Package 'mapsf'

October 9, 2026

Title Thematic Cartography

Version 1.3.0

Description Create and integrate thematic maps in your workflow. This package helps to design various cartographic representations such as proportional symbols, choropleth or typology maps. It also offers several functions to display layout elements that improve the graphic presentation of maps (e.g. scale bar, north arrow, title, labels). 'mapsf' maps 'sf' objects on 'base' graphics.

License GPL (>= 3)

URL https://riatelab.github.io/mapsf/

BugReports https://github.com/riatelab/mapsf/issues/

Depends R (>= 3.6.0)

Imports classInt, graphics, maplegend (>= 0.6.3), s2, sf, stats, utils, grDevices

Suggests terra, Ckmeans.1d.dp, png, jpeg, knitr, rmarkdown, svglite, tinytest, covr

Encoding UTF-8

VignetteBuilder knitr

Language en-US

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Timothée Giraud [cre, aut] (ORCID: <https://orcid.org/0000-0002-1932-3323>),
Hugues Pecout [ctb] (ORCID: <https://orcid.org/0000-0002-0246-0954>, Logo),
Ronan Ysebaert [ctb] (ORCID: <https://orcid.org/0000-0002-7344-5911>, Cheat sheet),
Elina Marveaux [ctb] (ORCID: <https://orcid.org/0009-0000-8667-3019>, Themes),
Ian Fellows [cph] (No overlap algorithm for labels, from wordcloud package),

Danielle Navarro [cph] (ORCID: <https://orcid.org/0000-0001-7648-6578>,
Bézier curve algorithm for text annotations)

Maintainer Timothée Giraud <timothee.giraud@cnrs.fr>

Repository CRAN

Date/Publication 2026-10-09 12:30:08 UTC

Contents

mapsf . . . 3
mapsf-deprecated . . . 5
mf_annotation . . . 6
mf_arrow . . . 7
mf_background . . . 8
mf_credits . . . 9
mf_distr . . . 10
mf_frame . . . 11
mf_get_borders . . . 12
mf_get_breaks . . . 13
mf_get_links . . . 14
mf_get_mtq . . . 15
mf_get_pal . . . 16
mf_get_pencil . . . 17
mf_get_ratio . . . 18
mf_graticule . . . 19
mf_inset_on . . . 20
mf_label . . . 21
mf_layout . . . 23
mf_legend . . . 24
mf_logo . . . 27
mf_map . . . 28
mf_png . . . 38
mf_raster . . . 39
mf_scale . . . 42
mf_shadow . . . 44
mf_svg . . . 45
mf_text . . . 46
mf_theme . . . 49
mf_title . . . 52
mf_worldmap . . . 53

Index **55**

mapsf *Package description*

Description

Create and integrate thematic maps in your workflow. This package helps to design various cartographic representations such as proportional symbols, choropleth or typology maps. It also offers several functions to display layout elements that improve the graphic presentation of maps (e.g. scale bar, north arrow, title, labels). `mapsf` maps `sf` objects on `base` graphics.

A "Get Started" **vignette** contains commented scripts on how to create various maps: `vignette(topic = "mapsf", package = "mapsf")`

Symbology

These functions display cartographic layers.

- `mf_map()` Plot a map
- `mf_label()` Plot labels
- `mf_raster()` Plot a raster
- `mf_graticule()` Plot graticules

Map layout

These functions are dedicated to the map layout design.

- `mf_theme()` Set a theme
- `mf_shadow()` Plot a shadow
- `mf_background()` Plot a background image
- `mf_annotation()` Plot an annotation
- `mf_arrow()` Plot a north arrow
- `mf_credits()` Plot credits
- `mf_layout()` Plot a map layout
- `mf_title()` Plot a title
- `mf_scale()` Plot a scale bar
- `mf_inset_on()` / `mf_inset_off()` Plot an inset
- `mf_worldmap()` Plot a point on a world map
- `mf_legend()` Plot a legend

Utility functions

- `mf_svg()` Export a map in SVG file format
- `mf_png()` Export a map in SVG file format
- `mf_distr()` Plot a distribution
- `mf_get_links()` Get a link layer from a data.frame of links
- `mf_get_pal()` Get color palettes
- `mf_get_breaks()` Get class intervals
- `mf_get_mtq()` Get the 'mtq' dataset
- `mf_get_ratio()` Get map width and height values
- `mf_get_pencil()` Get a pencil layer from polygons
- `mf_get_borders()` Get a border layer from polygons

Author(s)

Maintainer: Timothée Giraud <timothee.giraud@cnrs.fr> (ORCID)

Authors:

- Timothée Giraud <timothee.giraud@cnrs.fr> (ORCID)

Other contributors:

- Hugues Pecout (ORCID) (Logo) [contributor]
- Ronan Ysebaert (ORCID) (Cheat sheet) [contributor]
- Elina Marveaux (ORCID) (Themes) [contributor]
- Ian Fellows (No overlap algorithm for labels, from wordcloud package) [copyright holder]
- Danielle Navarro (ORCID) (Bézier curve algorithm for text annotations) [copyright holder]

See Also

Useful links:

- https://riatelab.github.io/mapsf/
- Report bugs at https://github.com/riatelab/mapsf/issues/

`mapsf-deprecated` *Deprecated functions in mapsf*

Description

These functions and features still work but will be removed in the next major version of the package, which is scheduled in January 2027.

`mf_map` **sub-functions:**

Instead of using the following deprecated functions, one can use mf_map with the corresponding type:

- `mf_base()` => **base**
- `mf_choro()` => **choro**
- `mf_prop()` => **prop**
- `mf_typo()` => **typo**
- `mf_symb()` => **symb**
- `mf_grad()` => **grad**
- `mf_prop_typo()` => **prop_typo**
- `mf_prop_choro()` => **prop_choro**
- `mf_symb_choro()` => **symb_choro**

`mf_init`**:**

`mf_init` is deprecated. It is possible to use `mf_map` instead (`mf_map(x, type = "base", col = NA, border = NA)`.

It is also possible to use the `extent` argument of mf_map.

`mf_export`**:**

`mf_export` is deprecated, use mf_png or mf_svg instead.

`mf_annotation`**:**

`mf_annotation` is deprecated, use mf_text instead.

Double legends:

The use of separated legends for map types **prop_choro**, **prop_typo** and **symb_choro** is deprecated. Use `leg_pos = NA` and mf_legend if separated legends are needed.

Theming system:

In mf_theme, the following themes are deprecated: "default", "brutal", "ink", "dark", "agolalight", "candy", "darkula", "iceberg", "green", "nevermind", "jsk" and "barcelona".
The following arguments are also deprecated: "bg", "fg", "tab", "pos", "inner", "line", "cex" and "font".

Although the map theming system has been radically changed in version 1.0.0 of the package, you can still use the old themes by referencing them by name. If you need to use the *pre* v1.0.0 default theme, set `x` to "default".

If an old theme is set, only deprecated arguments are used and others are ignored.
If current and deprecated arguments are mixed, only deprecated arguments are used and others are ignored.
All references and usages of the old theming system will be removed in the next major version.

`mf_annotation` *Deprecated - Plot an annotation*

Description

This function is deprecated. Please use mf_text instead. instead.

Plot an annotation on a map.

Usage

```
mf_annotation(
  x,
  txt,
  pos = "topright",
  cex = 0.8,
  col_arrow,
  col_txt,
  halo = FALSE,
  bg,
  s = 1,
  ...
)
```

Arguments

`x`	an sf object with 1 row, a couple of coordinates (c(x, y)) or "interactive"
`txt`	the text to display
`pos`	position of the text, one of "topleft", "topright", "bottomright", "bottomleft" or "center"
`cex`	size of the text
`col_arrow`	arrow color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
`col_txt`	text color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
`halo`	add a halo around the text
`bg`	halo color, hex code or color name given by colors. The default color is the background color (see mf_theme).
`s`	arrow size (min=1)
`...`	further text arguments.

Value

No return value, an annotation is displayed.

Note

Annotations cannot be displayed on unprojected (long/lat) maps.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_text(
  x = mtq[2, ],
  txt = "pos = 'bottomleft'\nline = 2\nclockwise = FALSE",
  pos = "bottomleft",
  offset = 6,
  clockwise = FALSE,
  line = 2,
  box = FALSE
)
mf_text(
  x = mtq[28, ],
  txt = "pos = 'topright'\nline = 3\nclockwise = FALSE",
  pos = "topright",
  offset = 10,
  clockwise = FALSE,
  line = 3,
  halo = TRUE,
  align = "left"
)
```

---

mf_arrow *Plot a north arrow*

Description

Plot a north arrow.

Usage

```
mf_arrow(pos = "topleft", col, cex = 1, adj = c(0, 0), align)
```

Arguments

pos	position. It can be one of 'topleft', 'top','topright', 'right', 'bottomright', 'bottom','bottomleft', 'left', 'interactive' or a vector of two coordinates in map units (c(x, y))
col	arrow color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).

cex	arrow size
adj	adjust the position of the north arrow in x and y directions
align	object of class sf or sfc used to adjust the arrow to the real north

Value

No return value, a north arrow is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_arrow(pos = "topright")
```

---

mf_background *Plot a background image*

Description

Plot a background image on an existing plot

Usage

```
mf_background(filename, ...)
```

Arguments

filename	filename of the background image, PNG or JPG/JPEG format.
...	ignored

Value

No return value, a background image is displayed.

Examples

```
if (require("jpeg")) {
  mtq <- mf_get_mtq()
  mf_map(mtq, col = NA, border = NA)
  mf_background(system.file("img/background.jpg", package = "mapsf"))
  mf_map(mtq, lwd = 3, col = NA, border = "white", add = TRUE)
  mf_credits(
    txt = "Background photo by Noita Digital on Unsplash",
    col = "white"
  )
}
```

`mf_credits` *Plot credits*

Description

Plot credits (sources, author, year...).

