pos_out()`. Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is `"left"`, `"center"` or `"right"` (or upper case, meaning tighter to the map frame), the second `"top"`, `"center"` or `"bottom"`. Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use `group_id` and specify `component` in `tm_components()`.
`group_id`	Component group id name. All components (e.g. legends, titles, etc) with the same `group_id` will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in `tm_components()` where its argument `id` should correspond to `group_id`.
`width, height`	width and height of the component.
`stack`	stack with other map components, either `"vertical"` or `"horizontal"`.
`z`	z index, e.g. the place of the component relative to the other componets
`...`	passed on to `tm_title()`

Details

Note that these charts are different from charts drawn inside the map. Those are called glyphs (to be implemented).

See Also

Vignette about charts

Examples

```
tm_shape(World) +
  tm_polygons("HPI",
    fill.scale = tm_scale_intervals(),
    fill.chart = tm_chart_histogram())
```

`tm_check_fix` *tmap options*

Description

Get or set the tmap options globally. For map specific options, we recommend to use `tm_options()` or `tm_layout()` via which the layout-related options can be set. `tmap_options()` functions similar to `base::options()`.

Usage

```
tm_check_fix()

tmap_options(
  ...,
  crs,
  facet.max,
  free.scales,
  raster.max_cells,
  raster.warp,
  show.messages,
  show.warnings,
  output.format,
  output.size,
  output.dpi,
  animation.dpi,
  value.const,
  value.na,
  value.null,
  value.blank,
  values.var,
  values.range,
  value.neutral,
  values.scale,
  scales.var,
  scale.misc.args,
  continuous.nclass_per_legend_break,
  continuous.nclasses,
  label.format,
  label.na,
  scale,
```

```
    asp,
    bg,
    bg.color,
    outer.bg,
    outer.bg.color,
    frame,
    frame.color,
    frame.alpha,
    frame.lwd,
    frame.r,
    frame.double_line,
    outer.margins,
    inner.margins,
    inner.margins.extra,
    meta.margins,
    meta.auto_margins,
    between_margin,
    panel.margin,
    xlab.show,
    xlab.text,
    xlab.size,
    xlab.color,
    xlab.rotation,
    xlab.space,
    xlab.fontface,
    xlab.fontfamily,
    xlab.alpha,
    xlab.side,
    ylab.show,
    ylab.text,
    ylab.size,
    ylab.color,
    ylab.rotation,
    ylab.space,
    ylab.fontface,
    ylab.fontfamily,
    ylab.alpha,
    ylab.side,
    panel.type,
    panel.wrap.pos,
    panel.xtab.pos,
    unit,
    color.sepia_intensity,
    color.saturation,
    color_vision_deficiency_sim,
    text.fontface,
    text.fontfamily,
    r,
```

```
component.position,
component.offset,
component.stack_margin,
component.autoscale,
component.resize_as_group,
component.frame_combine,
component.stack,
legend.stack,
chart.stack,
component.equalize,
component.frame,
component.frame.color,
component.frame.alpha,
component.frame.lwd,
component.frame.r,
component.bg,
component.bg.color,
component.bg.alpha,
legend.show,
legend.orientation,
legend.position,
legend.width,
legend.height,
legend.reverse,
legend.na.show,
legend.title.color,
legend.title.size,
legend.title.fontface,
legend.title.fontfamily,
legend.title.alpha,
legend.xlab.color,
legend.xlab.size,
legend.xlab.rot,
legend.xlab.fontface,
legend.xlab.fontfamily,
legend.xlab.alpha,
legend.ylab.color,
legend.ylab.size,
legend.ylab.rot,
legend.ylab.fontface,
legend.ylab.fontfamily,
legend.ylab.alpha,
legend.text.color,
legend.text.size,
legend.text.fontface,
legend.text.fontfamily,
legend.text.alpha,
legend.frame,
```

```
    legend.frame.color,
    legend.frame.alpha,
    legend.frame.lwd,
    legend.frame.r,
    legend.bg,
    legend.bg.color,
    legend.bg.alpha,
    legend.only,
    legend.absolute_fontsize,
    legend.settings.portrait,
    legend.settings.landscape,
    add_legend.position,
    chart.show,
    chart.plot.axis.x,
    chart.plot.axis.y,
    chart.position,
    chart.width,
    chart.height,
    chart.reverse,
    chart.na.show,
    chart.title.color,
    chart.title.size,
    chart.title.fontface,
    chart.title.fontfamily,
    chart.title.alpha,
    chart.xlab.color,
    chart.xlab.size,
    chart.xlab.fontface,
    chart.xlab.fontfamily,
    chart.xlab.alpha,
    chart.ylab.color,
    chart.ylab.size,
    chart.ylab.fontface,
    chart.ylab.fontfamily,
    chart.ylab.alpha,
    chart.text.color,
    chart.text.size,
    chart.text.fontface,
    chart.text.fontfamily,
    chart.text.alpha,
    chart.frame,
    chart.frame.color,
    chart.frame.alpha,
    chart.frame.lwd,
    chart.frame.r,
    chart.bg,
    chart.bg.color,
    chart.bg.alpha,
```

```
chart.object.color,
title.size,
title.color,
title.fontface,
title.fontfamily,
title.alpha,
title.padding,
title.frame,
title.frame.color,
title.frame.alpha,
title.frame.lwd,
title.frame.r,
title.position,
title.width,
credits.size,
credits.color,
credits.fontface,
credits.fontfamily,
credits.alpha,
credits.padding,
credits.position,
credits.width,
credits.height,
compass.north,
compass.type,
compass.text.size,
compass.size,
compass.show.labels,
compass.cardinal.directions,
compass.text.color,
compass.color.dark,
compass.color.light,
compass.lwd,
compass.margins,
compass.position,
inset.position,
logo.height,
logo.margins,
logo.between_margin,
logo.position,
inset_map.height,
inset_map.width,
inset_map.margins,
inset_map.between_margin,
inset_map.position,
inset_map.frame,
inset.height,
inset.width,
```

```
    inset.margins,
    inset.between_margin,
    inset.frame,
    inset.bg,
    inset.bg.color,
    inset.bg.alpha,
    inset_grob.height,
    inset_grob.width,
    inset_gg.height,
    inset_gg.width,
    scalebar.breaks,
    scalebar.width,
    scalebar.allow_clipping,
    scalebar.text.size,
    scalebar.text.color,
    scalebar.text.fontface,
    scalebar.text.fontfamily,
    scalebar.color.dark,
    scalebar.color.light,
    scalebar.lwd,
    scalebar.size,
    scalebar.margins,
    scalebar.position,
    grid.show,
    grid.labels.pos,
    grid.x,
    grid.y,
    grid.n.x,
    grid.n.y,
    grid.crs,
    grid.col,
    grid.lwd,
    grid.alpha,
    grid.labels.show,
    grid.labels.size,
    grid.labels.col,
    grid.labels.fontface,
    grid.labels.fontfamily,
    grid.labels.rot,
    grid.labels.format,
    grid.labels.cardinal,
    grid.labels.margin.x,
    grid.labels.margin.y,
    grid.labels.space.x,
    grid.labels.space.y,
    grid.labels.inside_frame,
    grid.ticks,
    grid.lines,
```

```
grid.ndiscr,
mouse_coordinates.position,
minimap.server,
minimap.toggle,
minimap.position,
panel.show,
panel.labels,
panel.label.size,
panel.label.color,
panel.label.fontface,
panel.label.fontfamily,
panel.label.alpha,
panel.label.bg,
panel.label.bg.color,
panel.label.bg.alpha,
panel.label.frame,
panel.label.frame.color,
panel.label.frame.alpha,
panel.label.frame.lwd,
panel.label.frame.r,
panel.label.height,
panel.label.rot,
qtm.scalebar,
qtm.minimap,
qtm.mouse_coordinates,
earth_boundary,
earth_boundary.color,
earth_boundary.lwd,
earth_datum,
space,
space.color,
space_overlay,
check_and_fix,
basemap.show,
basemap.server,
basemap.alpha,
basemap.zoom,
tiles.show,
tiles.server,
tiles.alpha,
tiles.zoom,
attr.color,
crs_extra,
crs_global,
crs_basemap,
use_gradient,
use_browser,
use_WebGL,
```

```
  control.position,
  control.bases,
  control.overlays,
  control.collapse,
  set_bounds,
  set_view,
  set_zoom_limits,
  use_circle_markers,
  leaflet.options,
  title = NULL,
  main.title = NULL,
  main.title.size = NULL,
  main.title.color = NULL,
  main.title.fontface = NULL,
  main.title.fontfamily = NULL,
  main.title.position = NULL,
  fontface = NULL,
  fontfamily = NULL
)

tmap_options_mode(
  mode = NA,
  style = NULL,
  mode.specific = TRUE,
  default.options = FALSE
)

tmap_options_diff()

tmap_options_reset()

tmap_options_save(style)
```

Arguments

`...`	List of tmap options to be set, or option names (characters) to be returned (see details)
`crs`	Map crs (see `tm_shape()`). `NA` means the crs is specified in `tm_shape()`. The crs that is used by the transformation functions is defined in `tm_shape()`.
`facet.max`	Maximum number of facets
`free.scales`	For backward compatibility: if this value is set, it will be used to impute the free arguments in the layer functions
`raster.max_cells`	Maximum number of raster grid cells. Can be mode specific `c(plot = 3000, view = 1000, 1000)` (the last value is the fall back default)
`raster.warp`	Should rasters be warped or transformed in case a different projection (crs) is used? Warping creates a new regular raster in the target crs, whereas transforming creates a (usually non-regular) raster in the target crs. The former is lossy,

but much faster and is therefore the default. When a different projection (crs) is used, a (usually) regular raster will be

`show.messages` Show messages?

`show.warnings` Show warnings?

`output.format` Output format

`output.size` Output size

`output.dpi` Output dpi

`animation.dpi` Output dpi for animations

`value.const` Default visual value constants e.g. the default fill color for `tm_shape(World) + tm_polygons()`. A list is required with per visual variable a value.

`value.na` Default visual values that are used to visualize NA data values. A list is required with per visual variable a value.

`value.null` Default visual values that are used to visualize null (out-of-scope) data values. A list is required with per visual variable a value.

`value.blank` Default visual values that correspond to blank. For color these are `"#00000000"` meaning transparent. A list is required with per visual variable a value.

`values.var` Default values when a data variable to mapped to a visual variable, e.g. a color palette. A list is required with per visual variable a value.

`values.range` Default range for values. See `values.range` of `tm_scale_categorical()`. A list is required with per visual variable a value.

`value.neutral` Default values for when a data variable to mapped to a visual variable, e.g. a color palette. A list is required with per visual variable a value.

`values.scale` Default scales (as in object sizes) for values. See `values.range` of `tm_scale_categorical()`. A list is required with per visual variable a value.

`scales.var` Default scale functions per visual variable and type of data variable. A list is required with per visual variable per data type.

`scale.misc.args`
Default values of scale function-specific arguments. A list is required with per scale function and optional per visual variable.

`continuous.nclass_per_legend_break`
The number of continuous legend breaks within one 'unit' (label). The default value is 50.

`continuous.nclasses`
the number of classes of a continuous scale. Should be odd. The default value is 101.

`label.format` Format for the labels. These are the default values for `tm_label_format()`

`label.na` Default label for missing values.

`scale` Overall scale of the map

`asp` Aspect ratio of each map. When `asp` is set to `NA` (default) the aspect ratio will be adjusted to the used shapes. When set to 0 the aspect ratio is adjusted to the size of the device divided by the number of columns and rows.

`bg` Draw map background?

bg.color	Background color of the map.
outer.bg	Draw map background (outside the frame)?
outer.bg.color	Background color of map outside the frame.
frame	Draw map frame?
frame.color	The color of the frame.
frame.alpha	The alpha transparency of the frame.
frame.lwd	The line width of the frame. See graphics::par, option 'lwd'.
frame.r	The r (radius) of the frame.
frame.double_line	The double line of the frame. TRUE or FALSE.
outer.margins	The margins of the outer space (outside the frame. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
inner.margins	The margins of the inner space (inside the frame). A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
inner.margins.extra	The extra arguments of the margins of the inner space (inside the frame). A list of arguments.
meta.margins	The margins of the meta. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
meta.auto_margins	The auto_margins of the meta.
between_margin	Margin between the map.
panel.margin	The margin of the panel.
xlab.show	The visibility of the xlab. TRUE or FALSE.
xlab.text	The text of the xlab.
xlab.size	The size of the xlab.
xlab.color	The color of the xlab.
xlab.rotation	The rotation of the xlab.
xlab.space	The space of the xlab. In terms of number of text line heights.
xlab.fontface	The font face of the xlab. See graphics::par, option 'font'.
xlab.fontfamily	The font family of the xlab. See graphics::par, option 'family'.
xlab.alpha	The alpha transparency of the xlab.
xlab.side	The side of the xlab.
ylab.show	The visibility of the ylab. TRUE or FALSE.
ylab.text	The text of the ylab.
ylab.size	The size of the ylab.
ylab.color	The color of the ylab.

`ylab.rotation` The rotation of the ylab.

`ylab.space` The space of the ylab. In terms of number of text line heights.

`ylab.fontface` The font face of the ylab. See `graphics::par`, option 'font'.

`ylab.fontfamily`
The font family of the ylab. See `graphics::par`, option 'family'.

`ylab.alpha` The alpha transparency of the ylab.

`ylab.side` The side of the ylab.

`panel.type` The type of the panel.

`panel.wrap.pos` The panel positions for wrapped facets created with `tm_facets_grid()`. One of `"left"`, `"right"`, `"top"` (default) or `"bottom"`.

`panel.xtab.pos` The panel positions for grid facets created with `tm_facets_grid()`. Vector of two, where the first determines the locations of row panels (`"left"` or `"right"`) and the second the location of column panels (`"top"` or '"bottom")

`unit` Unit of the coordinate

`color.sepia_intensity`
The sepia_intensity of the color.

`color.saturation`
The saturation of the color.

`color_vision_deficiency_sim`
Color vision deficiency simulation. Either `"protan"`, `"deutan"`, or `"tritan"`.

`text.fontface` The font face of the text. See `graphics::par`, option 'font'.

`text.fontfamily`
The font family of the text. See `graphics::par`, option 'family'.

`r` The r (radius) (overall).

`component.position`
The position of the component. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`component.offset`
The offset of the component.

`component.stack_margin`
The stack_margin of the component.

`component.autoscale`
The autoscale of the component.

`component.resize_as_group`
The resize_as_group of the component.

`component.frame_combine`
The frame_combine of the component.

`component.stack`
The stack of the component.

`legend.stack` The stack of the legend.

`chart.stack` The stack of the chart.

component.equalize
: The equalize of the component.

component.frame
: The frame of the component.

component.frame.color
: The color of the frame of the component.

component.frame.alpha
: The alpha transparency of the frame of the component.

component.frame.lwd
: The line width of the frame of the component. See graphics::par, option 'lwd'.

component.frame.r
: The r (radius) of the frame of the component.

component.bg
: The bg of the component.

component.bg.color
: The color of the bg of the component.

component.bg.alpha
: The alpha transparency of the bg of the component.

legend.show
: The visibility of the legend. TRUE or FALSE.

legend.orientation
: The orientation of the legend.

legend.position
: The position of the legend. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

legend.width
: The width of the legend.

legend.height
: The height of the legend.

legend.reverse
: The reverse of the legend.

legend.na.show
: The visibility of the na of the legend. TRUE or FALSE.

legend.title.color
: The color of the title of the legend.

legend.title.size
: The size of the title of the legend.

legend.title.fontface
: The font face of the title of the legend. See graphics::par, option 'font'.

legend.title.fontfamily
: The font family of the title of the legend. See graphics::par, option 'family'.

legend.title.alpha
: The alpha transparency of the title of the legend.

legend.xlab.color
: The color of the xlab of the legend.

legend.xlab.size
: The size of the xlab of the legend.

legend.xlab.rot
: The rot of the xlab of the legend.

legend.xlab.fontface
: The font face of the xlab of the legend. See graphics::par, option 'font'.

legend.xlab.fontfamily
: The font family of the xlab of the legend. See graphics::par, option 'family'.

legend.xlab.alpha
: The alpha transparency of the xlab of the legend.

legend.ylab.color
: The color of the ylab of the legend.

legend.ylab.size
: The size of the ylab of the legend.

legend.ylab.rot
: The rot of the ylab of the legend.

legend.ylab.fontface
: The font face of the ylab of the legend. See graphics::par, option 'font'.

legend.ylab.fontfamily
: The font family of the ylab of the legend. See graphics::par, option 'family'.

legend.ylab.alpha
: The alpha transparency of the ylab of the legend.

legend.text.color
: The color of the text of the legend.

legend.text.size
: The size of the text of the legend.

legend.text.fontface
: The font face of the text of the legend. See graphics::par, option 'font'.

legend.text.fontfamily
: The font family of the text of the legend. See graphics::par, option 'family'.

legend.text.alpha
: The alpha transparency of the text of the legend.

legend.frame
: The frame of the legend.

legend.frame.color
: The color of the frame of the legend.

legend.frame.alpha
: The alpha transparency of the frame of the legend.

legend.frame.lwd
: The line width of the frame of the legend. See graphics::par, option 'lwd'.

legend.frame.r
: The r (radius) of the frame of the legend.

legend.bg
: The bg of the legend.

legend.bg.color
: The color of the bg of the legend.

legend.bg.alpha
: The alpha transparency of the bg of the legend.

`legend.only` Should only legends be printed (so without map)?

`legend.absolute_fontsize`
: The absolute fontsize of the legend. So far, only used to calculate legend dimensions

`legend.settings.portrait`
: The portrait of the settings of the legend.

`legend.settings.landscape`
: The landscape of the settings of the legend.

`add_legend.position`
: The position of the add_legend. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`chart.show` The visibility of the chart. `TRUE` or `FALSE`.

`chart.plot.axis.x`
: The x of the axis of the plot of the chart.

`chart.plot.axis.y`
: The y of the axis of the plot of the chart.

`chart.position` The position of the chart. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`chart.width` The width of the chart.

`chart.height` The height of the chart.

`chart.reverse` The reverse of the chart.

`chart.na.show` The visibility of the na of the chart. `TRUE` or `FALSE`.

`chart.title.color`
: The color of the title of the chart.

`chart.title.size`
: The size of the title of the chart.

`chart.title.fontface`
: The font face of the title of the chart. See `graphics::par`, option 'font'.

`chart.title.fontfamily`
: The font family of the title of the chart. See `graphics::par`, option 'family'.

`chart.title.alpha`
: The alpha transparency of the title of the chart.

`chart.xlab.color`
: The color of the xlab of the chart.

`chart.xlab.size`
: The size of the xlab of the chart.

`chart.xlab.fontface`
: The font face of the xlab of the chart. See `graphics::par`, option 'font'.

`chart.xlab.fontfamily`
: The font family of the xlab of the chart. See `graphics::par`, option 'family'.

`chart.xlab.alpha`
: The alpha transparency of the xlab of the chart.

chart.ylab.color
: The color of the ylab of the chart.

chart.ylab.size
: The size of the ylab of the chart.

chart.ylab.fontface
: The font face of the ylab of the chart. See graphics::par, option 'font'.

chart.ylab.fontfamily
: The font family of the ylab of the chart. See graphics::par, option 'family'.

chart.ylab.alpha
: The alpha transparency of the ylab of the chart.

chart.text.color
: The color of the text of the chart.

chart.text.size
: The size of the text of the chart.

chart.text.fontface
: The font face of the text of the chart. See graphics::par, option 'font'.

chart.text.fontfamily
: The font family of the text of the chart. See graphics::par, option 'family'.

chart.text.alpha
: The alpha transparency of the text of the chart.

chart.frame
: The frame of the chart.

chart.frame.color
: The color of the frame of the chart.

chart.frame.alpha
: The alpha transparency of the frame of the chart.

chart.frame.lwd
: The line width of the frame of the chart. See graphics::par, option 'lwd'.

chart.frame.r
: The r (radius) of the frame of the chart.

chart.bg
: The bg of the chart.

chart.bg.color
: The color of the bg of the chart.

chart.bg.alpha
: The alpha transparency of the bg of the chart.

chart.object.color
: The color of the object of the chart.

title.size
: The size of the title.

title.color
: The color of the title.

title.fontface
: The font face of the title. See graphics::par, option 'font'.

title.fontfamily
: The font family of the title. See graphics::par, option 'family'.

title.alpha
: The alpha transparency of the title.

title.padding
: The padding of the title. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

title.frame
: The frame of the title.

`title.frame.color`
: The color of the frame of the title.

`title.frame.alpha`
: The alpha transparency of the frame of the title.

`title.frame.lwd`
: The line width of the frame of the title. See `graphics::par`, option 'lwd'.

`title.frame.r`
: The r (radius) of the frame of the title.

`title.position`
: The position of the title. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`title.width`
: The width of the title.

`credits.size`
: The size of the credits.

`credits.color`
: The color of the credits.

`credits.fontface`
: The font face of the credits. See `graphics::par`, option 'font'.

`credits.fontfamily`
: The font family of the credits. See `graphics::par`, option 'family'.

`credits.alpha`
: The alpha transparency of the credits.

`credits.padding`
: The padding of the credits. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`credits.position`
: The position of the credits. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`credits.width`
: The width of the credits.

`credits.height`
: The height of the credits.

`compass.north`
: The north of the compass.

`compass.type`
: The type of the compass.

`compass.text.size`
: The size of the text of the compass.

`compass.size`
: The size of the compass.

`compass.show.labels`
: The labels of the show of the compass.

`compass.cardinal.directions`
: The directions of the cardinal of the compass.

`compass.text.color`
: The color of the text of the compass.

`compass.color.dark`
: The dark of the color of the compass.

`compass.color.light`
: The light of the color of the compass.

`compass.lwd`
: The line width of the compass. See `graphics::par`, option 'lwd'.

`compass.margins`
: The margins of the compass. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`compass.position`
: The position of the compass. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset.position`
: The position of the inset. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`logo.height`
: The height of the logo.

`logo.margins`
: The margins of the logo. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`logo.between_margin`
: The between_margin of the logo.

`logo.position`
: The position of the logo. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset_map.height`
: The height of the inset_map.

`inset_map.width`
: The width of the inset_map.

`inset_map.margins`
: The margins of the inset_map. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inset_map.between_margin`
: The between_margin of the inset_map.

`inset_map.position`
: The position of the inset_map. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset_map.frame`
: The frame of the inset_map.

`inset.height`
: The height of the inset.

`inset.width`
: The width of the inset.

`inset.margins`
: The margins of the inset. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inset.between_margin`
: The between_margin of the inset.

`inset.frame`
: The frame of the inset.

`inset.bg`
: The bg of the inset.

`inset.bg.color`
: The color of the bg of the inset.

`inset.bg.alpha`
: The alpha transparency of the bg of the inset.

`inset_grob.height`
: The height of the inset_grob.

`inset_grob.width`
: The width of the inset_grob.

`inset_gg.height`
: The height of the inset_gg.

`inset_gg.width`
: The width of the inset_gg.

`scalebar.breaks`
: See `tm_scalebar()`

`scalebar.width`
: See `tm_scalebar()`

`scalebar.allow_clipping`
: See `tm_scalebar()`

`scalebar.text.size`
: The size of the text of the scalebar.

`scalebar.text.color`
: The color of the text of the scalebar.

`scalebar.text.fontface`
: The font face of the text of the scalebar. See `graphics::par`, option 'font'.

`scalebar.text.fontfamily`
: The font family of the text of the scalebar. See `graphics::par`, option 'family'.

`scalebar.color.dark`
: The dark of the color of the scalebar.

`scalebar.color.light`
: The light of the color of the scalebar.

`scalebar.lwd`
: The line width of the scalebar. See `graphics::par`, option 'lwd'.

`scalebar.size`
: The size of the scalebar.

`scalebar.margins`
: The margins of the scalebar. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`scalebar.position`
: The position of the scalebar. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`grid.show`
: The visibility of the grid. `TRUE` or `FALSE`.

`grid.labels.pos`
: The pos of the labels of the grid.

`grid.x`
: The x of the grid.

`grid.y`
: The y of the grid.

`grid.n.x`
: The x of the n of the grid.

`grid.n.y`
: The y of the n of the grid.

`grid.crs`
: The coordinate reference system (CRS) of the grid.

`grid.col`
: The color of the grid.

`grid.lwd`
: The line width of the grid. See `graphics::par`, option 'lwd'.

`grid.alpha`
: The alpha transparency of the grid.

`grid.labels.show`
: The visibility of the labels of the grid. `TRUE` or `FALSE`.

`grid.labels.size`
: The size of the labels of the grid.

`grid.labels.col`
: The color of the labels of the grid.

`grid.labels.fontface`
: The font face of the labels of the grid. See `graphics::par`, option 'font'.

`grid.labels.fontfamily`
: The font family of the labels of the grid. See `graphics::par`, option 'family'.

`grid.labels.rot`
: The rot of the labels of the grid.

`grid.labels.format`
: The format of the labels of the grid.

`grid.labels.cardinal`
: The cardinal of the labels of the grid.

`grid.labels.margin.x`
: The x of the margin of the labels of the grid.

`grid.labels.margin.y`
: The y of the margin of the labels of the grid.

`grid.labels.space.x`
: The x of the space of the labels of the grid.

`grid.labels.space.y`
: The y of the space of the labels of the grid.

`grid.labels.inside_frame`
: The inside_frame of the labels of the grid.

`grid.ticks` The ticks of the grid.

`grid.lines` The lines of the grid.

`grid.ndiscr` The ndiscr of the grid.

`mouse_coordinates.position`
: The position of the mouse_coordinates. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`minimap.server` The server of the minimap.

`minimap.toggle` The toggle of the minimap.

`minimap.position`
: The position of the minimap. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`panel.show` The visibility of the panel. `TRUE` or `FALSE`.

`panel.labels` The labels of the panel.

`panel.label.size`
: The size of the label of the panel.

panel.label.color
: The color of the label of the panel.

panel.label.fontface
: The font face of the label of the panel. See graphics::par, option 'font'.

panel.label.fontfamily
: The font family of the label of the panel. See graphics::par, option 'family'.

panel.label.alpha
: The alpha transparency of the label of the panel.

panel.label.bg
: The bg of the label of the panel.

panel.label.bg.color
: The color of the bg of the label of the panel.

panel.label.bg.alpha
: The alpha transparency of the bg of the label of the panel.

panel.label.frame
: The frame of the label of the panel.

panel.label.frame.color
: The color of the frame of the label of the panel.

panel.label.frame.alpha
: The alpha transparency of the frame of the label of the panel.

panel.label.frame.lwd
: The line width of the frame of the label of the panel. See graphics::par, option 'lwd'.

panel.label.frame.r
: The r (radius) of the frame of the label of the panel.

panel.label.height
: The height of the label of the panel.

panel.label.rot
: Rotation angles of the panel labels. Vector of four values that determine the panel label rotation when they are placed left, top, right, and bottom. The default angles are 90, 0, 270 and 0 respectively. Note that the second value is the most common, since labels are by default shown on top (see panel.wrap.pos). In cross-table facets created with tm_facets_grid(), the first two values are used by default (see panel.xtab.pos).

qtm.scalebar
: The scalebar of the qtm.

qtm.minimap
: The minimap of the qtm.

qtm.mouse_coordinates
: The mouse_coordinates of the qtm.

earth_boundary
: The earth boundary

earth_boundary.color
: The color of the earth_boundary.

earth_boundary.lwd
: The line width of the earth_boundary. See graphics::par, option 'lwd'.

earth_datum
: Earth datum

space
: Should the space be drawn? Only applicable is earth_boundary is enabled.

space.color	The color of the space.
space_overlay	Should the space be drawn as overlay (to make sure spatial features or rasters do not exceed the earth boundary), or as background? By default TRUE when a raster is warped.
check_and_fix	Should attempt to fix an invalid shapefile
basemap.show	The visibility of the basemap. TRUE or FALSE.
basemap.server	The server of the basemap.
basemap.alpha	The alpha transparency of the basemap.
basemap.zoom	The zoom of the basemap.
tiles.show	The visibility of the tiles. TRUE or FALSE.
tiles.server	The server of the tiles.
tiles.alpha	The alpha transparency of the tiles.
tiles.zoom	The zoom of the tiles.
attr.color	The color of the attr.
crs_extra	Only used internally (work in progress)
crs_global	The used crs for world maps
crs_basemap	The crs_basemap (overall).
use_gradient	Use gradient fill using linearGradient()
use_browser	If TRUE it opens an external browser, and FALSE (default) it opens the internal IDEs (e.g. RStudio) browser.
use_WebGL	use webGL for points, lines, and polygons. For large spatial objects, this is much faster than the standard leaflet layer functions. However, it can not always be used for two reasons. First, the number of visual variables are limited; only fill, size, and color (for lines) are supported. Second, projected CRS's are not supported. Furthermore, it has the drawback that polygon borders are not as sharp. By default only TRUE for large spatial objects (1000 or more features) when the mentioned criteria are met. By default TRUE if no other visual variables are used.
control.position	The position of the control. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details
control.bases	base layers
control.overlays	overlay layers
control.collapse	Should the box be collapsed or expanded?
set_bounds	logical that determines whether maximum bounds are set, or a bounding box. Not applicable in plot mode. In view mode, this is passed on to setMaxBounds()
set_view	numeric vector that determines the view. Either a vector of three: lng, lat, and zoom, or a single value: zoom. See setView(). Only applicable if bbox is not specified

`set_zoom_limits`
: numeric vector of two that set the minimum and maximum zoom levels (see tileOptions()).

`use_circle_markers`
: If `TRUE` (default) circle shaped symbols (e.g. `tm_dots()` and `tm_symbols()`) will be rendered as addCircleMarkers() instead of addMarkers(). The former is faster, the latter can support any symbol since it is based on icons

`leaflet.options`
: options passed on to leafletOptions()

`title`
: deprecated See `tm_title()`

`main.title`
: deprecated See `tm_title()`

`main.title.size`, `main.title.color`, `main.title.fontface`, `main.title.fontfamily`, `main.title.position`
: deprecated. Use the `title.` options instead.

