Package 'tmap'

February 13, 2018

License GPL-3

Title Thematic Maps

Type Package

LazyLoad yes

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.

Version 1.11-1

Date 2018-02-13

Depends R (>= 3.0), methods

Imports tmaptools (>= 1.2-1), sp, raster (>= 2.5-2), grid, rgdal, rgeos, RColorBrewer, classInt, spdep, htmltools, htmlwidgets, leaflet (>= 1.1.0), mapview, stats

Suggests rmarkdown, knitr, png, cartogram, ggplot2, dplyr, tidyr

URL https://github.com/mtennekes/tmap

BugReports https://github.com/mtennekes/tmap/issues

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Martijn Tennekes [aut, cre],
Joel Gombin [ctb],
Sebastian Jeworutzki [ctb],
Kent Russell [ctb],
Richard Zijdeman [ctb],
John Clouse [ctb]

Maintainer Martijn Tennekes <mtennekes@gmail.com>

Repository CRAN

Date/Publication 2018-02-13 13:19:03 UTC

R topics documented:

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.

Details

This page provides a brief overview of all package functions. See `vignette("tmap-nutshell")` for a short introduction with examples. See `vignette("tmap-modes")` for a short demo of the two output modes: plot and interactive view.

Quick plotting method

qtm	Plot a thematic map

Main plotting method

Shape specification:

tm_shape	Specify a shape object

Aesthetics base layers:

tm_polygons	Create a polygon layer (with borders)
tm_symbols	Create a layer of symbols
tm_lines	Create a layer of lines
tm_raster	Create a raster layer
tm_text	Create a layer of text labels

Aesthetics derived layers:

tm_fill	Create a polygon layer (without borders)
tm_borders	Create polygon borders
tm_bubbles	Create a layer of bubbles

tm_squares	Create a layer of squares
tm_dots	Create a layer of dots
tm_markers	Create a layer of markers
tm_iso	Create a iso/contour lines

Facetting (small multiples)

tm_facets	Define facets

Attributes:

tm_grid	Create grid lines
tm_scale_bar	Create a scale bar
tm_compass	Create a map compass
tm_credits	Create a text for credits
tm_logo	Create a logo
tm_xlab and tm_ylab	Create axis labels

Layout element:

tm_layout	Adjust the layout (main function)
tm_legend	Adjust the legend
tm_view	Configure the interactive view mode

Change options:

tmap_mode	Set the tmap mode: "plot" or "view"
ttm	Toggle between the modes
tmap_style	Set the default style

Create icons:

tmap_icons	Specify icons for markers or proportional symbols

Output functions

print	Plot in graphics device or view interactively in web browser or RStudio's viewer pane
last_map	Redraw the last map
tmap_leaflet	Obtain a leaflet widget object
animation_tmap	Create an animation
save_tmap	Save thematic maps (both in plot and view mode)

Spatial datasets

World	World country data (spatial polygons)
Europe	European country data (spatial polygons)
NLD_prov	Netherlands province data (spatial polygons)
NLD_muni	Netherlands municipal data (spatial polygons)
metro	Metropolitan araes (spatial points)
rivers	Rivers (spatial lines)
land	Global land cover (spatial grid)

Author(s)

Martijn Tennekes <mtennekes@gmail.com>

See Also

```
vignette("tmap-nutshell")
vignette("tmap-modes")
```

+.tmap *Stacking of tmap elements*

Description

The plus operator allows you to stack tmap-elements, and groups of tmap-elements.

Usage

```
## S3 method for class 'tmap'
e1 + e2
```

Arguments

e1	first tmap-element
e2	second tmap-element

See Also

tmap-element and vignette("tmap-nutshell")

---

animation_tmap *Create animation*

Description

Create a gif or mpeg animation from a tmap plot. The free tool ImageMagick is required.

Usage

```
animation_tmap(tm, filename = "animation.gif", width = NA, height = NA,
  delay = 40)
```

Arguments

tm	tmap object. In order to create a series of tmap plots, which will be the frames of the animation, it is important to set nrow and ncol in tm_facets, for otherwise a small multiples plot is generated. Commonly, where one map is shown at a time, both nrow and ncol are set to 1.
filename	filename of the video (should be a .gif or .mpg file)
width	width of the animation file (in pixels)
height	height of the animation file (in pixels)
delay	delay time between images (in 1/100th of a second)

Note

Not only tmap plots are supported, but any series of R plots.

Examples

```
## Not run:
data(World, metro, Europe)

m1 <- tm_shape(Europe) +
      tm_polygons("yellow") +
  tm_facets(along = "name")

animation_tmap(m1, filename="European countries.gif", width=800, delay=40)

m2 <- tm_shape(World) +
```

```
            tm_polygons() +
        tm_shape(metro) +
            tm_bubbles(paste0("pop", seq(1970, 2030, by=10)),
                border.col = "black", border.alpha = .5) +
        tm_facets(free.scales.symbol.size = FALSE, nrow=1,ncol=1) +
        tm_format_World(scale=.5)

animation_tmap(m2, filename="World population.gif", width=1200, delay=100)

## End(Not run)
```

---

land *Spatial data of global land cover*

Description

Spatial data of global land cover, of class SpatialGridDataFrame. The data includes a population times series from 1950 to (forecasted) 2030. All metro areas with over 1 million inhabitants in 2010 are included.

Usage

```
data(land)
```

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. See http://www.iscgm.org/gm/glcnmo.html#use.

Source

http://www.iscgm.org/gm/glcnmo.html

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).

last_map *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's last_plot, although there is a difference: last_map returns the last call instead of the stacked tmap-elements.

Usage

```
last_map()
```

Value

call

See Also

save_tmap

metro *Spatial data of metropolitan areas*

Description

Spatial data of metropolitan areas, of class SpatialPointsDataFrame. The data includes a population times series from 1950 to (forecasted) 2030. All metro areas with over 1 million inhabitants in 2010 are included.

Usage

```
data(metro)
```

Source

http://esa.un.org/unpd/wup/CD-ROM/

References

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

print.tmap *Draw thematic map*

Description

Draw thematic map. If the tmap mode is set to "plot" (see tmap_mode), the map is plot in the current graphics device. If the mode is set to "view", the map is shown interactively as an htmlwidget.

Usage

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

knit_print.tmap(x, ..., options = NULL)
```

Arguments

x	tmap object. A tmap object is created with qtm or by stacking tmap-elements.
vp	viewport to draw the plot in. This is particularly useful for insets.
return.asp	Logical that determines whether the aspect ratio of the map is returned. In that case, grid.newpage() will be called, but without plotting of the map. This is used by save_tmap to determine the aspect ratio of the map.
mode	the mode of tmap: "plot" (static) or "view" (interactive). See tmap_mode for details.
show	logical that determines whether to show to map. Obviously TRUE by default, but show=FALSE can be useful for just obtaining the returned objects.
knit	should knit_print be enabled, or the normal print function?
options	options passed on to knitprint
...	not used

Value

If mode=="plot", then a list is returned with the processed shapes and the metadata. If mode=="view", a leaflet object is returned (see also tmap_leaflet)

qtm *Quick thematic map plot*

Description

Draw a thematic map quickly.

Usage

```
qtm(shp, fill = NA, symbols.size = NULL, symbols.col = NULL,
  symbols.shape = NULL, dots.col = NULL, text = NULL, text.size = 1,
  text.col = NA, lines.lwd = NULL, lines.col = NULL, raster = NA,
  borders = NA, by = NULL, scale = NA, title = NA, projection = NULL,
  format = NULL, style = NULL, basemaps = NA, bubble.size = NULL,
  bubble.col = NULL, ...)
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame) 5. SpatialPixels(DataFrame) 6. RasterLayer, RasterStack, or RasterBrick In "view" mode (see tmap_mode) there are two other options. 1) If omitted, an interactive map without thematic layers is opened. 2) In addition, if a character is provided, this character is used as a search query for OpenStreetMap nominatim. This will position the interactive map accordingly. Arguments of tm_view, such as set.view can be passed on directly.tm_
fill	either a color to fill the polygons, or name of the data variable in shp to draw a choropleth. Only applicable when shp is type 1 (see above). Set fill=NULL to draw only polygon borders. See also argument borders.
symbols.size	either the size of the symbols or a name of the data variable in shp that specifies the sizes of the symbols. See also the size arugment of tm_symbols. Only applicable when shp is type 1, 2, or 3 (see above).
symbols.col	either the color of the symbols or a name of the data variable in shp that specifies the colors of the symbols. See also the col arugment of tm_symbols. Only applicable when shp is type 1, 2, or 3 (see above).
symbols.shape	either the shape of the symbols or a name of the data variable in shp that specifies the shapes of the symbols. See also the shape arugment of tm_symbols. Only applicable when shp is type 1, 2, or 3 (see above).
dots.col	name of the data variable in shp for the dot map that specifies the colors of the dots. If dots.col is specified instead symbols.col, dots instead of bubbles are drawn (unless symbols.shape is specified).

text	Name of the data variable that contains the text labels. Only applicable when shp is type 1, 2, or 3 (see above).
text.size	Font size of the text labels. Either a constant value, or the name of a numeric data variable. Only applicable when shp is type 1, 2, or 3 (see above).
text.col	name of the data variable in shp for the that specifies the colors of the text labels. Only applicable when shp is type 1, 2, or 3 (see above).
lines.lwd	either a line width or a name of the data variable that specifies the line width. Only applicable when shp is type 3 (see above).
lines.col	either a line color or a name of the data variable that specifies the line colors. Only applicable when shp is type 3 (see above).
raster	either a color or a name of the data variable that specifices the raster colors. Only applicable when shp is type 4, 5, or 6 (see above).
borders	color of the polygon borders. Use NULL to omit the borders.
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, sy mbol sizes, border widths, and line widths are controled by this value. The parameters symbols.size, text.size, and lines.lwd can be scaled seperately with respectively symbols.scale, text.scale, and lines.scale. See also ....
title	main title. For legend titles, use X.style, where X is the layer name (see ...).
projection	Either a CRS object or a character value. If it is a character, it can either be a PROJ.4 character string or a shortcut. See get_proj4 for a list of shortcut values. By default, the projection is used that is defined in the shp object itself, which can be obtained with get_projection.
format	tm_layout wrapper used for format. Currently available in tmap: "World", "Europe", "NLD", "World_wide", "Europe_wide", "NLD_wide". Own wrappers can be used as well (see details).
style	tm_layout wrapper used for style. Available in tmap: "bw", "classic". Own wrappers can be used as well (see details).
basemaps	basemaps for the view mode. See tm_view
bubble.size	deprecated. Please use symbols.size.
bubble.col	deprecated. Please use symbols.col.
...	arguments passed on to the tm_* functions. The prefix of these arguments should be with the layer function name without "tm_" and a period. For instance, the palette for polygon fill color is called fill.palette. The following prefixes are supported: shape., fill., borders., polygons., symbols., dots., lines., raster., text., layout., grid., facets., and view.. Arguments that have a unique name, i.e. that does not exist in any other layer function, e.g. convert2density, can also be called without prefix.

Details

This function is a convenient wrapper of the main plotting method of stacking tmap-elements. 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 can be specified (see ...). It is also possible to stack tmap-elements on a qtm plot. See examples.

For format, any character value, say "xxx" can be used if the wrapper function "tm_format_xxx" exists. The same applies for the argument style.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, rivers, metro)

# just the map
qtm(World)

# choropleth
qtm(World, fill = "economy", format="World", style="col_blind")

qtm(World, fill="HPI", fill.n=9, fill.palette="div", fill.auto.palette.mapping=FALSE,
fill.title="Happy Planet Index", fill.id="name", format="World", style="gray")

# bubble map
## Not run:
qtm(World, borders = NULL) +
qtm(metro, symbols.size = "pop2010",
    symbols.title.size= "Metropolitan Areas",
    symbols.id= "name",
    format = "World")

## End(Not run)

# TIP: check out these examples in view mode, enabled with tmap_mode("view")
```

---

rivers *Spatial data of rivers*

Description

Spatial data of rivers, of class SpatialLinesDataFrame

Usage

```
data(rivers)
```

Source

http://www.naturalearthdata.com

---

save_tmap *Save tmap*

Description

Save tmap to a file, such as png, jpg, or pdf.

Usage

```
save_tmap(tm = NULL, filename = NULL, width = NA, height = NA,
  units = NA, dpi = 300, outer.margins = NA, asp = NULL, scale = NA,
  insets_tm = NULL, insets_vp = NULL, verbose = TRUE, ...)
```

Arguments

tm	tmap object
filename	filename including extension, and optionally the path. The extensions pdf, eps, svg, wmf (Windows only), png, jpg, bmp, or tiff are supported. If the extension is missing, the file will be saved as png image
width	width. Units are set with the argument units. If set to NA and height is specified, it will be height * aspect ratio. If both width and height are not specified, then the width of the current plotting window will be taken.
height	height. Units are set with the argument units. If set to NA and width is specified, it will be width / aspect ratio. If both width and height are not specified, then the height of the current plotting window will be taken.
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.
outer.margins	overrides the outer.margins argument of tm_layout (unless set to NA)
asp	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	overrides the scale argument of tm_layout (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.
verbose	should information messages be returned?
...	arguments passed on to device functions or to saveWidget

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 * ")"),
                 legend.hist=FALSE) +
     tm_borders("black", alpha=.5) +
 tm_shape(NLD_prov) +
     tm_borders("grey25", lwd=2) +
     tm_format_NLD(inner.margins = c(.02, .15, .06, .15)) +
     tm_scale_bar(position = c("left", "bottom")) +
     tm_compass(position=c("right", "bottom")) +
     tm_style_classic()

save_tmap(m, "choropleth.png", height=7)

data(World)
m2 <- tm_shape(World) +
tm_fill("well_being", id="name", title="Well-being") +
tm_format_World()

# save image
save_tmap(m2, "World_map.png", width=1920, height=1080, asp=0)

# cut left inner margin to make sure Antarctica is snapped to frame
save_tmap(m2 + tm_layout(inner.margins = c(0, -.1, 0.05, 0.01)),
        "World_map2.png", width=1920, height=1080, asp=0)

# save interactive plot
save_tmap(m2, "World_map.html")

## End(Not run)
```

---

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

```
style_catalogue(path = "./tmap_style_previews", styles = NA,
  include.global.styles = TRUE)

style_catalog(path = "./tmap_style_previews", styles = NA,
  include.global.styles = TRUE)
```

Arguments

path	path where the png images are stored
styles	vector of tm_style_XXX function names. By default, it contains all styles that are included in tmap, and, if include.global.styles=TRUE, also the user defined tm_style_XXX function names.
include.global.styles	See above

---

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?

tmap-element *tmap element*

Description

Building block for drawing thematic maps. All element functions have the prefix `tm_`.