Usage

```
mf_credits(
  txt = "Source(s) & Author(s)",
  pos = "bottomleft",
  col,
  cex = 0.6,
  font = 3,
  bg = NA
)
```

Arguments

`txt`	text of the credits, use '\n' to add line breaks
`pos`	position, one of 'bottomleft', 'bottomright' or 'rightbottom'
`col`	color of the text, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
`cex`	size of the text
`font`	font of the text
`bg`	background color, hex code or color name given by colors

Value

No return value, credits are displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_credits(txt = "Author\nSources - Year")
```

`mf_distr` *Plot a distribution*

Description

This function displays the statistical distribution of a variable with a histogram, a box plot, a strip chart and a density curve on the same plot.

This graphic can be useful to choose an appropriate classification method for choropleth maps.

User-defined class boundaries can also be displayed on the plot.

Usage

```
mf_distr(
  x,
  nbins,
  bw,
  breaks,
  pal,
  alpha = 1,
  rev = FALSE,
  main = "Distribution",
  yaxt = TRUE,
  ylab = "Density",
  seed = 46
)
```

Arguments

`x`	a numeric variable
`nbins`	number of bins in the histogram
`bw`	bandwidth of the density curve
`breaks`	a vector of class boundaries. If `breaks` is used, the boxplot is not displayed.
`pal`	a color, a set of colors (hex codes) or a palette name. Palette names can be obtained with hcl.pals. The default color is either the highlight color if `breaks` is not used, or the background color otherwise (see mf_theme).
`alpha`	opacity, in the range [0,1] (0 means transparent and 1 means opaque). Default is set to 1.
`rev`	logical indicating whether the ordering of the colors should be reversed
`main`	plot title
`yaxt`	if FALSE the y axis is not displayed
`ylab`	y axis label
`seed`	seed used to position the points on the strip chart

Value

The number of bins of the histogram and the bandwidth of the density curve are (invisibly) returned in a list.

Examples

```
(mf_distr(rnorm(1000)))
mf_distr(rbeta(1000, .6, 7))
mf_distr(rbeta(1000, 5, .6))
a <- rbeta(1000, .6, 7)
bks <- mf_get_breaks(a, nbreaks = 5, breaks = "quantile")
mf_distr(a, breaks = bks)
mf_distr(a,
  breaks = bks, pal = "Teal", yaxt = FALSE,
  main = 'Classification method : "quantile"'
)
```

---

mf_frame *Plot a frame*

Description

Plot a frame around an existing map.

Usage

```
mf_frame(extent = "map", col, lwd = 1.5, lty = 1, ...)
```

Arguments

extent	type of frame, either 'map' or 'figure'
col	line color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
lwd	line width
lty	line type
...	other arguments from graphics::box()

Value

No return value, a frame is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_title()
mf_frame(extent = "map")
mf_map(mtq)
mf_title()
mf_frame(extent = "figure")
```

`mf_get_borders` *Get a border layer from polygons*

Description

This function extracts borders between contiguous polygons.

Usage

```
mf_get_borders(x)
```

Arguments

x	an sf object of POLYGONS, using a projected CRS

Value

An sf object (MULTILINESTRING) of borders is returned.

Note

If the polygon layer contains topology errors (such as contiguous polygons not sharing exactly the same boundary) the function may not return all boundaries correctly. It is possible to use `st_snap()` or other functions to try and correct these errors.

Examples

```
mtq <- mf_get_mtq()
mtq_b <- mf_get_borders(mtq)
mf_map(mtq)
mf_map(mtq_b, col = 1:5, lwd = 4, add = TRUE)
```

`mf_get_breaks` *Get class intervals*

Description

A function to classify continuous variables.

This function is a wrapper for `classIntervals` with some additional methods.

Usage

```
mf_get_breaks(x, nbreaks, breaks, k = 1, central = FALSE, ...)
```

Arguments

`x`	a vector of numeric values. NA and Inf values are not used in the classification.
`nbreaks`	a number of classes
`breaks`	a classification method; the main methods are "quantile", "equal", "msd", "ckmeans" (natural breaks), "Q6" and "geom". See Details for the full list.
`k`	number of standard deviation for "msd" method (see Details)
`central`	creation of a central class for "msd" method (see Details)
`...`	further arguments of `classIntervals`

Details

`classInt` **methods:**

"fixed", "sd", "equal", "pretty", "quantile", "kmeans", "hclust", "bclust", "fisher", "jenks", "dpih", "headtails", "maximum", and "box" are `classIntervals` methods. You may need to pass additional arguments for some of them.

Natural breaks method:

The "jenks", "fisher" and "ckmeans" methods are based on the same concept of **natural breaks** and and produce similar groupings. The use of "ckmeans" is recommended.

- The "jenks" method produces class boundaries falling on data points and is slow.
- The "fisher" method produces class boundaries located more conveniently between data points, and is faster than the "jenks" method.
- The "ckmeans" method produces exactly the same class boundaries as the "fisher" method, but is much faster. It uses the optimal univariate k-means method from the `Ckmeans.1d.dp` package. If the "ckmeans" method is selected but the `Ckmeans.1d.dp` package is not installed then the "fisher" method is used.

The relative speeds of these three methods may vary depending on the number of data points and the number of classes.

Other methods:

The "msd" method is based on the **mean** and the **standard deviation** of a numeric vector. The `nbreaks` parameter is not relevant, use `k` and `central` instead. `k` indicates the extent of each class in share of standard deviation. If `central=TRUE` then the mean value is the center of a class else the mean is a break value.

The "q6" method uses the following `quantile` probabilities: 0, 0.05, 0.275, 0.5, 0.725, 0.95, 1.

The "Q6" method uses the following `quantile` probabilities: 0, 0.05, 0.25, 0.5, 0.75, 0.95, 1.

The "geom" method is based on a geometric progression along the variable values, all values must be strictly greater than zero.

The "arith" method is based on an arithmetic progression along the variable values.

The "em" method is based on nested averages computation.

Class boundaries:

Breaks defined by a numeric vector or a classification method are left-closed: breaks defined by `c(2, 5, 10, 15, 20)` will be mapped as [2 - 5[, [5 - 10[, [10 - 15[, [15 - 20].

Value

A numeric vector of breaks

See Also

classIntervals

Examples

```
mtq <- mf_get_mtq()
mf_get_breaks(x = mtq$MED, nbreaks = 6, breaks = "quantile")
```

---

mf_get_links *Get a link layer from a data.frame of links*

Description

Create a link layer from a data.frame of links and an sf object.

Usage

```
mf_get_links(x, df, x_id, df_id)
```

Arguments

x	an sf object, a simple feature collection.
df	a data.frame that contains identifiers of starting and ending points.
x_id	name of the identifier variable in x, default to the first column (optional)
df_id	names of the identifier variables in df, character vector of length 2, default to the two first columns. (optional)

Value

An sf object is returned, it is composed of df and the sfc (LINESTRING) of links.

Examples

```
mtq <- mf_get_mtq()
mob <- read.csv(system.file("csv/mob.csv", package = "mapsf"))
# Select links from Fort-de-France (97209))
mob_97209 <- mob[mob$i == 97209, ]
# Create a link layer
mob_links <- mf_get_links(x = mtq, df = mob_97209)
# Plot the links
mf_map(mtq)
mf_map(mob_links, col = "red4", lwd = 2, add = TRUE)
```

---

mf_get_mtq *Get the 'mtq' dataset*

Description

Import the mtq dataset (Martinique municipalities).

Usage

```
mf_get_mtq(x = "polygons")
```

Arguments

x	one of "polygons", "points", "lines"

Details

This a wrapper around `st_read(system.file("gpkg/mtq.gpkg", package = "mapsf"), layer = x, quiet = TRUE)`.

For polygons (municipalities shapes) and points (municipalities centroids):

- **INSEE_COM**: Municipality identifier
- **STATUS**: Municipality administrative status
- **LIBGEO**: Municipality name

- **POP**: Total population, 2015
- **MED**: Median disposable income adjusted per equivalent household member, in euros, 2015
- **CHOM**: Unemployed population, 2015
- **ACT**: Active population, 2015

For lines (professional mobility flows from Fort-de-France to other municipalities):

- **i**: Municipality of residence identifier
- **j**: Municipality of workplace identifier
- **fij**: Flows of workers (employed population, 15 y.o. or more, 2015, only flows \> 100)
- **sj**: Administrative status of the workplace municipality

Value

an sf object

Source

For polygons (municipalities shapes) and points (municipalities centroids):
Base comparateur de territoires (data, upload date: 2018-09-25) & ADMIN EXPRESS-COG (geometry, 2018 edition).
Citation: Insee and IGN, 2018

For lines (professional mobility flows from Fort-de-France to other municipalities):
Flux de mobilité - déplacements domicile-lieu de travail (upload date: 2018-08-01)
Citation: Insee, 2018

Examples

```
mtq <- mf_get_mtq()
```

---

mf_get_pal *Get color palettes*

Description

`mf_get_pal` builds sequential, diverging and qualitative color palettes. Diverging color palettes can be dissymmetric (different number of colors in each of the two gradients).

Usage

```
mf_get_pal(
  n,
  palette,
  alpha = NULL,
  rev = c(FALSE, FALSE),
  neutral,
  breaks,
  mid
)
```

Arguments

n	the number of colors (>= 1) to be in the palette
palette	a valid palette name. See hcl.pals to get available palette names. The name is matched to the list of available palettes, ignoring upper vs. lower case, spaces, dashes, etc. in the matching.
alpha	opacity, in the range [0,1] (0 means transparent and 1 means opaque). Default is set to 1.
rev	logical indicating whether the ordering of the colors should be reversed
neutral	a color, if two gradients are used, the 'neutral' color can be added between them
breaks	a vector of class limit
mid	a numeric value use to divide the palette in two colors

Value

A vector of colors.