`fontface`, `fontfamily`
: renamed to `text.fontface` and `text.fontfamily`

`mode`
: mode, e.g. `"plot"` or `"view"`

`style`
: style, see `tmap_style()` for available styles

`mode.specific`
: Should only mode-specific options be returned? `TRUE` by default.

`default.options`
: return the default options or the current options?

Examples

```
# get all options
opt = tmap_options()

# print as a tree
if (requireNamespace("lobstr")) {
 lobstr::tree(opt)
}

# a fancy set of options:
tmap_options(
  bg.color = "steelblue",
  outer.bg.color = "salmon",
  frame.color = "purple3",
  frame.lwd = 5,
  compass.type = "8star",
  legend.bg.color = "gold",
  legend.position = tm_pos_in(pos.h = "left", pos.v = "top")
 )

if (requireNamespace("lobstr")) {
 lobstr::tree(
  tmap_options_diff()
 )
}
```

```
tm_shape(World) +
 tm_polygons("footprint")

tmap_options_save("fancy")

# the default style:
tmap_style("white")

tm_shape(World) +
 tm_polygons("footprint")

tmap_style("fancy")

tm_shape(World) +
 tm_polygons("footprint")

# reset all options
tmap_options_reset()
```

`tm_circles` *Map layer: geographic circles*

Description

Map layer that draws circles with geographically fixed radii — i.e. the radius is expressed in meters and the circles scale with zoom in interactive (view) mode. This is in contrast to `tm_bubbles()`, where the symbol size is a fixed number of screen pixels.

Usage

```
tm_circles(
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.chart = tm_chart_none(),
  size.free = NA,
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.chart = tm_chart_none(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  lwd = tm_const(),
```

```
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.chart = tm_chart_none(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.chart = tm_chart_none(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.chart = tm_chart_none(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  plot.order = tm_plot_order("size"),
  zindex = NA,
  group = NA,
  group.control = "check",
  popup = tm_popup(),
  popup.vars = NA,
  popup.format = tm_label_format(),
  hover = NA,
  id = "",
  blend = "over",
  options = opt_tm_circles(),
  ...
)

opt_tm_circles(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE
)
```

Arguments

`size`, `size.scale`, `size.legend`, `size.chart`, `size.free`

Visual variable that determines the size. See details. *Unit:* Meters. Accepts a plain numeric vector (values already in meters) or a `units` object from the **units** package (any linear unit, e.g. `units::as_units(50, "km")`), which is converted to meters automatically. *Unit:* Meters. Accepts a plain numeric vector (values already in meters) or a `units` object from the **units** package (any linear unit, e.g. `units::as_units(50, "km")`), which is converted to meters automatically.

`fill`, `fill.scale`, `fill.legend`, `fill.chart`, `fill.free`
: Visual variable that determines the fill color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`lwd`, `lwd.scale`, `lwd.legend`, `lwd.chart`, `lwd.free`
: Visual variable that determines the line width. See details. *Unit:* Base R line-width units; 1 lwd is approx. 0.75 pt at 96 dpi. Controlled by `values.scale`.

`lty`, `lty.scale`, `lty.legend`, `lty.chart`, `lty.free`
: Visual variable that determines the line type. See details. *Unit:* Integer (1-6) or name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

`fill_alpha`, `fill_alpha.scale`, `fill_alpha.legend`, `fill_alpha.chart`, `fill_alpha.free`
: Visual variable that determines the fill color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`plot.order`
: Specification in which order the spatial features are drawn. See `tm_plot_order()` for details.

`zindex`
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

`group`
: Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

`group.control`
: In view mode, the group control determines how layer groups can be switched on and off. Options: `"radio"` for radio buttons (meaning only one group can be shown), `"check"` for check boxes (so multiple groups can be shown), and `"none"` for no control (the group cannot be (de)selected).

`popup`
: popup specification for `"view"` mode, the output of `tm_popup()`. It determines the data variables shown in the popup table, the popup title, and (in the future) the popup layout. This replaces the deprecated arguments `popup.vars` and `popup.format`.

`popup.vars`
: (Deprecated.) Use `popup` with `tm_popup()` instead (via its `vars` argument). Names of data variables that are shown in the popups in `"view"` mode. Set `popup.vars` to `TRUE` to show all variables in the shape object. Set `popup.vars` to `FALSE` to disable popups. Set `popup.vars` to a character vector of variable names to show those variables in the popups. The default (`NA`) depends on whether visual variables (e.g. `fill`) are used. If so, only those are shown. If not, all variables in the shape object are shown.

`popup.format`
: (Deprecated.) Use `popup` with `tm_popup()` instead (via its `format` argument). List of formatting options for the popup values. Output of `tm_label_format()`. Only applicable for numeric data variables. If one list of formatting options is provided, it is applied to all numeric variables of `popup.vars`. Also, a (named) list of lists can be provided. In that case, each list of formatting options is applied to the named variable.

hover	name of the data variable that specifies the hover labels (view mode only). Set to FALSE to disable hover labels. By default FALSE, unless id is specified. In that case, it is set to id,
id	name of the data variable that specifies the indices of the spatial features. Only used for "view" mode.
blend	Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section for the supported values.
options	Options passed on to opt_tm_circles().
...	To catch deprecated arguments from version < 4.0.
points_only	Should only point geometries of the shape object be plotted? Default "ifany" means TRUE whenever the geometry collection contains points.
point_per	How spatial points are generated for non-point geometries. One of "feature" (one point per feature, default), "segment" (one per sub-feature), or "largest" (only the largest sub-feature).
on_surface	For polygon inputs, should the centroid be guaranteed to lie on the surface? If TRUE (slower), centroids outside the polygon are replaced via sf::st_point_on_surface().

Details

Supported visual variables: fill (fill colour), col (border colour), size (radius in meters), lwd (line width), lty (line type), fill_alpha (fill transparency), col_alpha (border transparency).

See Also

- tm_bubbles() for screen-fixed proportional circles (pixel radius).
- tm_symbols() for the general symbol layer with configurable shapes.
- tm_scale_asis() to pass data values directly as metre radii.

Examples

```
## Three concentric layers of geographic circles at different administrative
## levels, each with a fixed radius that corresponds to a real-world distance.
## Because the radius is in meters the circles scale with zoom in view mode.
tm_shape(NLD_prov) +
 tm_circles(size = 5000, fill = "#0033ff", col = NULL) +
tm_shape(NLD_muni) +
 tm_circles(size = 2000, fill = "#99dd99", col = NULL) +
tm_shape(NLD_dist) +
 tm_circles(size = 1000, fill = "#ff8833", col = NULL)

## Use a units object — any linear unit is accepted and converted to meters
NLD_prov$one_mile <- units::as_units(1:12, "mi")
tm_shape(NLD_prov) +
 tm_circles(size = "one_mile", size.scale = tm_scale_asis())
```

tm_compass *Map component: compass*

Description

Map component that adds a compass

Usage

```
tm_compass(
  north,
  type,
  text.size,
  size,
  show.labels,
  cardinal.directions,
  text.color,
  color.dark,
  color.light,
  lwd,
  position,
  group_id,
  bg,
  bg.color,
  bg.alpha,
  stack,
  just,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  margins,
  z,
  ...
)
```

Arguments

north	north
type	compass type, one of: "arrow", "4star", "8star", "radar", "rose". The default is controlled by tm_layout (which uses "arrow" for the default style)
text.size	text.size
size	size
show.labels	show.labels

cardinal.directions	cardinal.directions
text.color	text.color
color.dark	color.dark
color.light	color.light
lwd	lwd
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
bg	Show background?
bg.color	Background color
bg.alpha	Background transparency
stack	stack with other map components, either "vertical" or "horizontal".
just	just
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
margins	margins
z	z index, e.g. the place of the component relative to the other componets
...	to catch deprecated arguments (alpha)

See Also

Vignette about components

`tm_components` *Group components*

Description

Group components

Usage

```
tm_components(
  group_id = "",
  position,
  stack,
  frame_combine,
  equalize,
  resize_as_group,
  stack_margin,
  offset,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  bg,
  bg.color,
  bg.alpha
)
```

Arguments

`group_id`	id of the component group. By default set to `""`, which will apply to all components. There are two other options. 1) To use the same (self-chosen) name that corresponds to the `group_id` argument of a component function, such as `tm_legend()` and `tm_title()`. 2) To specify one (or more) component function names, e.g. `"tm_legend"` or `c("tm_scalebar", "tm_compass")`.
`position`	The position specification of the components in this group: an object created with `tm_pos_in()` or `tm_pos_out()`. Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is `"left"`, `"center"` or `"right"` (or upper case, meaning tighter to the map frame), the second `"top"`, `"center"` or `"bottom"`. Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning.
`stack`	stacking `"horizontal"` or `"vertical"`
`frame_combine`	put frame around all components that are drawn on the same location. Whether a frame is drawn is still decided by the `frame` argument of the 'main' (first) component.

equalize	in case frame_combine is FALSE, should the separate frames be equalized, i.e. have the same width (when stacked vertically) or height (when stacked horizontally)?
resize_as_group	in case a component if rescaled because of the limited space, rescale the other components proportionally?
stack_margin	Margin between components
offset	Offset margin between frame and the components block
frame	Should a frame be drawn? By default TRUE for legends, charts and insets, and FALSE otherwise.
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
bg	Background color the components block. Is usually set in each component function, but if specified here, it will overwrite them.
bg.color	Background color the components block. Is usually set in each component function, but if specified here, it will overwrite them.
bg.alpha	Background alpha transparency of the components block. Is usually set in each component function, but if specified here, it will overwrite them.

Value

A tmap-element

---

tm_const *tmap function to define a constant visual value*

Description

tmap function to define a constant visual value

Usage

```
tm_const()
```

`tm_credits` *Map component: (credits) text*

Description

Map component that adds a text, typically used as credits. This function is the same as `tm_title()` but with different default values.

Usage

```
tm_credits(
  text,
  size,
  color,
  padding,
  fontface,
  fontfamily,
  alpha,
  stack,
  just,
  frame,
  frame.lwd,
  frame.r,
  bg,
  bg.color,
  bg.alpha,
  position,
  group_id,
  width,
  height,
  z,
  ...
)
```

Arguments

`text`	text
`size`	font size
`color`	font color
`padding`	padding
`fontface`	font face, bold, italic
`fontfamily`	font family
`alpha`	alpha transparency of the text
`stack`	stack with other map components, either `"vertical"` or `"horizontal"`.
`just`	just

frame	frame should a frame be drawn?
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
bg	Show background?
bg.color	Background color
bg.alpha	Background transparency
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
width, height	width and height of the component.
z	z index, e.g. the place of the component relative to the other componets
...	to catch deprecated arguments

See Also

Vignette about components

---

tm_crs *Set the map projection (CRS)*

Description

This function sets the map projection. It can also be set via tm_shape(), but tm_crs is generally recommended for most cases. It can also be determined automatically (see details); however, this is still work-in-progress.

Usage

```
tm_crs(crs = NA, property = NA, bbox = NULL)
```

Arguments

crs	Map projection (CRS). Can be set to an crs object (see sf::st_crs()), a proj4string, an EPSG number, the value "auto" (automatic crs recommendation), or one the the following generic projections: c("laea", "aeqd", "utm", "pconic", "eqdc", "stere"). See details.
property	Which property should the projection have? One of: "global", "area" (equal-area), "distance" (equidistant), "shape" (conformal). Only applicable if crs = "auto". See details.
bbox	bounding box. Three options: a sf::st_bbox() object, an Open Street Map query (passed on to tmaptools::geocode_OSM()), or "FULL", which means the whole earth , which means the whole earth (this also guarantees that transformations to another CRS keep the whole earth, unlike sf::st_bbox()).

Details

The map projection (crs) determines in which coordinate system the spatial object is processed and plotted. See vignette about CRS. The crs can be specified in two places: 1) tm_shape() and tm_crs(). In both cases, the map is plotted into the specified crs. The difference is that in the first option, the crs is also taken into account in spatial transformation functions, such as the calculation of centroids and cartograms. In the second option, the crs is only used in the plotting phase.

The automatic crs recommendation (which is still work-in-progress) is the following:

Property	Recommendation
global (for world maps)	A pseudocylindrical projection tmap option crs_global, by default "eqearth (Equal Earth). See
area (equal area)	Lambert Azimuthal Equal Area (laea)
distance (equidistant)	Azimuthal Equidistant (aeqd)
shape (conformal)	Stereographic (stere)

For further info about the available "generic" projects see: for utm: https://proj.org/en/9.4/operations/projections/utm.html for laea: https://proj.org/en/9.4/operations/projections/laea.html for aeqd: https://proj.org/en/9.4/operations/projections/aeqd.html for pconic: https://proj.org/en/9.4/operations/projections/pconic.html for eqdc: https://proj.org/en/9.4/operations/projections/eqdc.html

Note

Plans are to migrate the functionality regarding generic crs and automatic crs recommendation to a separate package.

See Also

vignette about CRS

Examples

```
SA = World[World$continent == "South America", ]

# latlon coordinates (WGS84)
tm_shape(SA) +
 tm_polygons() +
 tm_graticules() +
 tm_crs(4326)

tm_list = lapply(c("global", "area", "distance", "shape"), FUN = function(property) {
 tm_shape(SA) +
  tm_polygons() +
  tm_graticules() +
  tm_crs(property = property) +
 tm_title(property)
})

tmap_arrange(tm_list, nrow = 1)
```

---

tm_facets *Specify facets*

Description

- `tm_facets_wrap()` specify facets for one grouping variable (so one faceting dimension)
- `tm_facets_(hv)stack()` stacks the facets either horizontally or vertically (one grouping variable).
- `tm_facets_grid()` specify facets for two grouping variables in a grid of rows and columns.
- `tm_facets_pagewise()` same as wrap, but the facets are drawn on different plots (pages). Replaces the `along` argument from version 3.
- `tm_facets()` is the core function, but it is recommended to use the other functions.

Usage

```
tm_facets(
  by = NULL,
  rows = NULL,
  columns = NULL,
  pages = NULL,
  as.layers = FALSE,
  nrow = NA,
  ncol = NA,
  byrow = TRUE,
  orientation = NA,
  free.coords = NA,
  drop.units = TRUE,
  drop.empty.facets = TRUE,
```

```
  drop.NA.facets = FALSE,
  sync = TRUE,
  swipe = FALSE,
  na.text = NA,
  scale.factor = 2,
  type = NA,
  free.scales = NULL,
  ...
)

tm_facets_grid(rows = NULL, columns = NULL, pages = NULL, ...)

tm_facets_wrap(by = "VARS__", nrow = NA, ncol = NA, byrow = TRUE, ...)

tm_facets_pagewise(by = "VARS__", byrow = TRUE, ...)

tm_facets_stack(by = "VARS__", orientation = NA, ...)

tm_facets_hstack(by = "VARS__", ...)

tm_facets_vstack(by = "VARS__", ...)

tm_facets_flip(...)
```

Arguments

by	Group by variable (only for a facet wrap or facet stack)
rows	Variable that specifies the rows (only for a facet grid)
columns	Variable that specifies the columns (only for a facet grid)
pages	Variable that specifies the pages (only for a facet grid)
as.layers	show facets as layers?
nrow	Number of rows
ncol	Number of columns
byrow	Should facets be wrapped by row?
orientation	For facet stack: horizontal or vertical?
free.coords	Logical. If the by argument is specified, should each map has its own coordinate ranges? By default TRUE, unless facets are shown in as different layers (as.layers = TRUE)
drop.units	Logical. If the by argument is specified, should non-selected spatial units be dropped? If FALSE, they are plotted where mapped aesthetics are regarded as missing values. Not applicable for raster shapes. By default TRUE.
drop.empty.facets	Logical. If the by argument is specified, should empty facets be dropped? Empty facets occur when the by-variable contains unused levels. When TRUE and two by-variables are specified, empty rows and columns are dropped.

drop.NA.facets	Logical. If the by argument is specified, and all data values for specific facets are missing, should these facets be dropped? FALSE by default. In v3, it was called showNA.
sync	Logical. Should the navigation in view mode (zooming and panning) be synchronized? By default TRUE if the facets have the same bounding box. This is generally the case when rasters are plotted, or when free.coords is FALSE.
swipe	Logical. If TRUE, the two maps are overlaid with a draggable divider to reveal either map. Requires exactly two maps, and only works in the "mapbox" and "maplibre" modes provided by the **tmap.mapgl** package. Default is FALSE.
na.text	Text used for facets of missing values. In v3, it was textNA.
scale.factor	Number that determines how the elements (e.g. font sizes, symbol sizes, line widths) of the small multiples are scaled in relation to the scaling factor of the shapes. The elements are scaled to the scale.factorth root of the scaling factor of the shapes. So, for scale.factor = 1, they are scaled proportional to the scaling of the shapes. Since elements, especially text, are often too small to read, a higher value is recommended. By default, scale.factor = 2.
type	"grid", "wrap" or "stack"
free.scales	deprecated. Please use the .free arguments in the layer functions, e.g. fill.free in tm_polygons.
...	passed on to tm_facets()

Note

In older versions (< 4.1) tm_facets() with page specification was used to create animations frames and tmap_animation() to create the animation itself using inputs like the frame rate specification. As of version 4.2, the whole animation, including frame rate, is specified in tm_animate(). The animation can still be saved via tmap_animation().

See Also

tm_animate()

Vignette about facets

Examples

```
## Not run:
tm_shape(NLD_dist) +
  tm_polygons("edu_appl_sci",
    fill.scale = tm_scale_intervals(values = "pu_gn", style = "kmeans", n = 7)) +
  tm_facets(by = "province") +
tm_shape(NLD_muni) +
  tm_borders(lwd = 3) +
  tm_facets(by = "province") +
tm_title("Population with a univeristy degree (incl appl. sciences), percentages")

tm_shape(World) +
  tm_polygons(c("gender", "press"),
    fill.scale = list(tm_scale_intervals(values = "bu_br_div", midpoint = 0.5),
```

```
      tm_scale_intervals(values = "pu_gn_div", midpoint = 50)),
    fill.legend = tm_legend("")) +
tm_layout(panel.labels = c("Gender Inequality Index (GII)", "World Press Freedom Index"))

## End(Not run)
```

`tm_graticules` *Coordinate grid / graticule lines*

Description

Draws horizontal and vertical lines according to a coordinate reference system (CRS). `tm_grid()` uses the CRS of the (master) shape object, and `tm_graticules()` uses latitude and longitude coordinates (EPSG 4326). It creates a `tmap-element` that draws coordinate grid lines. It serves as a layer that can be drawn anywhere between other layers.

Usage

```
tm_graticules(
  x = NA,
  y = NA,
  n.x = NA,
  n.y = NA,
  crs = 4326,
  labels.format = list(suffix = intToUtf8(176)),
  labels.cardinal = TRUE,
  ...
)

tm_grid(
  x = NA,
  y = NA,
  n.x = NA,
  n.y = NA,
  crs = NA,
  col = NA,
  lwd = 1,
  alpha = NA,
  labels.show = TRUE,
  labels.pos = c("left", "bottom"),
  labels.size = 0.6,
  labels.col = NA,
  labels.rot = c(0, 0),
  labels.format = list(big.mark = ","),
  labels.cardinal = FALSE,
  labels.margin.x = 0,
  labels.margin.y = 0,
```

```
  labels.space.x = NA,
  labels.space.y = NA,
  labels.inside_frame = FALSE,
  ticks = labels.show & !labels.inside_frame,
  lines = TRUE,
  ndiscr = 100,
  zindex = NA,
  group = NA,
  group.control = "none",
  ...
)
```

Arguments

x	X coordinates for vertical grid lines. If NA, it is specified with a pretty scale and n.x.
y	Y coordinates for horizontal grid lines. If NA, it is specified with a pretty scale and n.y.
n.x	Preferred number of grid lines for the x axis. For the labels, a pretty() sequence is used, so the number of actual labels may be different than n.x.
n.y	Preferred number of grid lines for the y axis. For the labels, a pretty() sequence is used, so the number of actual labels may be different than n.y.
crs	Projection character. If specified, the grid lines are projected accordingly. Many world maps are projected, but still have latitude longitude (EPSG 4326) grid lines.
labels.format	List of formatting options for the grid labels. Parameters are: **fun** Function to specify the labels. It should take a numeric vector, and should return a character vector of the same size. By default it is not specified. If specified, the list items scientific, format, and digits (see below) are not used. **scientific** Should the labels be formatted scientifically? If so, square brackets are used, and the format of the numbers is "g". Otherwise, format="f", and text.separator, text.less.than, and text.or.more are used. Also, the numbers are automatically rounded to millions or billions if applicable. **format** By default, "f", i.e. the standard notation xxx.xxx, is used. If scientific=TRUE then "g", which means that numbers are formatted scientifically, i.e. n.dddE+nn if needed to save space. **digits** Number of digits after the decimal point if format="f", and the number of significant digits otherwise. **...** Other arguments passed on to formatC()
labels.cardinal	Add the four cardinal directions (N, E, S, W) to the labels, instead of using negative coordinates for west and south (so it assumes that the coordinates are positive in the north-east direction).
...	Used to catch deprecated arguments from tmap v3.
col	Color of the grid lines.

lwd	Line width of the grid lines
alpha	Alpha transparency of the grid lines. Number between 0 and 1. By default, the alpha transparency of col is taken.
labels.show	Show tick labels. Either one value for both x and y axis, or a vector two: the first for x and latter for y.
labels.pos	position of the labels. Vector of two: the horizontal ("left" or "right") and the vertical ("top" or "bottom") position.
labels.size	Font size of the tick labels
labels.col	Font color of the tick labels
labels.rot	Rotation angles of the labels. Vector of two values: the first is the rotation angle (in degrees) of the tick labels on the x axis and the second is the rotation angle of the tick labels on the y axis. Only 0, 90, 180, and 270 are valid values.
labels.margin.x	Margin between tick labels of x axis and the frame. Note that when labels.inside_frame = FALSE and ticks = TRUE, the ticks will be adjusted accordingly.
labels.margin.y	Margin between tick labels of y axis and the frame. Note that when labels.inside_frame = FALSE and ticks = TRUE, the ticks will be adjusted accordingly.
labels.space.x	Space that is used for the labels and ticks for the x-axis when labels.inside_frame = FALSE. By default, it is determined automatically using the widths and heights of the tick labels. The unit of this parameter is text line height.
labels.space.y	Space that is used for the labels and ticks for the y-axis when labels.inside_frame = FALSE. By default, it is determined automatically using the widths and heights of the tick labels. The unit of this parameter is text line height.
labels.inside_frame	Show labels inside the frame? By default FALSE.
ticks	If labels.inside_frame = FALSE, should ticks can be drawn between the labels and the frame? Either one value for both x and y axis, or a vector two: the first for x and latter for y.
lines	If labels.inside_frame = FALSE, should grid lines can be drawn?
ndiscr	Number of points to discretize a parallel or meridian (only applicable for curved grid lines)
zindex	Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.
group	Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see group.control)
group.control	In view mode, the group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).

Details

#' In view mode, each layer is rendered in a Leaflet pane named "tmap{zindex}" (e.g., "tmap401", "tmap402"), with base tile layers placed in the standard "tile" pane.

Examples

```
## Not run:
current.mode <- tmap_mode("plot")

tm_shape(NLD_muni) +
 tm_polygons() +
 tm_grid()

tm_shape(NLD_muni) +
 tm_polygons() +
 tm_grid(crs = 4326)

tm_shape(NLD_muni) +
 tm_polygons() +
 tm_grid(crs = 3035, labels.inside.frame = TRUE)

tm_shape(World) +
 tm_polygons() +
 tm_facets(by = "continent") +
 tm_grid(labels.inside.frame = TRUE)

tm_shape(NLD_muni) +
 tm_polygons() +
 tm_graticules()

tm_shape(NLD_muni) +
 tm_polygons() +
 tm_graticules(labels.pos = c("right", "top"))


data(NLD_muni, World)

tmap_arrange(
 qtm(NLD_muni) + tm_grid(),
 qtm(NLD_muni) + tm_graticules()
)

qtm(World, shape.crs = "+proj=robin", style = "natural") +
 tm_graticules(ticks = FALSE) +
 tm_layout(frame=FALSE)

tmap_mode(current.mode)

## End(Not run)
```

---

tm_group *Layer group control*

Description

Controls the layer groups in interactive maps (view mode): the layer control box (radio buttons or check boxes) and at which zoom levels the layers are displayed at.

Usage

```
tm_group(name, control = NA, zoom_levels = NA)
```

Arguments

name	group name that corresponds with the group name specified in the layer functions (e.g. tm_polygons())
control	The group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).
zoom_levels	The zoom levels at which the group is displays at. When specified control will be set to "none".

See Also

vignette about layer groups

---

tm_inset *Map component: inset maps and other objects*

Description

Map component that adds an inset object, e.g. a mini map

Usage

```
tm_inset(
  x = NULL,
  height,
  width,
  margins,
  between_margin,
  position,
  group_id,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  bg,
```

```
  bg.color,
  bg.alpha,
  box_frame,
  box_frame.color,
  box_frame.alpha,
  box_frame.lwd,
  box_frame.lty,
  box_bg,
  box_bg.color,
  box_bg.alpha,
  main_frame,
  main_frame.r,
  main_frame.color,
  main_frame.alpha,
  main_frame.lwd,
  z
)
```

Arguments

x	object to draw. Can be: bounding box, tmap object, ggplot2 object, grob object, image file name.
height	height of the component in number of text line heights.
width	width of the component in number of text line heights.
margins	margins
between_margin	Margin between
position	The position specification of the component: an object created with `tm_pos_in()` or `tm_pos_out()`. Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is `"left"`, `"center"` or `"right"` (or upper case, meaning tighter to the map frame), the second `"top"`, `"center"` or `"bottom"`. Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use `group_id` and specify `component` in `tm_components()`.
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same `group_id` will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in `tm_components()` where its argument `id` should correspond to `group_id`.
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
bg	Show background?
bg.color	Background color
bg.alpha	Background transparency

box_frame — Should a box frame be drawn in the main map that shows where the inset is? TRUE by default

box_frame.color, box_frame.alpha, box_frame.lwd, box_frame.lty — Properties of the box frame

box_bg — Should the frame box have a background? FALSE by default

box_bg.color, box_bg.alpha — Properties of the box background

main_frame — Should a frame be drawn around the inset map? Note that this is different from the general map component frame (the argument frame)

main_frame.r, main_frame.color, main_frame.alpha, main_frame.lwd — Properties of the main frame

z — z index, e.g. the place of the component relative to the other componets

Examples

```
## map
bb = tmaptools::bb(NLD_prov[NLD_prov$name == "Utrecht",], ext = 1.05)

bb_Randstad =
  sf::st_bbox(c(xmin = 120000, xmax = 150000, ymin = 460000, ymax = 500000), crs = 28992)

tm_shape(NLD_dist) +
 tm_polygons(
  fill = "dwelling_value",
  fill.scale = tm_scale_continuous_pseudo_log(values = "-cols4all.pu_gn_div"),
  col = NULL) +
tm_shape(NLD_muni) +
 tm_borders(col = "black", lwd = 0.5) +
tm_shape(NLD_prov) +
 tm_borders(col = "black", lwd = 1.5) +
tm_inset(bb_Randstad, height = 12, width = 12, position = c("left", "top")) +
 tm_compass(position = c("left", "top"), )

## ggplot2
if (requireNamespace("ggplot2")) {
 library(ggplot2)
 p = ggplot(World, aes(x = gender, y = press, colour = continent)) +
  geom_point() +
  theme_bw()

 tm_shape(World) +
  tm_polygons(
   fill = "gender",
   fill.scale = tm_scale(values = "-cols4all.pu_gn_div")) +
  tm_inset(p, height = 15, width = 20, position = tm_pos_in("left", "bottom"))
}
```

tm_iso *Map layer: iso (contour)*

Description

Map layer that draws iso (contour) lines. Stack of tm_lines() and tm_labels_highlighted.

Usage

```
tm_iso(
  col = tm_const(),
  text = tm_vars(x = 1),
  ...,
  options_lines = opt_tm_lines(),
  options_labels = opt_tm_labels()
)
```

Arguments

col	Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.
text	Visual variable that determines the text. See details. *Unit:* Character string.
...	passed on to tm_lines() and tm_labels_highlighted(). For the text color and alpha transparency of the text labels, please use text_col and text_alpha instead of col and col_alpha.
options_lines	The options for tm_lines()
options_labels	The options for tm_labels_highlighted()

tm_label_format *tmap function to specify labels*

Description

tmap function to specify labels used in the scale functions, e.g. via the argument label.format in tm_scale_intervals().