Details

The fundamental, and hence required element is

- `tm_shape` that specifies the shape object, and also specifies the projection and bounding box

The elements that serve as aesthetics layers are

- `tm_fill` to fill the polygons
- `tm_borders` to draw polygon borders
- `tm_polygons` to draw polygons (it is a combination of `tm_fill` and `tm_borders`)
- `tm_bubbles` to draw bubbles
- `tm_lines` to draw lines
- `tm_text` to add text annotations
- `tm_raster` to draw a raster

The layers can be stacked by simply adding them with the + symbol. The combination of the elements described above form one group. Multiple groups can be stacked. Each group should start with `tm_shape`.

The attribute elements are

- `tm_grid` to specify coordinate grid lines
- `tm_credits` to add a credits/acknowledgements text label
- `tm_scale_bar` to add a measurement scale bar
- `tm_compass` to add a map compass

The element `tm_facets` specifies facets (small multiples). The element `tm_layout` is used to change the layout of the map.

See Also

```
vignette("tmap-nutshell")
```

```
vignette("tmap-modes")
```

The examples in each of the element functions

tmap_arrange *Arrange small multiples in grid layout*

Description

Arrange small multilples 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, sync = FALSE, asp = 0,
  outer.margins = 0.02)
```

Arguments

...	tmap objects. The number of multiples that can be plot is limited (see details).
ncol	number of columns
nrow	number of rows
sync	logical. Should the navigation in view mode (zooming and panning) be synchronized? By default FALSE.
asp	aspect ratio. If will overwrite the asp argument from tm_layout, unless set to NULL
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.

Details

The global option tmap.limits controlls 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

```
data(World)
w1 <- qtm(World, projection = "eck4", title="Eckert IV")
w2 <- qtm(World, projection = "merc", title="Mercator")
w3 <- qtm(World, projection = "wintri", title="Winkel-Tripel")
w4 <- qtm(World, projection = "robin", title="Robinsin")

current.mode <- tmap_mode("plot")
tmap_arrange(w1, w2, w3, w4)
tmap_mode(current.mode)
```

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, width = 48, height = 48, keep.asp = TRUE,
  just = c("center", "center"), as.local = TRUE, ...)

marker_icon()
```

Arguments

file	character value/vector containing the file path(s) or url(s).
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").
as.local	if the file is a url, should it be saved to local temporary file?
...	arguments passed on to icons. When iconWidth, iconHeight, iconAnchorX and iconAnchorY are specified, they override width and height, and just.

Value

icon data (see icons)

See Also

tm_symbols

tmap_leaflet *Create a leaflet widget from a tmap object*

Description

Create a leaflet widget from a tmap object. An interactive map (see tmap_mode) is an automatically generated leaflet widget. With this function, this leaflet widget is obtained, which can then be changed or extended by using leaflet's own methods.

Usage

```
tmap_leaflet(x, mode = "view", show = FALSE, ...)
```

Arguments

x	tmap object. A tmap object is created with qtm or by stacking tmap-elements.
mode	the mode of tmap, which is set to "view" in order to obtain the leaflet object. See tmap_mode for details.
show	should the leaflet map be shown? FALSE by default
...	arguments passed on to print.tmap

Value

leaflet object

See Also

tmap_mode, tm_view, print.tmap

Examples

```
# world choropleth/bubble map of the world
data(World, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

map1 <- tm_shape(metro) +
tm_bubbles("pop2010", col = "growth",
   border.col = "black", border.alpha = .5,
   style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
   palette="-RdYlBu", contrast=1,
   title.size="Metro population",
   title.col="Growth rate (%)", id="name") +
tm_layout(legend.bg.color = "grey90", legend.bg.alpha=.5, legend.frame=TRUE)


lf <- tmap_leaflet(map1)

# show leaflet widget
```

```
lf

# add marker
require(leaflet)
lf %>% leaflet::addMarkers(2.2945, 48.8582, popup = "Eiffel tower")

## Not run:
# alternative
eiffelTower <- geocode_OSM("Eiffel Tower, Paris", as.SPDF = TRUE)

map1 +
tm_shape(eiffelTower) +
tm_markers()

## End(Not run)
```

---

tmap_mode *Set tmap mode to static plotting or interactive viewing*

Description

Set tmap mode to static plotting or interactive viewing. The global option `tmap.mode` determines the whether thematic maps are plot in the graphics device, or shown as an interactive leaflet map (see also `tmap_options`. The function `tmap_mode` is a wrapper to set this global option. The convenient function `ttm` is a toggle switch between the two modes. Tip 1: use `tmap_mode` in scripts and `ttm` in the console. Tip 2: use `ttm` in combination with `last_map` to redraw the last map in the other mode.

Usage

```
tmap_mode(mode = c("plot", "view"))

ttm()
```

Arguments

mode	one of
	`"plot"` Thematic maps are shown in the graphics device. This is the default mode, and supports all tmap's features, such as small multiples (see `tm_facets`) and extensive layout settings (see `tm_layout`). It is recommended for saving static maps (see `save_tmap`).
	`"view"` Thematic maps are viewed interactively in the web browser or RStudio's Viewer pane. Maps are fully interactive with tiles from OpenStreetMap or other map providers. See `tm_view` for options related to the `"view"` mode. This mode generates a `leaflet` widget, which can also be directly obtained with `tmap_leaflet`. With RMarkdown, it is possible to publish it to an HTML page. There are a couple of contraints in comparison to `"plot"`:

- The map is always projected accoring to the Web Mercator projection. Although this projection is the de facto standard for interactive web-based mapping, it lacks the equal-area property, which is important for many thematic maps, especially choropleths (see examples from tm_shape).
- Small multiples are not supported
- The legend cannot be made for aesthetics regarding size, which are symbol size and line width.
- Text labels are not supported (yet)
- The layout options set with tm_layout) regarding map format are not used. However, the styling options still apply.

Value

the mode before changing

See Also

vignette("tmap-modes"), last_map to show the last map, tm_view for viewing options, and tmap_leaflet for obtaining a leaflet widget, and tmap_options for tmap options.

Examples

```
# world choropleth/bubble map of the world
data(World, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

map1 <- tm_shape(World) +
tm_polygons("income_grp", palette="-Blues", contrast=.7, id="name", title="Income group") +
tm_shape(metro) +
tm_bubbles("pop2010", col = "growth",
border.col = "black", border.alpha = .5,
style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
palette="-RdYlBu", contrast=1,
title.size="Metro population",
title.col="Growth rate (%)", id="name",
popup.vars = c("pop2010", "pop2020", "growth")) +
tm_layout(legend.bg.color = "grey90", legend.bg.alpha=.5, legend.frame=TRUE)

# initial mode: "plot"
current.mode <- tmap_mode("plot")

# plot map
map1

# switch to other mode: "view"
ttm()

# view map
map1
```

```
## Not run:
# choropleth of the Dutch population in interactive mode:
require(tmaptools)
data(NLD_muni, NLD_prov)
NLD_muni$pop_dens <- calc_densities(NLD_muni, var = "population")

tm_shape(NLD_muni) +
tm_fill(col="pop_dens",
style="kmeans",
title = "Population (per km^2)", id = "name") +
tm_borders("grey25", alpha=.5) +
tm_shape(NLD_prov) +
tm_borders("grey40", lwd=2)

## End(Not run)

# restore current mode
tmap_mode(current.mode)
```

`tmap_options` *Options for tmap*

Description

Get or set global options for tmap. The behaviour is similar to `options`: all tmap options are retrieved when this function is called without arguments. When arguments are specified, the corresponding options are set, and the old values are silently returned.

Usage

```
tmap_options(...)
```

Arguments

`...`	tmap options, using name = value. See below for the available tmap options. Alternatively, a named list can be provided.

Details

The following tmap options exist:

tmap.unit This is the default value for the `unit` argument of `tm_shape`. It specifies the unit of measurement, which is used in the scale bar and the calculation of density values. By default (when loading the package), it is `"metric"`. Other valid values are `"imperial"`, `"km"`, `"m"`, `"mi"`, and `"ft"`.

tmap.style This option determines the current style. See `tmap_style` for details.

tmap.mode This options determines the current mode. See `tmap_mode` for details.

tmap.limits This option determines how many facets (small multiples) are allowed for per mode. It should be a vector of two numeric values named `facets.view` and `facets.plot`. By default (i.e. when loading the package), it is set to `c(facets.view = 4, facets.plot = 64)`

See Also

tmap_mode, tmap_style

Examples

```
# save current options
current_options <- tmap_options()

# show current options
tmap_options()

# switch to other view
ttm()

# show current options
tmap_options()

# set style to cobalt
tmap_options(tmap.style = "cobalt")

# show current options
tmap_options()

# set style usign tmap_style
tmap_style("classic")

# show current options
tmap_options()

# set unit to imperial
tmap_options(tmap.unit = "imperial", tmap.limits = c(facets.view = 8, facets.plot = 16))

# show current options
tmap_options()

# restore options
tmap_options(current_options)
```

---

tmap_style *Set the default tmap style*

Description

Set the default tmap style, which is contained in the global option `tmap.style` (see also tmap_options. The default style (i.e. when loading the package) is `"white"`.

Usage

```
tmap_style(style)
```

Arguments

style	name of the style. The function tm_style_<style> should exist and be a wrapper of tm_layout. The default style when loading the package is "white", which corresponds to the function tm_style_white. See tm_layout for predefined styles, and style_catalogue for creating a catelogue.

Value

the style before changing

See Also

tm_layout for predefined styles, style_catalogue to create a style catelogue of all available styles, and tmap_options for tmap options.

tmap_options for tmap options

Examples

```
data(World)

current.style <- tmap_style("classic")
qtm(World, fill="life_exp", fill.title="Life expectancy")

tmap_style("cobalt")
qtm(World, fill="life_exp", fill.title="Life expectancy")

# restore current style
tmap_style(current.style)
```

---

tm_add_legend *Add manual legend*

Description

Creates a tmap-element that adds a manual legend.

Usage

```
tm_add_legend(type = c("fill", "symbol", "text", "line"), labels = NULL,
  col = NULL, size = NULL, shape = NULL, lwd = NULL, lty = NULL,
  text = NULL, alpha = NA, border.col = "black", border.lwd = 1,
  border.alpha = NA, title = "", is.portrait = TRUE,
  legend.format = list(), reverse = FALSE, z = NA)
```

Arguments

type	type of legend. One of "fill", "symbol", "text", or "line"
labels	legend labels
col	legend colors
size	legend symbol sizes (if type=="symbol")
shape	legend symbol shapes (if type=="symbol")
lwd	legend line widths (if type=="line")
lty	legend line types (if type=="line")
text	legend texts (if type=="text")
alpha	legend fill transparancy
border.col	legend border col (if type is "fill" or "symbol")
border.lwd	legend border width (if type is "fill" or "symbol")
border.alpha	legend border alpha (if type is "fill" or "symbol")
title	legend title
is.portrait	is legend portrait (TRUE) or landscape (FALSE)?
legend.format	options to format the legend, see tm_layout
reverse	are the legend items reversed (by default FALSE)?
z	legend stack position

See Also

tm_symbols for an example

---

tm_compass *Map compass*

Description

Creates a map compass.

Usage

```
tm_compass(north = 0, type = NA, fontsize = 0.8, size = NA,
  show.labels = 1, cardinal.directions = c("N", "E", "S", "W"),
  text.color = NA, color.dark = NA, color.light = NA, lwd = 1,
  position = NA, just = NA)
```

Arguments

north	north direction in degrees: 0 means up, 90 right, etc.
type	compass type, one of: "arrow", "4star", "8star", "radar", "rose". The default is controlled by tm_layout (which uses "arrow" for the default style)
fontsize	relative font size
size	size of the compass in number of text lines. The default values depend on the type: for "arrow" it is 2, for "4star" and "8star" it is 4, and for "radar" and "rose" it is 6.
show.labels	number that specifies which labels are shown: 0 means no labels, 1 (default) means only north, 2 means all four cardinal directions, and 3 means the four cardinal directions and the four intercardinal directions (e.g. north-east).
cardinal.directions	labels that are used for the cardinal directions north, east, south, and west.
text.color	color of the text. By default equal to the argument attr.color of tm_layout.
color.dark	color of the dark parts of the compass, typically (and by default) black.
color.light	color of the light parts of the compass, typically (and by default) white.
lwd	line width of the compass
position	position of the compass. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the left bottom corner of the compass. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.
just	Justification of the attribute 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/bottom alignment and 1 right/top alignment. This option is only used, if position is specified by numeric coordinates. The default value is controlled by the argument "attr.just" of tm_layout.

Examples

```
current.mode <- tmap_mode("plot")

data(NLD_muni)

qtm(NLD_muni, theme = "NLD") + tm_compass()
qtm(NLD_muni, theme = "NLD") + tm_compass(type="radar", position=c("left", "top"), show.labels = 3)

# restore current mode
tmap_mode(current.mode)
```

tm_credits *Credits text*

Description

Creates a text annotation that could be used for credits or acknowledgements.

Usage

```
tm_credits(text, size = 0.7, col = NA, alpha = NA, align = "left",
  bg.color = NA, bg.alpha = NA, fontface = NA, fontfamily = NA,
  position = NA, just = NA)
```

Arguments

text	text. Multiple lines can be created with the line break symbol "\n". Facets can have different texts: in that case a vector of characters is required. Use "" to omit the credits for specific facets.
size	relative text size
col	color of the text. By default equal to the argument attr.color of tm_layout.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of col is used (normally 1).
align	horizontal alignment: "left" (default), "center", or "right". Only applicable if text contains multiple lines
bg.color	background color for the text
bg.alpha	Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the bg.color is used (normally 1).
fontface	font face of the text. By default, determined by the fontface argument of tm_layout.
fontfamily	font family of the text. By default, determined by the fontfamily argument of tm_layout.
position	position of the text. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the center of the text. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.
just	Justification of the attribute 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/bottom alignment and 1 right/top alignment. This option is only used, if position is specified by numeric coordinates. The default value is controlled by the argument "attr.just" of tm_layout.

See Also

tm_xlab

Examples

```
current.mode <- tmap_mode("plot")

data(NLD_muni, NLD_prov)

tm_shape(NLD_muni) +
    tm_fill(col="population", convert2density=TRUE,
        style="kmeans", title = expression("Population (per " * km^2 * ")")) +
    tm_borders("grey25", alpha=.5) +
    tm_shape(NLD_prov) +
    tm_borders("grey40", lwd=2) +
tm_format_NLD(bg.color="white", frame = TRUE) +
tm_credits("(c) Statistics Netherlands (CBS) and\nKadaster Nederland", position=c("left", "bottom"))

# restore current mode
tmap_mode(current.mode)
```

---

tm_facets *Small multiples*

Description

Creates a tmap-element that specifies facets (small multiples). Small multiples can be created in two ways: 1) by specifying the by argument with one or two variable names, by which the data is grouped, 2) by specifying multiple variable names in any of the aesthetic argument of the layer functions (for instance, the argument col in tm_fill). This function further specifies the facets, for instance number of rows and columns, and whether the coordinate and scales are fixed or free (i.e. independent of each other).