Examples

```
cls <- mf_get_pal(n = c(3, 7), palette = c("Reds 2", "Greens"))
plot(1:10, rep(1, 10), bg = cls, pch = 22, cex = 4)
mtq <- mf_get_mtq()
bks <- mf_get_breaks(mtq$MED, breaks = "equal", nbreaks = 8)
pal <- mf_get_pal(
  breaks = bks, mid = 15000,
  palette = c("Dark Mint", "Burg"), neutral = "grey90"
)
mf_map(mtq, "MED", "choro", breaks = bks, pal = pal)
pal <- mf_get_pal(
  breaks = bks, mid = bks[4],
  palette = c("Dark Mint", "Burg")
)
mf_map(mtq, "MED", "choro", breaks = bks, pal = pal)
```

---

mf_get_pencil *Get a pencil layer from polygons*

Description

Create a pencil layer. This function transforms a POLYGON or MULTIPOLYGON sf object into a MULTILINESTRING one.

Usage

```
mf_get_pencil(x, size = 100, buffer = 0, lefthanded = TRUE, clip = FALSE)
```

Arguments

x	an sf object, a simple feature collection (POLYGON or MULTIPOLYGON).
size	density of the penciling. Median number of points used to build the MULTILINESTRING.
buffer	buffer around each polygon. This buffer (in map units) is used to take sample points. A negative value adds a margin between the penciling and the original polygons borders
lefthanded	if TRUE the penciling is done left-handed style.
clip	if TRUE, the penciling is cut by the original polygon.

Value

A MULTILINESTRING sf object is returned.

Examples

```
mtq <- mf_get_mtq()
mtq_pencil <- mf_get_pencil(x = mtq, clip = FALSE)
mf_map(mtq)
mf_map(mtq_pencil, add = TRUE)
```

---

mf_get_ratio *Get map width and height values*

Description

This function is to be used to get width and height values for maps created in reports (*.Rmd, *.qmd).
It uses the width / height ratio of a spatial object bounding box to find a matching ratio for the map.
If width is specified, then height is deduced from the width / height ratio of x, figure margins and title size.
If height is specified, then width is deduced from the width / height ratio of x, figure margins and title size.

Usage

```
mf_get_ratio(x, width, height, expandBB = rep(0, 4), theme = mf_theme())
```

Arguments

x	object of class sf, sfc or SpatRaster
width	width of the figure (inches), use only one of width or height
height	height of the figure (inches), use only one of width or height
expandBB	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of x width (for left and right values) or a share of x height (for bottom and top values).
theme	theme used for the map

Value

Width and height are returned in inches.

Examples

```
mtq <- mf_get_mtq()
mf_get_ratio(x = mtq, width = 5)
```

`mf_graticule` *Plot graticules*

Description

Display graticules and labels on a map.

Usage

```
mf_graticule(
  x,
  col,
  lwd = 1,
  lty = 1,
  expandBB = rep(0, 4),
  label = TRUE,
  pos = c("top", "left"),
  cex = 0.7,
  extent = x,
  bg,
  add = TRUE
)
```

Arguments

`x`	object of class `sf`, `sfc` or `SpatRaster`
`col`	graticules and labels color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
`lwd`	graticules line width
`lty`	graticules line type
`expandBB`	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of `x` width (for left and right values) or a share of `x` height (for bottom and top values).
`label`	whether to add labels (TRUE) or not (FALSE)
`pos`	labels positions ("bottom", "left", "top" and / or "right")
`cex`	labels size

extent	sf object used to define the map extent; defaults to x. extent and x must use the same CRS.
bg	background color of the map, hex code or color name given by colors, ignored if add = TRUE
add	whether to add the layer to an existing plot (TRUE) or not (FALSE)

Value

An (invisible) layer of graticules is returned (LINESTRING).

Use of graticules

From sf::st_graticule(): "In cartographic visualization, the use of graticules is not advised, unless the graphical output will be used for measurement or navigation, or the direction of North is important for the interpretation of the content, or the content is intended to display distortions and artifacts created by projection. Unnecessary use of graticules only adds visual clutter but little relevant information. Use of coastlines, administrative boundaries or place names permits most viewers of the output to orient themselves better than a graticule."

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq, expandBB = c(0, .1, .1, 0))
mf_graticule(mtq)

mf_graticule(
  x = mtq,
  col = "coral4",
  lwd = 2,
  lty = 2,
  expandBB = c(.1, 0, 0, .1),
  label = TRUE,
  pos = c("right", "bottom"),
  cex = .8,
  add = FALSE
)
mf_map(mtq, add = TRUE)
```

---

mf_inset_on *Plot an inset*

Description

This function is used to add an inset map to the current map.

Usage

```
mf_inset_on(x, pos = "topright", cex = 0.2, fig)

mf_inset_off()
```

Arguments

x	an sf object, or "worldmap" to use with mf_worldmap.
pos	position, one of "bottomleft", "left", "topleft", "top", "bottom", "bottomright", "right", "topright"
cex	share of the map width occupied by the inset
fig	coordinates of the inset region (in NDC, see in ?par())

Details

If x is used (with pos and cex), the width/height ratio of the inset will match the width/height ratio of x bounding box.
If fig is used, coordinates (xmin, xmax, ymin, ymax) are expressed as fractions of the mapping space (i.e. excluding margins).
If map layers have to be plotted after the inset (i.e after mf_inset_off()), please use add = TRUE.
It is not possible to plot an inset within an inset.
It is possible to plot anything (base plots) within the inset, not only map layers.

Value

No return value, an inset is initiated or closed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_inset_on(x = mtq[1, ], cex = .2)
mf_map(mtq[1, ])
mf_inset_off()

mf_map(mtq)
mf_inset_on(x = "worldmap", pos = "bottomleft")
mf_worldmap(x = mtq)
mf_inset_off()

mf_map(mtq)
mf_inset_on(fig = c(0, 0.25, 0, 0.25))
mf_map(x = mtq)
mf_inset_off()
```

---

mf_label *Plot labels*

Description

Put labels on a map.

Usage

```
mf_label(
  x,
  var,
  col,
  cex = 0.7,
  overlap = TRUE,
  lines = TRUE,
  halo = FALSE,
  bg,
  r = 0.1,
  q = 1,
  add = TRUE,
  extent = x,
  expandBB = rep(0.04, 4),
  ...
)
```

Arguments

x	object of class sf
var	name of the variable to map
col	labels color, it can be a single color or a vector of colors
cex	labels cex, it can be a single size or a vector of sizes
overlap	if FALSE, labels are moved so they do not overlap.
lines	if TRUE, then lines are plotted between x,y and the word, for those words not covering their x,y coordinate
halo	if TRUE, a 'halo' is displayed around the text and additional arguments bg and r can be modified to set the color and width of the halo.
bg	halo color, it can be a single color or a vector of colors
r	width of the halo, it can be a single value or a vector of values
q	quality of the non overlapping labels placement. Possible values are 0 (quick results), 1 (reasonable quality and speed), 2 (better quality), 3 (insane quality, can take a lot of time).
add	whether to add the labels to an existing plot (TRUE) or not (FALSE)
extent	sf object used to define the map extent; defaults to x. extent and x must use the same CRS.
expandBB	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of x width (for left and right values) or a share of x height (for bottom and top values).
...	further text arguments.

Value

No return value, labels are displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mtq$cex <- c(rep(.8, 8), 2, rep(.8, 25))
mf_label(
  x = mtq, var = "LIBGEO",
  col = "grey10", halo = TRUE, cex = mtq$cex,
  overlap = FALSE, lines = FALSE
)
```

`mf_layout` *Plot a map layout*

Description

Plot a map layout (title, credits, scalebar, north arrow, frame).

This function uses mf_title, mf_credits, mf_scale and mf_arrow with default values.

Usage

```
mf_layout(
  title = "Map Title",
  credits = "Authors & Sources",
  scale = TRUE,
  arrow = TRUE,
  frame = FALSE
)
```

Arguments

`title`	title of the map
`credits`	credits
`scale`	display a scale bar
`arrow`	display an arrow
`frame`	display a frame

Value

No return value, a map layout is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_layout()
```

`mf_legend` *Plot a legend*

Description

Plot different types of legend. The "type" argument defines the legend type. Please note that some arguments are available for all types of legend and some others are only relevant for specific legend types (see Details). `mf_legend()` is a wrapper for `maplegend::leg()`.

Usage

```
mf_legend(
  type,
  val,
  pos = "left",
  pal = "Inferno",
  alpha = 1,
  col = "tomato4",
  inches = 0.3,
  val_max = NULL,
  symbol = "circle",
  self_adjust = FALSE,
  lwd = 0.7,
  border = "#333333",
  pch = seq_along(val),
  cex = rep(1, length(val)),
  title = "Legend Title",
  title_cex = 0.8 * size,
  val_cex = 0.6 * size,
  val_rnd = 0,
  val_dec = ".",
  val_big = "",
  col_na = "white",
  cex_na = 1,
  pch_na = 4,
  no_data = FALSE,
  no_data_txt = "No Data",
  box_border = "#333333",
  box_cex = c(1, 1),
  horiz = FALSE,
  frame_border,
  frame = FALSE,
  bg,
  fg,
  size = 1,
  return_bbox = FALSE,
  adj = c(0, 0)
```

)