Usage

```
tm_label_format(
  fun,
  scientific,
  format,
  digits,
  interval.disjoint,
```

```
  big.num.abbr,
  prefix,
  suffix,
  text.separator,
  text.less.than,
  text.less.than_as.prefix,
  text.or.more,
  text.or.more_as.prefix,
  text.align,
  text.to.columns,
  html.escape,
  ...
)
```

Arguments

fun	Function to specify the labels. It should take a numeric vector, and should return a character vector of the same size. By default it is not specified. If specified, the list items scientific, format, and digits (see below) are not used.
scientific	Should the labels be formatted scientifically? If so, square brackets are used, and the format of the numbers is "g". Otherwise, format="f", and text.separator, text.less.than, and text.or.more are used. Also, the numbers are automatically rounded to millions or billions if applicable. By default, FALSE
format	By default, "f", i.e. the standard notation xxx.xxx, is used. If scientific = TRUE then "g", which means that numbers are formatted scientifically, i.e. n.dddE+nn if needed to save space.
digits	Number of digits after the decimal point if format="f", and the number of significant digits otherwise. By default NA, meaning as many as needed to have distinct numbers
interval.disjoint	In case of intervals (see tm_scale_intervals()), should the intervals appear disjoint, e.g. 0 to 999, 1000 - 1999, 2000 - 2999 (TRUE, default), or not: 0 - 1000, 1000 - 2000, 2000- 3000.
big.num.abbr	Vector that defines whether and which abbrevations are used for large numbers. It is a named numeric vector, where the name indicated the abbreviation, and the number the magnitude (in terms on numbers of zero). Numbers are only abbrevation when they are large enough. Set it to NA to disable abbrevations. The default is c("mln" = 6, "bln" = 9). For layers where style is set to log10 or log10_pretty, the default is NA.
prefix	Prefix of each number
suffix	Suffix of each number
text.separator	Character string to use to separate numbers in an interval legend (default: "to").
text.less.than	Character value(s) to use for 'less than'. Default "Less than". When a character vector of length 2 is specified, one for each word, these words are aligned when text.to.columns = TRUE
text.less.than_as.prefix	Should text.less.than be used as prefix?

text.or.more	Character value(s) to use to 'or more'. Default is "or more". When a character vector of length 2 is specified, one for each word, these words are aligned when text.to.columns = TRUE
text.or.more_as.prefix	Should text.or.more be used as prefix?
text.align	Not implemented in v4 (yet). Value that determines how the numbers are aligned, "left", "center" or "right". By default "left".
text.to.columns	Not implemented in v4 (yet). Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default FALSE.
html.escape	Logical that determins whther HTML code is escaped in the popups in view mode. By default TRUE. If set to FALSE HTML code can be added, e.g. to added white space via .
...	arguments passed on to formatC

Value

list with formatting options

---

tm_legend *Legend*

Description

Legend specification

Usage

```
tm_legend(
  title,
  show,
  orientation,
  reverse,
  na.show,
  position,
  group_id,
  width,
  height,
  z,
  title.color,
  title.size,
  title.fontface,
  title.fontfamily,
  title.alpha,
  title.padding,
  title.align,
```

```
  text.color,
  text.size,
  text.fontface,
  text.fontfamily,
  text.alpha,
  format,
  frame,
  frame.lwd,
  frame.r,
  bg,
  bg.color,
  bg.alpha,
  absolute_fontsize,
  item.height,
  item.width,
  item.space,
  item.na.height,
  item.na.width,
  item.na.space,
  item.shape,
  ticks,
  ticks.disable.na,
  ticks.col,
  ticks.lwd,
  margins,
  item_text.margin,
  ...
)

tm_legend_hide()

tm_legend_combine(variable)

tm_legend_bivariate(
  xlab,
  ylab,
  xlab.color,
  xlab.size,
  xlab.rot,
  xlab.fontface,
  xlab.fontfamily,
  xlab.alpha,
  xlab.padding,
  xlab.align,
  ylab.color,
  ylab.size,
  ylab.rot,
  ylab.fontface,
```

```
  ylab.fontfamily,
  ylab.alpha,
  ylab.padding,
  ylab.align,
  ...
)
```

Arguments

title	Legend title
show	Show legend?
orientation	Orientation of the legend: "portrait" or "landscape"
reverse	Should the legend items be reversed?
na.show	Show NA values in legend?
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
width	Width of the legend. Units are 'text line heights'. In case a negative number is specified, the units are (approximate) pixels. The relation between these two is configured via the option absolute_fontsize.
height	Height of the legend. Units are 'text line heights'. In case a negative number is specified, the units are (approximate) pixels. The relation between these two is configured via the option absolute_fontsize.
z	z index, e.g. the place of the component relative to the other componets
title.color	The color of the title of the legend.
title.size	The size of the title of the legend.
title.fontface	The font face of the title of the legend. See graphics::par, option 'font'.
title.fontfamily	The font family of the title of the legend. See graphics::par, option 'family'.
title.alpha	The alpha transparency of the title of the legend.
title.padding	The padding of the title of the legend. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
title.align	The align of the title of the legend.
text.color	The color of the text of the legend.

text.size	The size of the text of the legend.
text.fontface	The font face of the text of the legend. See graphics::par, option 'font'.
text.fontfamily	The font family of the text of the legend. See graphics::par, option 'family'.
text.alpha	The alpha transparency of the text of the legend.
format	Not used anymore: use the format argument of the tm_scale_*() functions instead.
frame	frame should a frame be drawn?
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
bg	Show background?
bg.color	The color of the bg of the legend.
bg.alpha	The alpha transparency of the bg of the legend.
absolute_fontsize	The absolute fontsize of the legend. So far, only used to calculate legend dimensions
item.height	The height of the item of the legend.
item.width	The width of the item of the legend.
item.space	The space of the item of the legend. In terms of number of text line heights.
item.na.height	The height of the na item of the legend.
item.na.width	The width of the na item of the legend.
item.na.space	The space of the na item of the legend. In terms of number of text line heights.
item.shape	The shape of the item of the legend.
ticks	List of vectors of size 2 that determines a tick mark line (for portrait legends). The values are the y-values of begin and endpoint of a tick mark. For a solid line, only one vector is required, for dashed lines one for each dash. See ggplot2 style example.
ticks.disable.na	Remove ticks for NA values
ticks.col	The color of the ticks of the legend.
ticks.lwd	The line width of the ticks of the legend. See graphics::par, option 'lwd'.
margins	The margins of the legend. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
item_text.margin	The margin of the space between item and text of the legend.
...	visual values, e.g. col, fill, lwd, can be specified. If so, they overrule the default visual values, which are determined by the drawn map objects (e.g. polygons)
variable	visual (or transformation) variable to combine the legend with: e.g. "fill" or "size"

xlab	label for the x dimension (rows)
ylab	label for the y dimension (columns)
xlab.color	The color of the xlab of the legend.
xlab.size	The size of the xlab of the legend.
xlab.rot	The rot of the xlab of the legend.
xlab.fontface	The font face of the xlab of the legend. See graphics::par, option 'font'.
xlab.fontfamily	The font family of the xlab of the legend. See graphics::par, option 'family'.
xlab.alpha	The alpha transparency of the xlab of the legend.
xlab.padding	The padding of the xlab of the legend. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
xlab.align	The align of the xlab of the legend.
ylab.color	The color of the ylab of the legend.
ylab.size	The size of the ylab of the legend.
ylab.rot	The rot of the ylab of the legend.
ylab.fontface	The font face of the ylab of the legend. See graphics::par, option 'font'.
ylab.fontfamily	The font family of the ylab of the legend. See graphics::par, option 'family'.
ylab.alpha	The alpha transparency of the ylab of the legend.
ylab.padding	The padding of the ylab of the legend. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
ylab.align	The align of the ylab of the legend.

Value

A tm_legend component

See Also

Vignette about legends

Examples

```
# Example using different settings from tm_legend()

tm_shape(World) +
  tm_polygons(
    fill = "HPI",
    fill.legend = tm_legend(
      title = "Home Price Index",
      title.color = "orange",
      bg.color = "purple",
      show = TRUE
```

```
 ),
    id = "name",
    # Format the labels using dollar sign
    fill.scale = tm_scale_intervals(
      label.format = function(x) format(x, big.mark = " ")),
)
```

`tm_lines` *Map layer: lines*

Description

Map layer that draws lines. Supported visual variables are: `col` (the color), `lwd` (line width), `lty` (line type), and `col_alpha` (color alpha transparency).

Usage

```
tm_lines(
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.chart = tm_chart_none(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.chart = tm_chart_none(),
  lty.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  linejoin = "round",
  lineend = "round",
  plot.order = tm_plot_order("lwd", reverse = TRUE, na.order = "bottom"),
  zindex = NA,
  group = NA,
  group.control = "check",
  popup = tm_popup(),
  popup.vars = NA,
  popup.format = tm_label_format(),
```

```
  hover = NA,
  id = "",
  blend = "over",
  options = opt_tm_lines(),
  ...
)

opt_tm_lines(lines.only = "ifany", hitbox = "auto")
```

Arguments

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`lwd`, `lwd.scale`, `lwd.legend`, `lwd.chart`, `lwd.free`
: Visual variable that determines the line width. See details. *Unit:* Base R line-width units; 1 lwd is approx. 0.75 pt at 96 dpi. Controlled by `values.scale`.

`lty`, `lty.scale`, `lty.legend`, `lty.chart`, `lty.free`
: Visual variable that determines the line type. See details. *Unit:* Integer (1-6) or name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`linejoin`, `lineend`
: line join and line end. See gpar() for details.

`plot.order`
: Specification in which order the spatial features are drawn. See `tm_plot_order()` for details.

`zindex`
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

`group`
: Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

`group.control`
: In view mode, the group control determines how layer groups can be switched on and off. Options: `"radio"` for radio buttons (meaning only one group can be shown), `"check"` for check boxes (so multiple groups can be shown), and `"none"` for no control (the group cannot be (de)selected).

`popup`
: popup specification for `"view"` mode, the output of `tm_popup()`. It determines the data variables shown in the popup table, the popup title, and (in the future) the popup layout. This replaces the deprecated arguments `popup.vars` and `popup.format`.

`popup.vars`
: (Deprecated.) Use `popup` with `tm_popup()` instead (via its `vars` argument). Names of data variables that are shown in the popups in `"view"` mode. Set `popup.vars` to `TRUE` to show all variables in the shape object. Set `popup.vars` to `FALSE` to disable popups. Set `popup.vars` to a character vector of variable names to show those variables in the popups. The default (`NA`) depends on whether visual variables (e.g. `fill`) are used. If so, only those are shown. If not, all variables in the shape object are shown.

popup.format (Deprecated.) Use popup with tm_popup() instead (via its format argument). List of formatting options for the popup values. Output of tm_label_format(). Only applicable for numeric data variables. If one list of formatting options is provided, it is applied to all numeric variables of popup.vars. Also, a (named) list of lists can be provided. In that case, each list of formatting options is applied to the named variable.

hover name of the data variable that specifies the hover labels (view mode only). Set to FALSE to disable hover labels. By default FALSE, unless id is specified. In that case, it is set to id,

id name of the data variable that specifies the indices of the spatial features. Only used for "view" mode.

blend Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section for the supported values.

options options passed on to the corresponding opt_<layer_function> function

... to catch deprecated arguments from version < 4.0

lines.only should only line geometries of the shape object (defined in tm_shape()) be plotted, or also other geometry types (like polygons)? By default "ifany", which means TRUE in case a geometry collection is specified.

hitbox Controls whether an invisible interaction layer with a larger clickable area (\"hitbox\") is added on top of the lines.

This can improve click and popup behaviour for thin or densely packed lines by increasing the effective mouse interaction width.

Possible values:

"none" No additional hitbox layer is added. Lines are clickable only at their visible width.

\item{"plusX"}{Adds \code{X} pixels to the visible line width to compute the interaction width: \code{line_width + X}. For example, \code{"plus8"} widens the clickable area by 4 pixels on each side.}

\item{"pmaxX"}{Ensures a minimum interaction width of \code{X} pixels: \code{pmax(line_width, X)}. For example, \code{"pmax8"} guarantees at least 8 pixels. Useful for very thin lines.}

\item{"auto"}{\code{"pmax8"} if and only if interactive features are enabled (popup or hover), lines are thin (median line width < 4), and there are fewer than 10000 features. Otherwise \code{"none"}.}

plus and pmax can be combined, e.g. "plus4pmax8" means pmax(line_width + 4, 8).

Adding a hitbox improves usability for thin lines but may reduce performance for very large datasets, as an additional invisible layer is rendered.

Details

The visual variable arguments (e.g. col) can be specified with a data variable name (e.g., a spatial vector attribute or a raster layer of the object specified in tm_shape()), with a visual value (for col,

a color is expected), or with a geometry-derived variable (see below). See vignette about visual variables.

Multiple values can be specified: in that case facets are created. These facets can be combined with other faceting data variables, specified with `tm_facets()`. See vignette about facets.

- The `*.scale` arguments determine the used scale to map the data values to visual variable values. These can be specified with one of the available `tm_scale_*()` functions. The default is specified by the tmap option (`tm_options()`) `scales.var`. See vignette about scales.
- The `*.legend` arguments determine the used legend, specified with `tm_legend()`. The default legend and its settings are determined by the tmap options (`tm_options()`) `legend.`. See vignette about legends.
- The `*.chart` arguments specify additional charts, specified with `tm_chart_`, e.g. `tm_chart_histogram()`. See vignette about charts.
- The `*.free` arguments determine whether scales are applied freely across facets, or shared. A logical value is required. They can also be specified with a vector of three logical values; these determine whether scales are applied freely per facet dimension. This is only useful when facets are applied (see `tm_facets()`). There are maximally three facet dimensions: rows, columns, and pages. This only applies for a facet grid (`tm_facets_grid()`). For instance, `col.free = c(TRUE, FALSE, FALSE)` means that for the visual variable `col`, each row of facets will have its own scale, and therefore its own legend. For facet wraps and stacks (`tm_facets_wrap()` and `tm_facets_stack()`) there is only one facet dimension, so the `*.free` argument requires only one logical value.

Currently, three geometry-derived variables are implemented:

- `"AREA"` (polygons only), which uses the feature area;
- `"LENGTH"` (lines only), which uses the feature length; and
- `"MAP_COLORS"`, which assigns values so that adjacent features receive different values, making it particularly suitable for coloring neighbouring polygons.

Note that geometry-derived variables do not generate a legend automatically. If a legend is required, compute the corresponding variable explicitly, for example with `sf::st_area()`, `sf::st_length()`, or `tmaptools::map_coloring()`, and use the resulting values instead.

Visual variable units:

Every visual variable maps data values to a specific output unit. Knowing the unit matters when supplying constant values via `tm_const()`, or output ranges via `values.range` / `values.scale` in the scale functions.

Variable	Output unit	Notes
`fill`, `col`, `bgcol`	color	name, hex, or palette string
`fill_alpha`, `col_alpha`, `bgcol_alpha`	proportion 0-1	0 = transparent, 1 = opaque
`size` (symbols, bubbles, squares, dots)	typographic lines	1 line approx. 1/6 inch; scaled by `values.scale`
`size` (circles)	meters	plain numeric or a `units` object
`size` (text, labels)	multiplier	1 = 12 pt (plot) / 12 px (view)
`lwd`	lwd	base R units; 1 lwd approx. 0.75 pt at 96 dpi
`lty`	–	integer 1-6 or name ("solid", "dashed", ...)
`shape`	–	integer `pch` 1-25 or single character

angle	degrees	0-360, clockwise from north
fontface	–	"plain", "bold", "italic", "bold.italic"

*Symbol size (*size *in* tm_symbols, tm_bubbles, tm_squares, tm_dots*):*
"Lines" is a typographic unit: one line is approximately 1/6 inch (the default base line-height in R graphics). The global multiplier tmap_options(values.scale = list(size.bubbles = 1.5)) scales all symbol sizes without changing the data mapping.

*Circle size (*size *in* tm_circles*):*
The value is a geographic radius in meters. A plain numeric vector is interpreted as meters; a units object (from the **units** package) is automatically converted, so units::as_units(1, "mi") gives a 1-mile radius. Because the radius is geographic, circles scale with zoom in interactive (view) mode – unlike bubble symbols which keep a fixed screen size.

*Text size (*size *in* tm_text, tm_labels*):*
The value is a multiplier of the base font size. size = 1 renders at 12 pt in plot mode (R's default par("ps")) and at 12 px in view mode (gp$cex * 12 px, see tmapLeafletDataPlot.tm_data_text); the two modes are consistent by design.

Layer blending (blend)**:**

Blend modes control how a layer's pixels are combined with the pixels beneath it. For each pixel, let S be the source (top layer) RGB value and D be the destination (bottom layer) RGB value, both normalised to $[0, 1]$.

blend	Formula	Use case
"over"	$S \cdot \alpha + D \cdot (1 - \alpha)$	Standard alpha compositing (default)
"multiply"	$S \times D$	Hillshading over colour raster; both layers darken each other
"screen"	$1 - (1 - S)(1 - D)$	Inverse of multiply; brightens
"overlay"	multiply if $D < 0.5$, screen if $D \geq 0.5$	Boosts contrast; preserves midtones
"darken"	$\min(S, D)$	Keeps the darker of the two layers per channel
"lighten"	$\max(S, D)$	Keeps the lighter of the two layers per channel

Requires R >= 4.2 and a compatible graphics device (e.g. png(type = "cairo"), svg()). In view mode, blending is applied via CSS mix-blend-mode. See grid::groupGrob() for the full list of supported operators.

zindex and pane names:

In view mode, each layer is rendered in a Leaflet pane named "tmap{zindex}" (e.g., "tmap401", "tmap402"), with base tile layers placed in the standard "tile" pane.

See Also

Terrain map example

Examples

```
tm_shape(World_rivers) +
 tm_lines(lwd = "strokelwd",
```

```
    lwd.scale = tm_scale_asis(values.scale = 0.2, value.neutral = 2),
    col = "scalerank",
    col.scale = tm_scale_categorical(values = "seaborn.dark"))

tm_shape(World) +
 tm_lines(col = "continent",
    col.scale = tm_scale_categorical(values = "seaborn.dark"),
    lty = "continent",
    lwd = 1.5,
    lty.legend = tm_legend_combine("col"))
```

`tm_logo` *Map component: logo*

Description

Map component that adds a logo.

Usage

```
tm_logo(
  file,
  height,
  margins,
  between_margin,
  stack,
  position,
  group_id,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  z
)
```

Arguments

`file`	either a filename or url of a png image. If multiple files/urls are provided with a character vector, the logos are placed near each other. To specify logos for small multiples use a list of character values/vectors. In order to stack logos vertically, multiple `tm_logo` elements can be stacked.
`height`	height of the logo in number of text line heights. The width is scaled based the height and the aspect ratio of the logo. If multiple logos are specified by a vector or list, the heights can be specified accordingly.
`margins`	margins
`between_margin`	Margin between

stack	stack with other map components, either "vertical" or "horizontal".
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
z	z index, e.g. the place of the component relative to the other componets

See Also

Vignette about components

Examples

```
data(World)

tm_shape(World) +
 tm_polygons("HPI", fill.scale = tm_scale_intervals(values = "brewer.rd_yl_gn")) +
 tm_logo(c("https://www.r-project.org/logo/Rlogo.png",
     system.file("help", "figures", "logo.png", package = "tmap"))) +
 tm_logo("https://happyplanetindex.org/wp-content/themes/hpi/public/images/hpi-logo.svg",
   height=5, position = c("left", "center"))
```

---

tm_minimap *Map component: minimap*

Description

Map component that adds a minimap in view mode.

Usage

```
tm_minimap(
  server,
  toggle,
  height,
  width,
  margins,
  between_margin,
  position,
  group_id,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  bg,
  bg.color,
  bg.alpha,
  z,
  ...
)
```

Arguments

server	name of the provider or an URL (see tm_tiles). By default, it shows the same map as the basemap, and moreover, it will automatically change when the user switches basemaps. Note the latter does not happen when server is specified.
toggle	should the minimap have a button to minimise it? By default TRUE.
width, height	width and height of the component.
margins	margins
between_margin	Margin between
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy

`frame.lwd` frame line width

`frame.r` Radius of the rounded frame corners. 0 means no rounding.

`bg` Show background?

`bg.color` Background color

`bg.alpha` Background transparency

`z` z index, e.g. the place of the component relative to the other componets

`...` Arguments passed on to `leaflet::addMiniMap`

`map` a map widget object

`collapsedWidth` The width of the toggle marker and the minimap when collapsed, in pixels. Defaults to 19.

`collapsedHeight` The height of the toggle marker and the minimap when collapsed, in pixels. Defaults to 19.

`zoomLevelOffset` The offset applied to the zoom in the minimap compared to the zoom of the main map. Can be positive or negative, defaults to -5.

`zoomLevelFixed` Overrides the offset to apply a fixed zoom level to the minimap regardless of the main map zoom. Set it to any valid zoom level, if unset zoomLevelOffset is used instead.

`centerFixed` Applies a fixed position to the minimap regardless of the main map's view / position. Prevents panning the minimap, but does allow zooming (both in the minimap and the main map). If the minimap is zoomed, it will always zoom around the centerFixed point. You can pass in a LatLng-equivalent object. Defaults to false.

`zoomAnimation` Sets whether the minimap should have an animated zoom. (Will cause it to lag a bit after the movement of the main map.) Defaults to false.

`toggleDisplay` Sets whether the minimap should have a button to minimize it. Defaults to false.

`autoToggleDisplay` Sets whether the minimap should hide automatically, if the parent map bounds does not fit within the minimap bounds. Especially useful when 'zoomLevelFixed' is set.

`minimized` Sets whether the minimap should start in a minimized position.

`aimingRectOptions` Sets the style of the aiming rectangle by passing in a Path.Options (`https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#path-options`) object. (Clickable will always be overridden and set to false.)

`shadowRectOptions` Sets the style of the aiming shadow rectangle by passing in a Path.Options (`https://web.archive.org/web/20220702182250/https://leafletjs.com/reference-1.3.4.html#path-option`) object. (Clickable will always be overridden and set to false.)

`strings` Overrides the default strings allowing for translation.

`tiles` URL for tiles or one of the pre-defined providers.

`mapOptions` Sets Leaflet options for the MiniMap map. It does not override the MiniMap default map options but extends them.

See Also

Vignette about components

tm_mouse_coordinates *Map component: mouse coordinates*

Description

Map component that adds mouse coordinates

Usage

```
tm_mouse_coordinates(stack, position, group_id, z)
```

Arguments

stack	stack with other map components, either "vertical" or "horizontal".
position	The position specification of the component: an object created with tm_pos_in() or tm_pos_out(). Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is "left", "center" or "right" (or upper case, meaning tighter to the map frame), the second "top", "center" or "bottom". Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use group_id and specify component in tm_components().
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same group_id will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in tm_components() where its argument id should correspond to group_id.
z	z index, e.g. the place of the component relative to the other componets

See Also

Vignette about components

tm_options *tmap options*

Description

tmap options

Usage

```
tm_options(
  crs,
  facet.max,
  free.scales,
  raster.max_cells,
  raster.warp,
  show.messages,
  show.warnings,
  output.format,
  output.size,
  output.dpi,
  animation.dpi,
  value.const,
  value.na,
  value.null,
  value.blank,
  values.var,
  values.range,
  value.neutral,
  values.scale,
  scales.var,
  scale.misc.args,
  continuous.nclass_per_legend_break,
  continuous.nclasses,
  label.format,
  label.na,
  scale,
  asp,
  bg,
  bg.color,
  outer.bg,
  outer.bg.color,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  frame.double_line,
  outer.margins,
  inner.margins,
  inner.margins.extra,
  meta.margins,
  meta.auto_margins,
  between_margin,
  panel.margin,
  xlab.show,
  xlab.text,
```

```
    xlab.size,
    xlab.color,
    xlab.rotation,
    xlab.space,
    xlab.fontface,
    xlab.fontfamily,
    xlab.alpha,
    xlab.side,
    ylab.show,
    ylab.text,
    ylab.size,
    ylab.color,
    ylab.rotation,
    ylab.space,
    ylab.fontface,
    ylab.fontfamily,
    ylab.alpha,
    ylab.side,
    panel.type,
    panel.wrap.pos,
    panel.xtab.pos,
    unit,
    color.sepia_intensity,
    color.saturation,
    color_vision_deficiency_sim,
    text.fontface,
    text.fontfamily,
    r,
    component.position,
    component.offset,
    component.stack_margin,
    component.autoscale,
    component.resize_as_group,
    component.frame_combine,
    component.stack,
    legend.stack,
    chart.stack,
    component.equalize,
    component.frame,
    component.frame.color,
    component.frame.alpha,
    component.frame.lwd,
    component.frame.r,
    component.bg,
    component.bg.color,
    component.bg.alpha,
    legend.show,
    legend.orientation,
```

```
legend.position,
legend.width,
legend.height,
legend.reverse,
legend.na.show,
legend.title.color,
legend.title.size,
legend.title.fontface,
legend.title.fontfamily,
legend.title.alpha,
legend.xlab.color,
legend.xlab.size,
legend.xlab.rot,
legend.xlab.fontface,
legend.xlab.fontfamily,
legend.xlab.alpha,
legend.ylab.color,
legend.ylab.size,
legend.ylab.rot,
legend.ylab.fontface,
legend.ylab.fontfamily,
legend.ylab.alpha,
legend.text.color,
legend.text.size,
legend.text.fontface,
legend.text.fontfamily,
legend.text.alpha,
legend.frame,
legend.frame.color,
legend.frame.alpha,
legend.frame.lwd,
legend.frame.r,
legend.bg,
legend.bg.color,
legend.bg.alpha,
legend.only,
legend.absolute_fontsize,
legend.settings.portrait,
legend.settings.landscape,
add_legend.position,
chart.show,
chart.plot.axis.x,
chart.plot.axis.y,
chart.position,
chart.width,
chart.height,
chart.reverse,
chart.na.show,
```

```
      chart.title.color,
      chart.title.size,
      chart.title.fontface,
      chart.title.fontfamily,
      chart.title.alpha,
      chart.xlab.color,
      chart.xlab.size,
      chart.xlab.fontface,
      chart.xlab.fontfamily,
      chart.xlab.alpha,
      chart.ylab.color,
      chart.ylab.size,
      chart.ylab.fontface,
      chart.ylab.fontfamily,
      chart.ylab.alpha,
      chart.text.color,
      chart.text.size,
      chart.text.fontface,
      chart.text.fontfamily,
      chart.text.alpha,
      chart.frame,
      chart.frame.color,
      chart.frame.alpha,
      chart.frame.lwd,
      chart.frame.r,
      chart.bg,
      chart.bg.color,
      chart.bg.alpha,
      chart.object.color,
      title.size,
      title.color,
      title.fontface,
      title.fontfamily,
      title.alpha,
      title.padding,
      title.frame,
      title.frame.color,
      title.frame.alpha,
      title.frame.lwd,
      title.frame.r,
      title.position,
      title.width,
      credits.size,
      credits.color,
      credits.fontface,
      credits.fontfamily,
      credits.alpha,
      credits.padding,
```

```
credits.position,
credits.width,
credits.height,
compass.north,
compass.type,
compass.text.size,
compass.size,
compass.show.labels,
compass.cardinal.directions,
compass.text.color,
compass.color.dark,
compass.color.light,
compass.lwd,
compass.margins,
compass.position,
inset.position,
logo.height,
logo.margins,
logo.between_margin,
logo.position,
inset_map.height,
inset_map.width,
inset_map.margins,
inset_map.between_margin,
inset_map.position,
inset_map.frame,
inset.height,
inset.width,
inset.margins,
inset.between_margin,
inset.frame,
inset.bg,
inset.bg.color,
inset.bg.alpha,
inset_grob.height,
inset_grob.width,
inset_gg.height,
inset_gg.width,
scalebar.breaks,
scalebar.width,
scalebar.allow_clipping,
scalebar.text.size,
scalebar.text.color,
scalebar.text.fontface,
scalebar.text.fontfamily,
scalebar.color.dark,
scalebar.color.light,
scalebar.lwd,
```

```
scalebar.size,
scalebar.margins,
scalebar.position,
grid.show,
grid.labels.pos,
grid.x,
grid.y,
grid.n.x,
grid.n.y,
grid.crs,
grid.col,
grid.lwd,
grid.alpha,
grid.labels.show,
grid.labels.size,
grid.labels.col,
grid.labels.fontface,
grid.labels.fontfamily,
grid.labels.rot,
grid.labels.format,
grid.labels.cardinal,
grid.labels.margin.x,
grid.labels.margin.y,
grid.labels.space.x,
grid.labels.space.y,
grid.labels.inside_frame,
grid.ticks,
grid.lines,
grid.ndiscr,
mouse_coordinates.position,
minimap.server,
minimap.toggle,
minimap.position,
panel.show,
panel.labels,
panel.label.size,
panel.label.color,
panel.label.fontface,
panel.label.fontfamily,
panel.label.alpha,
panel.label.bg,
panel.label.bg.color,
panel.label.bg.alpha,
panel.label.frame,
panel.label.frame.color,
panel.label.frame.alpha,
panel.label.frame.lwd,
panel.label.frame.r,
```

```
  panel.label.height,
  panel.label.rot,
  qtm.scalebar,
  qtm.minimap,
  qtm.mouse_coordinates,
  earth_boundary,
  earth_boundary.color,
  earth_boundary.lwd,
  earth_datum,
  space,
  space.color,
  space_overlay,
  check_and_fix,
  basemap.show,
  basemap.server,
  basemap.alpha,
  basemap.zoom,
  tiles.show,
  tiles.server,
  tiles.alpha,
  tiles.zoom,
  attr.color,
  crs_extra,
  crs_global,
  crs_basemap,
  title = NULL,
  main.title = NULL,
  main.title.size = NULL,
  main.title.color = NULL,
  main.title.fontface = NULL,
  main.title.fontfamily = NULL,
  main.title.position = NULL,
  fontface = NULL,
  fontfamily = NULL,
  style,
  ...
)
```

Arguments

crs	Map crs (see tm_shape()). NA means the crs is specified in tm_shape(). The crs that is used by the transformation functions is defined in tm_shape().
facet.max	Maximum number of facets
free.scales	For backward compatibility: if this value is set, it will be used to impute the free arguments in the layer functions
raster.max_cells	Maximum number of raster grid cells. Can be mode specific c(plot = 3000, view = 1000, 1000) (the last value is the fall back default)

`raster.warp` Should rasters be warped or transformed in case a different projection (crs) is used? Warping creates a new regular raster in the target crs, whereas transforming creates a (usually non-regular) raster in the target crs. The former is lossy, but much faster and is therefore the default. When a different projection (crs) is used, a (usually) regular raster will be

`show.messages` Show messages?

`show.warnings` Show warnings?

`output.format` Output format

`output.size` Output size

`output.dpi` Output dpi

`animation.dpi` Output dpi for animations

`value.const` Default visual value constants e.g. the default fill color for `tm_shape(World) + tm_polygons()`. A list is required with per visual variable a value.

`value.na` Default visual values that are used to visualize NA data values. A list is required with per visual variable a value.

`value.null` Default visual values that are used to visualize null (out-of-scope) data values. A list is required with per visual variable a value.

`value.blank` Default visual values that correspond to blank. For color these are `"#00000000"` meaning transparent. A list is required with per visual variable a value.

`values.var` Default values when a data variable to mapped to a visual variable, e.g. a color palette. A list is required with per visual variable a value.

`values.range` Default range for values. See `values.range` of `tm_scale_categorical()`. A list is required with per visual variable a value.

`value.neutral` Default values for when a data variable to mapped to a visual variable, e.g. a color palette. A list is required with per visual variable a value.

`values.scale` Default scales (as in object sizes) for values. See `values.range` of `tm_scale_categorical()`. A list is required with per visual variable a value.