Usage

```
tm_facets(by = NULL, along = NULL, ncol = NA, nrow = NA,
  free.coords = TRUE, drop.units = free.coords, drop.empty.facets = TRUE,
  sync = !free.coords, showNA = NA, textNA = "Missing",
  free.scales = is.null(by) && is.null(along),
  free.scales.fill = free.scales, free.scales.symbol.size = free.scales,
  free.scales.symbol.col = free.scales,
  free.scales.symbol.shape = free.scales,
  free.scales.text.size = free.scales, free.scales.text.col = free.scales,
  free.scales.line.col = free.scales, free.scales.line.lwd = free.scales,
  free.scales.raster = free.scales, inside.original.bbox = FALSE,
  scale.factor = 2, drop.shapes = drop.units)
```

Arguments

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).
along	data variable name by which the data is split and plotted on separate pages. This is especially useful for animations made with animation_tmap. The along argument can be used in combination with the by argument. It is only supported in "plot" mode (so not in "view" mode).
ncol	number of columns of the small multiples grid. Not applicable if by contains two variable names.
nrow	number of rows of the small multiples grid. Not applicable if by contains two variable names.
free.coords	logical. If the by argument is specified, should each map has its own coordinate ranges?
drop.units	logical. If the by argument is specified, should non-selected spatial units be dropped? If FALSE (default), they are plotted where mapped aesthetics are regared as missing values. Not applicable for raster shapes.
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 colums are dropped.
sync	logical. Should the navigation in view mode (zooming and panning) be synchronized? By default TRUE, unless free.coords is set to TRUE.
showNA	If the by argument is specified, should missing values of the by-variable be shown in a facet? If two by-variables are specified, should missing values be shown in an additional row and column? If NA, missing values only are shown if they exist. Similar to the useNA argument of table, where TRUE, FALSE, and NA correspond to "always", "no", and "ifany" respectively.
textNA	text used for facets of missing values.
free.scales	logical. Should all scales of the plotted data variables be free, i.e. independent of each other? Possible data variables are color from tm_fill, color and size from tm_symbols and line color from tm_lines.
free.scales.fill	logical. Should the color scale for the choropleth be free?
free.scales.symbol.size	logical. Should the symbol size scale for the symbol map be free?
free.scales.symbol.col	logical. Should the color scale for the symbol map be free?
free.scales.symbol.shape	logical. Should the symbol shape scale for the symbol map be free?
free.scales.text.size	logical. Should the text size scale be free?
free.scales.text.col	logical. Should the text color scale be free?

`free.scales.line.col`
: Should the line color scale be free?

`free.scales.line.lwd`
: Should the line width scale be free?

`free.scales.raster`
: Should the color scale for raster layers be free?

`inside.original.bbox`
: If `free.coords`, should the bounding box of each small multiple be inside the original bounding box?

`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.factor`th 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`.

`drop.shapes`
: deprecated: renamed to `drop.units`

Details

The global option `tmap.limits` controlls the limit of the number of facets that are plotted. By default, `tmap_options(tmap.limits=c(facets.plot=64, facets.view=4))`. The maximum number of interactive facets is set to four since otherwise it may become very slow.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, NLD_muni, NLD_prov, land, metro)

current.mode <- tmap_mode("plot") # small multiples don't work in view mode

# Facets defined by constant values
tm_shape(World) +
    tm_fill(c("forestgreen", "goldenrod")) +
tm_format_World(title=c("A green world", "A dry world"), bg.color="lightskyblue2",
    title.position=c("left", "bottom"))

# Facets defined by multiple variables
tm_shape(Europe) +
    tm_polygons(c("well_being", "life_exp"),
     style=c("pretty", "fixed"), breaks=list(NULL, c(65,70,75,80,85)),
     palette=list("Oranges", "Purples"),
     border.col = "black",
        title=c("Well-Being Index", "Life Expectancy")) +
```

```
tm_format_Europe()

## Not run:
tm_shape(NLD_muni) +
    tm_fill(c("pop_0_14", "pop_15_24", "pop_25_44", "pop_45_64", "pop_65plus"),
        style="kmeans",
        palette=list("Oranges", "Greens", "Blues", "Purples", "Greys"),
        title=c("Population 0 to 14", "Population 15 to 24", "Population 25 to 44",
            "Population 45 to 64", "Population 65 and older")) +
tm_shape(NLD_prov) +
    tm_borders() +
tm_format_NLD(frame = TRUE, asp=0)

## End(Not run)

# Facets defined by groupings
tm_shape(NLD_prov) +
    tm_polygons("gold2") +
    tm_facets(by="name")

## Not run:
tm_shape(NLD_muni) +
    tm_borders() +
    tm_facets(by="province") +
    tm_fill("population", style="kmeans", convert2density = TRUE) +
tm_shape(NLD_prov) +
    tm_borders(lwd=4) +
    tm_facets(by="name")

tm_shape(land) +
    tm_raster("black") +
    tm_facets(by="cover_cls")

## End(Not run)

# Facets defined by groupings defined by two variables
## Not run:
World$HPI3 <- cut(World$HPI, breaks = c(20, 35, 50, 65),
    labels = c("HPI low", "HPI medium", "HPI high"))
World$GDP3 <- cut(World$gdp_cap_est, breaks = c(0, 5000, 20000, Inf),
    labels = c("GDP low", "GDP medium", "GDP high"))

tm_shape(World) +
tm_fill("HPI3", palette="Dark2", colorNA="grey90", legend.show = FALSE) +
tm_facets(c("HPI3", "GDP3"), showNA=FALSE, free.coords = FALSE)

metro$pop1950cat <- cut(metro$pop1950, breaks=c(0.5, 1, 1.5, 2, 3, 5, 10, 40)*1e6)
metro$pop2020cat <- cut(metro$pop2020, breaks=c(0.5, 1, 1.5, 2, 3, 5, 10, 40)*1e6)

tm_shape(World) +
tm_fill() +
tm_shape(metro) +
tm_dots("red", size = .5) +
```

```
tm_facets(c("pop1950cat", "pop2020cat"), free.coords = FALSE) +
tm_layout(panel.label.rot = c(0, 90), panel.label.size = 2)

## End(Not run)

# example: Meuse data
## Not run:
require(sp)
require(tmaptools)
data(meuse)
coordinates(meuse) <- ~x+y
proj4string(meuse) <- get_proj4("rd")

meuse_osm <- read_osm(meuse, ext=1.1)

qtm(meuse_osm) +
tm_shape(meuse) +
tm_bubbles(size=c("cadmium", "copper", "lead", "zinc"),
    col=c("orange", "orange3", "grey40", "grey70"),
    border.col="black",
        border.alpha = .75,
        scale=.7)

## End(Not run)

# restore current mode
tmap_mode(current.mode)
```

`tm_fill` *Draw polygons*

Description

Creates a tmap-element that draws the polygons. `tm_fill` fills the polygons. Either a fixed color is used, or a color palette is mapped to a data variable. `tm_borders` draws the borders of the polygons. `tm_polygons` fills the polygons and draws the polygon borders.

Usage

```
tm_fill(col = NA, alpha = NA, palette = NULL, convert2density = FALSE,
  area = NULL, n = 5, style = ifelse(is.null(breaks), "pretty", "fixed"),
  breaks = NULL, interval.closure = "left", labels = NULL,
  auto.palette.mapping = TRUE, contrast = NA, max.categories = 12,
  colorNA = NA, textNA = "Missing", showNA = NA, thres.poly = 0,
  title = NA, legend.show = TRUE, legend.format = list(),
  legend.is.portrait = TRUE, legend.reverse = FALSE, legend.hist = FALSE,
  legend.hist.title = NA, legend.z = NA, legend.hist.z = NA, id = NA,
  popup.vars = NA, popup.format = list(), ...)
```

```
tm_borders(col = NA, lwd = 1, lty = "solid", alpha = NA)

tm_polygons(col = NA, alpha = NA, border.col = NA, border.alpha = NA,
  ...)
```

Arguments

col	For tm_fill, it is one of • a single color value • the name of a data variable that is contained in shp. Either the data variable contains color values, or values (numeric or categorical) that will be depicted by a color palette (see palette. In the latter case, a choropleth is drawn. • "MAP_COLORS". In this case polygons will be colored such that adjacent polygons do not get the same color. See the underlying function map_coloring for details. For tm_borders, it is a single color value that specifies the border line color. If multiple values are specified, small multiples are drawn (see details).
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
palette	a palette name or a vector of colors. See tmaptools::palette_explorer() for the named palettes. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette, which typically depends on the style. The type of palette from aes.palette is automatically determined, but can be overwritten: use "seq" for sequential, "div" for diverging, and "cat" for categorical.
convert2density	boolean that determines whether col is converted to a density variable. Should be TRUE when col consists of absolute numbers. The area size is either approximated from the shape object, or given by the argument area.
area	Name of the data variable that contains the area sizes in squared kilometer.
n	preferred number of classes (in case col is a numeric variable).
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".
breaks	in case style=="fixed", breaks should be specified. The breaks argument can also be used when style="cont". In that case, the breaks are mapped evenly to the sequential or divering color palette.
interval.closure	value that determines whether where the intervals are closed: "left" or "right". Only applicable if col is a numerc variable.

`labels` labels of the classes.

`auto.palette.mapping`
When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. When categorical color palettes are used, this method stretches the palette if there are more levels than colors.

`contrast` vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when `auto.palette.mapping=TRUE`). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

`max.categories` in case `col` is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than `max.categories` and `auto.palette.mapping` is `FALSE`, then levels are combined.

`colorNA` color used for missing values. Use `NULL` for transparency.

`textNA` text used for missing values.

`showNA` logical that determines whether missing values are named in the legend. By default (`NA`), this depends on the presence of missing values.

`thres.poly` number that specifies the threshold at which polygons are taken into account. The number itself corresponds to the proportion of the area sizes of the polygons to the total polygon size. By default, all polygons are drawn. To ignore polygons that are not visible in a normal plot, a value like `1e-05` is recommended.

`title` title of the legend element

`legend.show` logical that determines whether the legend is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. 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 scientically? 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 scientically, 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.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

... Other arguments passed on to `formatC`

`legend.is.portrait`
: logical that determines whether the legend is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.reverse`
: logical that determines whether the items are shown in reverse order, i.e. from bottom to top when `legend.is.portrait = TRUE` and from right to left when `legend.is.portrait = FALSE`

`legend.hist`
: logical that determines whether a histogram is shown

`legend.hist.title`
: title for the histogram. By default, one title is used for both the histogram and the normal legend.

`legend.z`
: index value that determines the position of the legend element with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

`legend.hist.z`
: index value that determines the position of the histogram legend element

`id`
: name of the data variable that specifies the indices of the polygons. Only used for `"view"` mode (see `tmap_mode`).

`popup.vars`
: names of data variables that are shown in the popups in `"view"` mode. If `convert2density=TRUE`, the derived density variable name is suffixed with `_density`. If `NA` (default), only aesthetic variables (i.e. specified by `col` and `lwd`) are shown). If they are not specified, all variables are shown. Set popup.vars to `FALSE` to disable popups. When a vector of variable names is provided, the names (if specified) are printed in the popups.

`popup.format`
: list of formatting options for the popup values. See the argument `legend.format` for options. 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.

`...`
: for `tm_polygons`, these arguments passed to either `tm_fill` or `tm_borders`. For `tm_fill`, these arguments are passed on to `map_coloring`.

`lwd`
: border line width (see `par`)

`lty`
: border line type (see `par`)

`border.col`
: border line color

`border.alpha`
: transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `col` is used (normally 1).

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments. The aesthetic argument of tm_fill (and tm_polygons) is col. In the latter case, the arguments, except for thres.poly, and the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe)

# Constant fill
tm_shape(World) + tm_fill("darkolivegreen3") + tm_format_World(title="A green World")

# Borders only
tm_shape(Europe) + tm_borders()

# Data variable containing colours values
Europe$isNLD <- ifelse(Europe$name=="Netherlands", "darkorange", "darkolivegreen3")
tm_shape(Europe) +
    tm_fill("isNLD") +
tm_layout("Find the Netherlands!")

# Categorical data variable
if (require(RColorBrewer)) {
pal <- brewer.pal(10, "Set3")[c(10, 8, 4, 5)]
tm_shape(Europe) +
tm_polygons("EU_Schengen", palette=pal, title = "European Countries", showNA=FALSE) +
tm_format_Europe()
}

tm_shape(World) +
tm_polygons("economy", title="Economy", id="name") +
tm_text("iso_a3", size="AREA", scale=1.5) +
tm_format_World()

# Numeric data variable
tm_shape(World) +
tm_polygons("HPI", palette="RdYlGn", style="cont", n=8, auto.palette.mapping=FALSE,
title="Happy Planet Index", id="name") +
tm_text("iso_a3", size="AREA", scale=1.5) +
```

```
tm_format_World() +
tm_style_grey()

## Not run:
data(NLD_muni, NLD_prov)
tm_shape(NLD_muni) +
tm_fill(col="population", convert2density=TRUE,
style="kmeans", title = expression("Population (per " * km^2 * ")"),
legend.hist=TRUE, id="name") +
tm_borders("grey25", alpha=.5) +
tm_shape(NLD_prov) +
tm_borders("grey40", lwd=2) +
tm_format_NLD_wide(bg.color="white", frame = FALSE, legend.hist.bg.color="grey90")

# Map coloring algorithm
tm_shape(NLD_prov) +
    tm_fill("name", legend.show = FALSE) +
tm_shape(NLD_muni) +
    tm_polygons("MAP_COLORS", palette="Greys", alpha = .25) +
tm_shape(NLD_prov) +
    tm_borders(lwd=2) +
    tm_text("name", shadow=TRUE) +
tm_format_NLD(title="Dutch provinces and\nmunicipalities", bg.color="white")

# Cartogram
if (require(cartogram)) {
NLD_prov_pop <- cartogram(NLD_prov, "population")
tm_shape(NLD_prov_pop) +
tm_polygons("origin_non_west", title = "Non-western origin (%)")
}

## End(Not run)

# TIP: check out these examples in view mode, enabled with tmap_mode("view")
```

`tm_grid` *Coordinate grid lines*

Description

Creates a tmap-element that draws coordinate grid lines. It serves as a layer that can be drawn anywhere between other layers. By default the coordinate system of the (master) shape object is used, which results in horizontal and vertical lines. Alternatively, grid lines can be reprojected, for instance to latitude longitude coordinates, and hence be curved.

Usage

```
tm_grid(x = NA, y = NA, n.x = NA, n.y = NA, projection = NA,
  col = NA, lwd = 1, alpha = NA, labels.size = 0.6, labels.col = NA,
  labels.rot = c(0, 0), labels.format = list(big.mark = ","),
  labels.margin.x = 0, labels.margin.y = 0, labels.inside.frame = TRUE)
```

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	prefered number of grid lines for the x axis.
n.y	prefered number of grid lines for the y axis.
projection	projection character. If specified, the grid lines are projected accordingly. See set_projection for projection details. Many world maps are projected, but still have latitude longitude ("longlat") grid lines.
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.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.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 scientically? 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 scientically, 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.margin.x	margin between tick labels of x axis and the frame
labels.margin.y	margin between tick labels of y axis and the frame
labels.inside.frame	Show labels inside the frame?

tm_iso *Draw iso (contour) lines with labels*

Description

This function is a wrapper of tm_lines and tm_text aimed to draw isopleths, which can be created with smooth_map.