Arguments

type	type of legend: • **prop** for proportional symbols, • **choro** for choropleth maps, • **cont** for continuous maps (e.g. raster), • **typo** for typology maps, • **symb** for symbols maps, • **prop_line** for proportional lines maps, • **grad_line** for graduated lines maps, • **histo** for histograms, • **choro_point** for choropleth points maps, • **choro_line** for choropleth lines maps, • **choro_symb** for choropleth on symbols maps • **typo_line** for typology lines maps.
val	vector of value(s) (for "prop" and "prop_line", at least c(min, max) for "cont"), vector of categories (for "symb", "typo", "typo_line"), break labels (for "choro", "choro_point", "choro_line", "choro_symb", and "grad_line"), histogram parameters (for "histo").
pos	position of the legend. It can be one of 'topleft', 'top', 'topright', 'right', 'bottomright', 'bottom','bottomleft', 'left', 'interactive' or a vector of two coordinates in map units (c(x, y)).
pal	a set of colors (hex codes) or a palette name (valid palette names can be obtained with hcl.pals.
alpha	if pal is a valid palette name, the alpha-transparency level in the range [0,1]
col	color of the symbols (for "prop") or color of the lines (for "prop_line" and "grad_line"), hex code or color name given by colors.
inches	size of the largest symbol (radius for circles, half width for squares) in inches
val_max	maximum value corresponding to the largest symbol
symbol	type of symbols, 'circle' or 'square'
self_adjust	if TRUE values are self-adjusted to keep min, max and intermediate rounded values
lwd	width(s) of the symbols borders (for "prop", "symb", "choro_point", "choro_symb"), width of the largest line (for "prop_line"), line width (for "choro_line" and "typo_line"), vector of line widths (for "grad_line")
border	symbol border color(s)
pch	type(s) of the symbols (0:25)
cex	size(s) of the symbols
title	title of the legend
title_cex	size of the legend title

val_cex	size of the values in the legend
val_rnd	number of decimal places of the values in the legend
val_dec	decimal separator
val_big	thousands separator
col_na	color for missing values
cex_na	size of the symbols for missing values
pch_na	type of the symbols for missing values
no_data	if TRUE a "missing value" box is plotted
no_data_txt	label for missing values
box_border	border color of legend boxes
box_cex	width and height size expansion of boxes, histogram circles, squares or lines
horiz	if TRUE plot an horizontal legend
frame_border	border color of the frame
frame	if TRUE the legend is plotted within a frame
bg	background color of the legend, hex code or color name given by colors.
fg	foreground color of the legend, hex code or color name given by colors.
size	size of the legend; 2 means two times bigger
return_bbox	return only bounding box of the legend. No legend is plotted.
adj	adjust the position of the legend in x and y directions

Details

Some arguments are available for all types of legend: `val`, `pos`, `title`, `title_cex`, `val_cex`, `frame`, `bg`, `fg`, `size`, `adj`, `alpha`, `return_bbox`).

Relevant arguments for each specific legend types:

- `mf_legend(type = "prop", val, inches, val_max, symbol, col, lwd, border, val_rnd, val_big, val_dec, self_adjust, horiz)`
- `mf_legend(type = "choro", val, pal, val_rnd, val_big, val_dec, col_na, no_data, no_data_txt, box_border, box_cex, horiz)`
- `mf_legend(type = "cont", val, pal, val_rnd, val_big, val_dec, col_na, no_data, no_data_txt, box_border, box_cex, horiz)`
- `mf_legend(type = "typo", val, pal, col_na, no_data, no_data_txt, box_border, box_cex)`
- `mf_legend(type = "symb", val, pal, pch, cex, lwd, pch_na, cex_na, col_na, no_data, no_data_txt)`
- `mf_legend(type = "prop_line", val, col, lwd, val_rnd, val_big, val_dec)`
- `mf_legend(type = "grad_line", val, col, lwd, val_rnd, val_big, val_dec)`
- `mf_legend(type = "histo", val, pal, box_border, val_rnd, val_big, val_dec)`
- `mf_legend(type = "choro_point", val, pal, symbol, border, cex, val_rnd, val_big, val_dec, col_na, no_data, no_data_txt, horiz)`

- mf_legend(type = "choro_line", val, pal, lwd, val_rnd, val_big, val_dec, col_na, no_data, no_data_txt)
- mf_legend(type = "choro_symb", val, pal, pch, lwd, val_rnd, val_big, val_dec, col_na, no_data, no_data_txt)
- mf_legend(type = "typo_line", val, pal, lwd, col_na, no_data, no_data_txt, box_cex)

Value

No value is returned, a legend is displayed (except if return_bbox is used).

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_legend(type = "prop", pos = "topright", val = c(1, 5, 10), inches = .3)
mf_legend(
  type = "choro", pos = "bottomright", val = c(10, 20, 30, 40, 50),
  pal = hcl.colors(4, "Reds 2")
)
mf_legend(
  type = "typo", pos = "topleft", val = c("A", "B", "C", "D"),
  pal = hcl.colors(4, "Dynamic")
)
mf_legend(
  type = "symb", pos = "bottomleft", val = c("A", "B", "C"),
  pch = 21:23, cex = c(1, 2, 2),
  pal = hcl.colors(3, "Dynamic")
)
mf_legend(
  type = "grad_line", pos = "top", val = c(1, 2, 3, 4, 10, 15),
  lwd = c(0.2, 2, 4, 5, 10)
)
mf_legend(type = "prop_line", pos = "bottom", lwd = 20, val = c(5, 50, 100))
```

---

mf_logo *Plot a logo on a map*

Description

The logo can be a PNG or JPG/JPEG file.

Usage

```
mf_logo(filename, pos = "bottomright", cex = 1, adj = c(0, 0), resize = TRUE)
```

Arguments

filename	filename of the logo image, PNG or JPG/JPEG format.
pos	position of the logo, one of 'topleft', 'top','topright', 'right', 'bottomright', 'bottom', 'bottomleft', 'left' or a vector of two coordinates in map units (c(x, y)). Use 'interactive' to choose the legend position by clicking on the map.
cex	amount by which the logo width should be magnified or reduced relative to the default
adj	adjust the position of the logo in x and y directions
resize	if FALSE, the logo is displayed at its original size in pixels and cex is not used.

Value

No return value, a logo is displayed.

Examples

```
m <- mf_get_mtq()
mf_map(m)
mf_scale()
logo <- system.file("img", "Rlogo.png", package = "png")
mf_logo(logo, pos = "bottomleft", adj = c(0, 4))
mf_credits()
```

---

mf_map *Plot a map*

Description

mf_map() is the main function of the package, it displays map layers on a georeferenced plot.

mf_map() has three main arguments:

- x, an sf object,
- var, the name(s) of a variable(s) to map,
- type, the map type.

Usage

```
mf_map(x, var, type = "base",
       breaks, nbreaks, pal, alpha, rev, inches, val_max, symbol, col,
       lwd_max, val_order, pch, cex, border, lwd, col_na, cex_na, pch_na,
       expandBB, extent, bg, add,
       leg_pos, leg_title, leg_title_cex, leg_val_cex, leg_val_rnd,
       leg_val_dec, leg_val_big, leg_no_data, leg_frame, leg_frame_border,
       leg_horiz, leg_adj, leg_bg, leg_fg, leg_size,
       leg_box_border, leg_box_cex, ...)
```

Arguments

x	object of class sf
var	name(s) of the variable(s) to map (see Details)
type	map type • *base*: base maps • *choro*: choropleth maps • *typo*: typology maps • *prop*: proportional symbols maps • *prop_choro*: proportional symbols with choropleth coloration • *prop_typo*: proportional symbols with typology coloration • *symb*: symbols maps • *grad*: graduated symbols maps • *symb_choro*: symbols with choropleth coloration
breaks	break values or classification method name such as 'quantile', 'equal', 'msd', 'ckmeans' (natural breaks), 'Q6' or 'geom' (see Details)
nbreaks	number of classes.
pal	a set of colors or a palette name (see Details)
alpha	col or pal opacity, in the range [0,1] (0 means transparent and 1 means opaque). Default is set to 1.
rev	if pal is a palette name, whether the ordering of the colors should be reversed (TRUE) or not (FALSE)
inches	size of the largest symbol in inches (radius for circles, half width for squares)
val_max	maximum value corresponding to the largest symbol or line
symbol	type of proportional symbols, either "circle" or "square"
col	fill color of polygons, lines, points, proportional and graduated symbols. It can be a hex code or a color name given by colors.
lwd_max	width of the largest line
val_order	modalities order in the legend, a character vector that matches var modalities. Default to alphabetic order of modalities.
pch	type of point symbol (see Details)
cex	point symbol size, numerical value giving the amount by which symbols should be magnified relative to the default (see Details)
border	border color of polygons, points and symbols, a hex code or a color name given by colors.
lwd	line width of borders of polygons, points and symbols or lines (see Details)
col_na	color for missing values, a hex code or a color name given by colors.
cex_na	point symbol size for missing values on points
pch_na	type of point symbol for missing values on points
expandBB	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of x width (for left and right values) or a share of x height (for bottom and top values).

extent	sf object used to define the map extent; defaults to x. extent and x must use the same CRS.
bg	background color of the map, hex code or color name given by colors, ignored if add = TRUE
add	whether to add the layer to an existing plot or not (see Details)
leg_pos	position of the legend, one of 'topleft', 'top','topright', 'right', 'bottomright', 'bottom', 'bottomleft', 'left' or a vector of two coordinates in map units (c(x, y)). Use NA to avoid plotting the legend, use 'interactive' to choose the legend position by clicking on the map.
leg_title	legend title (see Details)
leg_title_cex	size of the title
leg_val_cex	size of the values
leg_val_rnd	number of decimal places of the values displayed in the legend (see Details)
leg_val_dec	decimal separator
leg_val_big	thousands separator
leg_no_data	label for missing values (see Details)
leg_frame	whether to add a frame to the legend (TRUE) or not (FALSE)
leg_frame_border	border color of the legend frame
leg_horiz	display the legend horizontally. It only applies to *prop* and *choro* map types (see Details)
leg_adj	adjust the position of the legend in x and y directions
leg_bg	color of the legend background
leg_fg	color of the legend foreground
leg_size	size of the legend. Combine this argument with leg_title_cex and leg_val_cex.
leg_box_border	border color of legend boxes (for types related to choropleth and typology)
leg_box_cex	width and height size expansion of boxes (for types related to choropleth and typology)
...	ignored

Details

x an sf object, for *base* map type sfc and sfg objects are also accepted.

var A vector of two names is required for the following map types.
For *prop_choro* map type, the first name refers to the proportional symbols, the second one to the choropleth coloration.
For *prop_typo* map type, the first name refers to the proportional symbols, the second one to the typology coloration.
For *symb_choro* map type the first name refers to the symbols categories, the second one to the choropleth coloration.

breaks It can be either a numeric vector with the actual breaks, or a classification method name. See mf_get_breaks for details on available methods.

pal `pal` can be a palette name (such as "Reds 2" or "Inferno"). These names can be obtained with hcl.pals and an overview of all available palettes is given in hcl.pals examples.
`pal` can be a vector of colors defined by their hex codes or color names given by colors: c("#7F000D", "#A9565A", "#CA9496", "#E2CBCB", "#F1F1F1") or c("red", "yellow", "green", "black")

pch For *base*, *choro*, *typo* and *grad* map types, a single value is needed.
For *symb* and *symb_choro* map types, a vector of point symbols with a length that matches the number of modalities is needed.
See pch for the available symbols.

cex For *base*, *choro*, *typo*, *prop* and *prop_typo* map types, it should be a single value.
For *symb* and *symb_choro* map types, it shoud be a single value or a vector of sizes (the vector length must match the number of modalities).
For the *grad* map type applied to points or polygons, it must be a vector of sizes (the vector length must match the number of classes).

lwd It must be a single value for all map types. The only exception is the *grad* map type applied to lines, for which it should be a vector of line widths. The vector length must match the number of classes.

add The default value is FALSE but if a graphics device is opened (i.e. if a map layer has already been plotted) then the default value switches to TRUE for the following map types: "prop", "symb", "grad", "prop_choro", "prop_typo" and "symb_choro". For clarity, it is better to set the argument explicitly.

leg_title This argument need two values when using *prop_choro*, the first values refers to the proportional symbols legend, the second one to the choropleth legend.
This argument need two values when using *prop_typo*, the first values refers to the proportional symbols legend, the second one to the typology legend.
This argument need two values when using *symb_choro*, the first value refers to the symbols legend, the second one to the choropleth legend.

leg_val_rnd The values are only rounded for the legend and raw values are used on the map.
This argument need two values when using *prop_choro*, the first value refers to the symbols legend, the second one to the choropleth legend.

leg_no_data This argument need two values when using *symb_choro*, the first value refers to the symbols legend, the second one to the choropleth legend.

leg_horiz This argument need two values when using *prop_choro*, the first value refers to the symbols legend, the second one to the choropleth legend.

Default colors and palettes:

Default colors and palettes are defined by the cartographic theme. mf_theme documentation explains how to inspect and modify these default colors and palettes.
Current theme parameters are set in `mapsf` options and are extracted with getOption() in the following section.

Relevant arguments and default values for each map type::

Relevant arguments and default values are different for each map type, and they also may differ given the type of object that is displayed (polygons, lines or points).

Base maps (default):

`mf_map()` can be used to display geographic layers (`sf` objects), using the default map type **base**.
For polygons:

```
mf_map(x, col = getOption("mapsf.foreground"),
       border = getOption("mapsf.highlight"),
       lwd = .7, lty = 1,
       alpha, expandBB, extent, bg, add = FALSE)
```

For points:

```
mf_map(x, col = getOption("mapsf.highlight),
       border = getOption("mapsf.foreground"),
       pch = 20, cex = 1, lwd = .7,
       alpha, expandBB, extent, bg, add = FALSE)
```

For lines:

```
mf_map(x, col = getOption("mapsf.highlight),
       lwd = .7, lty = 1,
       alpha, expandBB, extent, bg, add = FALSE)
```

Choropleth maps:
With the **choro** map type, `mf_map()` displays a choropleth map.
In choropleth maps, areas are shaded according to the variation of a quantitative variable.
They are used to represent ratios or indices.
For polygons:

```
mf_map(x, var, type = "choro",
       breaks = "quantile", nbreaks, pal = getOption("mapsf.pal_seq"),
       rev = FALSE, border = getOption("mapsf.highlight"),
       lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

For points:

```
mf_map(x, var, type = "choro",
       breaks = "quantile", nbreaks, pal = getOption("mapsf.pal_seq"),
       rev = FALSE, border = getOption("mapsf.background"),
       pch = 21, cex = 2, lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

For lines:

```
mf_map(x, var, type = "choro",
       breaks = "quantile", nbreaks, pal = getOption("mapsf.pal_seq"),
       rev = FALSE, lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

Typology maps:
With the **typo** map type, `mf_map()` displays a typology map.
In typology maps, areas are shaded according to the modalities of a qualitative variable.
For polygons:

```
mf_map(x, var, type = "typo",
       pal = getOption("mapsf.pal_quali"), rev = FALSE, val_order,
       border = getOption("mapsf.highlight"), lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

For points:

```
mf_map(x, var, type = "typo",
       pal = getOption("mapsf.pal_quali"), rev = FALSE, val_order,
       border = getOption("mapsf.background"),
       pch = 21, cex = 2, lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

For lines:

```
mf_map(x, var, type = "typo",
       pal = getOption("mapsf.pal_quali"), rev = FALSE,
       val_order, lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add = FALSE, leg_*)
```

Proportional symbols maps:

With the **prop** map type, `mf_map()` displays symbols (squares or circles) with areas proportional to a quantitative variable (stocks).

For polygons, centroids are used to plot proportional symbols.

For polygons and points:

```
mf_map(x, var, type = "prop",
       inches = .3, val_max, symbol = "circle",
       col = getOption("mapsf.highlight),
       border = getOption("mapsf.background), lwd = .7,
       expandBB, extent, bg, alpha, add, leg_*)
```

For lines:

```
mf_map(x, var, type = "prop",
       val_max, lwd_max = 20, col = getOption("mapsf.highlight),
       expandBB, extent, bg, alpha, add, leg_*)
```

Proportional symbols with choropleth coloration maps:

`mf_map()` with **prop_choro** type creates symbols that are proportional to values of a first variable and colored to reflect the classification of a second variable.

This map types uses two variables and some arguments need to be set for both variables (see `leg_title`, `leg_val_rnd` and `leg_horiz` Details).

For polygons, centroids are used to plot proportional symbols. This map type is not available for lines.

For polygons and points:

```
mf_map(x, var, type = "prop_choro",
       inches = .3, val_max,  symbol = "circle",
       pal = getOption("mapsf.pal_seq"), rev = FALSE,
       breaks = "quantile", nbreaks, border = getOption("mapsf.background"),
       lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add, leg_*)
```

Proportional symbols with typology coloration maps:

`mf_map()` with **prop_typo** type creates symbols that are proportional to values of a first variable and colored to reflect the modalities of a second qualitative variable.

This map types uses two variables and some `leg_title` need to be set for both variables (see `leg_title` Details).

For polygons, centroids are used to plot proportional symbols.

For polygons and points:

```
mf_map(x, var, type = "prop_typo",
       inches = .3, val_max, symbol = "circle",
       border = getOption("mapsf.background"),
       pal = getOption("mapsf.pal_quali"), rev = FALSE, val_order,
       lwd = .7, col_na = "white",
       alpha, expandBB, extent, bg, add, leg_*)
```

For lines:

```
mf_map(x, var, type = "typo",
       lwd_max = 15,
       pal = getOption("mapsf.pal_quali"), rev = FALSE, val_order,
       col_na = "white",
       alpha, expandBB, extent, bg, add, leg_*)
```

Symbols maps:
`mf_map()` can use symbols to display qualitative data, using **symb** map type.
For polygons, centroids are used to plot symbols. This map type is not available for lines.
For polygons and points:

```
mf_map(x, var, type = "symb",
       pch, cex = 2, lwd = .7, pal = getOption("mapsf.pal_quali"),
       rev = FALSE, border = getOption("mapsf.background"),
       val_order,
       col_na = "grey", pch_na = 4, cex_na = 1,
       alpha, expandBB, extent, bg, add, leg_*)
```

Graduated symbols maps:
With the **grad** map type, `mf_map()` displays graduated symbols on a map.
Graduated symbols are based on classified quantitative variables.
For polygons, centroids are used to plot graduated symbols.
For polygons and points:

```
mf_map(x, var, type = "grad",
       breaks = "quantile", nbreaks = 3,
       col = getOption("mapsf.highlight"),
       border = getOption("mapsf.background"),
       lwd = .7, pch = 21, cex = seq(1, 4, length.out = nbreaks),
       alpha, expandBB, extent, bg, add, leg_*)
```

For lines:

```
mf_map(x, var, type = "grad",
       breaks = "quantile", nbreaks = 3,
       col = getOption("mapsf.highlight"),
       lwd = seq(1, 4, length.out = nbreaks),
       alpha, expandBB, extent, bg, add, leg_*)
```

Symbols with choropleth coloration maps:
`mf_map()` with **symb_choro** type creates symbols that reflect modalities of a first qualitative variable and colored to reflect the classification of a second variable.
This map types uses two variables and some arguments need to be set for both variables (see `leg_title` and `leg_no_data` Details).
For polygons, centroids are used to plot symbols. This map type is not available for lines.
For polygons and points:

```
mf_map(x, var, type = "symb_choro",
       pch, cex = 2, lwd = .7,
       border = getOption("mapsf.background"), val_order,
       pal = getOption("mapsf.pal_seq"), rev = FALSE,
       breaks = "quantile", nbreaks,
       pch_na = 4, cex_na = 1, col_na = "white",
       alpha, expandBB, extent, bg, add, leg_*)
```

Value

x is (invisibly) returned.