`scales.var` Default scale functions per visual variable and type of data variable. A list is required with per visual variable per data type.

`scale.misc.args` Default values of scale function-specific arguments. A list is required with per scale function and optional per visual variable.

`continuous.nclass_per_legend_break` The number of continuous legend breaks within one 'unit' (label). The default value is 50.

`continuous.nclasses` the number of classes of a continuous scale. Should be odd. The default value is 101.

`label.format` Format for the labels. These are the default values for `tm_label_format()`

`label.na` Default label for missing values.

`scale` Overall scale of the map

asp	Aspect ratio of each map. When asp is set to NA (default) the aspect ratio will be adjusted to the used shapes. When set to 0 the aspect ratio is adjusted to the size of the device divided by the number of columns and rows.
bg	Draw map background?
bg.color	Background color of the map.
outer.bg	Draw map background (outside the frame)?
outer.bg.color	Background color of map outside the frame.
frame	Draw map frame?
frame.color	The color of the frame.
frame.alpha	The alpha transparency of the frame.
frame.lwd	The line width of the frame. See graphics::par, option 'lwd'.
frame.r	The r (radius) of the frame.
frame.double_line	The double line of the frame. TRUE or FALSE.
outer.margins	The margins of the outer space (outside the frame. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
inner.margins	The margins of the inner space (inside the frame). A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
inner.margins.extra	The extra arguments of the margins of the inner space (inside the frame). A list of arguments.
meta.margins	The margins of the meta. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).
meta.auto_margins	The auto_margins of the meta.
between_margin	Margin between the map.
panel.margin	The margin of the panel.
xlab.show	The visibility of the xlab. TRUE or FALSE.
xlab.text	The text of the xlab.
xlab.size	The size of the xlab.
xlab.color	The color of the xlab.
xlab.rotation	The rotation of the xlab.
xlab.space	The space of the xlab. In terms of number of text line heights.
xlab.fontface	The font face of the xlab. See graphics::par, option 'font'.
xlab.fontfamily	The font family of the xlab. See graphics::par, option 'family'.
xlab.alpha	The alpha transparency of the xlab.
xlab.side	The side of the xlab.

`ylab.show` The visibility of the ylab. `TRUE` or `FALSE`.

`ylab.text` The text of the ylab.

`ylab.size` The size of the ylab.

`ylab.color` The color of the ylab.

`ylab.rotation` The rotation of the ylab.

`ylab.space` The space of the ylab. In terms of number of text line heights.

`ylab.fontface` The font face of the ylab. See `graphics::par`, option 'font'.

`ylab.fontfamily`
The font family of the ylab. See `graphics::par`, option 'family'.

`ylab.alpha` The alpha transparency of the ylab.

`ylab.side` The side of the ylab.

`panel.type` The type of the panel.

`panel.wrap.pos` The panel positions for wrapped facets created with `tm_facets_grid()`. One of `"left"`, `"right"`, `"top"` (default) or `"bottom"`.

`panel.xtab.pos` The panel positions for grid facets created with `tm_facets_grid()`. Vector of two, where the first determines the locations of row panels (`"left"` or `"right"`) and the second the location of column panels (`"top"` or '"bottom")

`unit` Unit of the coordinate

`color.sepia_intensity`
The sepia_intensity of the color.

`color.saturation`
The saturation of the color.

`color_vision_deficiency_sim`
Color vision deficiency simulation. Either `"protan"`, `"deutan"`, or `"tritan"`.

`text.fontface` The font face of the text. See `graphics::par`, option 'font'.

`text.fontfamily`
The font family of the text. See `graphics::par`, option 'family'.

`r` The r (radius) (overall).

`component.position`
The position of the component. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`component.offset`
The offset of the component.

`component.stack_margin`
The stack_margin of the component.

`component.autoscale`
The autoscale of the component.

`component.resize_as_group`
The resize_as_group of the component.

`component.frame_combine`
The frame_combine of the component.

component.stack
: The stack of the component.

legend.stack
: The stack of the legend.

chart.stack
: The stack of the chart.

component.equalize
: The equalize of the component.

component.frame
: The frame of the component.

component.frame.color
: The color of the frame of the component.

component.frame.alpha
: The alpha transparency of the frame of the component.

component.frame.lwd
: The line width of the frame of the component. See graphics::par, option 'lwd'.

component.frame.r
: The r (radius) of the frame of the component.

component.bg
: The bg of the component.

component.bg.color
: The color of the bg of the component.

component.bg.alpha
: The alpha transparency of the bg of the component.

legend.show
: The visibility of the legend. TRUE or FALSE.

legend.orientation
: The orientation of the legend.

legend.position
: The position of the legend. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

legend.width
: The width of the legend.

legend.height
: The height of the legend.

legend.reverse
: The reverse of the legend.

legend.na.show
: The visibility of the na of the legend. TRUE or FALSE.

legend.title.color
: The color of the title of the legend.

legend.title.size
: The size of the title of the legend.

legend.title.fontface
: The font face of the title of the legend. See graphics::par, option 'font'.

legend.title.fontfamily
: The font family of the title of the legend. See graphics::par, option 'family'.

legend.title.alpha
: The alpha transparency of the title of the legend.

`legend.xlab.color`
: The color of the xlab of the legend.

`legend.xlab.size`
: The size of the xlab of the legend.

`legend.xlab.rot`
: The rot of the xlab of the legend.

`legend.xlab.fontface`
: The font face of the xlab of the legend. See `graphics::par`, option 'font'.

`legend.xlab.fontfamily`
: The font family of the xlab of the legend. See `graphics::par`, option 'family'.

`legend.xlab.alpha`
: The alpha transparency of the xlab of the legend.

`legend.ylab.color`
: The color of the ylab of the legend.

`legend.ylab.size`
: The size of the ylab of the legend.

`legend.ylab.rot`
: The rot of the ylab of the legend.

`legend.ylab.fontface`
: The font face of the ylab of the legend. See `graphics::par`, option 'font'.

`legend.ylab.fontfamily`
: The font family of the ylab of the legend. See `graphics::par`, option 'family'.

`legend.ylab.alpha`
: The alpha transparency of the ylab of the legend.

`legend.text.color`
: The color of the text of the legend.

`legend.text.size`
: The size of the text of the legend.

`legend.text.fontface`
: The font face of the text of the legend. See `graphics::par`, option 'font'.

`legend.text.fontfamily`
: The font family of the text of the legend. See `graphics::par`, option 'family'.

`legend.text.alpha`
: The alpha transparency of the text of the legend.

`legend.frame` The frame of the legend.

`legend.frame.color`
: The color of the frame of the legend.

`legend.frame.alpha`
: The alpha transparency of the frame of the legend.

`legend.frame.lwd`
: The line width of the frame of the legend. See `graphics::par`, option 'lwd'.

`legend.frame.r` The r (radius) of the frame of the legend.

`legend.bg` The bg of the legend.

`legend.bg.color`
: The color of the bg of the legend.

`legend.bg.alpha`
: The alpha transparency of the bg of the legend.

`legend.only`
: Should only legends be printed (so without map)?

`legend.absolute_fontsize`
: The absolute fontsize of the legend. So far, only used to calculate legend dimensions

`legend.settings.portrait`
: The portrait of the settings of the legend.

`legend.settings.landscape`
: The landscape of the settings of the legend.

`add_legend.position`
: The position of the add_legend. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`chart.show`
: The visibility of the chart. `TRUE` or `FALSE`.

`chart.plot.axis.x`
: The x of the axis of the plot of the chart.

`chart.plot.axis.y`
: The y of the axis of the plot of the chart.

`chart.position`
: The position of the chart. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`chart.width`
: The width of the chart.

`chart.height`
: The height of the chart.

`chart.reverse`
: The reverse of the chart.

`chart.na.show`
: The visibility of the na of the chart. `TRUE` or `FALSE`.

`chart.title.color`
: The color of the title of the chart.

`chart.title.size`
: The size of the title of the chart.

`chart.title.fontface`
: The font face of the title of the chart. See `graphics::par`, option 'font'.

`chart.title.fontfamily`
: The font family of the title of the chart. See `graphics::par`, option 'family'.

`chart.title.alpha`
: The alpha transparency of the title of the chart.

`chart.xlab.color`
: The color of the xlab of the chart.

`chart.xlab.size`
: The size of the xlab of the chart.

`chart.xlab.fontface`
: The font face of the xlab of the chart. See `graphics::par`, option 'font'.

chart.xlab.fontfamily
: The font family of the xlab of the chart. See graphics::par, option 'family'.

chart.xlab.alpha
: The alpha transparency of the xlab of the chart.

chart.ylab.color
: The color of the ylab of the chart.

chart.ylab.size
: The size of the ylab of the chart.

chart.ylab.fontface
: The font face of the ylab of the chart. See graphics::par, option 'font'.

chart.ylab.fontfamily
: The font family of the ylab of the chart. See graphics::par, option 'family'.

chart.ylab.alpha
: The alpha transparency of the ylab of the chart.

chart.text.color
: The color of the text of the chart.

chart.text.size
: The size of the text of the chart.

chart.text.fontface
: The font face of the text of the chart. See graphics::par, option 'font'.

chart.text.fontfamily
: The font family of the text of the chart. See graphics::par, option 'family'.

chart.text.alpha
: The alpha transparency of the text of the chart.

chart.frame
: The frame of the chart.

chart.frame.color
: The color of the frame of the chart.

chart.frame.alpha
: The alpha transparency of the frame of the chart.

chart.frame.lwd
: The line width of the frame of the chart. See graphics::par, option 'lwd'.

chart.frame.r
: The r (radius) of the frame of the chart.

chart.bg
: The bg of the chart.

chart.bg.color
: The color of the bg of the chart.

chart.bg.alpha
: The alpha transparency of the bg of the chart.

chart.object.color
: The color of the object of the chart.

title.size
: The size of the title.

title.color
: The color of the title.

title.fontface
: The font face of the title. See graphics::par, option 'font'.

title.fontfamily
: The font family of the title. See graphics::par, option 'family'.

title.alpha
: The alpha transparency of the title.

`title.padding` The padding of the title. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`title.frame` The frame of the title.

`title.frame.color`
The color of the frame of the title.

`title.frame.alpha`
The alpha transparency of the frame of the title.

`title.frame.lwd`
The line width of the frame of the title. See `graphics::par`, option 'lwd'.

`title.frame.r` The r (radius) of the frame of the title.

`title.position` The position of the title. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`title.width` The width of the title.

`credits.size` The size of the credits.

`credits.color` The color of the credits.

`credits.fontface`
The font face of the credits. See `graphics::par`, option 'font'.

`credits.fontfamily`
The font family of the credits. See `graphics::par`, option 'family'.

`credits.alpha` The alpha transparency of the credits.

`credits.padding`
The padding of the credits. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`credits.position`
The position of the credits. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`credits.width` The width of the credits.

`credits.height` The height of the credits.

`compass.north` The north of the compass.

`compass.type` The type of the compass.

`compass.text.size`
The size of the text of the compass.

`compass.size` The size of the compass.

`compass.show.labels`
The labels of the show of the compass.

`compass.cardinal.directions`
The directions of the cardinal of the compass.

`compass.text.color`
The color of the text of the compass.

`compass.color.dark`
The dark of the color of the compass.

`compass.color.light`
: The light of the color of the compass.

`compass.lwd`
: The line width of the compass. See `graphics::par`, option 'lwd'.

`compass.margins`
: The margins of the compass. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`compass.position`
: The position of the compass. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset.position`
: The position of the inset. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`logo.height`
: The height of the logo.

`logo.margins`
: The margins of the logo. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`logo.between_margin`
: The between_margin of the logo.

`logo.position`
: The position of the logo. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset_map.height`
: The height of the inset_map.

`inset_map.width`
: The width of the inset_map.

`inset_map.margins`
: The margins of the inset_map. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inset_map.between_margin`
: The between_margin of the inset_map.

`inset_map.position`
: The position of the inset_map. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`inset_map.frame`
: The frame of the inset_map.

`inset.height`
: The height of the inset.

`inset.width`
: The width of the inset.

`inset.margins`
: The margins of the inset. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inset.between_margin`
: The between_margin of the inset.

`inset.frame`
: The frame of the inset.

`inset.bg`
: The bg of the inset.

`inset.bg.color`
: The color of the bg of the inset.

`inset.bg.alpha` The alpha transparency of the bg of the inset.

`inset_grob.height`
: The height of the inset_grob.

`inset_grob.width`
: The width of the inset_grob.

`inset_gg.height`
: The height of the inset_gg.

`inset_gg.width` The width of the inset_gg.

`scalebar.breaks`
: See `tm_scalebar()`

`scalebar.width` See `tm_scalebar()`

`scalebar.allow_clipping`
: See `tm_scalebar()`

`scalebar.text.size`
: The size of the text of the scalebar.

`scalebar.text.color`
: The color of the text of the scalebar.

`scalebar.text.fontface`
: The font face of the text of the scalebar. See `graphics::par`, option 'font'.

`scalebar.text.fontfamily`
: The font family of the text of the scalebar. See `graphics::par`, option 'family'.

`scalebar.color.dark`
: The dark of the color of the scalebar.

`scalebar.color.light`
: The light of the color of the scalebar.

`scalebar.lwd` The line width of the scalebar. See `graphics::par`, option 'lwd'.

`scalebar.size` The size of the scalebar.

`scalebar.margins`
: The margins of the scalebar. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`scalebar.position`
: The position of the scalebar. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`grid.show` The visibility of the grid. `TRUE` or `FALSE`.

`grid.labels.pos`
: The pos of the labels of the grid.

`grid.x` The x of the grid.

`grid.y` The y of the grid.

`grid.n.x` The x of the n of the grid.

`grid.n.y` The y of the n of the grid.

`grid.crs` The coordinate reference system (CRS) of the grid.

`grid.col` The color of the grid.

`grid.lwd` The line width of the grid. See `graphics::par`, option 'lwd'.

`grid.alpha` The alpha transparency of the grid.

`grid.labels.show`
The visibility of the labels of the grid. `TRUE` or `FALSE`.

`grid.labels.size`
The size of the labels of the grid.

`grid.labels.col`
The color of the labels of the grid.

`grid.labels.fontface`
The font face of the labels of the grid. See `graphics::par`, option 'font'.

`grid.labels.fontfamily`
The font family of the labels of the grid. See `graphics::par`, option 'family'.

`grid.labels.rot`
The rot of the labels of the grid.

`grid.labels.format`
The format of the labels of the grid.

`grid.labels.cardinal`
The cardinal of the labels of the grid.

`grid.labels.margin.x`
The x of the margin of the labels of the grid.

`grid.labels.margin.y`
The y of the margin of the labels of the grid.

`grid.labels.space.x`
The x of the space of the labels of the grid.

`grid.labels.space.y`
The y of the space of the labels of the grid.

`grid.labels.inside_frame`
The inside_frame of the labels of the grid.

`grid.ticks` The ticks of the grid.

`grid.lines` The lines of the grid.

`grid.ndiscr` The ndiscr of the grid.

`mouse_coordinates.position`
The position of the mouse_coordinates. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`minimap.server` The server of the minimap.

`minimap.toggle` The toggle of the minimap.

`minimap.position`
The position of the minimap. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`panel.show` The visibility of the panel. `TRUE` or `FALSE`.

`panel.labels` The labels of the panel.

`panel.label.size`
: The size of the label of the panel.

`panel.label.color`
: The color of the label of the panel.

`panel.label.fontface`
: The font face of the label of the panel. See `graphics::par`, option 'font'.

`panel.label.fontfamily`
: The font family of the label of the panel. See `graphics::par`, option 'family'.

`panel.label.alpha`
: The alpha transparency of the label of the panel.

`panel.label.bg` The bg of the label of the panel.

`panel.label.bg.color`
: The color of the bg of the label of the panel.

`panel.label.bg.alpha`
: The alpha transparency of the bg of the label of the panel.

`panel.label.frame`
: The frame of the label of the panel.

`panel.label.frame.color`
: The color of the frame of the label of the panel.

`panel.label.frame.alpha`
: The alpha transparency of the frame of the label of the panel.

`panel.label.frame.lwd`
: The line width of the frame of the label of the panel. See `graphics::par`, option 'lwd'.

`panel.label.frame.r`
: The r (radius) of the frame of the label of the panel.

`panel.label.height`
: The height of the label of the panel.

`panel.label.rot`
: Rotation angles of the panel labels. Vector of four values that determine the panel label rotation when they are placed left, top, right, and bottom. The default angles are 90, 0, 270 and 0 respectively. Note that the second value is the most common, since labels are by default shown on top (see `panel.wrap.pos`). In cross-table facets created with `tm_facets_grid()`, the first two values are used by default (see `panel.xtab.pos`).

`qtm.scalebar` The scalebar of the qtm.

`qtm.minimap` The minimap of the qtm.

`qtm.mouse_coordinates`
: The mouse_coordinates of the qtm.

`earth_boundary` The earth boundary

`earth_boundary.color`
: The color of the earth_boundary.

`earth_boundary.lwd`
: The line width of the earth_boundary. See `graphics::par`, option 'lwd'.

earth_datum	Earth datum
space	Should the space be drawn? Only applicable is earth_boundary is enabled.
space.color	The color of the space.
space_overlay	Should the space be drawn as overlay (to make sure spatial features or rasters do not exceed the earth boundary), or as background? By default TRUE when a raster is warped.
check_and_fix	Should attempt to fix an invalid shapefile
basemap.show	The visibility of the basemap. TRUE or FALSE.
basemap.server	The server of the basemap.
basemap.alpha	The alpha transparency of the basemap.
basemap.zoom	The zoom of the basemap.
tiles.show	The visibility of the tiles. TRUE or FALSE.
tiles.server	The server of the tiles.
tiles.alpha	The alpha transparency of the tiles.
tiles.zoom	The zoom of the tiles.
attr.color	The color of the attr.
crs_extra	Only used internally (work in progress)
crs_global	The used crs for world maps
crs_basemap	The crs_basemap (overall).
title	deprecated See tm_title()
main.title	deprecated See tm_title()
main.title.size, main.title.color, main.title.fontface, main.title.fontfamily, main.title.position	deprecated. Use the title. options instead.
fontface, fontfamily	renamed to text.fontface and text.fontfamily
style	style see tm_style()
...	List of tmap options to be set, or option names (characters) to be returned (see details)

See Also

Vignette about layout, vignette about margins and aspect ratio and vignette about options

tm_place_legends_right *tmap layout: helper functions*

Description

tmap layout: helper functions

Usage

```
tm_place_legends_right(width = NA)

tm_place_legends_left(width = NA)

tm_place_legends_bottom(height = NA)

tm_place_legends_top(height = NA)

tm_place_legends_inside(pos.h = NULL, pos.v = NULL)

tm_extra_inner_margin(left = 0, right = 0, top = 0, bottom = 0)
```

Arguments

width	width
height	height
pos.h, pos.v	position (horizontal and vertical)
left, right, top, bottom	extra margins

tm_plot *Plot mode options*

Description

Plot mode options. This option is specific to the plot mode.

Usage

```
tm_plot(use_gradient, limit_latitude_3857)
```

Arguments

use_gradient	Use gradient fill using linearGradient()
limit_latitude_3857	Vector of two limit latitude values for maps printed in Web Mercator projection (EPSG 3857). If `c(-90, 90)` the poles will be inflated too much. The Web Mercator is defines as `c(-85.06, 85.06)`, but the default setting in tmap is `c(-84, 84)`.

---

tm_plot_order *Determine plotting order of features*

Description

Determine plotting order of features.

Usage

```
tm_plot_order(
  aes,
  reverse = TRUE,
  na.order = c("mix", "bottom", "top"),
  null.order = c("bottom", "mix", "top"),
  null.below.na = TRUE
)
```

Arguments

aes	Visual variable for which the values determine the plotting order. Example: bubble map where the `"size"` aesthetic is used. A data variable (say population) is mapped via a continuous scale (`tm_scale_continuous()`) to bubble sizes. The bubbles are plotted in order of size. How is determined by the other arguments. Use `"DATA"` to keep the same order as in the data. Another special value are `"AREA"` and `"LENGTH"` which are preserved for polygons and lines respectively: rather than a data variable the polygon area / line lengths determines the plotting order.
reverse	Logical that determines whether the visual values are plotted in reversed order. The visual values (specified with tmap option `"values.var"`) are by default reversed, so plotted starting from the last value. In the bubble map example, this means that large bubbles are plotted first, hence at the bottom.
na.order	Where should features be plotted that have an `NA` value for (at least) one other aesthetic variable? In the (order) `"mix"`, at the `"bottom"`, or on `"top"`? In the bubble map example: if fill color is missing for some bubble, where should those bubbles be plotted?
null.order	Where should non-selected (aka null) features be plotted?
null.below.na	Should null features be plotted below NA features?

tm_polygons *Map layer: polygons*

Description

Map layer that draws polygons. Supported visual variables are: `fill` (the fill color), `col` (the border color), `lwd` (line width), `lty` (line type), `fill_alpha` (fill color alpha transparency) and `col_alpha` (border color alpha transparency).

Usage

```
tm_polygons(
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.chart = tm_chart_none(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.chart = tm_chart_none(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.chart = tm_chart_none(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.chart = tm_chart_none(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  linejoin = "round",
  lineend = "round",
  plot.order = tm_plot_order("lwd", reverse = TRUE, na.order = "bottom"),
  zindex = NA,
  group = NA,
```

```
  group.control = "check",
  popup = tm_popup(),
  popup.vars = NA,
  popup.format = tm_label_format(),
  hover = NA,
  id = "",
  blend = "over",
  options = opt_tm_polygons(),
  ...
)

tm_fill(...)

tm_borders(col = tm_const(), ...)

opt_tm_polygons(polygons.only = "ifany")
```

Arguments

`fill`, `fill.scale`, `fill.legend`, `fill.chart`, `fill.free`
: Visual variable that determines the fill color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`lwd`, `lwd.scale`, `lwd.legend`, `lwd.chart`, `lwd.free`
: Visual variable that determines the line width. See details. *Unit:* Base R line-width units; 1 lwd is approx. 0.75 pt at 96 dpi. Controlled by `values.scale`.

`lty`, `lty.scale`, `lty.legend`, `lty.chart`, `lty.free`
: Visual variable that determines the line type. See details. *Unit:* Integer (1-6) or name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

`fill_alpha`, `fill_alpha.scale`, `fill_alpha.chart`, `fill_alpha.legend`, `fill_alpha.free`
: Visual variable that determines the fill color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`linejoin`, `lineend`
: Line join and line end. See gpar() for details.

`plot.order`
: Specification in which order the spatial features are drawn. See `tm_plot_order()` for details.

`zindex`
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

`group`
: Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

group.control	In view mode, the group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).
popup	popup specification for "view" mode, the output of tm_popup(). It determines the data variables shown in the popup table, the popup title, and (in the future) the popup layout. This replaces the deprecated arguments popup.vars and popup.format.
popup.vars	(Deprecated.) Use popup with tm_popup() instead (via its vars argument). Names of data variables that are shown in the popups in "view" mode. Set popup.vars to TRUE to show all variables in the shape object. Set popup.vars to FALSE to disable popups. Set popup.vars to a character vector of variable names to show those variables in the popups. The default (NA) depends on whether visual variables (e.g. fill) are used. If so, only those are shown. If not, all variables in the shape object are shown.
popup.format	(Deprecated.) Use popup with tm_popup() instead (via its format argument). List of formatting options for the popup values. Output of tm_label_format(). Only applicable for numeric data variables. If one list of formatting options is provided, it is applied to all numeric variables of popup.vars. Also, a (named) list of lists can be provided. In that case, each list of formatting options is applied to the named variable.
hover	name of the data variable that specifies the hover labels (view mode only). Set to FALSE to disable hover labels. By default FALSE, unless id is specified. In that case, it is set to id,
id	name of the data variable that specifies the indices of the spatial features. Only used for "view" mode.
blend	Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section for the supported values.
options	options passed on to the corresponding opt_<layer_function> function
...	to catch deprecated arguments from version < 4.0
polygons.only	should only polygon geometries of the shape object (defined in tm_shape()) be plotted? By default "ifany", which means TRUE in case a geometry collection is specified.

Details

The visual variable arguments (e.g. col) can be specified with a data variable name (e.g., a spatial vector attribute or a raster layer of the object specified in tm_shape()), with a visual value (for col, a color is expected), or with a geometry-derived variable (see below). See vignette about visual variables.

Multiple values can be specified: in that case facets are created. These facets can be combined with other faceting data variables, specified with tm_facets(). See vignette about facets.

- The *.scale arguments determine the used scale to map the data values to visual variable values. These can be specified with one of the available tm_scale_*() functions. The default is specified by the tmap option (tm_options()) scales.var. See vignette about scales.

- The `*.legend` arguments determine the used legend, specified with `tm_legend()`. The default legend and its settings are determined by the tmap options (`tm_options()`) `legend.` . See vignette about legends.
- The `*.chart` arguments specify additional charts, specified with `tm_chart_`, e.g. `tm_chart_histogram()`. See vignette about charts.
- The `*.free` arguments determine whether scales are applied freely across facets, or shared. A logical value is required. They can also be specified with a vector of three logical values; these determine whether scales are applied freely per facet dimension. This is only useful when facets are applied (see `tm_facets()`). There are maximally three facet dimensions: rows, columns, and pages. This only applies for a facet grid (`tm_facets_grid()`). For instance, `col.free = c(TRUE, FALSE, FALSE)` means that for the visual variable `col`, each row of facets will have its own scale, and therefore its own legend. For facet wraps and stacks (`tm_facets_wrap()` and `tm_facets_stack()`) there is only one facet dimension, so the `*.free` argument requires only one logical value.

Currently, three geometry-derived variables are implemented:

- `"AREA"` (polygons only), which uses the feature area;
- `"LENGTH"` (lines only), which uses the feature length; and
- `"MAP_COLORS"`, which assigns values so that adjacent features receive different values, making it particularly suitable for coloring neighbouring polygons.

Note that geometry-derived variables do not generate a legend automatically. If a legend is required, compute the corresponding variable explicitly, for example with `sf::st_area()`, `sf::st_length()`, or `tmaptools::map_coloring()`, and use the resulting values instead.

Visual variable units:

Every visual variable maps data values to a specific output unit. Knowing the unit matters when supplying constant values via `tm_const()`, or output ranges via `values.range` / `values.scale` in the scale functions.

Variable	Output unit	Notes
`fill`, `col`, `bgcol`	color	name, hex, or palette string
`fill_alpha`, `col_alpha`, `bgcol_alpha`	proportion 0-1	0 = transparent, 1 = opaque
`size` (symbols, bubbles, squares, dots)	typographic lines	1 line approx. 1/6 inch; scaled by `values.scale`
`size` (circles)	meters	plain numeric or a `units` object
`size` (text, labels)	multiplier	1 = 12 pt (plot) / 12 px (view)
`lwd`	lwd	base R units; 1 lwd approx. 0.75 pt at 96 dpi
`lty`	–	integer 1-6 or name ("solid", "dashed", ...)
`shape`	–	integer `pch` 1-25 or single character
`angle`	degrees	0-360, clockwise from north
`fontface`	–	"plain", "bold", "italic", "bold.italic"

Symbol size (`size` *in* `tm_symbols`, `tm_bubbles`, `tm_squares`, `tm_dots`*):*
"Lines" is a typographic unit: one line is approximately 1/6 inch (the default base line-height in R graphics). The global multiplier `tmap_options(values.scale = list(size.bubbles = 1.5))` scales all symbol sizes without changing the data mapping.

Circle size (`size` *in* `tm_circles`*):*
The value is a geographic radius in meters. A plain numeric vector is interpreted as meters; a `units` object (from the **units** package) is automatically converted, so `units::as_units(1, "mi")` gives a 1-mile radius. Because the radius is geographic, circles scale with zoom in interactive (view) mode – unlike bubble symbols which keep a fixed screen size.

Text size (`size` *in* `tm_text`, `tm_labels`*):*
The value is a multiplier of the base font size. `size = 1` renders at 12 pt in plot mode (R's default `par("ps")`) and at 12 px in view mode (`gp$cex * 12` px, see `tmapLeafletDataPlot.tm_data_text`); the two modes are consistent by design.

Layer blending (`blend`)**:**

Blend modes control how a layer's pixels are combined with the pixels beneath it. For each pixel, let S be the source (top layer) RGB value and D be the destination (bottom layer) RGB value, both normalised to $[0, 1]$.

`blend`	Formula	Use case
`"over"`	$S \cdot \alpha + D \cdot (1 - \alpha)$	Standard alpha compositing (default)
`"multiply"`	$S \times D$	Hillshading over colour raster; both layers darken each other
`"screen"`	$1 - (1 - S)(1 - D)$	Inverse of multiply; brightens
`"overlay"`	multiply if $D < 0.5$, screen if $D \geq 0.5$	Boosts contrast; preserves midtones
`"darken"`	$\min(S, D)$	Keeps the darker of the two layers per channel
`"lighten"`	$\max(S, D)$	Keeps the lighter of the two layers per channel

Requires R >= 4.2 and a compatible graphics device (e.g. `png(type = "cairo")`, `svg()`). In view mode, blending is applied via CSS `mix-blend-mode`. See `grid::groupGrob()` for the full list of supported operators.

zindex and pane names:

In view mode, each layer is rendered in a Leaflet pane named `"tmap{zindex}"` (e.g., `"tmap401"`, `"tmap402"`), with base tile layers placed in the standard `"tile"` pane.