Usage

```
tm_iso(col = NA, text = "level", size = 0.5, remove.overlap = TRUE,
  along.lines = TRUE, overwrite.lines = TRUE, ...)
```

Arguments

col	line color. See tm_lines.
text	text to display. By default, it is the variable named "level" of the shape that is created with smooth_map
size	text size (see tm_text)
remove.overlap	see tm_text
along.lines	see tm_text
overwrite.lines	see tm_text
...	arguments passed on to tm_lines or tm_text

See Also

smooth_map

tm_layout *Layout of cartographic maps*

Description

This element specifies the map layout. The main function tm_layout controlls title, margins, aspect ratio, colors, frame, legend, among many other things. The function tm_legend is a shortcut to access all legend. arguments without this prefix. The other functions are wrappers for two purposes: the tm_format_ functions specify position related layout settings such as margins, and the tm_style_ functions specify general styling related layout settings such as colors and font. Typically, the former functions are shape dependent, and the latter functions are shape independent. See details for predefined styles and formats. With the global option tmap.style, a default style can be specified. Multiple tm_layout elements (or wrapper functions) can be stacked: called arguments will be overwritten.

Usage

```
tm_layout(title = NA, scale = 1, title.size = 1.3, bg.color = "white",
  aes.color = c(fill = "grey85", borders = "grey40", symbols = "grey60", dots
  = "black", lines = "red", text = "black", na = "grey75"),
  aes.palette = list(seq = "YlOrBr", div = "RdYlGn", cat = "Set3"),
  attr.color = "black", sepia.intensity = 0, saturation = 1,
  frame = TRUE, frame.lwd = 1, frame.double.line = FALSE, asp = NA,
  outer.margins = rep(0.02, 4), inner.margins = NA, between.margin = 0.5,
  outer.bg.color = NULL, fontface = "plain", fontfamily = "",
  compass.type = "arrow", earth.boundary = FALSE,
  earth.boundary.color = attr.color, earth.boundary.lwd = 1,
  earth.datum = "WGS84", space.color = NULL, legend.show = TRUE,
  legend.only = FALSE, legend.outside = NA,
  legend.outside.position = "right", legend.outside.size = 0.3,
  legend.position = NULL, legend.stack = c("vertical", "horizontal"),
  legend.just = c("left", "bottom"), legend.width = 0.4,
  legend.height = 0.9, legend.hist.height = 0.3,
  legend.hist.width = legend.width, legend.title.size = 1.1,
  legend.text.size = 0.7, legend.hist.size = 0.7, legend.format = list(fun
  = NULL, scientific = FALSE, digits = NA, text.separator = "to", text.less.than
  = c("Less", "than"), text.or.more = c("or", "more"), text.align = NA,
  text.to.columns = FALSE), legend.frame = FALSE,
  legend.text.color = attr.color, legend.bg.color = NA,
  legend.bg.alpha = 1, legend.hist.bg.color = NA,
  legend.hist.bg.alpha = 1, title.snap.to.legend = NA,
  title.position = c("left", "top"), title.color = attr.color,
  title.bg.color = NA, title.bg.alpha = 1, panel.show = NA,
  panel.labels = NA, panel.label.size = 1, panel.label.color = "black",
  panel.label.bg.color = "grey80", panel.label.height = 1.25,
  panel.label.rot = c(90, 0), main.title = NA, main.title.size = 1.5,
  main.title.color = "black", main.title.position = "left",
  attr.outside = FALSE, attr.outside.position = "bottom",
  attr.outside.size = NA, attr.position = c("right", "bottom"),
  attr.just = c("left", "bottom"), design.mode = FALSE,
  basemaps = c("CartoDB.Positron", "OpenStreetMap", "Esri.WorldTopoMap"),
  basemaps.alpha = c(1, 1, 1), bg.overlay = NULL, bg.overlay.alpha = 0)

tm_legend(...)

tm_format_World(title = NA, inner.margins = c(0, 0.05, 0.025, 0.01),
  legend.position = c("left", "bottom"), attr.position = c("right",
  "bottom"), scale = 0.8, ...)

tm_format_World_wide(title = NA, inner.margins = c(0, 0.2, 0.025, 0.01),
  legend.position = c("left", "bottom"), attr.position = c("right",
  "bottom"), scale = 0.8, ...)

tm_format_Europe(title = NA, title.position = c("left", "top"),
```

```
  legend.position = c("right", "top"), legend.just = c("right", "top"),
  attr.position = c("right", "bottom"), legend.frame = TRUE,
  inner.margins = c(0, 0, 0, 0), ...)

tm_format_Europe2(title = NA, title.position = c("left", "top"),
  legend.position = c("left", "top"), legend.outside = TRUE,
  legend.outside.size = 0.2, legend.just = c("right", "top"),
  attr.position = c("right", "bottom"), inner.margins = c(0, 0, 0, 0), ...)

tm_format_Europe_wide(title = NA, title.position = c("left", 0.85),
  title.snap.to.legend = TRUE, legend.position = c("left", 0.85),
  legend.just = c("left", "top"), attr.position = c("left", "bottom"),
  inner.margins = c(0, 0.25, 0, 0), ...)

tm_format_NLD(title = NA, frame = FALSE, inner.margins = c(0.02, 0.2,
  0.06, 0.02), legend.position = c("left", "top"), attr.position = c("left",
  "bottom"), ...)

tm_format_NLD_wide(title = NA, frame = FALSE, inner.margins = c(0.02, 0.3,
  0.06, 0.02), legend.position = c("left", "top"), attr.position = c("left",
  "bottom"), ...)

tm_style_white(...)

tm_style_gray(bg.color = "grey85", aes.color = c(fill = "grey70", borders =
  "grey20", symbols = "grey50", dots = "black", lines = "red", text = "black",
  na = "grey60"), ...)

tm_style_natural(bg.color = "lightskyblue1", aes.color = c(fill =
  "darkolivegreen3", borders = "black", symbols = "tomato2", dots = "firebrick",
  lines = "steelblue", text = "black", na = "white"), aes.palette = list(seq =
  "YlGn", div = "RdYlGn", cat = "Set3"), attr.color = "black",
  space.color = "white", legend.frame = TRUE, legend.bg.color = "grey90",
  earth.boundary = TRUE, basemaps = "Esri.NatGeoWorldMap",
  basemaps.alpha = 1, ...)

tm_style_grey(bg.color = "grey85", aes.color = c(fill = "grey70", borders =
  "grey20", symbols = "grey50", dots = "black", lines = "red", text = "black",
  na = "grey60"), ...)

tm_style_cobalt(bg.color = "#002240", aes.color = c(fill = "#0088FF",
  borders = "#002240", symbols = "#FF9D00", dots = "#FF9D00", lines = "#FFEE80",
  text = "white", na = "grey60"), aes.palette = list(seq = "YlGn", div =
  "RdYlGn", cat = "Set3"), attr.color = "white",
  basemaps = "CartoDB.DarkMatter", basemaps.alpha = 0.5, ...)

tm_style_col_blind(bg.color = "white", aes.color = c(fill = "grey85",
  borders = "black", symbols = "#D55E00", dots = "#0072B2", lines = "#009E73",
```

```
  text = "black", na = "white"), aes.palette = list(seq = "Blues", div =
  "RdBu", cat = c("#D55E00", "#56B4E9", "#E69F00", "#009E73", "#F0E442",
  "#0072B2", "#CC79A7")), attr.color = "black", ...)

tm_style_albatross(bg.color = "#00007F", aes.color = c(fill = "#4C4C88",
  borders = "#00004C", symbols = "#BFBFFF", dots = "#BFBFFF", lines = "#BFBFFF",
  text = "#FFE700", na = "grey60"), aes.palette = list(seq = "YlOrRd", div =
  "RdYlGn", cat = "Set3"), attr.color = "#BFBFFF",
  basemaps = "CartoDB.DarkMatter", basemaps.alpha = 0.5, ...)

tm_style_beaver(bg.color = "#FFFFFF", aes.color = c(fill = "#FFE200",
  borders = "#000000", symbols = "#A30000", dots = "#A30000", lines = "#A30000",
  text = "#000000", na = "#E0E0E0"), aes.palette = list(seq = "YlOrBr", div =
  "RdYlGn", cat = "Dark2"), attr.color = "black", ...)

tm_style_bw(saturation = 0, ...)

tm_style_classic(sepia.intensity = 0.7, fontfamily = "serif",
  frame.double.line = TRUE, compass.type = "rose",
  basemaps = "Esri.WorldTopoMap", basemaps.alpha = 0.5, ...)
```

Arguments

`title`	Global title of the map. For small multiples, multiple titles can be specified. The title is drawn inside the map. Alternatively, use `panel.labels` to print the map as a panel, with the title inside the panel header (especially useful for small multiples). Another alternative is the `main.title` which prints a title above the map. Titles for the legend items are specified at the layer functions (e.g. `tm_fill`).
`scale`	numeric value that serves as the global scale parameter. All font sizes, symbol sizes, border widths, and line widths are controled by this value. Each of these elements can be scaled independantly with the `scale`, `lwd`, or `size` arguments provided by the `tmap-elements`.
`title.size`	Relative size of the title
`bg.color`	Background color. By default it is `"white"`. A recommended alternative for choropleths is light grey (e.g., `"grey85"`).
`aes.color`	Default color values for the aesthetics layers. Should be a named vector with the names chosen from: `fill`, `borders`, `symbols`, `dots`, `lines`, `text`, `na`. Use `"#00000000"` for transparency.
`aes.palette`	Default color palettes for the aesthetics. It takes a list of three items: `seq` for sequential palettes, `div` for diverging palettes, and `cat` for categorical palettes. By default, Color Brewer palettes (see (see `tmaptools::palette_explorer()`)) are used. It is also possible provide a vector of colors for any of these items.
`attr.color`	Default color value for map attributes
`sepia.intensity`	Number between 0 and 1 that defines the amount of sepia effect, which gives the map a brown/yellowish flavour. By default this effect is disabled (`sepia.intensity=0`). All colored used in the map are adjusted with this effect.

`saturation` Number that determines how much saturation (also known as chroma) is used: `saturation=0` is greyscale and `saturation=1` is normal. A number larger than 1 results in very saturated maps. All colored used in the map are adjusted with this effect. Hacking tip: use a negative number.

`frame` Either a boolean that determines whether a frame is drawn, or a color value that specifies the color of the frame.

`frame.lwd` width of the frame

`frame.double.line` draw a double frame line border?

`asp` Aspect ratio. The aspect ratio of the map (width/height). If `NA`, it is determined by the bounding box (see argument `bbox` of `tm_shape`), the `outer.margins`, and the `inner.margins`. If `0`, then the aspect ratio is adjusted to the aspect ratio of the device.

`outer.margins` Relative margins between device and frame. Vector of four values specifying the bottom, left, top, and right margin. Values are between 0 and 1. When facets are created, the outer margins are the margins between the outer panels and the device borders (see also `between.margin`)

`inner.margins` Relative margins inside the frame. Vector of four values specifying the bottom, left, top, and right margin. Values are between 0 and 1. By default, 0 for each side if master shape is a raster, otherwise 0.02.

`between.margin` Margin between facets (small multiples) in number of text line heights. The height of a text line is automatically scaled down based on the number of facets.

`outer.bg.color` Background color outside the frame.

`fontface` font face of all text in the map.

`fontfamily` font family of the text labels.

`compass.type` type of compass, one of: `"arrow"`, `"4star"`, `"8star"`, `"radar"`, `"rose"`. Of course, only applicable if a compass is shown. The compass type can also be set within `tm_compass`.

`earth.boundary` Logical that determines whether the boundaries of the earth are shown or an object that specifies the boundaries. This object can be a vector of size four, a 2 by 2 matrix (bounding box), or an `extent` object. By default, the boundaries are `c(-180, 180, -90, 90)`. Useful for rojected world maps. Often, it is useful to crop both poles (e.g., with `c(-180, 180, -88, 88)`).

`earth.boundary.color` Color of the earth boundary.

`earth.boundary.lwd` Line width of the earth boundary.

`earth.datum` Geodetic datum to determine the earth boundary. By default `"WGS84"`, other frequently used datums are `"NAD83"` and `"NAD27"`. Any other `PROJ.4` character string can be used.

`space.color` Color of the space, i.e. the region inside the frame, and outsise the earth boundary.

`legend.show` Logical that determines whether the legend is shown.

legend.only logical. Only draw the legend (without map)? Particularly useful for small multiples with a common legend.

legend.outside Logical that determines whether the legend is plot outside of the map/facets. Especially useful when using facets that have a common legend (i.e. with free.scales=FALSE).

legend.outside.position
Character that determines the outside position of the legend. Only applicable when legend.outside=TRUE. One of: "right", "left", "top", or "bottom".

legend.outside.size
Numeric value that determines the relative size of the legend, when legend.outside=TRUE. If the first value of legend.outside.position is "top" or "bottom", then it is the width of the legend, else it is the height of the legend.

legend.position
Position of the legend. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y coordinates of the left bottom corner of the legend. The uppercase values correspond to the position without margins (so tighter to the frame). By default, it is automatically placed in the corner with most space based on the (first) shape object. If legend.outside=TRUE, this argument specifies the legend position within the outside panel.

legend.stack Value that determines how the legend items are stacked: "vertical" or "horizontal".

legend.just Justification of the legend 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/bottom alignment and 1 right/top alignment. This option is only used, if legend.position is specified by numeric coordinates.

legend.width width of the legend. If it is a negative number, it will be the exact legend width. If it is a positive number (by default), it will be the maximum legend width; the actual legend width will be decreased automatically based on the legend content and font sizes.

legend.height height of the legend. If it is a negative number, it will be the exact legend height. If it is a positive number (by default), it will be the maximum legend height; the actual legend height will be decreased automatically based on the legend content and font sizes.

legend.hist.height
height of the histogram. This hight is initial. If the total legend is downscaled to legend.height, the histogram is downscaled as well.

legend.hist.width
width of the histogram. By default, it is equal to the legend.width.

legend.title.size
Relative font size for the legend title

legend.text.size
Relative font size for the legend text elements

`legend.hist.size`
: Relative font size for the choropleth histogram

`legend.format`
: list of formatting options for the legend numbers. Only applicable for layer functions (such as `tm_fill`) where `labels` is undefined. 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 scientically? 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 scientically, 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.

  text.separator Character string to use to separate numbers in the legend (default: "to").

  text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

  text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

  text.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

  text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

  ... Other arguments passed on to `formatC`

`legend.frame`
: either a logical that determines whether the legend is placed inside a frame, or a color that directly specifies the frame border color. The width of the frame is automatically determined, but is upper-bounded by `legend.width`.

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

`legend.bg.color`
: Background color of the legend. Use `TRUE` to match with the overall background color `bg.color`.

