

Package ‘bfsMaps’

July 22, 2025

Type Package

Title Plot Maps from Switzerland by Swiss Federal Statistical Office

Version 1.99.3

Date 2023-06-30

Description At the Swiss Federal Statistical Office (SFSO), spatial maps of Switzerland are available free of charge as 'Cartographic bases for small-scale thematic mapping'. This package contains convenience functions to import ESRI (Environmental Systems Research Institute) shape files using the package 'sf' and to plot them easily and quickly without having to worry too much about the technical details.

It contains utilities to combine multiple areas to one single polygon and to find neighbours for single regions. For any point on a map, a special locator can be used to determine to which municipality, district or canton it belongs.

Depends base, stats, R (>= 4.0.0), DescTools

Imports graphics, grDevices, sf

Suggests R.rsp

License GPL (>= 2)

LazyLoad yes

LazyData yes

URL <https://github.com/AndriSignorell/bfsMaps/>

BugReports <https://github.com/AndriSignorell/bfsMaps/issues>

RoxygenNote 6.1.1

Encoding UTF-8

VignetteBuilder R.rsp

NeedsCompilation no

Author Andri Signorell [aut, cre],
Juerg Guggenbuehl [ctb]

Maintainer Andri Signorell <andri@signorell.net>

Repository CRAN

Date/Publication 2023-06-30 22:20:02 UTC

Contents

bfsMaps-package	2
AddWaters	6
BfSStamp	7
CombinePolygons	8
d.bfsrg	9
DownloadBfSMaps	11
GetMap	12
kt	13
Neighbours	13
Plot Swiss Regions	14
PlotCH	18
PlotMapDot	20
PlotPremReg	21
SwissLocator	23
tkart	23
Index	24

bfsMaps-package	<i>Plotting Switzerland Maps from the Swiss Federal Statistical Office (SFSO)</i>
-----------------	---

Description

This package contains convenience functions for plotting Switzerland maps distributed free of charge by the Swiss Federal Office of Statistics (SFSO). It uses the package 'sf' for reading and plotting ESRI (Environmental Systems Research Institute) shapefiles.

Details

The generation of spatial images with maps normally requires several steps, which makes the handling for occasional users complex and confusing. Functions on a higher level of abstraction simplify the daily work. The purpose is to allow the user to get to the desired map as quickly and easily as possible.

The idea behind the functions is to load the specific map, assign the desired color to the regions and create the plot. The arguments are kept straightforward, what is needed is a vector with the specific ids of the regions and an equally sized vector for the colors.

There are specific functions for the most important spatial divisions in Switzerland. Cantons can be plotted with [PlotKant\(\)](#), political municipalities with [PlotPolg\(\)](#), large regions with [PlotGreg\(\)](#) and districts with [PlotBezK\(\)](#). Lakes and rivers in multiple categories can be added to existing images with [AddLakes\(\)](#), [AddRivers\(\)](#) or [AddWaters\(\)](#).

Before the maps can be drawn, a few preparations must be made:

- download the maps following the link 'Swiss Federal Office of Statistics - Base maps' (below) and unzip them into a folder
- declare the location as options(bfsMaps.base = "//path_to_my_maps/")

- names and shortnames of the maps are stored in a file named 'maps.csv', which can be stored either in the 'bfsMaps.base' folder or alternatively in the packages installation folder. An example file for the last map edition ('ThemaKart map boundaries - Set 2023') is included in the package and can be found in the packages .../extdata folder. If you are using a different edition, you have to adjust the file accordingly.

Author(s)

Andri Signorell <andri@signorell.net>

References

Swiss Federal Office of Statistics - Base maps: <https://www.bfs.admin.ch/bfs/de/home/statistiken/regionalstatistik/kartengrundlagen.html>

Swiss Federal Office of Statistics - Spatial divisions: <https://www.agvchapp.bfs.admin.ch/de/typologies/query>

Official directory of towns and cities (PLZ): <https://www.swisstopo.admin.ch/de/geodata/amtliche-verzeichnisse/ortschaftenverzeichnis.html>