See Also

Choropleth example (1) and choropleth example (2)

Examples

```
# load Africa country data
data(World)
Africa = World[World$continent == "Africa", ]
Africa_border = sf::st_make_valid(sf::st_union(sf::st_buffer(Africa, 0.001))) # slow and ugly

# without specifications
tm_shape(Africa_border) + tm_polygons()
tm_shape(Africa_border) + tm_fill()
tm_shape(Africa_border) + tm_borders()

# specification with visual variable values
tm_shape(Africa) +
```

```
  tm_polygons(fill = "limegreen", col = "purple", lwd = 2, lty = "solid", col_alpha = 0.3) +
  tm_text("name", options = opt_tm_text(remove_overlap = TRUE)) +
tm_shape(Africa_border) +
  tm_borders("darkred", lwd = 3)

# specification with a data variable
tm_shape(Africa) +
 tm_polygons(fill = "income_grp", fill.scale = tm_scale_categorical(values = "-tol.muted"))


# continuous color scale with landscape legend
tm_shape(Africa) +
  tm_polygons(fill = "inequality",
    fill.scale = tm_scale_continuous(values = "-scico.roma"),
    fill.legend = tm_legend(
     title = "", orientation = "landscape",
     position = tm_pos_out("center", "bottom"), frame = FALSE
     )) +
tm_shape(Africa_border) +
 tm_borders(lwd = 2) +
tm_title("Inequality index", position = tm_pos_in("right", "TOP"), frame = FALSE) +
tm_layout(frame = FALSE)

# bivariate scale
tm_shape(World) +
 tm_polygons(tm_vars(c("inequality", "well_being"), multivariate = TRUE))
```

---

tm_popup *Popup specification for interactive maps*

Description

tm_popup() specifies the popups that are shown in interactive ("view") mode when a feature is clicked. It is passed to the popup argument of the layer functions (tm_polygons(), tm_symbols(), tm_lines(), etc.). It replaces the (now deprecated) layer arguments popup.vars and popup.format.

Usage

```
tm_popup(
  vars = NA,
  title = NA,
  format = tm_label_format(),
  width = "auto",
  max.height = "25em",
  label.color = "#888888",
  value.align = "right"
)
```

Arguments

vars	Names of the data variables that are shown in the popup table. A (possibly named) character vector; when named, the names are used as labels in the popup table. Besides a character vector, the following special values are supported (identical to the former popup.vars argument): TRUE show all variables of the shape object; FALSE disable popups; NA **(default)** automatic: if visual variables (e.g. fill) are used, only those are shown, otherwise all variables of the shape object are shown.
title	Name of the data variable used as the popup title (the bold header shown above the popup table). This overrules the layer argument id, analogous to how hover overrules id for hover labels. The default (NA) means that the popup title is derived from id (the former, and still default, behaviour). A length-one character vector is expected; a named value is allowed and reserved for future use.
format	A list of formatting options for the popup values, the output of tm_label_format(). Only applicable to numeric data variables. If one list of formatting options is provided, it is applied to all numeric variables of vars. A (named) list of lists can also be provided; in that case, each list of formatting options is applied to the named variable.
width	Width of the popup content (view mode). A bare number is interpreted as pixels (e.g. 300 means "300px"); a character string is used as-is, so any CSS length is accepted ("300px", "20em", "50%"). The default "auto" lets the popup size to its content.
max.height	Maximum height of the popup table before it becomes vertically scrollable (view mode). A bare number is interpreted as em (e.g. 5 means "5em", roughly "show 5 lines"); a character string is used as-is. Default "25em". Use max.height = "none" (or NA/Inf) to remove the cap, so the popup grows to fit its content and never scrolls.
label.color	Color of the variable-name (label) column in the popup table. Default "#888888" (grey).
value.align	Horizontal alignment of the value column in the popup table, one of "right" (default), "left", or "center".

Value

A tm_popup object.

See Also

tm_polygons(), tm_symbols(), tm_lines()

tm_pos *Set the position of map components*

Description

Set the position of map components, such as legends, title, compass, scale bar, etc. `tm_pos()` is the function to position these components: `tm_pos_out()` places the components outside the map area, `tm_pos_in()` inside the map area, and `tm_pos_on_top()` on top of the map Each `position` argument of a map layer or component should be specified with one of these functions. The functions `tm_pos_auto_out()` and `tm_pos_auto_in()` are used to set the components automatically, and should be used via `tmap_options()`. See Details how the positioning works.

Usage

```
tm_pos(cell.h, cell.v, pos.h, pos.v, align.h, align.v, just.h, just.v)

tm_pos_in(pos.h, pos.v, align.h, align.v, just.h, just.v)

tm_pos_out(cell.h, cell.v, pos.h, pos.v, align.h, align.v, just.h, just.v)

tm_pos_on_top(pos.h, pos.v, align.h, align.v, just.h, just.v)

tm_pos_auto_out(cell.h, cell.v, pos.h, pos.v, align.h, align.v, just.h, just.v)

tm_pos_auto_in(align.h, align.v, just.h, just.v)
```

Arguments

`cell.h`, `cell.v`	The plotting area is overlaid with a 3x3 grid, of which the middle grid cell is the map area. Components can be drawn into any cell. `cell.h` specifies the horizontal position (column) can take values `"left"`, `"center"`, and `"right"`. `cell.v` specifies the vertical position (row) and can take values `"top"`, `"center"`, and `"bottom"`. See details for a graphical explanation.
`pos.h`, `pos.v`	The position of the component within the cell. The main options for `pos.h` are `"left"`, `"center"`, and `"right"`. For `cell.v` these are `"top"`, `"center"`, and `"bottom"`. These options can also be provided in upper case; in that case there is no offset (see the tmap option `component.offset`). Also numbers between 0 and 1 can be provided, which determine the position of the component inside the cell (with (0,0) being left bottom). The arguments `just.h` and `just.v` determine the justification point.
`align.h`, `align.v`	The alignment of the component in case multiple components are stacked. When they are stacked horizontally, `align.v` determines how components that are smaller in height than the available height (determined by the outer.margins if specified and otherwise by the highest component) are justified: `"top"`, `"center"`, or `"bottom"`. Similarly, `align.h` determines how components are justified horizontally when they are stacked vertically: `"left"`, `"center"`, or `"right"`.

`just.h`, `just.v` The justification of the components. Only used in case `pos.h` and `pos.v` are numbers.

Details

`tm_pos_in()` sets the position of the component(s) inside the maps area, which is equivalent to the center-center cell (in case there are facets, these are all drawn in this center-center cell).

`tm_pos_out()` sets the position of the component(s) outside the map.

`tm_pos_on_top()` is the same as `tm_pos_out`, but with the cell set to the center cell It may be therefore seem similar to `tm_pos_in()`, but with an essential difference: `tm_pos_in()` takes the map frame into account whereas `tm_pos_on_top()` does not. #' The amount of space that the top and bottom rows, and left and right columns occupy is determined by the `tm_layout()` arguments `meta.margins` and `meta.auto_margins`. The former sets the relative space of the bottom, left, top, and right side. In case these are set to `NA`, the space is set automatically based on 1) the maximum relative space specified by `meta.auto_margins` and 2) the presence and size of components in each cell. For instance, if there is one landscape oriented legend in the center-bottom cell, then the relative space of the bottom row is set to the height of that legend (given that it is smaller than the corresponding value of `meta.auto_margins`), while the other four sides are set to 0.

`tm_pos_auto_out()` is more complex: the `cell.h` and `cell.v` arguments should be set to one of the four corners. It does not mean that the components are drawn in a corner. The corner represents the sides of the map that the components are drawn. By default, legends are drawn either at the bottom or on the right-side of the map by default (see `tmap_options("legend.position")`). Only when there are row- and column-wise legends and a general legend (using `tm_facets_grid()`), the general legend is drawn in the corner, but in practice this case will be rare.

The arguments `pos.h` and `pos.v` determine where the components are drawn within the cell. Again, with `"left"`, `"center"`, and `"right"` for `pos.h` and `"top"`, `"center"`, and `"bottom"` for `pos.v`. The values can also be specified in upper-case, which influences the offset with the cell borders, which is determined by tmap option `component.offset`. By default, there is a small offset when components are drawn inside and no offset when they are drawn outside or with upper-case.

`tm_pos_auto_in()` automatically determines `pos.h` and `pos.v` given the available space inside the map. This is similar to the default positioning in tmap3.

In case multiple components are draw in the same cell and the same position inside that cell, they are stacked (determined which the `stack` argument in the legend or component function). The `align.h` and `align.v` arguments determine how these components will be justified with each other.

Note that legends and components may be different for a facet row or column. This is the case when `tm_facets_grid()` or `tm_facets_stack()` are applied and when scales are set to free (with the `.free` argument of the map layer functions). In case a legends or components are draw row- or column wise, and the position of the legends (or components) is right next to the maps, these legends (or components) will be aligned with the maps.

See Also

Vignette about positioning

`tm_raster` *Map layer: raster*

Description

Map layer that draws rasters. Supported visual variable is: `col` (the color).

Usage

```
tm_raster(
  col = tm_vars(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  zindex = NA,
  group = NA,
  group.control = "check",
  blend = "over",
  options = opt_tm_raster(),
  ...
)

opt_tm_raster(interpolate = FALSE)
```

Arguments

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`	Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.
`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`	Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).
`zindex`	Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.
`group`	Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)
`group.control`	In view mode, the group control determines how layer groups can be switched on and off. Options: `"radio"` for radio buttons (meaning only one group can be shown), `"check"` for check boxes (so multiple groups can be shown), and `"none"` for no control (the group cannot be (de)selected).

blend	Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section for the supported values.
options	options passed on to the corresponding opt_<layer_function> function
...	to catch deprecated arguments from version < 4.0
interpolate	Should the raster image be interpolated? Currently only applicable in view mode (passed on to grid)

Details

The visual variable arguments (e.g. col) can be specified with a data variable name (e.g., a spatial vector attribute or a raster layer of the object specified in tm_shape()), with a visual value (for col, a color is expected), or with a geometry-derived variable (see below). See vignette about visual variables.

Multiple values can be specified: in that case facets are created. These facets can be combined with other faceting data variables, specified with tm_facets(). See vignette about facets.

- The *.scale arguments determine the used scale to map the data values to visual variable values. These can be specified with one of the available tm_scale_*() functions. The default is specified by the tmap option (tm_options()) scales.var. See vignette about scales.
- The *.legend arguments determine the used legend, specified with tm_legend(). The default legend and its settings are determined by the tmap options (tm_options()) legend. . See vignette about legends.
- The *.chart arguments specify additional charts, specified with tm_chart_, e.g. tm_chart_histogram(). See vignette about charts.
- The *.free arguments determine whether scales are applied freely across facets, or shared. A logical value is required. They can also be specified with a vector of three logical values; these determine whether scales are applied freely per facet dimension. This is only useful when facets are applied (see tm_facets()). There are maximally three facet dimensions: rows, columns, and pages. This only applies for a facet grid (tm_facets_grid()). For instance, col.free = c(TRUE, FALSE, FALSE) means that for the visual variable col, each row of facets will have its own scale, and therefore its own legend. For facet wraps and stacks (tm_facets_wrap() and tm_facets_stack()) there is only one facet dimension, so the *.free argument requires only one logical value.

Currently, three geometry-derived variables are implemented:

- "AREA" (polygons only), which uses the feature area;
- "LENGTH" (lines only), which uses the feature length; and
- "MAP_COLORS", which assigns values so that adjacent features receive different values, making it particularly suitable for coloring neighbouring polygons.

Note that geometry-derived variables do not generate a legend automatically. If a legend is required, compute the corresponding variable explicitly, for example with sf::st_area(), sf::st_length(), or tmaptools::map_coloring(), and use the resulting values instead.

Visual variable units:

Every visual variable maps data values to a specific output unit. Knowing the unit matters when supplying constant values via `tm_const()`, or output ranges via `values.range` / `values.scale` in the scale functions.

Variable	Output unit	Notes
`fill`, `col`, `bgcol`	color	name, hex, or palette string
`fill_alpha`, `col_alpha`, `bgcol_alpha`	proportion 0-1	0 = transparent, 1 = opaque
`size` (symbols, bubbles, squares, dots)	typographic lines	1 line approx. 1/6 inch; scaled by `values.scale`
`size` (circles)	meters	plain numeric or a `units` object
`size` (text, labels)	multiplier	1 = 12 pt (plot) / 12 px (view)
`lwd`	lwd	base R units; 1 lwd approx. 0.75 pt at 96 dpi
`lty`	–	integer 1-6 or name ("solid", "dashed", ...)
`shape`	–	integer `pch` 1-25 or single character
`angle`	degrees	0-360, clockwise from north
`fontface`	–	"plain", "bold", "italic", "bold.italic"

Symbol size (`size` *in* `tm_symbols`, `tm_bubbles`, `tm_squares`, `tm_dots`*):*
"Lines" is a typographic unit: one line is approximately 1/6 inch (the default base line-height in R graphics). The global multiplier `tmap_options(values.scale = list(size.bubbles = 1.5))` scales all symbol sizes without changing the data mapping.

Circle size (`size` *in* `tm_circles`*):*
The value is a geographic radius in meters. A plain numeric vector is interpreted as meters; a `units` object (from the **units** package) is automatically converted, so `units::as_units(1, "mi")` gives a 1-mile radius. Because the radius is geographic, circles scale with zoom in interactive (view) mode – unlike bubble symbols which keep a fixed screen size.

Text size (`size` *in* `tm_text`, `tm_labels`*):*
The value is a multiplier of the base font size. `size = 1` renders at 12 pt in plot mode (R's default `par("ps")`) and at 12 px in view mode (`gp$cex * 12` px, see `tmapLeafletDataPlot.tm_data_text`); the two modes are consistent by design.

Layer blending (`blend`):

Blend modes control how a layer's pixels are combined with the pixels beneath it. For each pixel, let S be the source (top layer) RGB value and D be the destination (bottom layer) RGB value, both normalised to $[0, 1]$.

`blend`	Formula	Use case
`"over"`	$S \cdot \alpha + D \cdot (1 - \alpha)$	Standard alpha compositing (default)
`"multiply"`	$S \times D$	Hillshading over colour raster; both layers darken each other
`"screen"`	$1 - (1 - S)(1 - D)$	Inverse of multiply; brightens
`"overlay"`	multiply if $D < 0.5$, screen if $D \geq 0.5$	Boosts contrast; preserves midtones
`"darken"`	$\min(S, D)$	Keeps the darker of the two layers per channel
`"lighten"`	$\max(S, D)$	Keeps the lighter of the two layers per channel

Requires R >= 4.2 and a compatible graphics device (e.g. `png(type = "cairo")`, `svg()`). In view mode, blending is applied via CSS `mix-blend-mode`. See `grid::groupGrob()` for the full list of supported operators.

zindex and pane names:

In view mode, each layer is rendered in a Leaflet pane named "tmap{zindex}" (e.g., "tmap401", "tmap402"), with base tile layers placed in the standard "tile" pane.

Examples

```
## Not run:
# load land data
data(land, World)

tm_shape(land) +
 tm_raster("cover")

tm_shape(land) +
 tm_raster("elevation", col.scale = tm_scale_continuous(values = terrain.colors(9))) +
 tm_shape(World) +
 tm_borders()

## End(Not run)
```

---

tm_rgb *Map layer: rgb images*

Description

Map layer that an rgb image.. The used (multivariate) visual variable is col, which should be specified with 3 or 4 variables for tm_rgb() and tm_rgba() respectively. The first three correspond to the red, green, and blue channels. The optional fourth is the alpha transparency channel.

Usage

```
tm_rgb(
  col = tm_vars(n = 3, multivariate = TRUE),
  col.scale = tm_scale_rgb(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  blend = "over",
  options = opt_tm_rgb(),
  ...
)

tm_rgba(
```

```
  col = tm_vars(n = 4, multivariate = TRUE),
  col.scale = tm_scale_rgba(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  blend = "over",
  options = opt_tm_rgb()
)

opt_tm_rgb(interpolate = FALSE, saturation = 1)
```

Arguments

col, col.scale, col.legend, col.chart, col.free	Visual variable that determines the color. col is a multivariate variable, with 3 (tm_rgb) or 4 (tm_rgba) numeric data variables. These can be specified via tm_vars() with multivariate = TRUE
col_alpha, col_alpha.scale, col_alpha.legend, col_alpha.chart, col_alpha.free	Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).
blend	Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section of tm_polygons() for supported values.
options	options passed on to the corresponding opt_<layer_function> function
...	to catch deprecated arguments from version < 4.0
interpolate	Should the raster image be interpolated? Currently only applicable in view mode (passed on to grid)
saturation	The saturation of the rgb.

Examples

```
## Not run:
require(stars)
file = system.file("tif/L7_ETMs.tif", package = "stars")

L7 = stars::read_stars(file)

tm_shape(L7) +
 tm_rgb()

# the previous example was a shortcut of this call
tm_shape(L7) +
 tm_rgb(col = tm_vars("band", dimvalues = 1:3, multivariate = TRUE))

# alternative format: using a stars dimension instead of attributes
L7_alt = split(L7, "band")
tm_shape(L7_alt) +
 tm_rgb()
```

```
# with attribute names
tm_shape(L7_alt) +
 tm_rgb(col = tm_vars(c("X1", "X2", "X3"), multivariate = TRUE))

# with attribute indices
tm_shape(L7_alt) +
 tm_rgb(col = tm_vars(1:3, multivariate = TRUE))

if (requireNamespace("terra")) {
 L7_terra = terra::rast(file)

 tm_shape(L7_terra) +
  tm_rgb()

 # with layer names
 tm_shape(L7_terra) +
  tm_rgb(tm_vars(names(L7_terra)[1:3], multivariate = TRUE))

 # with layer indices
 tm_shape(L7_alt) +
  tm_rgb(col = tm_vars(1:3, multivariate = TRUE))

}

## End(Not run)
```

`tm_scale` *Scales: automatic scale*

Description

Scales in tmap are configured by the family of functions with prefix `tm_scale`. Such function should be used for the input of the `.scale` arguments in the layer functions (e.g. `fill.scale` in `tm_polygons()`). The function `tm_scale()` is a scale that is set automatically given by the data type (factor, numeric, and integer) and the visual variable. The tmap option `scales.var` contains information which scale is applied when.

Usage

```
tm_scale(...)
```

Arguments

`...`	arguments passed on to the applied scale function `tm_scale_*()`

See Also

`tm_scale_asis()`, `tm_scale_ordinal()`, `tm_scale_categorical()`, `tm_scale_intervals()`, `tm_scale_discrete()`, `tm_scale_continuous()`, `tm_scale_rank()`, `tm_scale_continuous_log()`, `tm_scale_continuous_log2()`, `tm_scale_continuous_log10()`, `tm_scale_continuous_log1p()`, `tm_scale_continuous_sqrt()`, `tm_scale_continuous_pseudo_log()`, `tm_scale_rgb()`, `tm_scale_bivariate()`

tm_scale_asis *Scales: as is*

Description

Scales in tmap are configured by the family of functions with prefix `tm_scale`. Such function should be used for the input of the `.scale` arguments in the layer functions (e.g. `fill.scale` in `tm_polygons()`). The function `tm_scale_asis()` is used to take data values as they are and use them as such for the visual variable.

Usage

```
tm_scale_asis(values.scale = NA, value.neutral = NA, value.na = NA, ...)
```

Arguments

`values.scale`	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as `size` of `tm_symbols()` and `lwd` of `tm_lines()`.
`value.neutral`	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both `fill` and `size` are used for `tm_symbols()` (using filled circles), the size legend items are filled with the `value.neutral` color from the `fill.scale` scale, and fill legend items are bubbles of size `value.neutral` from the `size.scale` scale.
`value.na`	(generic scale argument) Value used for missing values. See tmap option `"value.na"` for defaults per visual variable.
`...`	Arguments caught (and not used) from the automatic function `tm_scale()`

See Also

`tm_scale()`

tm_scale_bivariate *Scales: bivariate scale*

Description

Scales in tmap are configured by the family of functions with prefix `tm_scale`. Such function should be used for the input of the `.scale` arguments in the layer functions (e.g. `fill.scale` in `tm_polygons()`). The function `tm_scale_bivariate()` is used for `bivariate.scales`.

Usage

```
tm_scale_bivariate(
  scale1 = tm_scale(),
  scale2 = tm_scale(),
  values = NA,
  values.repeat = FALSE,
  values.range = NA,
  values.scale = 1,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  labels = NULL,
  label.na = NA,
  label.null = NA
)
```

Arguments

scale1, scale2	two tm_scale objects. Currently, all tm_scale_*() functions are supported except tm_scale_continuous().
values	(generic scale argument) The visual values. For colors (e.g. fill or col for tm_polygons()) this is a palette name from the cols4all package (see cols4all::c4a()) or vector of colors, for size (e.g. size for tm_symbols()) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. shape for tm_symbols()) these are a set of symbols, etc. The tmap option values.var contains the default values per visual variable and in some cases also per data type.
values.repeat	(generic scale argument) Should the values be repeated in case there are more categories?
values.range	(generic scale argument) Range of the values. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as c(0, 1). For instance, when a grey scale is used for color (from black to white), c(0,1) means that all colors are used, 0.25, 0.75 means that only colors from dark grey to light grey are used (more precisely "grey25" to "grey75"), and 0, 0.5 means that only colors are used from black to middle grey ("grey50"). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option values.range.
values.scale	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as size of tm_symbols() and lwd of tm_lines().
value.na	(generic scale argument) Value used for missing values. See tmap option "value.na" for defaults per visual variable.
value.null	(generic scale argument) Value used for NULL values. See tmap option "value.null" for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).

value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both fill and size are used for tm_symbols() (using filled circles), the size legend items are filled with the value.neutral color from the fill.scale scale, and fill legend items are bubbles of size value.neutral from the size.scale scale.
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values

See Also

tm_scale()

---

tm_scale_continuous *Scales: continuous scale*

Description

Scales in tmap are configured by the family of functions with prefix tm_scale. Such function should be used for the input of the .scale arguments in the layer functions (e.g. fill.scale in tm_polygons()). The function tm_scale_continuous() is used for continuous data. The functions tm_scale_continuous_<x>() use transformation functions x.

Usage

```
tm_scale_continuous(
  n = NULL,
  limits = NULL,
  outliers.trunc = NULL,
  ticks = NULL,
  trans = NULL,
  midpoint = NULL,
  values = NA,
  values.repeat = FALSE,
  values.range = NA,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  labels = NULL,
  label.na = NA,
  label.null = NA,
  label.format = tm_label_format(),
  trans.args = list()
)
```

```
tm_scale_continuous_log(..., base = exp(1))

tm_scale_continuous_log2(...)

tm_scale_continuous_log10(...)

tm_scale_continuous_log1p(...)

tm_scale_continuous_sqrt(...)

tm_scale_continuous_pseudo_log(..., base = exp(1), sigma = 1)
```

Arguments

n	Preferred number of tick labels. Only used if ticks is not specified
limits	Limits of the data values that are mapped to the continuous scale. When NA, the range of data values is taken. When only one value is provided, the range of data values with this provided value is taken. The default depends on the visual variable: it is 0 for all visual variables other than color when tm_scale_continuous is used. For the transformation scale functions, it is NA.
outliers.trunc	Should outliers be truncated? An outlier is a data value that is below or above the respectively lower and upper limit. A logical vector of two values is expected. The first and second value determines whether values lower than the lower limit respectively higher than the upper limit are truncated to the lower respectively upper limit. If FALSE (default), they are considered as missing values.
ticks	Tick values. If not specified, it is determined automatically with n
trans	Transformation function. One of "identity" (default), "log", and "log1p". Note: the base of the log scale is irrelevant, since the log transformed values are normalized before mapping to visual values.
midpoint	The data value that is interpreted as the midpoint. By default it is set to 0 if negative and positive values are present. Useful when values are diverging colors. In that case, the two sides of the color palette are assigned to negative respectively positive values. If all values are positive or all values are negative, then the midpoint is set to NA, which means that the value that corresponds to the middle color class (see style) is mapped to the middle color. If it is specified for sequential color palettes (e.g. "Blues"), then this color palette will be treated as a diverging color palette.
values	(generic scale argument) The visual values. For colors (e.g. fill or col for tm_polygons()) this is a palette name from the cols4all package (see cols4all::c4a()) or vector of colors, for size (e.g. size for tm_symbols()) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. shape for tm_symbols()) these are a set of symbols, etc. The tmap option values.var contains the default values per visual variable and in some cases also per data type.
values.repeat	(generic scale argument) Should the values be repeated in case there are more categories?

values.range	(generic scale argument) Range of the values, especially useful for color palettes. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as `c(0, 1)`. For instance, when a gray scale is used for color (from black to white), `c(0,1)` means that all colors are used, `0.25, 0.75` means that only colors from dark gray to light gray are used (more precisely `"grey25"` to `"grey75"`), and `0, 0.5` means that only colors are used from black to middle gray (`"grey50"`). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option `values.range`.
values.scale	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as `size` of `tm_symbols()` and `lwd` of `tm_lines()`.
value.na	(generic scale argument) Value used for missing values. See tmap option `"value.na"` for defaults per visual variable.
value.null	(generic scale argument) Value used for NULL values. See tmap option `"value.null"` for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).
value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both `fill` and `size` are used for `tm_symbols()` (using filled circles), the size legend items are filled with the `value.neutral` color from the `fill.scale` scale, and fill legend items are bubbles of size `value.neutral` from the `size.scale` scale.
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values
label.format	(generic scale argument) Label formatting. Output of `tm_label_format()`
trans.args	list of additional argument for the transformation (generic transformation arguments)
...	passed on to `tm_scale_continuous()`
base	base of logarithm
sigma	Scaling factor for the linear part of pseudo-log transformation.

See Also

`tm_scale()`

Examples

```
tm_shape(World) +
  tm_polygons(
   fill = "HPI",
    fill.scale = tm_scale_continuous(values = "scico.roma", midpoint = 30))

tm_shape(metro) +
  tm_bubbles(
```

```
    size = "pop1950",
     size.scale = tm_scale_continuous(
       values.scale = 1),
     size.legend = tm_legend("Population in 1950", frame = FALSE))

tm_shape(metro) +
  tm_bubbles(
    size = "pop1950",
    size.scale = tm_scale_continuous(
      values.scale = 2,
      limits = c(0, 12e6),
      ticks = c(1e5, 3e5, 8e5, 4e6, 1e7),
      labels = c("0 - 200,000", "200,000 - 500,000", "500,000 - 1,000,000",
        "1,000,000 - 10,000,000", "10,000,000 or more"),
      outliers.trunc = c(TRUE, TRUE)),
    size.legend = tm_legend("Population in 1950", frame = FALSE))
# Note that for this type of legend, we recommend tm_scale_intervals()
```

`tm_scale_discrete` *Scales: discrete scale*

Description

Scales in tmap are configured by the family of functions with prefix `tm_scale`. Such function should be used for the input of the `.scale` arguments in the layer functions (e.g. `fill.scale` in `tm_polygons()`). The function `tm_scale_discrete()` is used for discrete numerical data, such as integers.

Usage

```
tm_scale_discrete(
  ticks = NA,
  midpoint = NULL,
  values = NA,
  values.repeat = FALSE,
  values.range = NA,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  labels = NULL,
  label.na = NA,
  label.null = NA,
  label.format = list()
)
```

Arguments

ticks	Discrete values. If not specified, it is determined automatically: unique values are put on a discrete scale.
midpoint	The data value that is interpreted as the midpoint. By default it is set to 0 if negative and positive values are present. Useful when values are diverging colors. In that case, the two sides of the color palette are assigned to negative respectively positive values. If all values are positive or all values are negative, then the midpoint is set to NA, which means that the value that corresponds to the middle color class (see style) is mapped to the middle color. If it is specified for sequential color palettes (e.g. "Blues"), then this color palette will be treated as a diverging color palette.
values	(generic scale argument) The visual values. For colors (e.g. fill or col for tm_polygons()) this is a palette name from the cols4all package (see cols4all::c4a()) or vector of colors, for size (e.g. size for tm_symbols) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. shape for tm_symbols()) these are a set of symbols, etc. The tmap option values.var contains the default values per visual variable and in some cases also per data type.
values.repeat	(generic scale argument) Should the values be repeated in case there are more categories?
values.range	(generic scale argument) Range of the values. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as c(0, 1). For instance, when a gray scale is used for color (from black to white), c(0,1) means that all colors are used, 0.25, 0.75 means that only colors from dark gray to light gray are used (more precisely "grey25" to "grey75"), and 0, 0.5 means that only colors are used from black to middle grey ("grey50"). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option values.range.
values.scale	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as size of tm_symbols() and lwd of tm_lines().
value.na	(generic scale argument) Value used for missing values. See tmap option "value.na" for defaults per visual variable.
value.null	(generic scale argument) Value used for NULL values. See tmap option "value.null" for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).
value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both fill and size are used for tm_symbols() (using filled circles), the size legend items are filled with the value.neutral color from the fill.scale scale, and fill legend items are bubbles of size value.neutral from the size.scale scale.
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values
label.format	(generic scale argument) Label formatting. Output of tm_label_format()

See Also

tm_scale()

---

tm_scale_intervals *Scales: interval scale*

Description

Scales in tmap are configured by the family of functions with prefix tm_scale. Such function should be used for the input of the .scale arguments in the layer functions (e.g. fill.scale in tm_polygons()). The function tm_scale_intervals() is used for numerical data.