`legend.bg.alpha`
: Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `legend.bg.color` is used (normally 1).

legend.hist.bg.color
: Background color of the histogram

legend.hist.bg.alpha
: Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the legend.hist.bg.color is used (normally 1).

title.snap.to.legend
: Logical that determines whether the title is part of the legend. By default FALSE, unless the legend is drawn outside the map (see legend.outside).

title.position
: Position of the title. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y coordinates of the tile. The uppercase values correspond to the position without margins (so tighter to the frame). By default the title is placed on top of the legend (determined by legend.position).

title.color
: color of the title

title.bg.color
: background color of the title. Use TRUE to match with the overall background color bg.color. By default, it is TRUE if legend.frame is TRUE or a color.

title.bg.alpha
: Transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the title.bg.color is used (normally 1).

panel.show
: Logical that determines if the map(s) are shown as panels. If TRUE, the title will be placed in the panel header instead of inside the map. By default, it is TRUE when small multiples are created with the by variable. (See tm_facets)

panel.labels
: Panel labels. Only applicable when panel.show is TRUE. For cross tables facets, it should be a list containing the row names in the first, and column names in the second item.

panel.label.size
: Relative font size of the panel labels

panel.label.color
: Font color of the panel labels

panel.label.bg.color
: Background color of the panel labels

panel.label.height
: Height of the labels in number of text line heights.

panel.label.rot
: Rotation angles of the panel labels. Vector of two values: the first is the rotation angle (in degrees) of the row panels, which are only used in cross-table facets (when tm_facets's by is specified with two variables). The second is the rotation angle of the column panels.

main.title
: Title that is printed above the map (or small multiples). When multiple pages are generated (see along argument of tm_facets), a vector can be provided. By default, the main title is only printed when this along argument is specified.

main.title.size
: Size of the main title

main.title.color
: Color of the main title

`main.title.position`
: Position of the main title. Either a numeric value between 0 (left) and 1 (right), or a character value: `"left"`, `"center"`, or `"right"`.

`attr.outside`
: Logical that determines whether the attributes are plot outside of the map/facets.

`attr.outside.position`
: Character that determines the outside position of the attributes: `"top"` or `"bottom"`. Only applicable when `attr.outside=TRUE`. If the legend is also drawn outside (with `legend.outside=TRUE`) and on the same side of the map (e.g. also `"top"` or `"bottom"`), the attributes are placed between the map and the legend. This can be changed by setting `attr.outside.position` to `"TOP"` or `"BOTTOM"`: in this case, the attributes are placed above respecvitely below the legend.

`attr.outside.size`
: Numeric value that determines the relative height of the attribute viewport, when `attr.outside=TRUE`.

`attr.position`
: Position of the map attributes, which are `tm_credits`, `tm_scale_bar` and `tm_compass`. Vector of two values, specifing the x and y coordinates. The first value is `"left"`, `"LEFT"`, `"center"`, `"right"`, or `"RIGHT"`, and the second value `"top"`, `"TOP"`, `"center"`, `"bottom"`, or `"BOTTOM"`. The uppercase values correspond to the position without margins (so tighter to the frame). Positions can also be set separately in the map attribute fuctions. If `attr.outside=TRUE`, this argument specifies the position of the attributes within the outside panel.

`attr.just`
: Justification of the attributes 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/bottom alignment and 1 right/top alignment. This option is only used, if `attr.position` is specified by numeric coordinates. It can also be specified per attribute function.

`design.mode`
: Logical that enables the design mode. If `TRUE`, inner and outer margins, legend position, aspect ratio are explicitely shown. Also, feedback text in the console is given.

`basemaps`
: vector of one or more names of baselayer maps used in the interactive view mode. See `tm_view`.

`basemaps.alpha`
: alpha transparancy (opacity) of the basemaps. See `tm_view`.

`bg.overlay`
: Not used anymore. See `tm_view`.

`bg.overlay.alpha`
: Not used anymore. See `tm_view`.

`...`
: other arguments from `tm_layout`

Details

Predefined styles:

`tm_style_white`	White background, commonly used colors (default)
`tm_style_gray/_grey`	Grey background, useful to highlight sequential palettes (e.g. in choropleths)
`tm_style_natural`	Emulation of natural view: blue waters and green land
`tm_style_bw`	Greyscale, obviously useful for greyscale printing

tm_style_classic	Classic styled maps (recommended)
tm_style_cobalt	Inspired by latex beamer style cobalt
tm_style_albatross	Inspired by latex beamer style albatross
tm_style_beaver	Inspired by latex beamer style beaver

Predefined formats

tm_format_World	Format specified for world maps
tm_format_World_wide	Format specified for world maps with more space for the legend
tm_format_Europe	Format specified for maps of Europe
tm_format_Europe_wide	Format specified for maps of Europe with more space for the legend
tm_format_NLD	Format specified for maps of the Netherlands
tm_format_NLD_wide	Format specified for maps of the Netherlands with more space for the legend

See Also

vignette("tmap-nutshell")

Examples

```
data(World, land)

tm_shape(World) +
    tm_fill("pop_est_dens", style="kmeans", title="Population density") +
tm_format_World(title="The World") + tm_style_albatross(frame.lwd=10)

## Not run:
tm_shape(land) +
tm_raster("elevation", breaks=c(-Inf, 250, 500, 1000, 1500, 2000, 2500, 3000, 4000, Inf),
palette = terrain.colors(9), title="Elevation", auto.palette.mapping = FALSE) +
tm_shape(World, is.master=TRUE) +
tm_borders("grey20") +
tm_grid(projection="longlat", labels.size = .5) +
tm_text("name", size="AREA") +
tm_compass(position = c(.65, .15), color.light = "grey90") +
tm_credits("Eckert IV projection", position = c("right", "BOTTOM")) +
tm_layout(inner.margins=c(.04,.03, .02, .01),
earth.boundary = TRUE,
space.color="grey90") +
tm_style_classic(bg.color="lightblue") +
tm_legend(position = c("left", "bottom"),
frame = TRUE,
bg.color="lightblue")

## End(Not run)

WorldOne <- rgeos::gUnaryUnion(World)
```

```
tm_shape(World, projection="wintri") +
tm_fill("HPI", palette="div", auto.palette.mapping = FALSE, n=7,
title = "Happy Planet Index") +
tm_shape(WorldOne) +
tm_borders() +
tm_grid(projection = "longlat") +
tm_credits("Winkel Tripel projection", position = c("right", "BOTTOM")) +
tm_style_natural(earth.boundary = c(-180,180,-87,87), inner.margins = .05) +
tm_legend(position=c("left", "bottom"), bg.color="grey95", frame=TRUE)

## Not run:
# global option tmap.style demo

# get current style
current.style <- tmap_style()

qtm(World, fill="economy", format="World")

tmap_style("col_blind")
qtm(World, fill="economy", format="World")

tmap_style("cobalt")
qtm(World, fill="economy", format="World")

# set to current style
tmap_style(current.style)

## End(Not run)

# TIP: check out these examples in view mode, enabled with tmap_mode("view")
```

`tm_lines` *Draw spatial lines*

Description

Creates a `tmap-element` that draw spatial lines.

Usage

```
tm_lines(col = NA, lwd = 1, lty = "solid", alpha = NA, scale = 1,
  lwd.legend = NULL, lwd.legend.labels = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  interval.closure = "left", palette = NULL, labels = NULL,
  auto.palette.mapping = TRUE, contrast = NA, max.categories = 12,
  colorNA = NA, textNA = "Missing", showNA = NA, title.col = NA,
  title.lwd = NA, legend.col.show = TRUE, legend.lwd.show = TRUE,
  legend.format = list(), legend.col.is.portrait = TRUE,
  legend.lwd.is.portrait = FALSE, legend.col.reverse = FALSE,
```

```
legend.lwd.reverse = FALSE, legend.hist = FALSE, legend.hist.title = NA,
legend.col.z = NA, legend.lwd.z = NA, legend.hist.z = NA, id = NA,
popup.vars = NA, popup.format = list())
```

Arguments

col	color of the lines. Either a color value or a data variable name. If multiple values are specified, small multiples are drawn (see details).
lwd	line width. Either a numeric value or a data variable. In the latter case, the class of the highest values (see style) will get the line width defined by scale. If multiple values are specified, small multiples are drawn (see details).
lty	line type.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
scale	line width multiplier number.
lwd.legend	vector of line widths that are shown in the legend. By default, this is determined automatically.
lwd.legend.labels	vector of labels for that correspond to lwd.legend.
n	preferred number of color scale classes. Only applicable when lwd is the name of a numeric variable.
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".
breaks	in case style=="fixed", breaks should be specified. The breaks argument can also be used when style="cont". In that case, the breaks are mapped evenly to the sequential or divering color palette.
interval.closure	value that determines whether where the intervals are closed: "left" or "right". Only applicable if col is a numerc variable.
palette	a palette name or a vector of colors. See tmaptools::palette_explorer() for the named palettes. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette, which typically depends on the style. The type of palette from aes.palette is automatically determined, but can be overwritten: use "seq" for sequential, "div" for diverging, and "cat" for categorical.
labels	labels of the classes
auto.palette.mapping	When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are

assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. In this case of line widths, obviously only the positive side is used. When categorical color palettes are used, this method stretches the palette if there are more levels than colors.

`contrast` vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when `auto.palette.mapping=TRUE`). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

`max.categories` in case `col` is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than `max.categories` and `auto.palette.mapping` is `FALSE`, then levels are combined.

`colorNA` color used for missing values. Use `NULL` for transparency.

`textNA` text used for missing values.

`showNA` logical that determines whether missing values are named in the legend. By default (`NA`), this depends on the presence of missing values.

`title.col` title of the legend element regarding the line colors

`title.lwd` title of the legend element regarding the line widths

`legend.col.show`

logical that determines whether the legend for the line colors is shown

`legend.lwd.show`

logical that determines whether the legend for the line widths is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. 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 scientically? 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 scientically, 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.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

... Other arguments passed on to `formatC`

`legend.col.is.portrait`
: logical that determines whether the legend element regarding the line colors is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.lwd.is.portrait`
: logical that determines whether the legend element regarding the line widths is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.col.reverse`
: logical that determines whether the items of the legend regarding the line colors sizes are shown in reverse order, i.e. from bottom to top when `legend.col.is.portrait = TRUE` and from right to left when `legend.col.is.portrait = FALSE`

`legend.lwd.reverse`
: logical that determines whether the items of the legend regarding the line widths are shown in reverse order, i.e. from bottom to top when `legend.lwd.is.portrait = TRUE` and from right to left when `legend.lwd.is.portrait = FALSE`

`legend.hist` logical that determines whether a histogram is shown regarding the line colors

`legend.hist.title`
: title for the histogram. By default, one title is used for both the histogram and the normal legend for line colors.

`legend.col.z` index value that determines the position of the legend element regarding the line colors with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

`legend.lwd.z` index value that determines the position of the legend element regarding the line widths. (See `legend.col.z`)

`legend.hist.z` index value that determines the position of the legend element regarding the histogram. (See `legend.col.z`)

`id` name of the data variable that specifies the indices of the lines. Only used for `"view"` mode (see `tmap_mode`).

`popup.vars` names of data variables that are shown in the popups in `"view"` mode. If `NA` (default), only aesthetic variables (i.e. specified by `col` and `lwd`) are shown). If they are not specified, all variables are shown. Set popup.vars to `FALSE` to disable popups. When a vector of variable names is provided, the names (if specified) are printed in the popups.

`popup.format` list of formatting options for the popup values. See the argument `legend.format` for options. 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.

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments. The aesthetic arguments of tm_lines are col and lwd. In the latter case, the arguments, except for the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, rivers)

qtm(rivers)

## Not run:
tm_shape(Europe) +
    tm_fill() +
tm_shape(rivers) +
    tm_lines(col="black", lwd="scalerank", scale=2, legend.lwd.show = FALSE) +
tm_layout("Rivers of Europe") +
tm_style_cobalt()

## End(Not run)
```

---

tm_logo *Logo*

Description

Creates a map logo. Multiple logos can be specified which are shown next to each other. Logos placed on top of each other can be specified with stacking tm_logo elements.

Usage

```
tm_logo(file, height = 3, halign = "center", margin = 0.2,
  position = NA, just = NA)
```

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.
halign	if logos in one row have different heights, halign specifies the vertical alignment. Possible values are "top", "center" and "bottom".
margin	margin around the logo in number of text line heights.
position	position of the logo. Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the center of the text. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.
just	Justification of the attribute 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/bottom alignment and 1 right/top alignment. This option is only used, if position is specified by numeric coordinates. The default value is controlled by the argument "attr.just" of tm_layout.

Examples

```
## Not run:
data(NLD_muni, NLD_prov)

tm_shape(NLD_muni) +
tm_polygons("origin_native", border.alpha=0.5, style="cont", title="Native Dutch (%)") +
tm_logo("http://statline.cbs.nl/Statweb/Images/cbs_logo.png",
        position=c("left", "bottom"), height = 2) +
tm_layout(bg.color="gray98")

data(World)

tm_shape(World) +
tm_polygons("HPI", palette="RdYlGn", auto.palette.mapping=FALSE) +
tm_logo(c("https://www.r-project.org/logo/Rlogo.png",
        system.file("img/tmap.png", package="tmap"))) +
tm_logo("http://blog.kulikulifoods.com/wp-content/uploads/2014/10/logo.png",
        height=5, position = c("left", "top")) +
tm_format_World()

## End(Not run)
```

tm_raster *Draw a raster*

Description

Creates a tmap-element that draws a raster. For coloring, there are three options: 1) a fixed color is used, 2) a color palette is mapped to a data variable, 3) RGB values are used. The function tm_raster is designed for option 2, while tm_rgb is used for option 3.

Usage