Examples

```
mtq <- mf_get_mtq()
pts <- mf_get_mtq("points")
flows <- mf_get_mtq("lines")
# basic examples
# type = "base"
mf_map(mtq)
# type = "prop"
mf_map(mtq)
mf_map(mtq, var = "POP", type = "prop")
# type = "choro"
mf_map(mtq, var = "MED", type = "choro")
# type = "typo"
mf_map(mtq, "STATUS", "typo")
# type = "symb"
mf_map(mtq)
mf_map(mtq, "STATUS", "symb")
# type = "grad"
mf_map(mtq)
mf_map(mtq, var = "POP", type = "grad")
# type = "prop_choro"
mf_map(mtq)
mf_map(mtq, var = c("POP", "MED"), type = "prop_choro")
# type = "prop_typo"
mf_map(mtq)
mf_map(mtq, var = c("POP", "STATUS"), type = "prop_typo")
# type = "symb_choro
mf_map(mtq)
mf_map(mtq, var = c("STATUS", "MED"), type = "symb_choro")


# Base map type
mf_map(mtq, lty = 3)
mf_map(pts, col = "red", border = "white", pch = 21, add = TRUE)
mf_map(flows, col = "coral", lwd = 2, add = TRUE)

# Choropleth map type
# polygons
mtq[6, "MED"] <- NA
```

```
mf_map(
  x = mtq, var = "MED", type = "choro",
  col_na = "grey90", pal = "Cividis",
  breaks = "equal", nbreaks = 5, border = "white",
  lwd = .5, leg_pos = "topleft",
  leg_title = "Median Income", leg_title_cex = 1,
  leg_val_cex = .9, leg_val_rnd = -2, leg_no_data = "No data",
  leg_box_cex = c(.5, 3), leg_box_border = NA, leg_frame = FALSE
)
# points
mf_map(mtq)
mf_map(
  x = pts, var = "MED", type = "choro",
  pch = 21, cex = 3, lwd = 1.2,
  pal = "Teal", border = "white",
  leg_horiz = FALSE, leg_val_big = " ",
  leg_val_rnd = -2, leg_pos = "topright",
  leg_frame = TRUE, add = TRUE
)
# lines
mf_map(mtq, extent = flows)
mf_map(
  x = flows, var = "fij", type = "choro",
  breaks = "equal", nbreaks = 3, add = TRUE,
  lwd = 5, pal = "Burg", leg_horiz = TRUE,
  leg_box_cex = c(.7, 1),
  leg_val_rnd = 0, leg_pos = "bottomleft"
)

# Typology map type
# polygons
mtq[6, "STATUS"] <- NA
mf_map(
  x = mtq, var = "STATUS", type = "typo",
  col_na = "grey90", border = "white",
  pal = c("#FFE93F", "#00214E", "#7C7C7C"),
  val_order = c("Prefecture", "Sub-prefecture", "Simple municipality"),
  lwd = .5, leg_pos = "bottomleft",
  leg_title = "", leg_title_cex = 1,
  leg_val_cex = .9, leg_no_data = "No data",
  leg_box_cex = c(.5, 3), leg_box_border = NA
)
# points
mf_map(
  x = pts, var = "STATUS", type = "typo",
  cex = 3, pal = "Dark 3", border = "grey",
  leg_pos = "bottomleft"
)
# lines
mf_map(mtq, extent = flows)
mf_map(
  x = flows, var = "sj", type = "typo",
  add = TRUE,
```

```
  lwd = 2, pal = c("red", "blue"),
  leg_pos = "bottomleft"
)

# Proportional symbols map type
mf_map(mtq)
mf_map(
  x = mtq, var = "POP", type = "prop",
  inches = .4, symbol = "circle", val_max = 90000,
  col = "tomato1", border = "blue", lwd = 1,
  leg_pos = "topright", leg_title = "Population",
  leg_title_cex = 1, leg_val_cex = .8, leg_val_rnd = 0,
  leg_frame = TRUE, add = TRUE
)

# Proportional symbols with choropleth coloration map type
mf_map(mtq)
mf_map(
  x = mtq, var = c("POP", "MED"), type = "prop_choro",
  inches = .2,
  val_max = 90000, symbol = "circle",
  col_na = "grey90", pal = "Cividis",
  breaks = "msd", nbreaks = 4, lwd = 1,
  leg_pos = "topright",
  leg_title = c("Population", "Median Income"),
  leg_val_rnd = c(0, 1),
  leg_horiz = c(TRUE, FALSE),
  leg_title_cex = .9,
  leg_val_dec = ",",
  leg_val_cex = .8,
  leg_size = 1,
  add = TRUE
)

# Proportional symbols with typology coloration map type
mf_map(mtq, extent = flows, expandBB = c(0, .5, 0, 0))
mf_map(flows, c("fij", "sj"), "prop_typo",
  val_order = c("Sub-prefecture", "Simple municipality"),
  pal = c("steelblue", "lightblue"), lwd_max = 30,
  leg_pos = "topleft", leg_title = c("commuters", "destination")
)
mf_map(
  x = mtq, var = c("POP", "STATUS"), type = "prop_typo",
  inches = .2, border = "tomato4", lwd = 1,
  pal = c("darkblue", "steelblue", "lightblue"),
  val_order = c("Prefecture", "Sub-prefecture", "Simple municipality"),
  leg_pos = "bottomleft",
  leg_title = c("Population", ""),
  leg_no_data = "No dada",
  add = TRUE
)

# Symbols map type
```

```
mf_map(mtq)
mf_map(mtq, "STATUS", "symb",
  pal = "Berlin", border = "white", lwd = 1,
  cex = c(4, 3, 2), pch = c(21:23), col_na = "red",
  val_order = c("Prefecture", "Sub-prefecture", "Simple municipality"),
  leg_title = ""
)

# Graduated symbols map type
mf_map(mtq, bg = "cornsilk2")
mf_map(flows, "fij", "grad",
  breaks = "geom", nbreaks = 3,
  lwd = c(1, 3, 7),
  leg_title = "N. commuters",
  leg_pos = "bottomleft", leg_val_rnd = 0
)
mf_map(mtq, "POP", "grad",
  breaks = c(686, 5000, 25000, 82502),
  cex = c(1, 2, 4), pch = 22, col = "steelblue",
  leg_title = "Population", leg_pos = "topright",
  leg_frame = TRUE
)

# Symbols with choropleth coloration map type
mf_map(mtq)
mtq$STATUS[4] <- NA
mf_map(mtq, c("STATUS", "MED"),
  type = "symb_choro", lwd = 1,
  pal = "Reds 3", breaks = "quantile", nbreaks = 4,
  cex = c(2, 1, 1), pch = c(20, 21, 23), pch_na = 22,
  leg_pos = "topright", border = "white", col_na = "blue",
  val_order = c("Prefecture", "Sub-prefecture", "Simple municipality")
)
```

`mf_png` *Export a map in PNG format*

Description

Export a map with the extent of a spatial object in PNG format.

PNG is a raster graphics file format and PNG export should be used for maps that do not require further modification.

If `width` is specified, then `height` is deduced from the width/height ratio of `x`. Alternatively, if `height` is specified, then `width` is deduced from the width/height ratio of `x`. This helps to produce maps without too much wasted space.

Use `dev.off()` to finish the export (see Examples).

Usage

```
mf_png(
  x,
  filename = "map.png",
  width,
  height,
  expandBB = rep(0, 4),
  res = 96,
  ...
)
```

Arguments

x	object of class sf, sfc or SpatRaster
filename	path to the exported file
width	width of the figure (pixels)
height	height of the figure (pixels)
expandBB	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of x width (for left and right values) or a share of x height (for bottom and top values).
res	nominal resolution in ppi
...	further parameters

Value

No return value, a PNG device is initiated.

Examples

```
mtq <- mf_get_mtq()
(filename <- tempfile(fileext = ".png"))
mf_png(mtq, filename = filename)
mf_map(mtq)
mf_title()
dev.off()
```

---

mf_raster *Plot a raster*

Description

Plot a raster object (SpatRaster from terra).

Usage

```
mf_raster(
  x,
  type,
  nbreaks,
  breaks = "equal",
  val_order,
  pal,
  alpha = NULL,
  rev = FALSE,
  expandBB = rep(0, 4),
  bg,
  leg_pos = "right",
  leg_title = names(x),
  leg_title_cex = 0.8,
  leg_val_cex = 0.6,
  leg_val_rnd = 1,
  leg_val_dec = ".",
  leg_val_big = "",
  leg_frame = FALSE,
  leg_frame_border,
  leg_horiz = FALSE,
  leg_adj = c(0, 0),
  leg_box_border,
  leg_box_cex = c(1, 1),
  leg_fg,
  leg_bg,
  leg_size = 1,
  add = FALSE,
  ...
)
```

Arguments

x	a SpatRaster
type	type of raster map, one of "continuous", "classes", or "interval". Default type for a numeric and categorial raster are "continuous" and "classes" respectively.
nbreaks	number of classes (for type = "interval" only)
breaks	either a numeric vector with the actual breaks (for type = "continuous" and type = "interval"), or a classification method name (for type = "interval" only; see mf_get_breaks for details).
val_order	modalities order in the legend, a character vector that matches var modalities. Default to alphabetic order of modalities (for type = "classes" only).
pal	a set of colors (hex codes) or a palette name. Palette names can be obtained with hcl.pals. The default palette is the pal_quali palette for type = "classes" and pal_seq otherwise (see mf_theme).

alpha	pal‘ opacity, in the range [0,1] (0 means transparent and 1 means opaque). Default is set to 1.
rev	if pal is a palette name, whether the ordering of the colors should be reversed (TRUE) or not (FALSE)
expandBB	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of x width (for left and right values) or a share of x height (for bottom and top values).
bg	background color of the map, hex code or color name given by colors, ignored if add = TRUE
leg_pos	position of the legend, one of 'topleft', 'top','topright', 'right', 'bottomright', 'bottom', 'bottomleft', 'left' or a vector of two coordinates in map units (c(x, y)). Use NA to avoid plotting the legend, use 'interactive' to choose the legend position by clicking on the map.
leg_title	legend title
leg_title_cex	size of the title
leg_val_cex	size of the values
leg_val_rnd	number of decimal places of the values displayed in the legend
leg_val_dec	decimal separator
leg_val_big	thousands separator
leg_frame	whether to add a frame to the legend (TRUE) or not (FALSE)
leg_frame_border	border color of the legend frame
leg_horiz	display the legend horizontally (for proportional symbols and choropleth types)
leg_adj	adjust the position of the legend in x and y directions
leg_box_border	border color of legend boxes (for types related to choropleth and typology)
leg_box_cex	width and height size expansion of boxes
leg_fg	color of the legend foreground
leg_bg	color of the legend background
leg_size	size of the legend. Combine this argument with leg_title_cex and leg_val_cex.
add	whether to add the layer to an existing plot (TRUE) or not (FALSE)
...	bgalpha, smooth, maxcell or other arguments passed to terra::plotRGB or terra::plot

Value

x is (invisibly) returned.