Usage

```
tm_scale_intervals(
  n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"),
  style.args = list(),
  breaks = NULL,
  interval.closure = "left",
  label.style = "discrete",
  label.select = TRUE,
  midpoint = NULL,
  as.count = FALSE,
  values = NA,
  values.repeat = FALSE,
  values.range = NA,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  labels = NULL,
  label.na = NA,
  label.null = NA,
  label.format = tm_label_format()
)
```

Arguments

n	Number of intervals. For some styles (see argument style below) it is the preferred number rather than the exact number.
style	Method to create intervals. Options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", "jenks", "dpih", "headtails", and "log10_pretty". See the details in classInt::classIntervals() (extra arguments can be passed on via style.args).
style.args	List of extra arguments passed on to classInt::classIntervals().

breaks — Interval breaks (only used and required when style = "fixed")

interval.closure — value that determines whether where the intervals are closed: "left" or "right". If as.count = TRUE, inverval.closure is always set to "left".

label.style — Either "discrete" or "continuous". If discrete (default) intervals will be labeled. If continuous, the legend will be similar to a continuous scale legend (but with discrete colors), where the breaks are labeled.

label.select — Which labels are shown? A logical vector is expected, which is repeated along the number of labels. By default TRUE, which means all.

midpoint — The data value that is interpreted as the midpoint. By default it is set to 0 if negative and positive values are present. Useful when values are diverging colors. In that case, the two sides of the color palette are assigned to negative respectively positive values. If all values are positive or all values are negative, then the midpoint is set to NA, which means that the value that corresponds to the middle color class (see style) is mapped to the middle color. If it is specified for sequential color palettes (e.g. "Blues"), then this color palette will be treated as a diverging color palette.

as.count — Should the data variable be processed as a count variable? For instance, if style = "pretty", n = 2, and the value range of the variable is 0 to 10, then the column classes for as.count = TRUE are 0; 1 to 5; 6 to 10 (note that 0 is regarded as an own category) whereas for as.count = FALSE they are 0 to 5; 5 to 10. Only applicable if style is "pretty", "fixed", or "log10_pretty". By default FALSE.

values — (generic scale argument) The visual values. For colors (e.g. fill or col for tm_polygons()) this is a palette name from the cols4all package (see cols4all::c4a()) or vector of colors, for size (e.g. size for tm_symbols) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. shape for tm_symbols) these are a set of symbols, etc. The tmap option values.var contains the default values per visual variable and in some cases also per data type.

values.repeat — (generic scale argument) Should the values be repeated in case there are more categories?

values.range — (generic scale argument) Range of the values. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as c(0, 1). For instance, when a gray scale is used for color (from black to white), c(0,1) means that all colors are used, 0.25, 0.75 means that only colors from dark gray to light gray are used (more precisely "gray25" to "gray75"), and 0, 0.5 means that only colors are used from black to middle grey ("grey50"). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option values.range.

values.scale — (generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as size of tm_symbols() and lwd of tm_lines().

value.na — (generic scale argument) Value used for missing values. See tmap option "value.na" for defaults per visual variable.

value.null	(generic scale argument) Value used for NULL values. See tmap option "value.null" for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).
value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both fill and size are used for tm_symbols() (using filled circles), the size legend items are filled with the value.neutral color from the fill.scale scale, and fill legend items are bubbles of size value.neutral from the size.scale scale.
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values
label.format	(generic scale argument) Label formatting. Output of tm_label_format()

See Also

tmap basics: scales

tm_scale()

---

tm_scale_ordinal *Scales: categorical and ordinal scale*

Description

Scales in tmap are configured by the family of functions with prefix tm_scale. Such function should be used for the input of the .scale arguments in the layer functions (e.g. fill.scale in tm_polygons()). The functions tm_scale_categorical() and tm_scale_ordinal() are used for categorical data. The only difference between these functions is that the former assumes unordered categories whereas the latter assumes ordered categories. For colors (the visual variable fill or col), different default color palettes are used (see the tmap option values.var).

Usage

```
tm_scale_ordinal(
  n.max = 30,
  values = NA,
  values.repeat = FALSE,
  values.range = 1,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  levels = NULL,
  levels.drop = FALSE,
  labels = NULL,
```

```
  label.na = NA,
  label.null = NA,
  label.format = list()
)

tm_scale_categorical(
  n.max = 30,
  values = NA,
  values.repeat = TRUE,
  values.range = NA,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  levels = NULL,
  levels.drop = FALSE,
  labels = NULL,
  label.na = NA,
  label.null = NA,
  label.format = list()
)
```

Arguments

`n.max`	Maximum number of categories (factor levels). In case there are more, they are grouped into `n.max` groups.
`values`	(generic scale argument) The visual values. For colors (e.g. `fill` or `col` for `tm_polygons()`) this is a palette name from the `cols4all` package (see `cols4all::c4a()`) or vector of colors, for size (e.g. `size` for `tm_symbols()`) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. `shape` for `tm_symbols()`) these are a set of symbols, etc. The tmap option `values.var` contains the default values per visual variable and in some cases also per data type.
`values.repeat`	(generic scale argument) Should the values be repeated in case there are more categories?
`values.range`	(generic scale argument) Range of the values. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as `c(0, 1)`. For instance, when a gray scale is used for color (from black to white), `c(0,1)` means that all colors are used, `0.25, 0.75` means that only colors from dark gray to light gray are used (more precisely `"grey25"` to `"grey75"`), and `0, 0.5` means that only colors are used from black to middle gray (`"gray50"`). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option `values.range`.
`values.scale`	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as `size` of `tm_symbols()` and `lwd` of `tm_lines()`.
`value.na`	(generic scale argument) Value used for missing values. See tmap option `"value.na"` for defaults per visual variable.

value.null	(generic scale argument) Value used for NULL values. See tmap option "value.null" for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).
value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both fill and size are used for tm_symbols() (using filled circles), the size legend items are filled with the value.neutral color from the fill.scale scale, and fill legend items are bubbles of size value.neutral from the size.scale scale.
levels	Levels to show. Other values are treated as missing.
levels.drop	Should unused levels be dropped (and therefore are not assigned to a visual value and shown in the legend)?
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values
label.format	(generic scale argument) Label formatting. Output of tm_label_format()

See Also

tm_scale()

---

tm_scale_rank *Scales: rank scale*

Description

Scales in tmap are configured by the family of functions with prefix tm_scale. Such function should be used for the input of the .scale arguments in the layer functions (e.g. fill.scale in tm_polygons()). The function tm_scale_rank() is used to rank numeric data.

Usage

```
tm_scale_rank(
  n = NULL,
  ticks = NULL,
  values = NA,
  values.repeat = FALSE,
  values.range = NA,
  values.scale = NA,
  value.na = NA,
  value.null = NA,
  value.neutral = NA,
  labels = NULL,
  label.na = NA,
  label.null = NA,
```

```
  label.format = tm_label_format(),
  unit = "rank"
)
```

Arguments

n	Preferred number of tick labels. Only used if ticks is not specified
ticks	Tick values. If not specified, it is determined automatically with n
values	(generic scale argument) The visual values. For colors (e.g. fill or col for tm_polygons()) this is a palette name from the cols4all package (see cols4all::c4a()) or vector of colors, for size (e.g. size for tm_symbols()) these are a set of sizes (if two values are specified they are interpret as range), for symbol shapes (e.g. shape for tm_symbols()) these are a set of symbols, etc. The tmap option values.var contains the default values per visual variable and in some cases also per data type.
values.repeat	(generic scale argument) Should the values be repeated in case there are more categories?
values.range	(generic scale argument) Range of the values, especially useful for color palettes. Vector of two numbers (both between 0 and 1) where the first determines the minimum and the second the maximum. Full range, which means that all values are used, is encoded as c(0, 1). For instance, when a gray scale is used for color (from black to white), c(0,1) means that all colors are used, 0.25, 0.75 means that only colors from dark gray to light gray are used (more precisely "grey25" to "grey75"), and 0, 0.5 means that only colors are used from black to middle gray ("grey50"). When only one number is specified, this is interpreted as the second number (where the first is set to 0). Default values can be set via the tmap option values.range.
values.scale	(generic scale argument) Scaling of the values. Only useful for size-related visual variables, such as size of tm_symbols() and lwd of tm_lines().
value.na	(generic scale argument) Value used for missing values. See tmap option "value.na" for defaults per visual variable.
value.null	(generic scale argument) Value used for NULL values. See tmap option "value.null" for defaults per visual variable. Null data values occur when out-of-scope features are shown (e.g. for a map of Europe showing a data variable per country, the null values are applied to countries outside Europe).
value.neutral	(generic scale argument) Value that can be considered neutral. This is used for legends of other visual variables of the same map layer. E.g. when both fill and size are used for tm_symbols() (using filled circles), the size legend items are filled with the value.neutral color from the fill.scale scale, and fill legend items are bubbles of size value.neutral from the size.scale scale.
labels	(generic scale argument) Labels
label.na	(generic scale argument) Label for missing values
label.null	(generic scale argument) Label for null (out-of-scope) values
label.format	(generic scale argument) Label formatting. Output of tm_label_format()
unit	The unit name of the values. By default "rank".

See Also

tm_scale()

---

tm_scale_rgb *Scales: RGB*

Description

Scales in tmap are configured by the family of functions with prefix tm_scale. Such function should be used for the input of the .scale arguments in the layer functions (e.g. fill.scale in tm_polygons()). The function tm_scale_rgb() is used to transform r, g, b band variables to colors. This function is adopted from (and works similar as) stars::st_rgb()

Usage

```
tm_scale_rgb(
  value.na = NA,
  stretch = FALSE,
  probs = c(0, 1),
  max_color_value = 255L
)

tm_scale_rgba(
  value.na = NA,
  stretch = FALSE,
  probs = c(0, 1),
  max_color_value = 255
)
```

Arguments

value.na	value for missing values
stretch	should each (r, g, b) band be stretched? Possible values: "percent" (same as TRUE), "histogram", FALSE. In the first case, the values are stretched to probs[1]...probs[2]. In the second case, a histogram equalization is performed
probs	probability (quantile) values when stretch = "percent"
max_color_value	maximum value

See Also

tm_scale() and stars::st_rgb()

Examples

```
## Not run:
require(stars)
file = system.file("tif/L7_ETMs.tif", package = "stars")

L7 = stars::read_stars(file)

tm_shape(L7) +
 tm_rgb(col.scale = tm_scale_rgb(probs = c(0, .99), stretch = TRUE))

tm_shape(L7) +
 tm_rgb(col.scale = tm_scale_rgb(stretch = "histogram"))

## End(Not run)
```

---

tm_scalebar *Map component: scale bar*

Description

Map component that adds a scale bar.

Usage

```
tm_scalebar(
  breaks,
  width,
  allow_clipping,
  text.size,
  text.color,
  color.dark,
  color.light,
  lwd,
  position,
  group_id,
  bg,
  bg.color,
  bg.alpha,
  size = "deprecated",
  stack,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  margins,
  z
)
```

Arguments

breaks	Scale bar break positions. E.g. `c(0, 10, 50)` places breaks at 0, 10, and 50 units. The unit is controlled by the `unit` argument from `tm_shape()`. When `NULL` (default), break positions are chosen automatically.
width	Width of the scale bar, in number of text line heights (roughly equivalent to character widths). When `breaks` are specified, `width` is only useful for fine-tuning, e.g. to prevent label clipping or reduce excess whitespace.
allow_clipping	Should clipping of the last label be allowed? If `TRUE` (default), the last break label including its unit suffix is printed even when it extends beyond the frame. If `FALSE`, that label is suppressed and the unit suffix is appended to the second-to-last label instead.
text.size	text size
text.color	text.color
color.dark	color.dark
color.light	color.light
lwd	linewidth
position	The position specification of the component: an object created with `tm_pos_in()` or `tm_pos_out()`. Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is `"left"`, `"center"` or `"right"` (or upper case, meaning tighter to the map frame), the second `"top"`, `"center"` or `"bottom"`. Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use `group_id` and specify `component` in `tm_components()`.
group_id	Component group id name. All components (e.g. legends, titles, etc) with the same `group_id` will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in `tm_components()` where its argument `id` should correspond to `group_id`.
bg	Show background?
bg.color	Background color
bg.alpha	Background transparency
size	Deprecated (use `text.size` instead)
stack	stack with other map components, either `"vertical"` or `"horizontal"`.
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width
frame.r	Radius of the rounded frame corners. 0 means no rounding.
margins	margins
z	z index, e.g. the place of the component relative to the other componets

See Also

Vignette about components

tm_seq *Specify a numeric sequence*

Description

Specify a numeric sequence, for numeric scales like `tm_scale_continuous()`. This function is needed when there is a non-linear relationship between the numeric data values and the visual variables. E.g. to make relationship with the area of bubbles linear, the square root of input variables should be used to calculate the radius of the bubbles.

Usage

```
tm_seq(
  from = 0,
  to = 1,
  power = c("lin", "sqrt", "sqrt_perceptual", "quadratic")
)
```

Arguments

`from`, `to`	The numeric range, default 0 and 1 respectively
`power`	The power component, a number or or one of `"lin"`, `"sqrt"`, `"sqrt_perceptual"`, `"quadratic"`, which correspond to 1, 0.5, 0.5716, 2 respectively. See details.

Details

The perceived area of larger symbols is often underestimated. Flannery (1971) experimentally derived a method to compensate this for symbols. This compensation is obtained by using the power exponent of 0.5716 instead of 0.5, or by setting `power` to `"sqrt_perceptual"`

tm_sf *Map layer: simple features*

Description

Map layer that draws simple features as they are. Supported visual variables are: `fill` (the fill color), `col` (the border color), `size` the point size, `shape` the symbol shape, `lwd` (line width), `lty` (line type), `fill_alpha` (fill color alpha transparency) and `col_alpha` (border color alpha transparency).

Usage

```
tm_sf(
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.free = NA,
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.free = NA,
  shape = tm_const(),
  shape.scale = tm_scale(),
  shape.legend = tm_legend(),
  shape.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.free = NA,
  linejoin = "round",
  lineend = "round",
  plot.order.list = list(polygons = tm_plot_order("AREA"), lines =
    tm_plot_order("LENGTH"), points = tm_plot_order("size")),
  options = opt_tm_sf(),
  zindex = NA,
  group = NA,
  group.control = "check",
  blend = "over",
  ...
)

opt_tm_sf(
  polygons.only = "yes",
```

```
  lines.only = "yes",
  points_only = "yes",
  point_per = "feature",
  points.icon.scale = 3,
  points.just = NA,
 points.grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)
```

Arguments

`fill`, `fill.scale`, `fill.legend`, `fill.free`
: Visual variable that determines the fill color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`col`, `col.scale`, `col.legend`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`size`, `size.scale`, `size.legend`, `size.free`
: Visual variable that determines the size. See details. *Unit:* Typographic lines ("lines"); 1 line is approx. 1/6 inch. Controlled by `values.scale` and `tmap_options(values.scale = ...)`.

`shape`, `shape.scale`, `shape.legend`, `shape.free`
: Visual variable that determines the shape. See details. *Unit:* Integer `pch` code (1-25) or a single character used as a plotting symbol.

`lwd`, `lwd.scale`, `lwd.legend`, `lwd.free`
: Visual variable that determines the line width. See details. *Unit:* Base R line-width units; 1 lwd is approx. 0.75 pt at 96 dpi. Controlled by `values.scale`.

`lty`, `lty.scale`, `lty.legend`, `lty.free`
: Visual variable that determines the line type. See details. *Unit:* Integer (1-6) or name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

`fill_alpha`, `fill_alpha.scale`, `fill_alpha.legend`, `fill_alpha.free`
: Visual variable that determines the fill color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`linejoin`, `lineend`
: line join and line end. See gpar() for details.

`plot.order.list`
: Specification in which order the spatial features are drawn. This consists of a list of three elementary geometry types: for polygons, lines and, points. For each of these types, which are drawn in that order, a `tm_plot_order()` is required.

`options`
: options passed on to the corresponding `opt_<layer_function>` function

`zindex`
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

`group`
: Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

`group.control` In view mode, the group control determines how layer groups can be switched on and off. Options: `"radio"` for radio buttons (meaning only one group can be shown), `"check"` for check boxes (so multiple groups can be shown), and `"none"` for no control (the group cannot be (de)selected).

`blend` Compositing operator for layer blending, applied to each sublayer (polygons, lines, points). Default `"over"` applies no blending. See the "Layer blending" section of `tm_polygons()` for supported values.

`...` passed on to `tm_polygons()`, `tm_lines()`, and `tm_dots()`

`polygons.only` should only polygon geometries of the shape object (defined in `tm_shape()`) be plotted? By default `"ifany"`, which means `TRUE` in case a geometry collection is specified.

`lines.only` should only line geometries of the shape object (defined in `tm_shape()`) be plotted, or also other geometry types (like polygons)? By default `"ifany"`, which means `TRUE` in case a geometry collection is specified.

`points_only` should only point geometries of the shape object (defined in `tm_shape()`) be plotted? By default `"ifany"`, which means `TRUE` in case a geometry collection is specified.

`point_per` specification of how spatial points are mapped when the geometry is a multi line or a multi polygon. One of `"feature"`, `"segment"` or `"largest"`. The first generates a spatial point for every feature, the second for every segment (i.e. subfeature), the third only for the largest segment (subfeature). Note that the last two options can be significant slower.

`points.icon.scale` scaling number that determines how large the icons (or grobs) are in plot mode in comparison to proportional symbols (such as bubbles). For view mode, use the argument `grob.dim`

`points.just` justification of the points relative to the point coordinates. Either one of the following values: `"left"` , `"right"`, `"center"`, `"bottom"`, and `"top"`, or a vector of two values where first value specifies horizontal and the second value vertical justification. Besides the mentioned values, also numeric values between 0 and 1 can be used. 0 means left justification for the first value and bottom justification for the second value. Note that in view mode, only one value is used.

`points.grob.dim` vector of four values that determine how grob objects (see details) are shown in view mode. The first and second value are the width and height of the displayed icon. The third and fourth value are the width and height of the rendered png image that is used for the icon. Generally, the third and fourth value should be large enough to render a ggplot2 graphic successfully. Only needed for the view mode.

Details

The visual variable arguments (e.g. `col`) can be specified with a data variable name (e.g., a spatial vector attribute or a raster layer of the object specified in `tm_shape()`), with a visual value (for `col`, a color is expected), or with a geometry-derived variable (see below). See vignette about visual variables.

Multiple values can be specified: in that case facets are created. These facets can be combined with other faceting data variables, specified with tm_facets(). See vignette about facets.

- The *.scale arguments determine the used scale to map the data values to visual variable values. These can be specified with one of the available tm_scale_*() functions. The default is specified by the tmap option (tm_options()) scales.var. See vignette about scales.
- The *.legend arguments determine the used legend, specified with tm_legend(). The default legend and its settings are determined by the tmap options (tm_options()) legend. . See vignette about legends.
- The *.chart arguments specify additional charts, specified with tm_chart_, e.g. tm_chart_histogram(). See vignette about charts.
- The *.free arguments determine whether scales are applied freely across facets, or shared. A logical value is required. They can also be specified with a vector of three logical values; these determine whether scales are applied freely per facet dimension. This is only useful when facets are applied (see tm_facets()). There are maximally three facet dimensions: rows, columns, and pages. This only applies for a facet grid (tm_facets_grid()). For instance, col.free = c(TRUE, FALSE, FALSE) means that for the visual variable col, each row of facets will have its own scale, and therefore its own legend. For facet wraps and stacks (tm_facets_wrap() and tm_facets_stack()) there is only one facet dimension, so the *.free argument requires only one logical value.

Currently, three geometry-derived variables are implemented:

- "AREA" (polygons only), which uses the feature area;
- "LENGTH" (lines only), which uses the feature length; and
- "MAP_COLORS", which assigns values so that adjacent features receive different values, making it particularly suitable for coloring neighbouring polygons.

Note that geometry-derived variables do not generate a legend automatically. If a legend is required, compute the corresponding variable explicitly, for example with sf::st_area(), sf::st_length(), or tmaptools::map_coloring(), and use the resulting values instead.

Visual variable units:

Every visual variable maps data values to a specific output unit. Knowing the unit matters when supplying constant values via tm_const(), or output ranges via values.range / values.scale in the scale functions.

Variable	Output unit	Notes
fill, col, bgcol	color	name, hex, or palette string
fill_alpha, col_alpha, bgcol_alpha	proportion 0-1	0 = transparent, 1 = opaque
size (symbols, bubbles, squares, dots)	typographic lines	1 line approx. 1/6 inch; scaled by values.scale
size (circles)	meters	plain numeric or a units object
size (text, labels)	multiplier	1 = 12 pt (plot) / 12 px (view)
lwd	lwd	base R units; 1 lwd approx. 0.75 pt at 96 dpi
lty	–	integer 1-6 or name ("solid", "dashed", ...)
shape	–	integer pch 1-25 or single character
angle	degrees	0-360, clockwise from north
fontface	–	"plain", "bold", "italic", "bold.italic"

Symbol size (`size` *in* `tm_symbols`, `tm_bubbles`, `tm_squares`, `tm_dots`*):*
"Lines" is a typographic unit: one line is approximately 1/6 inch (the default base line-height in R graphics). The global multiplier `tmap_options(values.scale = list(size.bubbles = 1.5))` scales all symbol sizes without changing the data mapping.

Circle size (`size` *in* `tm_circles`*):*
The value is a geographic radius in meters. A plain numeric vector is interpreted as meters; a `units` object (from the **units** package) is automatically converted, so `units::as_units(1, "mi")` gives a 1-mile radius. Because the radius is geographic, circles scale with zoom in interactive (view) mode – unlike bubble symbols which keep a fixed screen size.

Text size (`size` *in* `tm_text`, `tm_labels`*):*
The value is a multiplier of the base font size. `size = 1` renders at 12 pt in plot mode (R's default `par("ps")`) and at 12 px in view mode (`gp$cex * 12` px, see `tmapLeafletDataPlot.tm_data_text`); the two modes are consistent by design.

Layer blending (`blend`)**:**
Blend modes control how a layer's pixels are combined with the pixels beneath it. For each pixel, let S be the source (top layer) RGB value and D be the destination (bottom layer) RGB value, both normalised to $[0, 1]$.

`blend`	Formula	Use case
`"over"`	$S \cdot \alpha + D \cdot (1 - \alpha)$	Standard alpha compositing (default)
`"multiply"`	$S \times D$	Hillshading over colour raster; both layers darken each other
`"screen"`	$1 - (1 - S)(1 - D)$	Inverse of multiply; brightens
`"overlay"`	multiply if $D < 0.5$, screen if $D \geq 0.5$	Boosts contrast; preserves midtones
`"darken"`	$\min(S, D)$	Keeps the darker of the two layers per channel
`"lighten"`	$\max(S, D)$	Keeps the lighter of the two layers per channel

Requires R >= 4.2 and a compatible graphics device (e.g. `png(type = "cairo")`, `svg()`). In view mode, blending is applied via CSS `mix-blend-mode`. See `grid::groupGrob()` for the full list of supported operators.

zindex and pane names:
In view mode, each layer is rendered in a Leaflet pane named `"tmap{zindex}"` (e.g., `"tmap401"`, `"tmap402"`), with base tile layers placed in the standard `"tile"` pane.

Examples

```
data(World)

World$geometry[World$continent == "Africa"] <-
  sf::st_centroid(World$geometry[World$continent == "Africa"])
World$geometry[World$continent == "South America"] <-
  sf::st_cast(World$geometry[World$continent == "South America"],
    "MULTILINESTRING", group_or_split = FALSE)

tm_shape(World, crs = "+proj=robin") +
 tm_sf()
```

`tm_shape` *Shape (spatial object) specification*

Description

Specify a shape, which is a spatial object from one of these spatial object class packages: `sf`, `stars`, or `terra`.

Usage

```
tm_shape(
  shp = NULL,
  bbox = NULL,
  crs = NULL,
  is.main = NA,
  layer = NULL,
  name = NULL,
  unit = NULL,
  filter = NULL,
  ...
)
```

Arguments

`shp`	Spatial data object. Typically an object from **sf**, **terra**, or **stars**. Additional spatial data types can be supported via extension packages, such as **tmap.networks** and **tmap.sources** (experimental). These may include, for example, remote or streaming data sources.
`bbox`	Bounding box of the map. Only used when `shp` is the main shape (see `is.main`). Three options are supported: • a `sf::st_bbox()` object, • a character string specifying a location, passed to `tmaptools::geocode_OSM()`, • `"FULL"`, which represents the whole earth. This option ensures that reprojection retains the full global extent, unlike a regular bounding box.
`crs`	Map projection (CRS). Can be set to an `crs` object (see `sf::st_crs()`), a proj4string, an EPSG number, the value `"auto"` (automatic crs recommendation), or one the the following generic projections: `c("laea", "aeqd", "utm", "pconic", "eqdc", "stere")`. See details.
`is.main`	Is `shp` the main shape, which determines the crs and bounding box of the map? By default, `TRUE` if it is the first `tm_shape` call
`layer`	Name of the layer to use. This is primarily relevant for multi-layer or remote data sources (e.g. PMTiles or vector tiles), where multiple layers may be available.
`name`	of the spatial object

unit	Unit of distance measurement, used by tm_scalebar(). Either a specific unit string such as "km", "m", "mi", "yd", "ft", or "in" (see units::valid_udunits() for all options), or one of two automatic options: "metric" (default) selects the most readable unit from km, m, and mm; "imperial" selects from mi, yd, ft, and in. In both cases the unit is chosen as the largest one for which the map width expressed in that unit is at least 10.
filter	Optional filter expression used to subset features. The exact syntax depends on the data source. For in-memory objects (e.g. **sf**), this is typically evaluated in R, whereas for remote sources it may be translated to a query and executed on the server side.
...	passed on to bb (e.g. ext can be used to enlarge or shrink a bounding box)

Details

The map projection (crs) determines in which coordinate system the spatial object is processed and plotted. See vignette about CRS. The crs can be specified in two places: 1) tm_shape() and tm_crs(). In both cases, the map is plotted into the specified crs. The difference is that in the first option, the crs is also taken into account in spatial transformation functions, such as the calculation of centroids and cartograms. In the second option, the crs is only used in the plotting phase.

The automatic crs recommendation (which is still work-in-progress) is the following:

Property	Recommendation
global (for world maps)	A pseudocylindrical projection tmap option crs_global, by default "eqearth (Equal Earth). See
area (equal area)	Lambert Azimuthal Equal Area (laea)
distance (equidistant)	Azimuthal Equidistant (aeqd)
shape (conformal)	Stereographic (stere)

For further info about the available "generic" projects see: for utm: https://proj.org/en/9.4/operations/projections/utm.html for laea: https://proj.org/en/9.4/operations/projections/laea.html for aeqd: https://proj.org/en/9.4/operations/projections/aeqd.html for pconic: https://proj.org/en/9.4/operations/projections/pconic.html for eqdc: https://proj.org/en/9.4/operations/projections/eqdc.html

Note

as of tmap 4.0, simplify has been removed. Please use tmaptools::simplify_shape() instead

See Also

vignette about CRS

Examples

```
tm_shape(World, crs = "auto") +
 tm_polygons()

tm_shape(World, crs = 3035, bb = "Europe") +
 tm_polygons()
```

```
tm_shape(World, crs = "+proj=robin", filter = World$continent=="Africa") +
 tm_polygons()
```

`tm_style` *Layout options*

Description

Specify the layout of the maps. `tm_layout()` is identical as `tm_options()` but only contain the tmap options that are directly related to the layout. `tm_style()` sets the style for the map. A style is a specified set of options (that can be changed afterwards with `tm_layout()`). These functions are used within a plot a plot call (stacked with the + operator). Their counterparts `tmap_options()` and `tmap_style()` can be used to set the (layout) options globally.

Usage

```
tm_style(style, ...)

tm_layout(
  scale,
  asp,
  legend.only,
  bg,
  bg.color,
  outer.bg,
  outer.bg.color,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  frame.double_line,
  outer.margins,
  inner.margins,
  inner.margins.extra,
  meta.margins,
  meta.auto_margins,
  between_margin,
  text.fontfamily,
  text.fontface,
  r,
  attr.color,
  panel.margin,
  panel.type,
  panel.wrap.pos,
  panel.xtab.pos,
```

```
  color.sepia_intensity,
  color.saturation,
  color_vision_deficiency_sim,
  panel.show,
  panel.labels,
  panel.label.size,
  panel.label.color,
  panel.label.fontface,
  panel.label.fontfamily,
  panel.label.alpha,
  panel.label.bg,
  panel.label.bg.color,
  panel.label.bg.alpha,
  panel.label.frame,
  panel.label.frame.color,
  panel.label.frame.alpha,
  panel.label.frame.lwd,
  panel.label.frame.r,
  panel.label.height,
  panel.label.rot,
  earth_boundary,
  earth_boundary.color,
  earth_boundary.lwd,
  earth_datum,
  space,
  space.color,
  ...
)
```

Arguments

style	name of the style
...	List of tmap options to be set, or option names (characters) to be returned (see details)
scale	Overall scale of the map
asp	Aspect ratio of each map. When asp is set to NA (default) the aspect ratio will be adjusted to the used shapes. When set to 0 the aspect ratio is adjusted to the size of the device divided by the number of columns and rows.
legend.only	Should only legends be printed (so without map)?
bg	Draw map background?
bg.color	Background color of the map.
outer.bg	Draw map background (outside the frame)?
outer.bg.color	Background color of map outside the frame.
frame	Draw map frame?
frame.color	The color of the frame.
frame.alpha	The alpha transparency of the frame.

`frame.lwd` The line width of the frame. See `graphics::par`, option 'lwd'.

`frame.r` The r (radius) of the frame.

`frame.double_line`
The double line of the frame. `TRUE` or `FALSE`.

`outer.margins` The margins of the outer space (outside the frame. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inner.margins` The margins of the inner space (inside the frame). A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`inner.margins.extra`
The extra arguments of the margins of the inner space (inside the frame). A list of arguments.

`meta.margins` The margins of the meta. A vector of 4 values: bottom, left, top, right. The unit is the device height (for bottom and top) or width (for left and right).

`meta.auto_margins`
The auto_margins of the meta.

`between_margin` Margin between the map.

`text.fontfamily`
The font family of the text. See `graphics::par`, option 'family'.

`text.fontface` The font face of the text. See `graphics::par`, option 'font'.

`r` The r (radius) (overall).

`attr.color` The color of the attr.

`panel.margin` The margin of the panel.

`panel.type` The type of the panel.

`panel.wrap.pos` The panel positions for wrapped facets created with `tm_facets_grid()`. One of `"left"`, `"right"`, `"top"` (default) or `"bottom"`.

`panel.xtab.pos` The panel positions for grid facets created with `tm_facets_grid()`. Vector of two, where the first determines the locations of row panels (`"left"` or `"right"`) and the second the location of column panels (`"top"` or '"bottom")

`color.sepia_intensity`
The sepia_intensity of the color.

`color.saturation`
The saturation of the color.

`color_vision_deficiency_sim`
Color vision deficiency simulation. Either `"protan"`, `"deutan"`, or `"tritan"`.

`panel.show` The visibility of the panel. `TRUE` or `FALSE`.

`panel.labels` The labels of the panel.

`panel.label.size`
The size of the label of the panel.

`panel.label.color`
The color of the label of the panel.

`panel.label.fontface`
: The font face of the label of the panel. See `graphics::par`, option 'font'.

`panel.label.fontfamily`
: The font family of the label of the panel. See `graphics::par`, option 'family'.

`panel.label.alpha`
: The alpha transparency of the label of the panel.

`panel.label.bg` The bg of the label of the panel.

`panel.label.bg.color`
: The color of the bg of the label of the panel.

`panel.label.bg.alpha`
: The alpha transparency of the bg of the label of the panel.