```
tm_raster(col = NA, alpha = NA, palette = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  interval.closure = "left", labels = NULL, auto.palette.mapping = TRUE,
  contrast = NA, max.categories = 12, colorNA = NULL, saturation = 1,
  interpolate = FALSE, textNA = "Missing", showNA = NA, title = NA,
  legend.show = TRUE, legend.format = list(), legend.is.portrait = TRUE,
  legend.reverse = FALSE, legend.hist = FALSE, legend.hist.title = NA,
  legend.z = NA, legend.hist.z = NA)

tm_rgb(alpha = NA, saturation = 1, interpolate = TRUE, ...)
```

Arguments

col	three options: a single color value, the name of a data variable that is contained in shp, or the name of a variable in shp that contain color values. In the second case the values (numeric or categorical) that will be depicted by a color palette (see palette. If omitted, and if shp contains three numeric layers that range between 0 and 255, these are interpreted as RGB values, else, the first data variable is selected. If multiple values are specified, small multiples are drawn (see details). By default, it is the name of the first data variable.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
palette	a palette name or a vector of colors. See tmaptools::palette_explorer() for the named palettes. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette, which typically depends on the style. The type of palette from aes.palette is automatically determined, but can be overwritten: use "seq" for sequential, "div" for diverging, and "cat" for categorical.
n	preferred number of classes (in case col is a numeric variable)
style	method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former

maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".

breaks — in case style=="fixed", breaks should be specified. The breaks argument can also be used when style="cont". In that case, the breaks are mapped evenly to the sequential or divering color palette.

interval.closure — value that determines whether where the intervals are closed: "left" or "right". Only applicable if col is a numerc variable.

labels — labels of the classes

auto.palette.mapping — When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. When categorical color palettes are used, this method stretches the palette if there are more levels than colors.

contrast — vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

max.categories — in case col is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than max.categories and auto.palette.mapping is FALSE, then levels are combined.

colorNA — color used for missing values. Use NULL for transparency.

saturation — Number that determines how much saturation (also known as chroma) is used: saturation=0 is greyscale and saturation=1 is normal. This saturation value is multiplied by the overall saturation of the map (see tm_layout).

interpolate — Should the raster image be interpolated? By default FALSE for tm_raster and TRUE for tm_rgb.

textNA — text used for missing values.

showNA — logical that determines whether missing values are named in the legend. By default (NA), this depends on the presence of missing values.

title — title of the legend element

legend.show — logical that determines whether the legend is shown

legend.format — list of formatting options for the legend numbers. Only applicable if labels is undefined. 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 scientically? 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 scientically, 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.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, "left", "center" or "right". By default "left" for legends in portrait format (legend.is.protrait = TRUE), and "center" otherwise.

text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default FALSE.

... Other arguments passed on to formatC

legend.is.portrait
: logical that determines whether the legend is in portrait mode (TRUE) or landscape (FALSE)

legend.reverse
: logical that determines whether the items of the legend regarding the text sizes are shown in reverse order, i.e. from bottom to top when legend.is.portrait = TRUE and from right to left when legend.is.portrait = FALSE

legend.hist
: logical that determines whether a histogram is shown

legend.hist.title
: title for the histogram. By default, one title is used for both the histogram and the normal legend.

legend.z
: index value that determines the position of the legend element with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

legend.hist.z
: index value that determines the position of the histogram legend element

...
: arguments passed on from tm_raster to tm_rgb

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments. The aesthetic argument of tm_raster is col. In the latter case, the arguments, except for the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can

take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

Value

tmap-element

See Also

vignette("tmap-nutshell")

Examples

```
data(World, land, metro)

pal8 <- c("#33A02C", "#B2DF8A", "#FDBF6F", "#1F78B4", "#999999", "#E31A1C", "#E6E6E6", "#A6CEE3")
tm_shape(land, ylim = c(-88,88)) +
    tm_raster("cover_cls", palette = pal8, title = "Global Land Cover") +
tm_shape(metro) + tm_dots(col = "#E31A1C") +
tm_shape(World) +
    tm_borders(col = "black") +
tm_layout(scale = .8,
legend.position = c("left","bottom"),
    legend.bg.color = "white", legend.bg.alpha = .2,
    legend.frame = "gray50")

## Not run:
pal20 <- c("#003200", "#3C9600", "#006E00", "#556E19", "#00C800", "#8CBE8C",
   "#467864", "#B4E664", "#9BC832", "#EBFF64", "#F06432", "#9132E6",
   "#E664E6", "#9B82E6", "#B4FEF0", "#646464", "#C8C8C8", "#FF0000",
   "#FFFFFF", "#5ADCDC")
tm_shape(land) +
tm_raster("cover", max.categories = 20, palette = pal20, title = "Global Land Cover") +
tm_layout(scale=.8, legend.position = c("left","bottom"))

## End(Not run)


tm_shape(land, ylim = c(-88,88)) +
    tm_raster("trees", palette = "Greens", title = "Percent Tree Cover") +
tm_shape(World) +
    tm_borders() +
tm_layout(legend.position = c("left", "bottom"), bg.color = "lightblue")

# TIP: check out these examples in view mode, enabled with tmap_mode("view")

## Not run:
# doesn't work in view mode, since it does not support small multiples
tm_shape(land) +
tm_raster("black") +
tm_facets(by="cover_cls")

## End(Not run)
```

tm_scale_bar *Scale bar*

Description

Creates a scale bar. By default, the coordinate units are assumed to be meters, and the map units in kilometers. This can be changed in tm_shape.

Usage

```
tm_scale_bar(breaks = NULL, width = NA, size = 0.5, text.color = NA,
  color.dark = "black", color.light = "white", lwd = 1, position = NA,
  just = NA)
```

Arguments

breaks	breaks of the scale bar. If not specified, breaks will be automatically be chosen given the prefered width of the scale bar. Not available for view mode.
width	(prefered) width of the scale bar. Only applicable when breaks=NULL. In plot mode, it corresponds the relative width; the default is 0.25 so one fourth of the map width. In view mode, it corresponds to the width in pixels; the default is 100.
size	relative text size (which is upperbound by the available label width)
text.color	color of the text. By default equal to the argument attr.color of tm_layout.
color.dark	color of the dark parts of the scale bar, typically (and by default) black.
color.light	color of the light parts of the scale bar, typically (and by default) white.
lwd	line width of the scale bar
position	position of the scale bar Vector of two values, specifing the x and y coordinates. Either this vector contains "left", "LEFT", "center", "right", or "RIGHT" for the first value and "top", "TOP", "center", "bottom", or "BOTTOM" for the second value, or this vector contains two numeric values between 0 and 1 that specifies the x and y value of the left bottom corner of the scale bar. The uppercase values correspond to the position without margins (so tighter to the frame). The default value is controlled by the argument "attr.position" of tm_layout.
just	Justification of the attribute 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/bottom alignment and 1 right/top alignment. This option is only used, if position is specified by numeric coordinates. The default value is controlled by the argument "attr.just" of tm_layout.

Examples

```
current.mode <- tmap_mode("plot")

data(NLD_muni)
qtm(NLD_muni, theme = "NLD") + tm_scale_bar(position=c("left", "bottom"))

data(Europe)
tm_shape(Europe, unit = "mi") +
tm_polygons() +
tm_scale_bar()  + tm_layout(attr.outside = TRUE)

# restore current mode
tmap_mode(current.mode)
```

---

tm_shape *Specify the shape object*

Description

Creates a tmap-element that specifies the shape object. Also the projection and covered area (bounding box) can be set. It is possible to use multiple shape objects within one plot (see tmap-element).

Usage

```
tm_shape(shp, name = NULL, is.master = NA, projection = NULL,
  bbox = NULL, unit = getOption("tmap.unit"), simplify = 1,
  line.center.type = c("segment", "midpoint"), ...)
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame) 5. SpatialPixels(DataFrame) 6. RasterLayer, RasterStack, or RasterBrick Simple features (sf objects) are also supported. For drawing layers tm_fill and tm_borders, 1 is required. For drawing layer tm_lines, 3 is required. Layers tm_symbols and tm_text accept 1 to 3. For layer tm_raster, 4, 5, or 6 is required.
name	name of the shape object (character) as it appears in the legend in "view" mode. Default value is the name of shp.

`is.master` logical that determines whether this `tm_shape` is the master shape element. The bounding box, projection settings, and the unit specifications of the resulting thematic map are taken from the `tm_shape` element of the master shape object. By default, the first master shape element with a raster shape is the master, and if there are no raster shapes used, then the first `tm_shape` is the master shape element.

`projection` Either a `CRS` object or a character value. If it is a character, it can either be a `PROJ.4` character string or a shortcut. See `get_proj4` for a list of shortcut values. By default, the projection is used that is defined in the `shp` object itself, which can be obtained with `get_projection`.

`bbox` bounding box. One of the following:

- A bounding box (either 2 by 2 matrix or an `Extent` object).
- Open Street Map search query. The bounding is automatically generated by querying `q` from Open Street Map Nominatim. See `http://wiki.openstreetmap.org/wiki/Nominatim`.

If unspecified, the current bounding box of `shp` is taken. The bounding box is feed to `bb` (as argument `x`. The other arguments of `bb` can be specified directly as well (see `...`).

`unit` desired units of the map. One of `"metric"` (default), `"imperial"`, `"km"`, `"m"`, `"mi"` and `"ft"`. For other units, please specify `orig` and `to`, which are passed on to `projection_units`. Used to specify the scale bar (see `tm_scale_bar`) and to calculate densities for choropleths (see argument `convert2density` in `tm_fill`).

`simplify` simplification factor for spatial polygons and spatial lines. A number between 0 and 1 that indicates how many coordinates are kept. See the underlying function `simplify_shape`, from which the arguments `keep.units` and `keep.subunits` can be passed on (see `...`).

`line.center.type`

vector of two values specifying how the center of spatial lines is determined Only applicable if `shp` is a `SpatialLines(DataFrame)`, and symbols, dots, and/or text labels are used for this shape. The two values are:

`"feature", "single"` If `"feature"` is specified, a pair of coordinates (used for symbols, dots, and text labels) is chosen for each feature (i.e., a row in the `SpatialLines(DataFrame)`). If `"segment"` is specified, a pair of coordinates is chosed for each line segment.

`"midpoint"` **or** `"centroid"` The midpoint is the middle point on the line, so the coordinates (used for symbols, dots, and text labels) correspond to the midpoints of the line segments. In case the first value is `"feature"`, then per feature, the midpoint of the line segment that is closest to the centroid is taken.

`...` Arguments passed on to `bb` (e.g. `ext` can be used to enlarge or shrinke a bounding box), `projection_units` (the arguments `orig` and `to`), and `simplify_shape` (the arguments `keep.units` and `keep.subunits`)

Value

`tmap-element`

See Also

read_shape to read ESRI shape files, set_projection, vignette("tmap-nutshell")

Examples

```
current.mode <- tmap_mode("plot")

data(World, metro, rivers)

tm_shape(World, projection="longlat") +
    tm_polygons() +
tm_layout("Long lat coordinates (WGS84)", inner.margins=c(0,0,.1,0), title.size=.8)

World$highlighted <- ifelse(World$iso_a3 %in% c("GRL", "AUS"), "gold", "gray75")
tm_shape(World, projection="merc", ylim=c(.1, 1), relative = TRUE) +
    tm_polygons("highlighted") +
tm_layout("Web Mercator projection. Although widely used, it is discouraged for
statistical purposes. In reality, Australia is 3 times larger than Greenland!",
    inner.margins=c(0,0,.1,0), title.size=.6)

tm_shape(World, projection="wintri") +
    tm_polygons() +
tm_layout(
"Winkel-Tripel projection, adapted as default by the National Geographic Society for world maps.",
    inner.margins=c(0,0,.1,0), title.size=.8)

tm_shape(World) +
    tm_polygons() +
tm_layout("Eckhart IV projection. Recommended in statistical maps for its equal-area property.",
    inner.margins=c(0,0,.1,0), title.size=.8)


# different levels of simplification
## Not run:
tm1 <- tm_shape(World, simplify = 0.05) + tm_polygons() + tm_layout("Simplification: 0.05")
tm2 <- tm_shape(World, simplify = 0.1) + tm_polygons() + tm_layout("Simplification: 0.1")
tm3 <- tm_shape(World, simplify = 0.25) + tm_polygons() + tm_layout("Simplification: 0.25")
tm4 <- tm_shape(World, simplify = 0.5) + tm_polygons() + tm_layout("Simplification: 0.5")

require(tmaptools)
tmap_arrange(tm1, tm2, tm3, tm4)

## End(Not run)

# three groups of layers, each starting with tm_shape
## Not run:
tm_shape(World) +
    tm_fill("darkolivegreen3") +
tm_shape(metro) +
    tm_bubbles("pop2010", col = "grey30", scale=.5) +
tm_shape(rivers) +
    tm_lines("lightcyan1") +
```

```
tm_layout(bg.color="lightcyan1", inner.margins=c(0,0,.02,0), legend.show = FALSE)

## End(Not run)

# restore current mode
tmap_mode(current.mode)
```

`tm_symbols` *Draw symbols*

Description

Creates a `tmap-element` that draws symbols, including symbols and dots. The color, size, and shape of the symbols can be mapped to data variables.

Usage

```
tm_symbols(size = 1, col = NA, shape = 21, alpha = NA,
  border.col = NA, border.lwd = 1, border.alpha = NA, scale = 1,
  perceptual = FALSE, clustering = FALSE, size.max = NA, size.lim = NA,
  sizes.legend = NULL, sizes.legend.labels = NULL, n = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  interval.closure = "left", palette = NULL, labels = NULL,
  auto.palette.mapping = TRUE, contrast = NA, max.categories = 12,
  colorNA = NA, textNA = "Missing", showNA = NA, shapes = 21:25,
  shapes.legend = NULL, shapes.legend.fill = NA, shapes.labels = NULL,
  shapeNA = 4, shape.textNA = "Missing", shapes.n = 5,
  shapes.style = ifelse(is.null(shapes.breaks), "pretty", "fixed"),
  shapes.breaks = NULL, shapes.interval.closure = "left",
  legend.max.symbol.size = 0.8, just = NA, jitter = 0, xmod = 0,
  ymod = 0, icon.scale = 3, grob.dim = c(width = 48, height = 48,
  render.width = 256, render.height = 256), title.size = NA, title.col = NA,
  title.shape = NA, legend.size.show = TRUE, legend.col.show = TRUE,
  legend.shape.show = TRUE, legend.format = list(),
  legend.size.is.portrait = FALSE, legend.col.is.portrait = TRUE,
  legend.shape.is.portrait = TRUE, legend.size.reverse = FALSE,
  legend.col.reverse = FALSE, legend.shape.reverse = FALSE,
  legend.hist = FALSE, legend.hist.title = NA, legend.size.z = NA,
  legend.col.z = NA, legend.shape.z = NA, legend.hist.z = NA, id = NA,
  popup.vars = NA, popup.format = list())

tm_squares(size = 1, col = NA, shape = 22, scale = 4/3, ...)

tm_bubbles(size = 1, col = NA, shape = 21, scale = 4/3,
  legend.max.symbol.size = 1, ...)

tm_dots(col = NA, size = 0.02, shape = 16, title = NA,
```

```
  legend.show = TRUE, legend.is.portrait = TRUE, legend.z = NA, ...)

tm_markers(shape = marker_icon(), col = NA, border.col = NULL,
  clustering = TRUE, text = NULL, text.just = c("center", "top"),
  markers.on.top.of.text = TRUE, ...)
```

Arguments

size	a single value or a shp data variable that determines the symbol sizes. The reference value size=1 corresponds to the area of symbols that have the same height as one line of text. If a data variable is provided, the symbol sizes are scaled proportionally (or perceptually, see perceptual) where by default the symbol with the largest data value will get size=1 (see also size.max). If multiple values are specified, small multiples are drawn (see details).
col	color(s) of the symbol. Either a color (vector), or categorical variable name(s). If multiple values are specified, small multiples are drawn (see details).
shape	shape(s) of the symbol. Either direct shape specification(s) or a data variable name(s) that is mapped to the symbols specified by the shapes argument. See details for the shape specification.
alpha	transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
border.col	color of the symbol borders.
border.lwd	line width of the symbol borders. If NA, no symbol borders are drawn.
border.alpha	transparency number, regarding the symbol borders, between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1).
scale	symbol size multiplier number.
perceptual	logical that determines whether symbols are scales with a perceptually (TRUE) or mathematically (FALSE, default value). The perceived area of larger symbols is often underestimated. Flannery (1971) experimentally derived a method to compensate this for symbols, which is enabled by this argument.
clustering	value that determines whether the symbols are clustered in "view" mode. It does not work proportional bubbles (i.e. tm_bubbles). One of: TRUE, FALSE, or the output of markerClusterOptions.
size.max	value that is mapped to size=1. By default (NA), the maximum data value is chosen. Only applicable when size is the name of a numeric variable of shp
size.lim	vector of two limit values of the size variable. Only symbols are drawn whose value is greater than or equal to the first value. Symbols whose values exceed the second value are drawn at the size of the second value. Only applicable when size is the name of a numeric variable of shp
sizes.legend	vector of symbol sizes that are shown in the legend. By default, this is determined automatically.
sizes.legend.labels	vector of labels for that correspond to sizes.legend.