Examples

```
if (require("terra")) {
  # multi band
  logo <- rast(system.file("ex/logo.tif", package = "terra"))
  mf_raster(logo)
```

```
  # one band
  elev <- rast(system.file("ex/elev.tif", package = "terra"))

  ## continuous
  mf_raster(elev)
  mf_raster(elev,
    pal = "Burg", expandBB = c(.2, 0, 0, 0),
    leg_pos = "bottom", leg_horiz = TRUE
  )

  ## continuous with colors and breaks
  mf_raster(elev,
    type = "continuous",
    breaks = c(141, 400, 547),
    pal = c("darkseagreen1", "black", "red")
  )

  ## interval
  mf_raster(elev,
    type = "interval",
    nbreaks = 5, breaks = "equal", pal = "Teal"
  )

  ## classes
  elev2 <- classify(elev, c(140, 400, 450, 549))
  lev_evel <- data.frame(ID = 0:2, elevation = c("Low", "High", "Very High"))
  levels(elev2) <- lev_evel
  mf_raster(elev2)
  mf_raster(elev2,
    pal = c("salmon4", "olivedrab", "yellow3"),
    val_order = c("Very High", "High", "Low")
  )
}
```

`mf_scale` *Plot a scale bar*

Description

Plot a scale bar.

Usage

```
mf_scale(
  size,
  pos = "bottomright",
  lwd = 1.5,
  cex = 0.6,
  col,
```

```
  crs_units = "m",
  scale_units = "km",
  adj = c(0, 0),
  x
)
```

Arguments

size	size of the scale bar in scale units (scale_units, default to km). If size is not set, an automatic size is used.
pos	position. It can be one of 'bottomright', 'bottomleft', 'interactive' or a vector of two coordinates in map units (c(x, y)).
lwd	line width of the scale bar
cex	size of the scale bar text
col	color of the scale bar (line and text), hex code or color name given by colors. The default color is the highlight color (see mf_theme).
crs_units	units used in the CRS of the currently plotted layer. Possible values are "m" and "ft" (see Details).
scale_units	units displayed in the scale bar. It can be "mi" for miles, "ft" for feet, "m" for meters, or "km" for kilometers (default).
adj	adjust the position of the scale bar in x and y directions
x	object of class crs, sf or sfc. If set, the CRS of x will be used instead of crs_units to define CRS units.

Details

Most CRS use the meter as unit. Some US CRS use feet or US survey feet. If unsure of the unit used in the CRS you can use the x argument of the function. Alternatively, you can use `sf::st_crs(zz, parameters = TRUE)$units_gdal` to see which units are used in the zz layer.

The scale bar cannot be displayed on unprojected (long/lat) maps or on maps without documented CRS.

Value

No return value, a scale bar is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_scale()

library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))[1, ]

nc_foot <- st_transform(nc, 2264) # NC state plane, US foot
mf_map(nc_foot)
```

```
mf_scale(size = 5, crs_units = "ft", scale_units = "mi")
mf_map(nc_foot)
mf_scale(size = 5, x = nc_foot, scale_units = "mi")

nc_meter <- st_transform(nc, 32119) # NC state plane, m
mf_map(nc_meter)
mf_scale(size = 5, crs_units = "m", scale_units = "mi")
mf_scale(size = 5, crs_units = "m", scale_units = "km", pos = "bottomleft")
```

`mf_shadow` *Plot a shadow*

Description

Plot the shadow of a polygon layer.

Usage

```
mf_shadow(
  x,
  col,
  cex = 1,
  add = FALSE,
  extent = x,
  bg,
  expandBB = rep(0.04, 4)
)
```

Arguments

`x`	an sf or sfc polygon object
`col`	shadow color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
`cex`	shadow extent
`add`	whether to add the layer to an existing plot (TRUE) or not (FALSE)
`extent`	`sf` object used to define the map extent; defaults to `x`. `extent` and `x` must use the same CRS.
`bg`	background color of the map, hex code or color name given by colors, ignored if `add = TRUE`
`expandBB`	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of `x` width (for left and right values) or a share of `x` height (for bottom and top values).

Value

x is (invisibly) returned.

Examples

```
mtq <- mf_get_mtq()
mf_shadow(mtq)
mf_map(mtq, add = TRUE)
```

`mf_svg` *Export a map in SVG format*

Description

Export a map with the extent of a spatial object in SVG format.

SVG export is the perfect solution for editing maps with desktop vector graphics software. SVG is a vector graphics file format.

If `width` is specified, then `height` is deduced from the width/height ratio of `x`. Alternatively, if `height` is specified, then `width` is deduced from the width/height ratio of `x`. This helps to produce maps without too much wasted space.

Use 'dev.off() to finish the export (see Examples).

Usage

```
mf_svg(
  x,
  filename = "map.svg",
  width,
  height,
  expandBB = rep(0, 4),
  svglite = TRUE,
  ...
)
```

Arguments

`x`	object of class `sf`, `sfc` or `SpatRaster`
`filename`	path to the exported file
`width`	width of the figure (inches)
`height`	height of the figure (inches)
`expandBB`	expansion of the map area in each direction (bottom, left, top, right). The expansion is expressed as a share of `x` width (for left and right values) or a share of `x` height (for bottom and top values).
`svglite`	if TRUE, the export is done with the `svglite` package if it is installed (see Details)
`...`	further parameters

Details

The default driver for building SVG files, `grDevices::svg()`, has limitations regarding speed, file size, editability, and font support. The `svglite` package aims to solve these issues but it is not lightweight in terms of dependencies, so it is not imported by `mapsf`, but rather suggested.

However, we strongly recommend its use if the aim is to edit the maps after export.

Value

No return value, an SVG device is initiated.

Examples

```
mtq <- mf_get_mtq()
(filename <- tempfile(fileext = ".svg"))
mf_svg(mtq, filename = filename)
mf_map(mtq)
mf_title()
dev.off()
```

---

mf_text *Plot a text*

Description

Plot a text on the map.

Usage

```
mf_text(
  x,
  txt = "Text",
  cex = 0.8,
  col_txt,
  pos = "center",
  offset = 0,
  align = "center",
  font = 1,
  family = "sans",
  halo = FALSE,
  col_halo,
  cex_halo = cex,
  box = FALSE,
  col_box,
  col_box_border,
  lwd_box = 2,
  line = 0,
  clockwise = TRUE,
```

```
  lwd = 2,
  col_line,
  arrow = TRUE,
  cex_arrow = 1,
  adj = c(0, 0)
)
```

Arguments

x	an sf object (the first row is used), a couple of coordinates (c(x, y)), a position (one of 'topleft', 'top','topright', 'right', 'bottomright', 'bottom', 'bottomleft', 'left'), or "interactive" for interactive placement.
txt	text to display
cex	text size
col_txt	text color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
pos	position of the text relative to x (one of 'center', 'topleft', 'top', 'topright', 'right', 'bottomright', 'bottom', 'bottomleft', 'left'). Ignored if x is a position.
offset	offset between the text and x. Ignored if x is a position.
align	text alignement, one of "left", "right" or "center"
font	text font
family	text family
halo	add a halo around the text
col_halo	halo color, hex code or color name given by colors. The default color is the background color (see mf_theme).
cex_halo	halo width
box	add a box around the text
col_box	box color, hex code or color name given by colors. The default color is the background color (see mf_theme).
col_box_border	box border color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
lwd_box	line width of the box border
line	type of the line drawn between the text and x. 0 for no line, 1 for a straight line, 2 for a line with an angle, 3 for a curved line. Ignored if x is a position.
clockwise	direction of the curve for types 2 and 3
lwd	line width
col_line	line color, hex code or color name given by colors. The default color is the highlight color (see mf_theme).
arrow	add an arrow to the line
cex_arrow	arrow size
adj	adjust the text position in x and y directions

Value

No return value, an text is displayed.

Examples

```
library(mapsf)
mf_theme("base")
mtq <- mf_get_mtq()
mtq_p <- mf_get_mtq("points")
mf_map(mtq, expandBB = c(0, .1, 0, .3))
mf_map(mtq_p[c(2, 3, 17, 28), ],
  pch = 4, lwd = 1.5, cex = .5, col = "red",
  add = TRUE
)
mf_title("Title of the map", banner = TRUE)
mf_frame()
mf_text(x = "topright", txt = "x = 'topright'")
mf_text(
  x = "bottomleft",
  txt = "x = 'bottomleft'\nadj = c(4,4)\nalign = 'left'",
  adj = c(4, 4),
  align = "left"
)
mf_text(
  x = c(728000, 1625500),
  txt = "x = c(X, Y)",
  pos = "right",
  line = 2,
  offset = 5
)
mf_text(
  mtq[3, ],
  txt = "pos = 'top'\nhalo = TRUE",
  pos = "top",
  align = "center",
  halo = TRUE
)
mf_text(
  mtq[3, ],
  txt = "pos = 'bottomleft'\nhalo = TRUE\nalign = 'right'",
  pos = "bottomleft",
  align = "right",
  halo = TRUE
)
mf_text(
  x = mtq[17, ],
  txt = "pos = 'bottomright'\nline = 1\nbox = TRUE",
  pos = "bottomright",
  offset = 10,
  line = 1,
  box = TRUE
)
```