`panel.label.frame`
: The frame of the label of the panel.

`panel.label.frame.color`
: The color of the frame of the label of the panel.

`panel.label.frame.alpha`
: The alpha transparency of the frame of the label of the panel.

`panel.label.frame.lwd`
: The line width of the frame of the label of the panel. See `graphics::par`, option 'lwd'.

`panel.label.frame.r`
: The r (radius) of the frame of the label of the panel.

`panel.label.height`
: The height of the label of the panel.

`panel.label.rot`
: Rotation angles of the panel labels. Vector of four values that determine the panel label rotation when they are placed left, top, right, and bottom. The default angles are 90, 0, 270 and 0 respectively. Note that the second value is the most common, since labels are by default shown on top (see `panel.wrap.pos`). In cross-table facets created with `tm_facets_grid()`, the first two values are used by default (see `panel.xtab.pos`).

`earth_boundary` The earth boundary

`earth_boundary.color`
: The color of the earth_boundary.

`earth_boundary.lwd`
: The line width of the earth_boundary. See `graphics::par`, option 'lwd'.

`earth_datum` Earth datum

`space` Should the space be drawn? Only applicable is earth_boundary is enabled.

`space.color` The color of the space.

See Also

Vignette about layout, vignette about margins and aspect ratio and vignette about options

Examples

```
tm_shape(World) +
  tm_polygons() +
tm_layout(
  bg.color = "steelblue",
  outer.bg.color = "gold",
  frame.lwd = 3,
  inner.margins = 0)

tm_shape(World) +
  tm_polygons(fill = "HPI") +
tm_style("classic")

tm_shape(World) +
 tm_polygons(fill = "HPI") +
 tm_style("cobalt")
```

`tm_symbols` *Map layer: symbols*

Description

Map layer that draws symbols Supported visual variables are: `fill` (the fill color), `col` (the border color), `size` the symbol size, `shape` the symbol shape, `lwd` (line width), `lty` (line type), `fill_alpha` (fill color alpha transparency) and `col_alpha` (border color alpha transparency).

Usage

```
tm_symbols(
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.chart = tm_chart_none(),
  size.free = NA,
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.chart = tm_chart_none(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  shape = tm_const(),
  shape.scale = tm_scale(),
  shape.legend = tm_legend(),
  shape.chart = tm_chart_none(),
```

```
  shape.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.chart = tm_chart_none(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.chart = tm_chart_none(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.chart = tm_chart_none(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  plot.order = tm_plot_order("size"),
  zindex = NA,
  group = NA,
  group.control = "check",
  popup = tm_popup(),
  popup.vars = NA,
  popup.format = tm_label_format(),
  hover = NA,
  id = "",
  blend = "over",
  options = opt_tm_symbols(),
  ...
)

tm_dots(
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.free = NA,
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.free = NA,
```

```
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.free = NA,
  plot.order = tm_plot_order("DATA"),
  zindex = NA,
  group = NA,
  group.control = "check",
  options = opt_tm_dots(),
  ...
)

tm_bubbles(
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.free = NA,
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.free = NA,
  plot.order = tm_plot_order("size"),
  zindex = NA,
  group = NA,
```

```
  group.control = "check",
  options = opt_tm_bubbles(),
  ...
)

tm_squares(
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.free = NA,
  fill = tm_const(),
  fill.scale = tm_scale(),
  fill.legend = tm_legend(),
  fill.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.free = NA,
  lwd = tm_const(),
  lwd.scale = tm_scale(),
  lwd.legend = tm_legend(),
  lwd.free = NA,
  lty = tm_const(),
  lty.scale = tm_scale(),
  lty.legend = tm_legend(),
  lty.free = NA,
  fill_alpha = tm_const(),
  fill_alpha.scale = tm_scale(),
  fill_alpha.legend = tm_legend(),
  fill_alpha.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.free = NA,
  plot.order = tm_plot_order("size"),
  zindex = NA,
  group = NA,
  group.control = "check",
  options = opt_tm_squares(),
  ...
)

tm_markers(
  text = tm_const(),
  text.scale = tm_scale(),
  text.legend = tm_legend(),
  text.chart = tm_chart_none(),
  text.free = NA,
```

```
    size = tm_const(),
    size.scale = tm_scale(),
    size.legend = tm_legend(),
    size.chart = tm_chart_none(),
    size.free = NA,
    col = tm_const(),
    col.scale = tm_scale(),
    col.legend = tm_legend(),
    col.chart = tm_chart_none(),
    col.free = NA,
    col_alpha = tm_const(),
    col_alpha.scale = tm_scale(),
    col_alpha.legend = tm_legend(),
    col_alpha.chart = tm_chart_none(),
    col_alpha.free = NA,
    fontface = tm_const(),
    fontface.scale = tm_scale(),
    fontface.legend = tm_legend(),
    fontface.chart = tm_chart_none(),
    fontface.free = NA,
    fontfamily = "",
    bgcol = tm_const(),
    bgcol.scale = tm_scale(),
    bgcol.legend = tm_legend(),
    bgcol.chart = tm_chart_none(),
    bgcol.free = NA,
    bgcol_alpha = tm_const(),
    bgcol_alpha.scale = tm_scale(),
    bgcol_alpha.legend = tm_legend(),
    bgcol_alpha.chart = tm_chart_none(),
    bgcol_alpha.free = NA,
    xmod = 0,
    xmod.scale = tm_scale(),
    xmod.legend = tm_legend_hide(),
    xmod.chart = tm_chart_none(),
    xmod.free = NA,
    ymod = 0,
    ymod.scale = tm_scale(),
    ymod.legend = tm_legend_hide(),
    ymod.chart = tm_chart_none(),
    ymod.free = NA,
    angle = 0,
    angle.scale = tm_scale(),
    angle.legend = tm_legend_hide(),
    angle.chart = tm_chart_none(),
    angle.free = NA,
    plot.order = tm_plot_order("AREA", reverse = FALSE, na.order = "bottom"),
    zindex = NA,
```

```
  group = NA,
  group.control = "check",
  options = opt_tm_markers(),
  ...
)

opt_tm_markers(
  markers_on_top_of_text = FALSE,
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  shadow = FALSE,
  shadow.offset.x = 0.1,
  shadow.offset.y = 0.1,
  just = "center",
  along_lines = TRUE,
  bg.padding = 0.4,
  clustering = TRUE,
  point.label = TRUE,
  point.label.gap = 0.4,
  point.label.method = "SANN",
  remove_overlap = FALSE,
  dots.just = NA,
  dots.icon.scale = 3,
 dots.grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)

opt_tm_symbols(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  clustering = FALSE,
  icon.scale = 3,
  just = NA,
  hitbox = "auto",
  grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)

opt_tm_dots(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  clustering = FALSE,
  icon.scale = 3,
  just = NA,
  hitbox = "auto",
  grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)
```

```
opt_tm_bubbles(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  clustering = FALSE,
  icon.scale = 3,
  just = NA,
  hitbox = "auto",
  grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)

opt_tm_squares(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  clustering = FALSE,
  icon.scale = 3,
  just = NA,
  hitbox = "auto",
  grob.dim = c(width = 48, height = 48, render.width = 256, render.height = 256)
)
```

Arguments

`size`, `size.scale`, `size.legend`, `size.chart`, `size.free`
: Visual variable that determines the size. See details. *Unit:* Typographic lines ("lines"); 1 line is approx. 1/6 inch. Controlled by `values.scale` and `tmap_options(values.scale = ...)`.

`fill`, `fill.scale`, `fill.legend`, `fill.chart`, `fill.free`
: Visual variable that determines the fill color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`shape`, `shape.scale`, `shape.legend`, `shape.chart`, `shape.free`
: Visual variable that determines the shape. See details. *Unit:* Integer `pch` code (1-25) or a single character used as a plotting symbol.

`lwd`, `lwd.scale`, `lwd.legend`, `lwd.chart`, `lwd.free`
: Visual variable that determines the line width. See details. *Unit:* Base R line-width units; 1 lwd is approx. 0.75 pt at 96 dpi. Controlled by `values.scale`.

`lty`, `lty.scale`, `lty.legend`, `lty.chart`, `lty.free`
: Visual variable that determines the line type. See details. *Unit:* Integer (1-6) or name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

`fill_alpha`, `fill_alpha.scale`, `fill_alpha.legend`, `fill_alpha.chart`, `fill_alpha.free`
: Visual variable that determines the fill color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque). the fill color

alpha transparency See details.

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`plot.order`
: Specification in which order the spatial features are drawn. See `tm_plot_order()` for details.

`zindex`
: Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.

`group`
: Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see `group.control`)

`group.control`
: In view mode, the group control determines how layer groups can be switched on and off. Options: `"radio"` for radio buttons (meaning only one group can be shown), `"check"` for check boxes (so multiple groups can be shown), and `"none"` for no control (the group cannot be (de)selected).

`popup`
: popup specification for `"view"` mode, the output of `tm_popup()`. It determines the data variables shown in the popup table, the popup title, and (in the future) the popup layout. This replaces the deprecated arguments `popup.vars` and `popup.format`.

`popup.vars`
: (Deprecated.) Use `popup` with `tm_popup()` instead (via its `vars` argument). Names of data variables that are shown in the popups in `"view"` mode. Set `popup.vars` to `TRUE` to show all variables in the shape object. Set `popup.vars` to `FALSE` to disable popups. Set `popup.vars` to a character vector of variable names to show those variables in the popups. The default (`NA`) depends on whether visual variables (e.g. `fill`) are used. If so, only those are shown. If not, all variables in the shape object are shown.

`popup.format`
: (Deprecated.) Use `popup` with `tm_popup()` instead (via its `format` argument). List of formatting options for the popup values. Output of `tm_label_format()`. Only applicable for numeric data variables. If one list of formatting options is provided, it is applied to all numeric variables of `popup.vars`. Also, a (named) list of lists can be provided. In that case, each list of formatting options is applied to the named variable.

`hover`
: name of the data variable that specifies the hover labels (view mode only). Set to `FALSE` to disable hover labels. By default `FALSE`, unless `id` is specified. In that case, it is set to `id`,

`id`
: name of the data variable that specifies the indices of the spatial features. Only used for `"view"` mode.

`blend`
: Compositing operator for layer blending. Default `"over"` applies no blending. See the "Layer blending" section for the supported values.

`options`
: options passed on to the corresponding `opt_<layer_function>` function

`...`
: to catch deprecated arguments from version < 4.0

`text`, `text.scale`, `text.legend`, `text.chart`, `text.free`
: Visual variable that determines the text. See details. *Unit:* Character string.

`fontface`, `fontface.scale`, `fontface.legend`, `fontface.chart`, `fontface.free`
: Visual variable that determines the font face. See details. *Unit:* "plain", "bold", "italic", or "bold.italic".

`fontfamily`
: The font family. See gpar() for details.

`bgcol`, `bgcol.scale`, `bgcol.legend`, `bgcol.chart`, `bgcol.free`
: Visual variable that determines the background color. See Details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`bgcol_alpha`, `bgcol_alpha.scale`, `bgcol_alpha.legend`, `bgcol_alpha.chart`, `bgcol_alpha.free`
: Visual variable that determines the background color transparency. See Details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`xmod`, `xmod.scale`, `xmod.legend`, `xmod.chart`, `xmod.free`
: Transformation variable that determines the x offset. See details. *Unit:* Line heights, relative to the label anchor. Positive = right.

`ymod`, `ymod.scale`, `ymod.legend`, `ymod.chart`, `ymod.free`
: Transformation variable that determines the y offset. See details. *Unit:* Line heights, relative to the label anchor. Positive = up. the text. See details.

`angle`, `angle.scale`, `angle.legend`, `angle.chart`, `angle.free`
: Rotation angle *Unit:* Degrees, clockwise from north (0-360).

`markers_on_top_of_text`
: should markers be plot on top of the text (by default `FALSE`)

`points_only`
: should only point geometries of the shape object (defined in `tm_shape()`) be plotted? By default `"ifany"`, which means `TRUE` in case a geometry collection is specified.

`point_per`
: specification of how spatial points are mapped when the geometry is a multi line or a multi polygon. One of `"feature"`, `"segment"` or `"largest"`. The first generates a spatial point for every feature, the second for every segment (i.e. subfeature), the third only for the largest segment (subfeature). Note that the last two options can be significant slower.

`on_surface`
: In case of polygons, centroids are computed. Should the points be on the surface? If `TRUE`, which is slower than the default `FALSE`, centroids outside the surface are replaced with points computed with `sf::st_point_on_surface()`.

`shadow`
: Shadow behind the text. Logical.

`shadow.offset.x`, `shadow.offset.y`
: Shadow offset in line heights

`just`
: justification of the text relative to the point coordinates. Either one of the following values: `"left"` , `"right"`, `"center"`, `"bottom"`, and `"top"`, or a vector of two values where first value specifies horizontal and the second value vertical justification. Besides the mentioned values, also numeric values between 0 and 1 can be used. 0 means left justification for the first value and bottom justification for the second value. Note that in view mode, only one value is used.

`along_lines`
: logical that determines whether labels are rotated along the spatial lines. Only applicable if a spatial lines shape is used.

`bg.padding`
: The padding of the background in terms of line heights.

clustering — in interactive modes (e.g. "view" mode), should clustering be applied at lower zoom levels? Either FALSE (default), TRUE, or a mode specific specification, e.g. for "view" mode markerClusterOptions.

point.label — logical that determines whether the labels are placed automatically. By default FALSE for tm_text, and TRUE for tm_labels if the number of labels is less than 500 (otherwise it will be too slow).

point.label.gap — numeric that determines the gap between the point and label

point.label.method — the optimization method, either "SANN" for simulated annealing (the default) or "GA" for a genetic algorithm.

remove_overlap — logical that determines whether the overlapping labels are removed

dots.just — justification of the text relative to the point coordinates. Either one of the following values: "left" , "right", "center", "bottom", and "top", or a vector of two values where first value specifies horizontal and the second value vertical justification. Besides the mentioned values, also numeric values between 0 and 1 can be used. 0 means left justification for the first value and bottom justification for the second value. Note that in view mode, only one value is used.

dots.icon.scale — scaling number that determines how large the icons (or grobs) are in plot mode in comparison to proportional symbols (such as bubbles). In view mode, the size is determined by the icon specification (see tmap_icons) or, if grobs are specified by grob.width and grob.height

dots.grob.dim — vector of four values that determine how grob objects (see details) are shown in view mode. The first and second value are the width and height of the displayed icon. The third and fourth value are the width and height of the rendered png image that is used for the icon. Generally, the third and fourth value should be large enough to render a ggplot2 graphic successfully. Only needed for the view mode.

icon.scale — scaling number that determines how large the icons (or grobs) are in plot mode in comparison to proportional symbols (such as bubbles). For view mode, use the argument grob.dim

hitbox — Controls whether an invisible interaction layer with a larger clickable area (\"hitbox\") is added on top of the symbols.

This can improve click and popup behaviour for small or densely packed symbols by enlarging the effective mouse interaction area.

Possible values:

"none" No additional hitbox layer is added. Symbols are clickable only at their visible size.

\item{"plusX"}{Adds \code{X} pixels to the visible symbol diameter to compute the interaction size. For example, \code{"plus8"} widens the clickable area by 4 pixels around each symbol edge.}

\item{"pmaxX"}{Ensures a minimum interaction diameter of \code{X} pixels: \code{pmax(symbol_diameter, X)}. Useful only for very small symbols, as

it adds no margin to symbols already larger than \code{X}.}

\item{"auto"}{\code{"pmax12"} if and only if interactive features are enabled (popup or hover), symbols are small (median visible diameter < 12px), and there are fewer than 10000 features. Otherwise \code{"none"}.}

plus and pmax can be combined, e.g. "plus4pmax8". The hitbox is not applied when clustering is enabled.

grob.dim vector of four values that determine how grob objects (see details) are shown in view mode. The first and second value are the width and height of the displayed icon. The third and fourth value are the width and height of the rendered png image that is used for the icon. Generally, the third and fourth value should be large enough to render a ggplot2 graphic successfully. Only needed for the view mode.

Details

A symbol shape specification is one of the following three options.

1. A numeric value that specifies the plotting character of the symbol. See parameter pch of points and the last example to create a plot with all options. Note that this is not supported for the "view" mode.
2. A grob object, which can be a ggplot2 plot object created with ggplotGrob. To specify multiple shapes, a list of grob objects is required. Tip: for proportional symbols, such as donuts or pies, see the extension package tmap.glyphs.
3. An icon specification, which can be created with tmap_icons.

To specify multiple shapes (needed for the shapes argument), a vector or list of these shape specification is required. The shape specification options can also be mixed. For the shapes argument, it is possible to use a named vector or list, where the names correspond to the value of the variable specified by the shape argument. For small multiples, a list of these shape specification(s) should be provided.

See Also

Bubble map example and terrain map example

Examples

```
########################
## plot symbol shapes
########################

# create grid of 25 points in the Atlantic
atlantic_grid = cbind(expand.grid(x = -51:-47, y = 20:24), id = seq_len(25))
x = sf::st_as_sf(atlantic_grid, coords = c("x", "y"), crs = 4326)

tm_shape(x, bbox = tmaptools::bb(x, ext = 1.2)) +
 tm_symbols(shape = "id",
      size = 2,
      lwd = 2,
```

```
    fill = "orange",
    col = "black",
    shape.scale = tm_scale_asis()) +
tm_text("id", ymod = -2)
```

`tm_text` *Map layer: text*

Description

Map layer that draws symbols Supported visual variables are: `text` (the text itself) `col` (color), `size` (font size), and `fontface` (font face).

Usage

```
tm_text(
  text = tm_const(),
  text.scale = tm_scale(),
  text.legend = tm_legend(),
  text.chart = tm_chart_none(),
  text.free = NA,
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.chart = tm_chart_none(),
  size.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  fontface = tm_const(),
  fontface.scale = tm_scale(),
  fontface.legend = tm_legend(),
  fontface.chart = tm_chart_none(),
  fontface.free = NA,
  fontfamily = NA,
  bgcol = tm_const(),
  bgcol.scale = tm_scale(),
  bgcol.legend = tm_legend(),
  bgcol.chart = tm_chart_none(),
  bgcol.free = NA,
```

```
  bgcol_alpha = tm_const(),
  bgcol_alpha.scale = tm_scale(),
  bgcol_alpha.legend = tm_legend(),
  bgcol_alpha.chart = tm_chart_none(),
  bgcol_alpha.free = NA,
  xmod = 0,
  xmod.scale = tm_scale(),
  xmod.legend = tm_legend_hide(),
  xmod.chart = tm_chart_none(),
  xmod.free = NA,
  ymod = 0,
  ymod.scale = tm_scale(),
  ymod.legend = tm_legend_hide(),
  ymod.chart = tm_chart_none(),
  ymod.free = NA,
  angle = 0,
  angle.scale = tm_scale(),
  angle.legend = tm_legend_hide(),
  angle.chart = tm_chart_none(),
  angle.free = NA,
  plot.order = tm_plot_order("size", reverse = FALSE),
  zindex = NA,
  group = NA,
  group.control = "check",
  blend = "over",
  options = opt_tm_text(),
  ...
)

tm_labels(
  text = tm_const(),
  text.scale = tm_scale(),
  text.legend = tm_legend(),
  text.chart = tm_chart_none(),
  text.free = NA,
  size = tm_const(),
  size.scale = tm_scale(),
  size.legend = tm_legend(),
  size.chart = tm_chart_none(),
  size.free = NA,
  col = tm_const(),
  col.scale = tm_scale(),
  col.legend = tm_legend(),
  col.chart = tm_chart_none(),
  col.free = NA,
  col_alpha = tm_const(),
  col_alpha.scale = tm_scale(),
  col_alpha.legend = tm_legend(),
```

```
  col_alpha.chart = tm_chart_none(),
  col_alpha.free = NA,
  fontface = tm_const(),
  fontface.scale = tm_scale(),
  fontface.legend = tm_legend(),
  fontface.chart = tm_chart_none(),
  fontface.free = NA,
  fontfamily = "",
  bgcol = tm_const(),
  bgcol.scale = tm_scale(),
  bgcol.legend = tm_legend(),
  bgcol.chart = tm_chart_none(),
  bgcol.free = NA,
  bgcol_alpha = tm_const(),
  bgcol_alpha.scale = tm_scale(),
  bgcol_alpha.legend = tm_legend(),
  bgcol_alpha.chart = tm_chart_none(),
  bgcol_alpha.free = NA,
  xmod = 0,
  xmod.scale = tm_scale(),
  xmod.legend = tm_legend_hide(),
  xmod.chart = tm_chart_none(),
  xmod.free = NA,
  ymod = 0,
  ymod.scale = tm_scale(),
  ymod.legend = tm_legend_hide(),
  ymod.chart = tm_chart_none(),
  ymod.free = NA,
  angle = 0,
  angle.scale = tm_scale(),
  angle.legend = tm_legend_hide(),
  angle.chart = tm_chart_none(),
  angle.free = NA,
  plot.order = tm_plot_order("AREA", reverse = FALSE, na.order = "bottom"),
  zindex = NA,
  group = NA,
  group.control = "check",
  options = opt_tm_labels(),
  ...
)

tm_labels_highlighted(
  text = tm_const(),
  text.scale = tm_scale(),
  text.legend = tm_legend(),
  text.chart = tm_chart_none(),
  text.free = NA,
  size = tm_const(),
```

```
    size.scale = tm_scale(),
    size.legend = tm_legend(),
    size.chart = tm_chart_none(),
    size.free = NA,
    col = tm_const(),
    col.scale = tm_scale(),
    col.legend = tm_legend(),
    col.chart = tm_chart_none(),
    col.free = NA,
    col_alpha = tm_const(),
    col_alpha.scale = tm_scale(),
    col_alpha.legend = tm_legend(),
    col_alpha.chart = tm_chart_none(),
    col_alpha.free = NA,
    fontface = tm_const(),
    fontface.scale = tm_scale(),
    fontface.legend = tm_legend(),
    fontface.chart = tm_chart_none(),
    fontface.free = NA,
    fontfamily = "",
    bgcol = tm_const(),
    bgcol.scale = tm_scale(),
    bgcol.legend = tm_legend(),
    bgcol.chart = tm_chart_none(),
    bgcol.free = NA,
    bgcol_alpha = tm_const(),
    bgcol_alpha.scale = tm_scale(),
    bgcol_alpha.legend = tm_legend(),
    bgcol_alpha.chart = tm_chart_none(),
    bgcol_alpha.free = NA,
    xmod = 0,
    xmod.scale = tm_scale(),
    xmod.legend = tm_legend_hide(),
    xmod.chart = tm_chart_none(),
    xmod.free = NA,
    ymod = 0,
    ymod.scale = tm_scale(),
    ymod.legend = tm_legend_hide(),
    ymod.chart = tm_chart_none(),
    ymod.free = NA,
    angle = 0,
    angle.scale = tm_scale(),
    angle.legend = tm_legend_hide(),
    angle.chart = tm_chart_none(),
    angle.free = NA,
    plot.order = tm_plot_order("AREA", reverse = FALSE, na.order = "bottom"),
    zindex = NA,
    group = NA,
```

```
  group.control = "check",
  options = opt_tm_labels(),
  ...
)

opt_tm_text(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  shadow = FALSE,
  shadow.col = NA,
  shadow.offset.x = 0.05,
  shadow.offset.y = 0.05,
  halo = FALSE,
  halo.col = NA,
  halo.width = 0.05,
  halo.blur = 0,
  halo.alpha = 0.8,
  just = "center",
  along_lines = FALSE,
  bg.padding = 0.4,
  bg.border = FALSE,
  bg.border.col = "black",
  bg.border.lwd = 1,
  clustering = FALSE,
  point.label = FALSE,
  point.label.gap = 0,
  point.label.method = "SANN",
  remove_overlap = FALSE
)

opt_tm_labels(
  points_only = "ifany",
  point_per = "feature",
  on_surface = FALSE,
  shadow = FALSE,
  shadow.col = NA,
  shadow.offset.x = 0.05,
  shadow.offset.y = 0.05,
  halo = FALSE,
  halo.col = NA,
  halo.width = 0.05,
  halo.blur = 0,
  halo.alpha = 0.8,
  just = "center",
  along_lines = TRUE,
  bg.padding = 0.4,
  bg.border = FALSE,
```

```
  bg.border.col = "black",
  bg.border.lwd = 1,
  clustering = FALSE,
  point.label = NA,
  point.label.gap = 0.4,
  point.label.method = "SANN",
  remove_overlap = FALSE
)
```

Arguments

`text`, `text.scale`, `text.legend`, `text.chart`, `text.free`
: Visual variable that determines the text. See details. *Unit:* Character string.

`size`, `size.scale`, `size.legend`, `size.chart`, `size.free`
: Visual variable that determines the size. See details. *Unit:* Multiplier of the base font size. `size = 1` renders at the default font size, which is 12 pt in plot mode (`par("ps")`) and 12 px in view mode (consistent by design). `size = 1.5` renders at 18 pt / px, etc.

`col`, `col.scale`, `col.legend`, `col.chart`, `col.free`
: Visual variable that determines the color. See details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`col_alpha`, `col_alpha.scale`, `col_alpha.legend`, `col_alpha.chart`, `col_alpha.free`
: Visual variable that determines the color transparency. See details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`fontface`, `fontface.scale`, `fontface.legend`, `fontface.chart`, `fontface.free`
: Visual variable that determines the font face. See details. *Unit:* "plain", "bold", "italic", or "bold.italic".

`fontfamily`
: The font family. See gpar() for details.

`bgcol`, `bgcol.scale`, `bgcol.legend`, `bgcol.chart`, `bgcol.free`
: Visual variable that determines the background color. See Details. *Unit:* Color – a color name, hex string, or (when mapped) a palette name.

`bgcol_alpha`, `bgcol_alpha.scale`, `bgcol_alpha.legend`, `bgcol_alpha.chart`, `bgcol_alpha.free`
: Visual variable that determines the background color transparency. See Details. *Unit:* Proportion – numeric 0-1 (0 = fully transparent, 1 = fully opaque).

`xmod`, `xmod.scale`, `xmod.legend`, `xmod.chart`, `xmod.free`
: Transformation variable that determines the x offset. See details. *Unit:* Line heights, relative to the label anchor. Positive = right.

`ymod`, `ymod.scale`, `ymod.legend`, `ymod.chart`, `ymod.free`
: Transformation variable that determines the y offset. See details. *Unit:* Line heights, relative to the label anchor. Positive = up. the text. See details.

`angle`, `angle.scale`, `angle.legend`, `angle.chart`, `angle.free`
: Rotation angle *Unit:* Degrees, clockwise from north (0-360).

`plot.order`
: Specification in which order the spatial features are drawn. See `tm_plot_order()` for details.

zindex	Controls the stacking order of map layers. Should be set to a value above 400. By default, layers are stacked in call order, starting at 401. See details.
group	Name of the group to which this layer belongs. This is only relevant in view mode, where layer groups can be switched (see group.control)
group.control	In view mode, the group control determines how layer groups can be switched on and off. Options: "radio" for radio buttons (meaning only one group can be shown), "check" for check boxes (so multiple groups can be shown), and "none" for no control (the group cannot be (de)selected).
blend	Compositing operator for layer blending. Default "over" applies no blending. See the "Layer blending" section for the supported values.
options	options passed on to the corresponding opt_<layer_function> function
...	to catch deprecated arguments from version < 4.0
points_only	should only point geometries of the shape object (defined in tm_shape()) be plotted? By default "ifany", which means TRUE in case a geometry collection is specified.
point_per	specification of how spatial points are mapped when the geometry is a multi line or a multi polygon. One of "feature", "segment" or "largest". The first generates a spatial point for every feature, the second for every segment (i.e. subfeature), the third only for the largest segment (subfeature). Note that the last two options can be significant slower.
on_surface	In case of polygons, centroids are computed. Should the points be on the surface? If TRUE, which is slower than the default FALSE, centroids outside the surface are replaced with points computed with sf::st_point_on_surface().
shadow	Shadow behind the text. Logical.
shadow.col	Color of the shadow.
shadow.offset.x, shadow.offset.y	Shadow offset in line heights
halo	Halo behind the text. In plot mode, it is just an outline, in view mode also a subtle glow.
halo.col	Color of the halo.
halo.width	Width (thickness) of the halo outline. In line heights
halo.blur	Blur radius of the halo glow (view mode only). In line heights. Should be sufficiently larger than halo.width in order to see the effect.
halo.alpha	Alpha transparency of the halo
just	justification of the text relative to the point coordinates. Either one of the following values: "left" , "right", "center", "bottom", and "top", or a vector of two values where first value specifies horizontal and the second value vertical justification. Besides the mentioned values, also numeric values between 0 and 1 can be used. 0 means left justification for the first value and bottom justification for the second value. Note that in view mode, only one value is used.
along_lines	logical that determines whether labels are rotated along the spatial lines. Only applicable if a spatial lines shape is used.
bg.padding	The padding of the background in terms of line heights.

bg.border	Should the background have borders?
bg.border.col	Color of the borders
bg.border.lwd	Line width of the borders
clustering	in interactive modes (e.g. "view" mode), should clustering be applied at lower zoom levels? Either FALSE (default), TRUE, or a mode specific specification, e.g. for "view" mode markerClusterOptions.
point.label	logical that determines whether the labels are placed automatically. By default FALSE for tm_text, and TRUE for tm_labels if the number of labels is less than 500 (otherwise it will be too slow).
point.label.gap	numeric that determines the gap between the point and label
point.label.method	the optimization method, either "SANN" for simulated annealing (the default) or "GA" for a genetic algorithm.
remove_overlap	logical that determines whether the overlapping labels are removed

Details

The visual variable arguments (e.g. col) can be specified with a data variable name (e.g., a spatial vector attribute or a raster layer of the object specified in tm_shape()), with a visual value (for col, a color is expected), or with a geometry-derived variable (see below). See vignette about visual variables.

Multiple values can be specified: in that case facets are created. These facets can be combined with other faceting data variables, specified with tm_facets(). See vignette about facets.