`n` preferred number of color scale classes. Only applicable when `col` is a numeric variable name.

`style` method to process the color scale when `col` is a numeric variable. Discrete options are `"cat"`, `"fixed"`, `"sd"`, `"equal"`, `"pretty"`, `"quantile"`, `"kmeans"`, `"hclust"`, `"bclust"`, `"fisher"`, and `"jenks"`. A numeric variable is processed as a categorial variable when using `"cat"`, i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in `classIntervals`. Continuous options are `"cont"` and `"order"`. The former maps the values of `col` to a smooth gradient, whereas the latter maps the order of values of `col` to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".

`breaks` in case `style=="fixed"`, breaks should be specified. The `breaks` argument can also be used when `style="cont"`. In that case, the breaks are mapped evenly to the sequential or divering color palette.

`interval.closure`
value that determines whether where the intervals are closed: `"left"` or `"right"`. Only applicable if `col` is a numerc variable.

`palette` a palette name or a vector of colors. See `tmaptools::palette_explorer()` for the named palettes. Use a `"-"` as prefix to reverse the palette. The default palette is taken from `tm_layout`'s argument `aes.palette`, which typically depends on the style. The type of palette from `aes.palette` is automatically determined, but can be overwritten: use `"seq"` for sequential, `"div"` for diverging, and `"cat"` for categorical.

`labels` labels of the classes

`auto.palette.mapping`
When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. When categorical color palettes are used, this method stretches the palette if there are more levels than colors.

`contrast` vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when `auto.palette.mapping=TRUE`). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

`max.categories` in case `col` is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than `max.categories` and `auto.palette.mapping` is `FALSE`, then levels are combined.

`colorNA` colour for missing values. Use `NULL` for transparency.

`textNA` text used for missing values of the color variable.

`showNA` logical that determines whether missing values are named in the legend. By default (`NA`), this depends on the presence of missing values.

shapes	palette of symbol shapes. Only applicable if shape is a (vector of) categorical variable(s). See details for the shape specification. By default, the filled symbols 21 to 25 are taken.
shapes.legend	symbol shapes that are used in the legend (instead of the symbols specified with shape. Especially useful when shapes consist of grobs that have to be represented by neutrally colored shapes (see also shapes.legend.fill.
shapes.legend.fill	Fill color of legend shapes (see shapes.legend)
shapes.labels	Legend labels for the symbol shapes
shapeNA	the shape (a number or grob) for missing values. By default a cross (number 4).
shape.textNA	text used for missing values of the shape variable.
shapes.n	preferred number of shape classes. Only applicable when shape is a numeric variable name.
shapes.style	method to process the shape scale when shape is a numeric variable. See style argument for options
shapes.breaks	in case shapes.style=="fixed", breaks should be specified
shapes.interval.closure	value that determines whether where the intervals are closed: "left" or "right". Only applicable if shape is a numerc variable.
legend.max.symbol.size	Maximum size of the symbols that are drawn in the legend. For circles and bubbles, a value larger than one is recommended (and used for tm_bubbles)
just	justification of the symbols 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 is c("center", "center"). For icons, this value may already be speficied (see tmap_icons). The just, if specified, will overrides this.
jitter	number that determines the amount of jittering, i.e. the random noise added to the position of the symbols. 0 means no jittering is applied, any positive number means that the random noise has a standard deviation of jitter times the height of one line of text line.
xmod	horizontal position modification of the symbols, in terms of the height of one line of text. Either a single number for all polygons, or a numeric variable in the shape data specifying a number for each polygon. Together with ymod, it determines position modification of the symbols. See also jitter for random position modifications. In most coordinate systems (projections), the origin is located at the bottom left, so negative xmod move the symbols to the left, and negative ymod values to the bottom.
ymod	vertical position modification. See xmod.
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.heigth

`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 succesfully. Only needed for the view mode.

`title.size` title of the legend element regarding the symbol sizes

`title.col` title of the legend element regarding the symbol colors

`title.shape` title of the legend element regarding the symbol shapes

`legend.size.show` logical that determines whether the legend for the symbol sizes is shown

`legend.col.show` logical that determines whether the legend for the symbol colors is shown

`legend.shape.show` logical that determines whether the legend for the symbol shapes is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. 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 scientically? 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 scientically, 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.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

... Other arguments passed on to `formatC`

legend.size.is.portrait
: logical that determines whether the legend element regarding the symbol sizes is in portrait mode (TRUE) or landscape (FALSE)

legend.col.is.portrait
: logical that determines whether the legend element regarding the symbol colors is in portrait mode (TRUE) or landscape (FALSE)

legend.shape.is.portrait
: logical that determines whether the legend element regarding the symbol shapes is in portrait mode (TRUE) or landscape (FALSE)

legend.size.reverse
: logical that determines whether the items of the legend regarding the symbol sizes are shown in reverse order, i.e. from bottom to top when legend.size.is.portrait = TRUE and from right to left when legend.size.is.portrait = FALSE

legend.col.reverse
: logical that determines whether the items of the legend regarding the symbol colors are shown in reverse order, i.e. from bottom to top when legend.col.is.portrait = TRUE and from right to left when legend.col.is.portrait = FALSE

legend.shape.reverse
: logical that determines whether the items of the legend regarding the symbol shapes are shown in reverse order, i.e. from bottom to top when legend.shape.is.portrait = TRUE and from right to left when legend.shape.is.portrait = FALSE

legend.hist
: logical that determines whether a histogram is shown regarding the symbol colors

legend.hist.title
: title for the histogram. By default, one title is used for both the histogram and the normal legend for symbol colors.

legend.size.z
: index value that determines the position of the legend element regarding the symbol sizes with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

legend.col.z
: index value that determines the position of the legend element regarding the symbol colors. (See legend.size.z)

legend.shape.z
: index value that determines the position of the legend element regarding the symbol shapes. (See legend.size.z)

legend.hist.z
: index value that determines the position of the histogram legend element. (See legend.size.z)

id
: name of the data variable that specifies the indices of the symbols. Only used for "view" mode (see tmap_mode).

popup.vars
: names of data variables that are shown in the popups in "view" mode. If NA (default), only aesthetic variables (i.e. specified by col and lwd) are shown). If they are not specified, all variables are shown. Set popup.vars to FALSE to disable popups. When a vector of variable names is provided, the names (if specified) are printed in the popups.

popup.format
: list of formatting options for the popup values. See the argument legend.format for options. 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.
...	arguments passed on to tm_symbols. For tm_markers, arguments can also be passed on to tm_text. In that case, they have to be prefixed with text., e.g. the col argument should be names text.col
title	shortcut for title.col for tm_dots
legend.show	shortcut for legend.col.show for tm_dots
legend.is.portrait	shortcut for legend.col.is.portrait for tm_dots
legend.z	shortcut for legend.col.z shortcut for tm_dots
text	text of the markers. Shown in plot mode, and as popup text in view mode.
text.just	justification of marker text (see just argument of tm_text). Only applicable in plot mode.
markers.on.top.of.text	For tm_markers, should the markers be drawn on top of the text labels?

Details

Small multiples can be drawn in two ways: either by specifying the by argument in tm_facets, or by defining multiple variables in the aesthetic arguments, which are size, col, and shape. In the latter case, the arguments, except for the ones starting with legend., can be specified for small multiples as follows. If the argument normally only takes a single value, such as n, then a vector of those values can be specified, one for each small multiple. If the argument normally can take a vector, such as palette, then a list of those vectors (or values) can be specified, one for each small multiple.

A shape specification is one of the following three options. 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 speficied by the shape argument.

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.
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. See example of a proportional symbol map with ggplot2 plots.
3. An icon specification, which can be created with tmap_icons.

For small multiples, a list of these shape specification(s) should be provided.

Value

tmap-element

References

Flannery J (1971). The Relative Effectiveness of Some Common Graduated Point Symbols in the Presentation of Quantitative Data. Canadian Cartographer, 8 (2), 96-109.

See Also

vignette("tmap-nutshell")

Examples

```
data(World, Europe, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

tm_shape(World) +
    tm_fill("grey70") +
tm_shape(metro) +
    tm_bubbles("pop2010", col = "growth",
        border.col = "black", border.alpha = .5,
        style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
        palette="-RdYlBu", contrast=1,
        title.size="Metro population",
        title.col="Growth rate (%)") +
tm_format_World()

tm_shape(metro) +
tm_symbols(size = "pop2010", col="pop2010", shape="pop2010",
legend.format = list(text.align="right", text.to.columns = TRUE)) +
tm_layout(legend.outside = TRUE, legend.outside.position = "bottom", legend.stack = "horizontal")

## Not run:
require(tmaptools)
x <- sample_dots(World, vars="gdp_md_est", convert2density = TRUE, w = 100000)
tm_shape(x) +
tm_dots() +
tm_layout("World GDP (one dot is 100 billon dollars)", title.position = c("right", "bottom"))

## End(Not run)

qtm(Europe, bbox="Italy") +
tm_shape(metro) +
tm_markers(text="name")


if (require(ggplot2) && require(dplyr) && require(tidyr) && require(tmaptools)) {
data(NLD_prov)

origin_data <- NLD_prov@data %>%
mutate(FID= factor(1:n())) %>%
select(FID, origin_native, origin_west, origin_non_west) %>%
gather(key=origin, value=perc, origin_native, origin_west, origin_non_west, factor_key=TRUE)

origin_cols <- get_brewer_pal("Dark2", 3)

grobs <- lapply(split(origin_data, origin_data$FID), function(x) {
ggplotGrob(ggplot(x, aes(x="", y=-perc, fill=origin)) +
geom_bar(width=1, stat="identity") +
scale_y_continuous(expand=c(0,0)) +
```

```
scale_fill_manual(values=origin_cols) +
theme_ps(plot.axes = FALSE))
})

names(grobs) <- NLD_prov$name

tm_shape(NLD_prov) +
tm_polygons() +
tm_symbols(size="population", shape="name",
shapes=grobs,
sizes.legend=c(.5, 1,3)*1e6,
scale=1,
legend.shape.show = FALSE,
legend.size.is.portrait = TRUE,
shapes.legend = 22,
title.size = "Population",
id = "name",
popup.vars = c("population", "origin_native",
   "origin_west", "origin_non_west")) +
tm_add_legend(type="fill",
col=origin_cols,
labels=c("Native", "Western", "Non-western"),
title="Origin") +
tm_format_NLD()
}



# TIP: check out these examples in view mode, enabled with tmap_mode("view")

## Not run:
if (require(rnaturalearth)) {

airports <- ne_download(scale=10, type="airports")
airplane <- tmap_icons(paste0("http://cdn.mysitemyway.com/etc-mysitemyway/icons/",
"legacy-previews/icons-256/retro-green-floral-icons-transport-travel/",
"040553-retro-green-floral-icon-transport-travel-transportation-airplane22.png"))

current.mode <- tmap_mode("view")
tm_shape(airports, bbox="Germany") +
tm_symbols(shape=airplane, size="natlscale",
legend.size.show = FALSE, scale=2, border.col = NULL, id="name", popup.vars = TRUE)
tmap_mode(current.mode)
}

## End(Not run)

################################################################################

## Not run:
# plot all available symbol shapes:
if (require(ggplot2)) {
ggplot(data.frame(p=c(0:25,32:127))) +
```

```
geom_point(aes(x=p%%16, y=-(p%/%16), shape=p), size=5, fill="red") +
geom_text(mapping=aes(x=p%%16, y=-(p%/%16+0.25), label=p), size=3) +
scale_shape_identity() +
theme(axis.title=element_blank(),
  axis.text=element_blank(),
  axis.ticks=element_blank(),
  panel.background=element_blank())
}

## End(Not run)
```

`tm_text` *Add text labels*

Description

Creates a `tmap-element` that adds text labels.

Usage

```
tm_text(text, size = 1, col = NA, root = 3, size.lim = NA,
  sizes.legend = NULL, sizes.legend.labels = NULL,
  sizes.legend.text = "Abc", n = 5, style = ifelse(is.null(breaks),
  "pretty", "fixed"), breaks = NULL, interval.closure = "left",
  palette = NULL, labels = NULL, labels.text = NA,
  auto.palette.mapping = TRUE, contrast = NA, max.categories = 12,
  colorNA = NA, textNA = "Missing", showNA = NA, fontface = NA,
  fontfamily = NA, alpha = NA, case = NA, shadow = FALSE,
  bg.color = NA, bg.alpha = NA, size.lowerbound = 0.4,
  print.tiny = FALSE, scale = 1, auto.placement = FALSE,
  remove.overlap = FALSE, along.lines = FALSE, overwrite.lines = FALSE,
  just = c("center", "center"), xmod = 0, ymod = 0, title.size = NA,
  title.col = NA, legend.size.show = TRUE, legend.col.show = TRUE,
  legend.format = list(), legend.size.is.portrait = FALSE,
  legend.col.is.portrait = TRUE, legend.size.reverse = FALSE,
  legend.col.reverse = FALSE, legend.hist = FALSE, legend.hist.title = NA,
  legend.size.z = NA, legend.col.z = NA, legend.hist.z = NA)
```

Arguments

`text`	name of the variable in the shape object that contains the text labels
`size`	relative size of the text labels (see note). Eiter one number, a name of a numeric variable in the shape data that is used to scale the sizes proportionally, or the value `"AREA"`, where the text size is proportional to the area size of the polygons.
`col`	color of the text labels. Either a color value or a data variable name. If multiple values are specified, small multiples are drawn (see details).

root — root number to which the font sizes are scaled. Only applicable if size is a variable name or "AREA". If root=2, the square root is taken, if root=3, the cube root etc.

size.lim — vector of two limit values of the size variable. Only text labels are drawn whose value is greater than or equal to the first value. Text labels whose values exceed the second value are drawn at the size of the second value. Only applicable when size is the name of a numeric variable of shp. See also size.lowerbound which is a threshold of the relative font size.

sizes.legend — vector of text sizes that are shown in the legend. By default, this is determined automatically.

sizes.legend.labels — vector of labels for that correspond to sizes.legend.

sizes.legend.text — vector of example text to show in the legend next to sizes.legend.labels. By default "Abc". When NA, examples from the data variable whose sizes are close to the sizes.legend are taken and "NA" for classes where no match is found.

n — preferred number of color scale classes. Only applicable when col is a numeric variable name.

style — method to process the color scale when col is a numeric variable. Discrete options are "cat", "fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", and "jenks". A numeric variable is processed as a categorial variable when using "cat", i.e. each unique value will correspond to a distinct category. For the other discrete options, see the details in classIntervals. Continuous options are "cont" and "order". The former maps the values of col to a smooth gradient, whereas the latter maps the order of values of col to a smooth gradient. They are the continuous variants of respectively the discrete methods "equal" and quantile".

breaks — in case style=="fixed", breaks should be specified. The breaks argument can also be used when style="cont". In that case, the breaks are mapped evenly to the sequential or divering color palette.

interval.closure — value that determines whether where the intervals are closed: "left" or "right". Only applicable if col is a numerc variable.

palette — a palette name or a vector of colors. See tmaptools::palette_explorer() for the named palettes. Use a "-" as prefix to reverse the palette. The default palette is taken from tm_layout's argument aes.palette, which typically depends on the style. The type of palette from aes.palette is automatically determined, but can be overwritten: use "seq" for sequential, "div" for diverging, and "cat" for categorical.

labels — labels of the color classes, applicable if col is a data variable name

labels.text — Example text to show in the legend next to the labels. When NA (default), examples from the data variable are taken and "NA" for classes where they don't exist.

auto.palette.mapping — When diverging colour palettes are used (i.e. "RdBu") this method automatically maps colors to values such that the middle colors (mostly white or yellow) are

assigned to values of 0, and the two sides of the color palette are assigned to negative respectively positive values. When categorical color palettes are used, this method stretches the palette if there are more levels than colors.