```
mf_text(
  x = mtq[17, ],
  txt = "pos = 'topright'\nline = 1\nalign = 'left'",
  pos = "topright",
  offset = 15,
  align = "left",
  line = 1
)
mf_text(
  x = mtq[2, ],
  txt = "pos = 'topleft'\nline = 2\nbox = TRUE\nclockwise = TRUE",
  pos = "topleft",
  offset = 8,
  clockwise = TRUE,
  line = 2,
  box = TRUE
)
mf_text(
  x = mtq[2, ],
  txt = "pos = 'bottomleft'\nline = 2\nclockwise = FALSE",
  pos = "bottomleft",
  offset = 6,
  clockwise = FALSE,
  line = 2,
  box = FALSE
)
mf_text(
  x = mtq[28, ],
  txt = "pos = 'topright'\nline = 3\nclockwise = FALSE",
  pos = "topright",
  offset = 10,
  clockwise = FALSE,
  line = 3,
  halo = TRUE,
  align = "left"
)
mf_text(
  x = mtq[28, ],
  txt = "pos = 'right'\nline = 3\nbox = TRUE\nclockwise = TRUE",
  pos = "right",
  offset = 10,
  clockwise = TRUE,
  line = 3,
  box = TRUE
)
```

---

mf_theme *Set a theme*

Description

A theme is a set of graphical parameters that are applied to maps created with `mapsf`. These parameters are:

- figure margins and frames,
- background, foreground and highlight colors,
- default sequential and qualitative palettes,
- title options (position, size, banner...).

`mapsf` offers some builtin themes. It's possible to modify an existing theme or to start a theme from scratch. It is also possible to set a custom theme using a list of arguments

Themes are persistent across maps produced by `mapsf` (e.g. they survive a `dev.off()` call).

Current theme parameters are set in `mapsf` options and nammed according to the following convention: "mapsf.mf_theme_arg_name". Use `getOption()` to return the value of a specific argument of the current theme (see examples).

Use `mf_theme(NULL)` or `mf_theme('base')` to reset to default theme settings.

Usage

```
mf_theme(
  x,
  mar,
  foreground,
  background,
  highlight,
  title_tab,
  title_pos,
  title_inner,
  title_line,
  title_cex,
  title_font,
  title_banner,
  frame,
  frame_lwd,
  frame_lty,
  pal_quali,
  pal_seq,
  ...
)
```

Arguments

`x`	name of a map theme. One of 'base', 'grey', 'sol_light', 'sol_dark', 'mint', 'dracula', 'rzine', 'pistachio'. The default theme is 'base'.
`mar`	a numeral vector of the form c(bottom, left, top, right) which gives the margin size specified in number of lines
`foreground`	foreground color, hex code or color name given by colors.

background	background color, hex code or color name given by colors.
highlight	highlight color, hex code or color name given by colors.
title_tab	if TRUE the title is displayed as a 'tab'
title_pos	title position, one of 'left', 'center', 'right'
title_inner	if TRUE the title is displayed inside the plot area; if FALSE the title is displayed in the top margin
title_line	number of margin lines used for the title
title_cex	size of the title
title_font	font of the title
title_banner	if TRUE the title is displayed as a banner
frame	either "none", "map" or "figure"; plot a frame around the map or the figure.
frame_lwd	line width for the frame
frame_lty	line type for the frame
pal_quali	default qualitative color palette (name or function). Palette names can be obtained with hcl.pals.
pal_seq	default sequential color palette (name or function). Palette names can be obtained with hcl.pals.
...	deprecated arguments ('bg', 'fg', 'tab', 'pos', 'inner', 'line', 'cex' and 'font'). See the Note section.

Value

mf_theme (invisibly) returns the list of current theme parameters.

Note

The following themes are deprecated: "default", "brutal", "ink", "dark", "agolalight", "candy", "darkula", "iceberg", "green", "nevermind", "jsk" and "barcelona".
The following arguments are deprecated: "bg", "fg", "tab", "pos", "inner", "line", "cex" and "font".

Although the map theming system has been radically changed in version 1.0.0 of the package, you can still use the old themes by referencing them by name. If you need to use the *pre* v1.0.0 default theme, set x to "default".
If an old theme is set, only deprecated arguments are used and others are ignored.
If current and deprecated arguments are mixed, only deprecated arguments are used and others are ignored.
All references and usages of the old theming system will be removed in the next major version.

Examples

```
mtq <- mf_get_mtq()

# Choosing a theme by name:
mf_theme("base")
mf_map(mtq)
```

```
mf_title()

# Specifying some values directly:
mf_theme(title_banner = TRUE)
mf_map(mtq)
mf_title()

# Using a mix of the above:
mf_theme("sol_dark", title_tab = TRUE, title_font = 1)
mf_map(mtq)
mf_title()

# Specifying a list with theme values:
theme <- list(
  mar = c(1, 1, 3, 1),
  title_tab = FALSE,
  title_pos = "left",
  title_inner = FALSE,
  title_line = 2,
  title_cex = 1.5,
  title_font = 2,
  title_banner = FALSE,
  frame = "figure",
  frame_lwd = 1,
  frame_lty = 1,
  foreground = "#fbfbfb",
  background = "grey75",
  highlight = "#0f5027",
  pal_quali = "Dark 3",
  pal_seq = "Greens"
)
mf_theme(theme)
mf_map(mtq, "MED", "choro")
mf_title()

# Obtaining a list of parameters for the current theme:
current_theme <- mf_theme()

# Obtaining individual parameters for the current theme:
getOption("mapsf.highlight")
getOption("mapsf.pal_seq")


# Use default theme:
mf_theme(NULL)
# or
mf_theme("base")
```

---

mf_title *Plot a title*

Description

Default values for `mf_title()` arguments are set with the map theme (see mf_theme).

Usage

```
mf_title(txt = "Map Title", pos, tab, bg, fg, cex, line, font, inner, banner)
```

Arguments

`txt`	title text
`pos`	position, one of 'left', 'center', 'right'
`tab`	if TRUE the title is displayed as a tab
`bg`	background color of the title, hex code or color name given by colors
`fg`	foreground color of the title, hex code or color name given by colors
`cex`	size of the title
`line`	number of margin lines used for the title
`font`	font of the title
`inner`	if TRUE the title is displayed inside the plot area; if FALSE the title is displayed in the top margin
`banner`	if TRUE the title is displayed as a banner

Value

No return value, a title is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_map(mtq)
mf_title()
```

`mf_worldmap` *Plot a point on a world map*

Description

Plot a point on a world map.

Usage

```
mf_worldmap(
  x,
  lon,
  lat,
  water_col = "lightblue",
  land_col = "grey60",
  border_col = "grey40",
  border_lwd = 0.8,
  ...
)
```

Arguments

x	object of class sf or sfc
lon	longitude
lat	latitude
water_col	color of the water, hex code or color name given by colors
land_col	color of the land, hex code or color name given by colors
border_col	color of the borders, hex code or color name given by colors
border_lwd	width of the borders, hex code or color name given by colors
...	further parameters related to the plotted point aspect (cex, pch, col...)

Value

No return value, a world map is displayed.

Examples

```
mtq <- mf_get_mtq()
mf_worldmap(mtq)
mf_worldmap(lon = 24, lat = 39)
mf_worldmap(
  lon = 106, lat = 26,
  pch = 4, lwd = 3, cex = 2, col = "tomato4",
  water_col = "#232525", land_col = "#A9B7C6",
  border_col = "white", border_lwd = 1
)
```

Index

classIntervals, *13, 14*
colors, *6, 7*, *9*, *11*, *19, 20*, *25, 26*, *29–31*, *41*, *43, 44*, *47*, *50, 51*, *53, 54*

graphics::box(), *11*

hcl.pals, *10*, *17*, *25*, *31*, *40*, *51*

mapsf, 3
mapsf-deprecated, 5
mapsf-package *(*mapsf*)*, 3
mf_annotation, 6
mf_annotation(), *3*
mf_arrow, 7, *23*
mf_arrow(), *3*
mf_background, 8
mf_background(), *3*
mf_credits, 9, *23*
mf_credits(), *3*
mf_distr, 10
mf_distr(), *4*
mf_frame, 11
mf_get_borders, 12
mf_get_borders(), *4*
mf_get_breaks, 13, *30*, *40*
mf_get_breaks(), *4*
mf_get_links, 14
mf_get_links(), *4*
mf_get_mtq, 15
mf_get_mtq(), *4*
mf_get_pal, 16
mf_get_pal(), *4*
mf_get_pencil, 17
mf_get_pencil(), *4*
mf_get_ratio, 18
mf_get_ratio(), *4*
mf_graticule, 19
mf_graticule(), *3*
mf_inset_off *(*mf_inset_on*)*, 20
mf_inset_off(), *3*
mf_inset_on, 20
mf_inset_on(), *3*
mf_label, 21
mf_label(), *3*
mf_layout, 23
mf_layout(), *3*
mf_legend, *5*, 24
mf_legend(), *3*
mf_logo, 27
mf_map, *5*, 28
mf_map(), *3*
mf_png, *5*, 38
mf_png(), *4*
mf_raster, 39
mf_raster(), *3*
mf_scale, *23*, 42
mf_scale(), *3*
mf_shadow, 44
mf_shadow(), *3*
mf_svg, *5*, 45
mf_svg(), *4*
mf_text, *5, 6*, 46
mf_theme, *5–7*, *9–11*, *19*, *31*, *40*, *43, 44*, *47*, 49, *53*
mf_theme(), *3*
mf_title, *23*, 52
mf_title(), *3*
mf_worldmap, *21*, 53
mf_worldmap(), *3*

pch, *31*

quantile, *14*

sf::st_graticule(), *20*

text, *6*, *22*