- The *.scale arguments determine the used scale to map the data values to visual variable values. These can be specified with one of the available tm_scale_*() functions. The default is specified by the tmap option (tm_options()) scales.var. See vignette about scales.
- The *.legend arguments determine the used legend, specified with tm_legend(). The default legend and its settings are determined by the tmap options (tm_options()) legend. . See vignette about legends.
- The *.chart arguments specify additional charts, specified with tm_chart_, e.g. tm_chart_histogram(). See vignette about charts.
- The *.free arguments determine whether scales are applied freely across facets, or shared. A logical value is required. They can also be specified with a vector of three logical values; these determine whether scales are applied freely per facet dimension. This is only useful when facets are applied (see tm_facets()). There are maximally three facet dimensions: rows, columns, and pages. This only applies for a facet grid (tm_facets_grid()). For instance, col.free = c(TRUE, FALSE, FALSE) means that for the visual variable col, each row of facets will have its own scale, and therefore its own legend. For facet wraps and stacks (tm_facets_wrap() and tm_facets_stack()) there is only one facet dimension, so the *.free argument requires only one logical value.

Currently, three geometry-derived variables are implemented:

- "AREA" (polygons only), which uses the feature area;

- "LENGTH" (lines only), which uses the feature length; and
- "MAP_COLORS", which assigns values so that adjacent features receive different values, making it particularly suitable for coloring neighbouring polygons.

Note that geometry-derived variables do not generate a legend automatically. If a legend is required, compute the corresponding variable explicitly, for example with sf::st_area(), sf::st_length(), or tmaptools::map_coloring(), and use the resulting values instead.

Visual variable units:

Every visual variable maps data values to a specific output unit. Knowing the unit matters when supplying constant values via tm_const(), or output ranges via values.range / values.scale in the scale functions.

Variable	Output unit	Notes
fill, col, bgcol	color	name, hex, or palette string
fill_alpha, col_alpha, bgcol_alpha	proportion 0-1	0 = transparent, 1 = opaque
size (symbols, bubbles, squares, dots)	typographic lines	1 line approx. 1/6 inch; scaled by values.scale
size (circles)	meters	plain numeric or a units object
size (text, labels)	multiplier	1 = 12 pt (plot) / 12 px (view)
lwd	lwd	base R units; 1 lwd approx. 0.75 pt at 96 dpi
lty	–	integer 1-6 or name ("solid", "dashed", ...)
shape	–	integer pch 1-25 or single character
angle	degrees	0-360, clockwise from north
fontface	–	"plain", "bold", "italic", "bold.italic"

*Symbol size (*size *in* tm_symbols, tm_bubbles, tm_squares, tm_dots*):*
"Lines" is a typographic unit: one line is approximately 1/6 inch (the default base line-height in R graphics). The global multiplier tmap_options(values.scale = list(size.bubbles = 1.5)) scales all symbol sizes without changing the data mapping.

*Circle size (*size *in* tm_circles*):*
The value is a geographic radius in meters. A plain numeric vector is interpreted as meters; a units object (from the **units** package) is automatically converted, so units::as_units(1, "mi") gives a 1-mile radius. Because the radius is geographic, circles scale with zoom in interactive (view) mode – unlike bubble symbols which keep a fixed screen size.

*Text size (*size *in* tm_text, tm_labels*):*
The value is a multiplier of the base font size. size = 1 renders at 12 pt in plot mode (R's default par("ps")) and at 12 px in view mode (gp$cex * 12 px, see tmapLeafletDataPlot.tm_data_text); the two modes are consistent by design.

Layer blending (blend)**:**

Blend modes control how a layer's pixels are combined with the pixels beneath it. For each pixel, let S be the source (top layer) RGB value and D be the destination (bottom layer) RGB value, both normalised to $[0, 1]$.

blend	Formula	Use case
"over"	$S \cdot \alpha + D \cdot (1 - \alpha)$	Standard alpha compositing (default)
"multiply"	$S \times D$	Hillshading over colour raster; both layers darken each other

"screen"	$1 - (1 - S)(1 - D)$	Inverse of multiply; brightens
"overlay"	multiply if $D < 0.5$, screen if $D \geq 0.5$	Boosts contrast; preserves midtones
"darken"	$\min(S, D)$	Keeps the darker of the two layers per channel
"lighten"	$\max(S, D)$	Keeps the lighter of the two layers per channel

Requires R >= 4.2 and a compatible graphics device (e.g. `png(type = "cairo")`, `svg()`). In view mode, blending is applied via CSS `mix-blend-mode`. See `grid::groupGrob()` for the full list of supported operators.

zindex and pane names:

In view mode, each layer is rendered in a Leaflet pane named `"tmap{zindex}"` (e.g., `"tmap401"`, `"tmap402"`), with base tile layers placed in the standard `"tile"` pane.

See Also

Terrain map example

Topographic map

Examples

```
tm_shape(World, bbox = World) +
 tm_text("name", size="pop_est", col="continent",
   col.scale = tm_scale_categorical(values = "seaborn.dark"),
   col.legend = tm_legend_hide(),
   size.scale = tm_scale_continuous(values.scale = 4),
   size.legend = tm_legend_hide())

metro$upside_down = ifelse(sf::st_coordinates(metro)[,2] < 0, 180, 0)
tm_shape(metro) +
 tm_text(text = "name", size = "pop2020",
   angle = "upside_down", size.legend = tm_legend_hide(),
   col = "upside_down",
   col.scale = tm_scale_categorical(values = c("#9900BB", "#228822")),
   col.legend = tm_legend_hide()) +
 tm_title_out("Which Hemisphere?", position = tm_pos_out("center", "top", pos.v = "bottom"))
```

---

tm_title *Map component: title*

Description

Map component that adds a title

Usage

```
tm_title(
  text,
  size,
  color,
  padding,
  fontface,
  fontfamily,
  alpha,
  stack,
  just,
  frame,
  frame.color,
  frame.alpha,
  frame.lwd,
  frame.r,
  bg,
  bg.color,
  bg.alpha,
  position,
  group_id,
  width,
  height,
  z
)

tm_title_in(text, ..., position = tm_pos_in("left", "top"))

tm_title_out(text, ..., position = tm_pos_out("center", "top"))
```

Arguments

text	text
size	font size
color	font color
padding	padding
fontface	font face, bold, italic
fontfamily	font family
alpha	alpha transparency of the text
stack	stack with other map components, either "vertical" or "horizontal".
just	just
frame	frame should a frame be drawn?
frame.color	frame color
frame.alpha	frame alpha transparancy
frame.lwd	frame line width

`frame.r`	Radius of the rounded frame corners. 0 means no rounding.
`bg`	Show background?
`bg.color`	Background color
`bg.alpha`	Background transparency
`position`	The position specification of the component: an object created with `tm_pos_in()` or `tm_pos_out()`. Or, as a shortcut, a vector of two values, specifying the x and y coordinates. The first is `"left"`, `"center"` or `"right"` (or upper case, meaning tighter to the map frame), the second `"top"`, `"center"` or `"bottom"`. Numeric values are also supported, where 0, 0 means left bottom and 1, 1 right top. See also vignette about positioning. In case multiple components should be combined (stacked), use `group_id` and specify `component` in `tm_components()`.
`group_id`	Component group id name. All components (e.g. legends, titles, etc) with the same `group_id` will be grouped. The specifications of how they are placed (e.g. stacking, margins etc.) are determined in `tm_components()` where its argument `id` should correspond to `group_id`.
`width, height`	width and height of the component.
`z`	z index, e.g. the place of the component relative to the other componets
`...`	passed on to `tm_title()`

See Also

Vignette about components

---

tm_vars *tmap function to specify variables*

Description

tmap function to specify all variables in the shape object

Usage

```
tm_vars(
  x = NA,
  dimvalues = NULL,
  n = NA,
  multivariate = FALSE,
  animate = FALSE
)
```

Arguments

x	variable names, variable indices, or a dimension name
dimvalues	dimension values
n	if specified the first n variables are taken (or the first n dimension values)
multivariate	in case multiple variables are specified, should they serve as facets (FALSE) or as a multivariate visual variable?
animate	should the variable(s) be animated? (experimental)

---

tm_view *View mode options*

Description

View mode options. These options are specific to the view mode.

Usage

```
tm_view(
  use_browser,
  use_WebGL,
  control.position,
  control.bases,
  control.overlays,
  control.collapse,
  set_bounds,
  set_view,
  set_zoom_limits,
  use_circle_markers,
  leaflet.options,
  ...
)
```

Arguments

use_browser	If TRUE it opens an external browser, and FALSE (default) it opens the internal IDEs (e.g. RStudio) browser.
use_WebGL	use webGL for points, lines, and polygons. For large spatial objects, this is much faster than the standard leaflet layer functions. However, it can not always be used for two reasons. First, the number of visual variables are limited; only fill, size, and color (for lines) are supported. Second, projected CRS's are not supported. Furthermore, it has the drawback that polygon borders are not as sharp. By default only TRUE for large spatial objects (1000 or more features) when the mentioned criteria are met. By default TRUE if no other visual variables are used.

`control.position`
: The position of the control. A tm_pos object, or a shortcut of two values: horizontal (left, center, right) and vertical (top, center, bottom). See tm_pos for details

`control.bases`
: base layers

`control.overlays`
: overlay layers

`control.collapse`
: Should the box be collapsed or expanded?

`set_bounds`
: logical that determines whether maximum bounds are set, or a bounding box. Not applicable in plot mode. In view mode, this is passed on to setMaxBounds()

`set_view`
: numeric vector that determines the view. Either a vector of three: `lng`, `lat`, and `zoom`, or a single value: `zoom`. See setView(). Only applicable if `bbox` is not specified

`set_zoom_limits`
: numeric vector of two that set the minimum and maximum zoom levels (see tileOptions()).

`use_circle_markers`
: If `TRUE` (default) circle shaped symbols (e.g. `tm_dots()` and `tm_symbols()`) will be rendered as addCircleMarkers() instead of addMarkers(). The former is faster, the latter can support any symbol since it is based on icons

`leaflet.options`
: options passed on to leafletOptions()

`...`
: to catch deprecated arguments

See Also

`tm_group()`

---

tm_xlab *Map: x and y labels*

Description

The x and y labels for maps

Usage

```
tm_xlab(text, size, color, rotation, space, fontface, fontfamily, alpha, side)

tm_ylab(text, size, color, rotation, space, fontface, fontfamily, alpha, side)
```

Arguments

text	text of the title
size	font size of the title
color	color
rotation	rotation in degrees
space	space between label and map in number of line heights
fontface	font face
fontfamily	font family
alpha	alpha transparency of the text
side	side: "top" or "bottom" for tm_xlab and "left" or "right" for tm_ylab

---

tmap-element *Stacking of tmap elements*

Description

The plus operator allows you to stack tmap elements (functions with a prefix tm_)

Usage

```
## S3 method for class 'tmap'
e1 + e2
```

Arguments

e1	first tmap element
e2	second tmap element

---

tmap_animation *Create animation*

Description

Create a gif animation or video from an animated tmap plot. First use tm_animate() or tm_animate_fast() to animate the plot, and then apply tmap_animation() to save it as a gif or video file (e.g. mp4).

Usage

```
tmap_animation(
  tm,
  filename = NULL,
  width = NA,
  height = NA,
  dpi = NA,
  outer.margins = NA,
  asp = NULL,
  scale = NA,
  ...
)
```

Arguments

tm	tmap or a list of tmap objects. If tm is a tmap object, animation frames should be created using either tm_animate() or tm_animate_fast().
filename	filename. If omitted (default), the animation will be shown in the viewer or browser. If specified, it should be a gif file or a video file (i.e. mp4). The package gifski is required to create a gif animation. The package av (which uses the FFmpeg library) is required for video formats. The mp4 format is recommended but many other video formats are supported, such as wmv, avi, and mkv.
width, height	Dimensions of the animation file (in pixels). Required when tm is a list, and recommended to specify in advance when tm is a tmap object. If not specified in the latter case, it will be determined by the aspect ratio of the map.
dpi	dots per inch. By default 100, but this can be set with the option animation.dpi in tmap_options().
outer.margins	(passed on to tmap_save()) overrides the outer.margins argument of tm_layout() (unless set to NA)
asp	(passed on to tmap_save()) if specified, it overrides the asp argument of tm_layout(). Tip: set to 0 if map frame should be placed on the edges of the image.
scale	(passed on to tmap_save()) overrides the scale argument of tm_layout() (unless set to NA)
...	arguments passed on to av::av_encode_video()

Note

Not only tmap plots are supported, but any series of R plots.

Examples

```
if (interactive()) {
 m1 <- tm_shape(NLD_prov) +
  tm_polygons("yellow") +
  tm_animate(frames = "name")

 tmap_animation(m1, filename = "countries.gif")
```

```
 m2 <- tm_shape(metro) +
  tm_symbols(size = paste0("pop", seq(1950, 2030, by=10)),
       size.free = FALSE,
       size.legend = tm_legend("Population")) +
  tm_layout(panel.labels = seq(1970, 2030, by=10)) +
  tm_animate()

 tmap_animation(m2, filename = "cities.gif")
}
```

---

tmap_arrange *Arrange small multiples in grid layout*

Description

Arrange small multiples in a grid layout. Normally, small multiples are created by specifying multiple variables for one aesthetic or by specifying the by argument (see tm_facets()). This function can be used to arrange custom small multiples in a grid layout.

Usage

```
tmap_arrange(
  ...,
  ncol = NA,
  nrow = NA,
  widths = NA,
  heights = NA,
  sync = FALSE,
  asp = 0,
  outer.margins = 0.02
)

## S3 method for class 'tmap_arrange'
knit_print(x, ..., options = NULL)

## S3 method for class 'tmap_arrange'
print(x, knit = FALSE, ..., options = NULL)
```

Arguments

...	tmap objects or one list of tmap objects. The number of multiples that can be plot is limited (see details).
ncol	number of columns
nrow	number of rows

widths	vector of column widths. It should add up to 1 and the length should be equal to ncol.
heights	vector of row heights. It should add up to 1 and the length should be equal to nrow.
sync	logical. Should the navigation in view mode (zooming and panning) be synchronized? By default FALSE.
asp	aspect ratio. The aspect ratio of each map. Normally, this is controlled by the asp argument from tm_layout() (also a tmap option). This argument will overwrite it, unless set to NULL. The default value for asp is 0, which means that the aspect ratio is adjusted to the size of the device divided by the number of columns and rows. When asp is set to NA, which is also the default value for tm_layout(), the aspect ratio will be adjusted to the used shapes.
outer.margins	outer.margins, numeric vector four or a single value. If defines the outer margins for each multiple. If will overwrite the outer.margins argument from tm_layout(), unless set to NULL.
x	a tmap_arrange object (returned from tmap_arrange()).
options	options passed on to knitr::knit_print()
knit	should knitr::knit_print() be enabled, or the normal base::print() function?

Details

The global option tmap.limits controls the limit of the number of facets that are plotted. By default, tmap_options(tmap.limits = c(facets.view=4, facets.plot=64)). The maximum number of interactive facets is set to four since otherwise it may become very slow.

Examples

```
tm1 = tm_shape(World) + tm_polygons("HPI")
tm2 = tm_shape(metro) + tm_bubbles(size = "pop2020")

tmap_arrange(tm1, tm2)
```

---

tmap_design_mode *Set the design mode*

Description

When the so-called "design mode" is enabled, the composition of the plot is shown explicitly in plot mode. The used color codings is printed in the console as well as information about plot size and aspect ratio.

Usage

```
tmap_design_mode(design.mode)
```

Arguments

design.mode	Logical value that determines the design mode. If omitted then the design mode is toggled.

Details

This function sets the global option tmap.design.mode. It can be used as toggle function without arguments.

See Also

tmap_options()

---

tmap_devel_mode *Set the development mode*

Description

When the so-called "development mode" is enabled, helpful messages and timings are printed in the console

Usage

```
tmap_devel_mode(devel.mode)
```

Arguments

devel.mode	logical value that determines the development mode. If omitted then the development mode is toggled.

---

tmap_icons *Specify icons*

Description

Specifies icons from a png images, which can be used as markers in thematic maps. The function marker_icon() is the specification of the default marker.

Usage

```
tmap_icons(
  file,
  names = NULL,
  width = 48,
  height = 48,
  keep.asp = TRUE,
  just = c("center", "center"),
  merge = NA,
  as.local = TRUE,
  ...
)

marker_icon()
```

Arguments

file	character value/vector containing the file path(s) or url(s).
names	names to be given to the icons. Useful when icons are assigned to factor levels.
width	width of the icon. If keep.asp, this is interpreted as the maximum width.
height	height of the icon. If keep.asp, this is interpreted as the maximum height.
keep.asp	keep the aspect ratio of the png image. If TRUE and the aspect ratio differs from width/height, either width or height is adjusted accordingly.
just	justification of the icons relative to the point coordinates. The first value specifies horizontal and the second value vertical justification. Possible values are: "left" , "right", "center", "bottom", and "top". Numeric values of 0 specify left alignment and 1 right alignment. The default value of just is c("center", "center").
merge	merge icons to one icon list (see return value)? If FALSE, a list is created per file. By default TRUE, unless names are specified.
as.local	if the file is a url, should it be saved to local temporary file?
...	arguments passed on to leaflet::icons(). When iconWidth, iconHeight, iconAnchorX, and iconAnchorY are specified, they override width and height, and just.

Value

icon data (see leaflet::icons())

See Also

tm_symbols()

tmap_last *Retrieve the last map to be modified or created*

Description

Retrieve the last map to be modified or created. Works in the same way as `ggplot2::last_plot()`, although there is a difference: `tmap_last()` returns the last call instead of the stacked `tmap-element`s.

Usage

```
tmap_last()
```

Value

call

See Also

`tmap_save()`

tmap_leaflet *Export tmap to the format of the used graphics mode*

Description

- `tmap_grob()` returns a `grob` object (`"plot"` mode)
- `tmap_leaflet()` a `leaflet` object (`"view"` mode).

Usage

```
tmap_leaflet(x, show = FALSE, ...)

tmap_grob(x, asp = NA, scale = NA, show = FALSE, ...)
```

Arguments

x	a tmap object.
show	show the map?
...	Arguments passed on to `print.tmap` `return.asp` should the aspect ratio be returned? `vp` viewport (for `"plot"` mode) `knit` A logical, should knit? `in.shiny` A logical, is the map drawn in **shiny**? `proxy` A logical, if `in.shiny`, is `tmapProxy()` used? `options` A vector of options
asp, scale	the desired aspect ratio and scale of the map. Only applicable for `"plot"` mode.

Value

- tmap_grob() returns a grob object ("plot" mode)
- tmap_leaflet() a leaflet object ("view" mode). In case small multiples are shown, a list is returned.

Examples

```
map = tm_shape(World) + tm_polygons()
tmap_leaflet(map, show = TRUE)
```

---

tmap_mode *Set tmap mode*

Description

- tmap_mode() gets (no argument) or sets the current mode.
- ttm() toggles between the two most recent modes.
- ttmp() same as ttm(), then calls tmap_last().
- rtm() rotates through all modes in the pool.
- rtmp() same as rtm(), then calls tmap_last().
- tmap_mode_pool() restricts which modes are cycled by ttm() and rtm(). Call without arguments to inspect the current pool, or pass NULL to reset.

It is recommended to use tmap_mode() in scripts and ttm()/ttmp() in the console.

Usage

```
tmap_mode(mode = NULL, silent = FALSE)

ttm()

rtm()

tmap_mode_pool(modes = NULL, silent = FALSE)

ttmp()

rtmp()
```

Arguments

mode	A string specifying the mode. See tmap_options() for available modes.
silent	Should the mode be switched silently? Default FALSE.
modes	Character vector of mode names (minimum 2), or NULL to reset.

Details

The default modes are "plot" (static, graphics device) and "view" (interactive, browser or RStudio Viewer). Additional modes such as "maplibre" and "mapbox" become available when **tmap.mapgl** is loaded.

Value

- tmap_mode() returns the current mode invisibly when called without argument, otherwise the previous mode.
- ttm(), rtm() return the previous mode invisibly.
- tmap_mode_pool() returns the previous pool invisibly.

References

Tennekes, M., 2018, tmap: Thematic Maps in R, Journal of Statistical Software, 84(6), 1-39, doi:10.18637/jss.v084.i06

See Also

vignette about modes

- tmap_last() to show the last map
- tm_view() for viewing options
- tmap_leaflet() for obtaining a leaflet widget
- tmap_options() for tmap options

Examples

```
current.mode = tmap_mode()

tmap_mode("plot")

tm_shape(World) + tm_polygons("HPI")

tmap_mode("view")

tm_shape(World) + tm_polygons("HPI")

ttm()

tm_shape(World) + tm_polygons("HPI")

tmap_mode(current.mode)
```

tmap_overview *Overview of tmap layers*

Description

Overview of tmap layers, organized by layer type.

Usage

```
tmap_overview()
```

Value

A list of three layer types are returned: data layers, aux layers, and components.

tmap_save *Save tmap*

Description

Save tmap to a file. This can be either a static plot (e.g. png) or an interactive map (html).

Usage

```
tmap_save(
  tm = NULL,
  filename = NA,
  device = NULL,
  width = NA,
  height = NA,
  units = NA,
  dpi = NA,
  outer.margins = NA,
  asp = NULL,
  scale = NA,
  insets_tm = NULL,
  insets_vp = NULL,
  add.titles = TRUE,
  in.iframe = FALSE,
  selfcontained = !in.iframe,
  verbose = NULL,
  ...
)
```

Arguments

tm	tmap object
filename	filename including extension, and optionally the path. The extensions pdf, eps, svg, wmf (Windows only), png, jpg, bmp, tiff, and html are supported. If the extension is missing, the file will be saved as a static plot in "plot" mode and as an interactive map (html) in another mode. The default format for static plots is png, but this can be changed using the option "output.format" in tmap_options(). If NA (the default), the file is saved as "tmap01" in the default format, and the number incremented if the file already exists.
device	graphic device to use. Either a device function (e.g., png or cairo_pdf) or a text indicating selected graphic device: "pdf", "eps", "svg", "wmf" (Windows only), "png", "jpg", "bmp", "tiff". If NULL, the graphic device is guessed based on the filename argument.
height, width	The dimensions of the plot (not applicable for html files). Units are set with the argument units. If one of them is not specified, this is calculated using the formula asp = width / height, where asp is the estimated aspect ratio of the map. If both are missing, they are set such that width * height is equal to the option "output.size" in tmap_options(). This is by default 49, meaning that is the map is a square (so aspect ratio of 1) both width and height are set to 7.
units	units for width and height ("in", "cm", or "mm"). By default, pixels ("px") are used if either width or height is set to a value greater than 50. Else, the units are inches ("in").
dpi	dots per inch. Only applicable for raster graphics. By default it is set to 300, but this can be changed using the option "output.dpi" in tmap_options().
outer.margins	overrides the outer.margins argument of tm_options() (unless set to NA)
asp	if specified, it overrides the asp argument of tm_options(). **Tip**: set to 0 if map frame should be placed on the edges of the image.
scale	overrides the scale argument of tm_options() (unless set to NA)
insets_tm	tmap object of an inset map, or a list of tmap objects of multiple inset maps. The number of tmap objects should be equal to the number of viewports specified with insets_vp.
insets_vp	viewport of an inset map, or a list of viewports of multiple inset maps. The number of viewports should be equal to the number of tmap objects specified with insets_tm.
add.titles	add titles to leaflet object.
in.iframe	should an interactive map be saved as an iframe? If so, two HTML files will be saved; one small parent HTML file with the iframe container, and one large child HTML file with the actual widget. See widgetframe::saveWidgetframe() for details. By default FALSE, which means that one large HTML file is saved (see saveWidget()).
selfcontained	when an interactive map is saved, should the resources (e.g. JavaScript libraries) be contained in the HTML file? If FALSE, they are placed in an adjacent directory (see also htmlwidgets::saveWidget()). Note that the HTML file will often still be large when selfcontained = FALSE, since the map data (polygons and

popups), which are also contained in the HTML file, usually take more space then the map resources.

verbose Deprecated. It is now controlled by the tmap option show.messages (see tmap_options())

... Arguments passed on to htmlwidgets::saveWidget, widgetframe::saveWidgetframe

widget Widget to save

file File to save HTML into

libdir Directory to copy HTML dependencies into (defaults to filename_files).

background Text string giving the html background color of the widget. Defaults to white.

title Text to use as the title of the generated page.

knitrOptions A list of **knitr** chunk options.

Value

the filename, invisibly, if export is successful.

Examples

```
## Not run:
 data(NLD_muni, NLD_prov)
 m <- tm_shape(NLD_muni) +
       tm_fill(col="population", convert2density=TRUE,
                 style="kmeans",
                 title=expression("Population (per " * km^2 * ")")) +
       tm_borders("black", alpha=.5) +
   tm_shape(NLD_prov) +
       tm_borders("grey25", lwd=2) +
  tm_style("classic") +
  tm_format("NLD", inner.margins = c(.02, .15, .06, .15)) +
     tm_scalebar(position = c("left", "bottom")) +
     tm_compass(position=c("right", "bottom"))

 tmap_save(m, "choropleth.png", height = 7) # height interpreted in inches
 tmap_save(m, "choropleth_icon.png", height = 100, scale = .1) # height interpreted in pixels

 data(World)
 m2 <- tm_shape(World) +
  tm_fill("well_being", id="name", title="Well-being") +
  tm_format("World")

 # save image
 tmap_save(m2, "World_map.png", width=1920, height=1080, asp=0)

 # cut left inner margin to make sure Antarctica is snapped to frame
 tmap_save(m2 + tm_layout(inner.margins = c(0, -.1, 0.05, 0.01)),
        "World_map2.png", width=1920, height=1080, asp=0)

 # save interactive plot
 tmap_save(m2, "World_map.html")

## End(Not run)
```

`tmap_style` *Set or get the default tmap style*

Description

Set or get the default tmap style. Without arguments, the current style is returned. Also the available styles are displayed. When a style is set, the corresponding tmap options (see `tmap_options()`) will be set accordingly. The default style (i.e. when loading the package) is `"white"`.

Usage

```
tmap_style(style)
```

Arguments

`style`	Name of the style. When omitted, `tmap_style()` returns the current style and also shows all available styles. When the style is specified,`tmap_style()` sets the style accordingly. Note that in that case, all tmap options (see `tmap_options()`) will be reset according to the style definition. See `tm_layout()` for predefined styles, and `tmap_style_catalogue()` for creating a catalogue.

Details

Note that `tm_style()` is used within a plot call (so it only affects that plot), whereas `tmap_style()` sets the style globally.

After loading a style, the options that defined this style (i.e. the difference with the default `"white"` style) can be obtained by `tmap_options_diff()`.

The documentation of `tmap_options()` (details and the examples) shows how to create a new style.

Value

The style before changing

See Also

- `tmap_options()` for tmap options
- `tmap_style_catalogue()` to create a style catalogue of all available styles.

Examples

```
tmap_style()

tm_shape(World) + tm_polygons("HPI")

tmap_style("cobalt")

tm_shape(World) + tm_polygons("HPI")
```

```
# for backwards compatibility, the styles of tmap versions 1-3 are also included:

tmap_style("v3")

tm_shape(World) + tm_polygons("HPI")

tmap_style("cobalt_v3")

tm_shape(World) + tm_polygons("HPI")
```

---

tmap_style_catalogue *Create a style catalogue*

Description

Create a style catalogue for each predefined tmap style. The result is a set of png images, one for each style.

Usage

```
tmap_style_catalogue(path = "./tmap_style_previews", styles = NA)

tmap_style_catalog(path = "./tmap_style_previews", styles = NA)
```

Arguments

path	path where the png images are stored
styles	vector of styles function names (see tmap_style()) for which a preview is generated. By default, a preview is generated for all loaded styles.

---

tmap_tip *Print a random tip to the console*

Description

Print a random tip to the console

Usage

```
tmap_tip()
```

Value

A message

`World` *World dataset*

Description

World dataset, class `sf`

Usage

```
World
```

Details

Variable	Source	Description
`iso_a3`	NED	ISO 3166-1 alpha-3 three-letter country code (see below)
`name`	NED	Country name
`sovereignt`	NED	Sovereignt country name
`continent`	NED	Continent (primary; some countries are transcontinental)
`area`	NED	Area in km2
`pop_est`	NED	Population estimation
`pop_est_dens`	NED	Population estimation per km2
`economy`	NED	Economy class
`income_grp`	NED	Income group
`gdp_cap_est`	NED	GDP per capita (estimated)
`life_exp`	HPI	Life expectancy. The average number of years an infant born in that country is expected to l
`well_being`	HPI	Well being. Self-reported from 0 (worst) to 10 (best)
`footprint`	HPI	Carbon footprint. Per capita greendwelling gas emissions associated with consumption and
`HPI`	HPI	Happy Planet Indicator. An index of human well-being and environmental impact that was
`inequality`	WB	Income inequality: Gini coefficient (World Bank variable SI.POV.GINI) A value of 0 repres
`gender`	UNDP/OWiD	Gender Inequality Index (GII) Composite metric using reproductive health, empowerment a
`press`	RSF	World Press Freedom Index. Degree of freedom that journalists, news organizations and ne

See sources for more detailed information about the variables.

This dataset, created Noveber 2024, is an update from the old version, which has been created around 2016. All variables from the old version are included, but updated. Furthermore, gender ineuqlity and press freedom have been added.

ISO country-code: two countries have user-assigned codes, namely: XKX is used for Kosovo (conform European Union and World Bank) (was UNK in the old version); XNC is used for Northern Cyprus (was CYN in the old version).

For some variables data were available from multiple years, but availability was different across countries. In those cases, the most recent values were taken.

Source

NED: Natural Earth Data https://www.naturalearthdata.com/

HPI: Happy Planet Index https://happyplanetindex.org/

UNDP: Human Development Report (2024) https://hdr.undp.org/content/human-development-report-2023-24

WB: World Bank https://data.worldbank.org

OWiD: Our World in Data https://ourworldindata.org

RSF: Reporters Without Borders https://rsf.org/en/index

World_rivers *Spatial data of rivers*

Description

Spatial data of rivers

Usage

```
World_rivers
```

Format

An object of class `sf` (inherits from `data.frame`) with 1632 rows and 5 columns.

Note

In tmap <= 3, this dataset was called `rivers`.

Source

https://www.naturalearthdata.com

Index