`contrast` vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when `auto.palette.mapping=TRUE`). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

`max.categories` in case `col` is the name of a categorical variable, this value determines how many categories (levels) it can have maximally. If the number of levels is higher than `max.categories` and `auto.palette.mapping` is `FALSE`, then levels are combined.

`colorNA` colour for missing values. Use `NULL` for transparency.

`textNA` text used for missing values.

`showNA` logical that determines whether missing values are named in the legend. By default (`NA`), this depends on the presence of missing values.

`fontface` font face of the text labels. By default, determined by the fontface argument of `tm_layout`.

`fontfamily` font family of the text labels. By default, determined by the fontfamily argument of `tm_layout`.

`alpha` transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the `fontcolor` is used (normally 1).

`case` case of the font. Use "upper" to generate upper-case text, "lower" to generate lower-case text, and `NA` to leave the text as is.

`shadow` logical that determines whether a shadow is depicted behind the text. The color of the shadow is either white or yellow, depending of the `fontcolor`.

`bg.color` background color of the text labels. By default, `bg.color=NA`, so no background is drawn.

`bg.alpha` number between 0 and 1 that specifies the transparancy of the text background (0 is totally transparent, 1 is solid background).

`size.lowerbound`

lowerbound for `size`. Only applicable when `size` is not a constant. If `print.tiny` is `TRUE`, then all text labels which relative text is smaller than `size.lowerbound` are depicted at relative size `size.lowerbound`. If `print.tiny` is `FALSE`, then text labels are only depicted if their relative sizes are at least `size.lowerbound` (in other words, tiny labels are omitted).

`print.tiny` boolean, see `size.lowerbound`

`scale` text size multiplier, useful in case `size` is variable or `"AREA"`.

`auto.placement` logical (or numeric) that determines whether the labels are placed automatically. If `TRUE`, the labels are placed next to the coordinate points with as little overlap as possible using the simulated annealing algorithm. Therefore, it is recommended for labeling spatial dots or symbols. If a numeric value is provided, this value

acts as a parameter that specifies the distance between the coordinate points and the text labels in terms of text line heights.

`remove.overlap` logical that determines whether the overlapping labels are removed

`along.lines` logical that determines whether labels are rotated along the spatial lines. Only applicabel if a spatial lines shape is used.

`overwrite.lines`
logical that determines whether the part of the lines below the text labels is removed. Only applicabel if a spatial lines shape is used.

`just` justification of the text 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.

`xmod` horizontal position modification of the text (relatively): 0 means no modification, and 1 corresponds to the height of one line of text. Either a single number for all polygons, or a numeric variable in the shape data specifying a number for each polygon. Together with `ymod`, it determines position modification of the text labels. In most coordinate systems (projections), the origin is located at the bottom left, so negative `xmod` move the text to the left, and negative `ymod` values to the bottom.

`ymod` vertical position modification. See xmod.

`title.size` title of the legend element regarding the text sizes

`title.col` title of the legend element regarding the text colors

`legend.size.show`
logical that determines whether the legend for the text sizes is shown

`legend.col.show`
logical that determines whether the legend for the text colors is shown

`legend.format` list of formatting options for the legend numbers. Only applicable if `labels` is undefined. 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 scientically? 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 scientically, 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.

text.separator Character string to use to separate numbers in the legend (default: "to").

text.less.than Character value(s) to use to translate "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.or.more Character value(s) to use to translate "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.align Value that determines how the numbers are aligned, `"left"`, `"center"` or `"right"`. By default `"left"` for legends in portrait format (`legend.is.protrait = TRUE`), and `"center"` otherwise.

text.to.columns Logical that determines whether the text is aligned to three columns (from, text.separator, to). By default `FALSE`.

... Other arguments passed on to `formatC`

`legend.size.is.portrait`
: logical that determines whether the legend element regarding the text sizes is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.col.is.portrait`
: logical that determines whether the legend element regarding the text colors is in portrait mode (`TRUE`) or landscape (`FALSE`)

`legend.size.reverse`
: logical that determines whether the items of the legend regarding the text sizes are shown in reverse order, i.e. from bottom to top when `legend.size.is.portrait = TRUE` and from right to left when `legend.size.is.portrait = FALSE`

`legend.col.reverse`
: logical that determines whether the items of the legend regarding the text colors are shown in reverse order, i.e. from bottom to top when `legend.col.is.portrait = TRUE` and from right to left when `legend.col.is.portrait = FALSE`

`legend.hist` logical that determines whether a histogram is shown regarding the text colors

`legend.hist.title`
: title for the histogram. By default, one title is used for both the histogram and the normal legend for text colors.

`legend.size.z` index value that determines the position of the legend element regarding the text sizes with respect to other legend elements. The legend elements are stacked according to their z values. The legend element with the lowest z value is placed on top.

`legend.col.z` index value that determines the position of the legend element regarding the text colors. (See `legend.size.z`)

`legend.hist.z` index value that determines the position of the histogram legend element. (See `legend.size.z`)

Value

`tmap-element`

Note

The absolute fontsize (in points) is determined by the (ROOT) viewport, which may depend on the graphics device.

See Also

vignette("tmap-nutshell")

Examples

```
current.mode <- tmap_mode("plot")

data(World, Europe, metro)

tm_shape(World) +
    tm_text("name", size="AREA")

tm_shape(Europe) +
    tm_polygons() +
    tm_text("iso_a3", size="AREA", col = "grey20", root=4, shadow = TRUE, scale=2,
        size.lowerbound = .1) +
tm_shape(Europe) +
    tm_text("name", size="AREA", root=4, scale=1,
        ymod=-1 * tmaptools::approx_areas(Europe, target = "norm")^(1/4))

tm_shape(Europe) +
tm_polygons() +
tm_shape(metro) +
tm_bubbles("pop2010", size.lim = c(0, 15e6),
   title.size = "European metropolitan areas") +
tm_shape(metro[metro$pop2010>=2e6, ]) +
tm_text("name", auto.placement = TRUE) +
tm_format_Europe()

tm_shape(World) +
tm_text("name", size="pop_est", col="continent", palette="Dark2",
title.size = "Population", title.col="Continent") +
tm_legend(outside = TRUE)

# restore current mode
tmap_mode(current.mode)
```

---

tm_view *Options for the interactive tmap viewer*

Description

Set the options for the interactive tmap viewer. Some of these options can also be set with tm_layout, since they are style dependent (e.g., the choice of basemaps). The function tm_view overrides these options when specified.

Usage

```
tm_view(alpha = NA, colorNA = NA, basemaps = NA, basemaps.alpha = NA,
  projection = 3857, symbol.size.fixed = FALSE, dot.size.fixed = TRUE,
  text.size.variable = FALSE, set.bounds = FALSE, set.view = NA,
  set.zoom.limits = NA, legend.position = c("right", "top"),
  control.position = c("left", "top"), popup.all.data = NULL,
  bg.overlay = NULL, bg.overlay.alpha = NULL)
```

Arguments

alpha	transparency (opacity) parameter applied to whole map. By default, it is set to 0.7 if basemaps are used, and 1 otherwise.
colorNA	default color for missing values in interactive mode. If the color of missing values is not defined in the layer functions (e.g. tm_fill), then the default color is taken from the na value of the aes.color argument in tm_layout. This colorNA argument (if not NA itself) overrides that default value. For interactive maps, it can be useful to set colorNA to NULL, which means transparent.
basemaps	vector of one or more names of baselayer maps, or a logical value. See http://leaflet-extras.github.io/leaflet-providers/preview. Also supports URL's for tile servers, such as "http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png". By default (NA), the choice of basemap(s) will be determined by the basemaps argument of tm_layout, which is style dependent. Use TRUE to select a large set of recommended basemaps. Use FALSE to omit basemap tiles. If a named vector is provided, the names are used in the layer control legend.
basemaps.alpha	transparency (opacity) value for the basemaps. Can be a vector of values, one for each basemap.
projection	projection. Either a EPSG number, or a leaflet_crs object created with leafletCRS. By default, the Web Mercator (3857) is used, since the vast majority of basemaps are rendered accordingly. Other standards are EPSG numbers 4326 (WGS84) and 3395 (Mercator). If set to 0, the projection of the master shape is used (see tm_shape) provided that a EPSG number can be extracted.
symbol.size.fixed	should symbol sizes be fixed while zooming?
dot.size.fixed	should dot sizes be fixed while zooming?
text.size.variable	should text size variables be allowed in view mode? By default FALSE, since in many applications, the main reason to vary text size is to prevent occlusion in plot mode, which is often not a problem in view mode due to the ability to zoom in.
set.bounds	logical that determines whether maximum bounds are set, or a numeric vector of four values that specify the lng1, lat1, lng2, and lat2 coordinates (see setMaxBounds).
set.view	numeric vector or three that determines the view: lng, lat, and zoom (see setView).
set.zoom.limits	numeric vector of two that set the minimum and maximum zoom levels (see tileOptions).

`legend.position`
: Character vector of two values, specifing the position of the legend. Use "left" or "right" for the first value and "top" or "bottom" for the second value. It overrides the value of `legend.position` of `tm_layout`, unless set to `NA`.

`control.position`
: Character vector of two values, specifing the position of the layer control UI. Use "left" or "right" for the first value and "top" or "bottom" for the second value.

`popup.all.data`
: not used anymore. As of version 1.6, the popups are specified by the argument `popup.vars` in the layer functions `tm_fill`, `tm_symbols`, and `tm_lines`.

`bg.overlay`
: not used anymore as of version 1.7. Instead of an overlay, a background color is set, which is determined by `bg.color` of `tm_layout`, which is style dependent.

`bg.overlay.alpha`
: not used anymore as of version 1.7. Instead of an overlay, a background color is set. The trade-off between background and basemaps can now be set by `basemaps.alpha`

See Also

`tmap_mode` and `vignette("tmap-modes")`

Examples

```
# world choropleth/bubble map of the world
data(World, metro)
metro$growth <- (metro$pop2020 - metro$pop2010) / (metro$pop2010 * 10) * 100

map1 <- tm_shape(metro) +
tm_bubbles("pop2010", col = "growth",
border.col = "black", border.alpha = .5,
style="fixed", breaks=c(-Inf, seq(0, 6, by=2), Inf),
palette="-RdYlBu", contrast=1,
title.size="Metro population",
title.col="Growth rate (%)", id="name",
    popup.vars=c("pop2010", "pop2020", "growth")) +
tm_legend(outside=TRUE)

current.mode <- tmap_mode("plot")

# plot map
map1

# view map with default view options
tmap_mode("view")
map1

# view map with changed view options
map1 + tm_view(alpha = 1, basemaps = "Stamen.Watercolor")

# restore current mode
tmap_mode(current.mode)
```

tm_xlab *Axis labels*

Description

Add axis labels

Usage

```
tm_xlab(text, size = 0.8, rotation = 0)

tm_ylab(text, size = 0.8, rotation = 90)
```

Arguments

text	text for the axis
size	fontsize, by default 0.8
rotation	rotation angle in degrees. By default, 0 for the x axis label and 90 for the y axis label.

Examples

```
data(World)

qtm(World, fill="#FFF8DC", projection="longlat", inner.margins=0) +
tm_grid(x = seq(-180, 180, by=20), y=seq(-90,90,by=10), col = "gray70") +
tm_xlab("Longitude") +
tm_ylab("Latitude")
```

World *World, Europe, and Netherlands map*

Description

Maps of the world, Europe, and the Netherlands (province and municipality level), class SpatialPolygonsDataFrame

Usage

```
data(World)

data(Europe)

data(NLD_prov)

data(NLD_muni)
```

Details

The default projections for these maps are Eckhart IV (World), Lambert azimuthal (Europe), and Rijksdriehoekstelsel (Netherlands). See below. To change the projection, use `set_projection`. Alternatively, the projection can be changed temporarily for plotting purposes by using the projection arugment of `tm_shape` (or `qtm`).

`World` World map. The default projection for this world map is Eckhart IV since area sizes are preserved, which is a very important property for choropleths.

`Europe` Europe map. The ETRS-LAEA projection is used by default for this map. This projection, as well as the bounding box, are also used in the maps published by Eurostat. Several countries are transcontinental and are partly located in Asia. From these countries, only data from Russia and Turkey have been included in this shape file since they are widely considered as European countries. The data is taken from the `World` data, where variables `"part"` and `"EU_Schengen"` have been added.

`NLD_prov` and `NLD_muni`, maps of the Netherlands at province and municipality level of 2013. The used projection is the Rijksdriehoekstelsel projection. **Important:** publication of these maps is only allowed when cited to Statistics Netherlands (CBS) and Kadaster Nederland as source.

Source

`http://www.naturalearthdata.com` for `World` and `Europe`

`http://www.happyplanetindex.org` for `World` and `Europe`

`http://www.cbs.nl` for `NLD_prov` and `NLD_muni`.

References

Statistics Netherlands (2014), The Hague/Heerlen, Netherlands, `http://www.cbs.nl`.

Kadaster, the Netherlands' Cadastre, Land Registry, and Mapping Agency (2014), Apeldoorn, Netherlands, `http://www.kadaster.nl`.

Index