Swiss Premium Regions: <https://www.priminfo.admin.ch/>

Examples

```
# Note:
# The examples can not be run without the map data being installed before!

try( {

# PlotKant simply tasks for the id and the color of the spatial region
# labels can be directly placed
PlotKant(id=c("ZH", "FR"), col=c("yellow","limegreen"), label=TRUE)
PlotKant(id="GR", col="orange", label=TRUE, add=TRUE)
AddLakes()
title("Switzerland with some cantons")
# mark the national border
PlotCH(col=NA, add=TRUE, lwd=2)

# The maps have all a general area and a vegetational area
PlotKant(c("VS", "BE"), SetAlpha(c("yellow","limegreen"),.50),
        col.vf=c("yellow","limegreen"), label=TRUE)

# The function returns the centroid points of the objects, which can be used
# to label the plot afterwards
xy <- PlotGreg(c(3,6), SetAlpha(c("plum1", "lightslateblue"),.50),
              col.vf=c("plum1", "lightslateblue"), labels=NA)
AddLakes()
BoxedText(xy$x, xy$y, labels = c("here", "there"), border=NA,
          col = SetAlpha("white", 0.8))
```

```

# Plot political communities
PlotPolg(border="grey85" )
PlotBezK(border="grey55", add=TRUE )
PlotKant(border="black", lwd=1, add=TRUE)
AddLakes()
AddRivers()

# Cantonal capitals
points(sf::st_coordinates(GetMap("stkt.pnt")$geometry),
       pch=21, col="grey", bg="red")

# Display vegetational area
PlotCH(col="wheat3", col.vf="wheat", border="wheat3", main="CH Vegetation Area")
AddRivers()
AddLakes()
PlotKant(col=NA, border="wheat4", add=TRUE, lwd=1)

# Use extended spatial divisions (language regions)
cols <- c("peachpuff2", "gainsboro", "honeydew3", "lightgoldenrodyellow")
PlotPolg(d.bfsrg$gem_id, col=cols[d.bfsrg$sprgeb_c], border="grey70",
         main="Language CH" )
PlotBezK(d.bfsrg$bez_k_c, col=NA, border="grey40", add=TRUE)
AddLakes(col="lightsteelblue1", border="lightskyblue" )
legend(x="topleft", legend=c("german", "french", "italian", "romanche"), bg="white",
       cex=0.8, fill= cols )

# Swiss premiumregions demonstrating combinations of polygons
PlotCH(col="white", main="Premiumregions CH")

plot(CombinePolg(id=d.bfsrg$gem_id, g=d.bfsrg$preg_c),
     col=c("white", "olivedrab4", "olivedrab3", "olivedrab2"), add=TRUE)

legend(x="topleft", fill=c("white", "olivedrab4", "olivedrab3", "olivedrab2"), cex=0.8,
       legend=c("Region 0", "Region 1", "Region 2", "Region 3") )

PlotKant(col=NA, border="grey40", add=TRUE)
AddLakes()

# Cities
cols <- as.vector(sapply(c(hred, hblue, hyellow),
                        SetAlpha, alpha=c(1, 0.7, 0.5)))

old <- Mar(right=20)
PlotPolg(id=d.bfsrg$gem_id, col=cols[as.numeric(d.bfsrg$gem_typ9_x)],
         border="grey70")
AddLakes(col="grey90", border="grey50")
PlotKant(add=TRUE, col=NA, border="grey30")
legend(x=2854724, y=1292274, fill=cols, border=NA, box.col=NA,
       y.intersp=c(1,1,1, 1.1,1.05,1.05, 1.1,1.07,1.07),
       legend=StrTrunc(levels(d.bfsrg$gem_typ9_x), 50),

```

```

      xjust=0, yjust=1, cex=0.8, xpd=NA)
par(mar=old)

# Degree of urbanisation
PlotPolg(col=SetAlpha(c(hred, hblue, hyellow), 0.8)[as.numeric(d.bfsrg$ddegurba_x)],
        main="Degree of Urbanisation 2022")
PlotKant(add=TRUE, border="grey30")
AddLakes(col = "grey90", border = "grey50")

# get cantons' area
area <- sf::st_area(GetMap("kant.map")) / 1E6

# plot cantons
xy <- PlotKant(col=colorRampPalette(c("white", "steelblue"),
                                   space = "rgb")(720)[trunc(area)/10],
              main=expression(paste( "Cantons' area in ", km^2)) )
AddLakes(col="grey90", border="grey60")
text(xy, labels=round(area,1), cex=0.7)

kant.gr <- GetMap("kant.map") |> (\. ) [.$name=="Graubünden", "geometry"]()
# prepare plot
plot(kant.gr, asp=1, axes=FALSE, xlab="", ylab="",
     main="Beautiful Grisons", col="steelblue", lwd=2)

loctext <- function(x, y, text){
  points(x, y, pch=15, col="lightgrey" )
  text(x, y, text, adj=c(0,0.5), col="white", font=2)
}
# the new swiss coordinates LV95 are:  x_new = x_old + 2e6, y_new = y_old + 1e6
loctext(2782783, 1185993, " Davos")
loctext(2761412, 1176112, " Valbella")
loctext(2784192, 1152424, " St. Moritz")
loctext(2714275, 1175027, " Rabiuss")

# Swiss metropolitan areas
cols <- c("royalblue1", "red", "bisque3", "yellow", "orange", "beige")
# we have to prepare the background here, for some reasons...
PlotCH(col="darkolivegreen1", border="grey", lwd=2, main="Swiss metropolitan areas")
# require other map
metr.map <- GetMap("metr.map")
plot(metr.map$geometry, add=TRUE, border="grey60", col=cols)
AddLakes(col="grey90", border="grey70")
legend( x="topleft", legend=c("Ländliche Gemeinde", metr.map$name),
       fill=c("darkolivegreen1", cols),
       bg="white", cex=0.8, xpd=TRUE )

# We can find the neighbor cantons, here for the canton Glarus (id=8)
nbs <- Neighbours(map=GetMap("kant.map"), id=8)

```

```
PlotKant(id = c(8, nbs), col=c("steelblue", rep("grey80", length(nbs))),
        main="Find Neighbours")

})
```

AddWaters

Add Waters to Switzerland Map

Description

Add lakes and rivers to an already existing Switzerland map. The lakes are defined in 2 categories 1 and 2, whereas category 1 contains the bigger ones, category 2 the smaller ones. The rivers are defined in 5 categories 1:5, whereas category 1 contains the largest rivers, category 5 the smallest ones.

Usage

```
AddWaters(lakes = 1, rivers = 1:5, col = NULL,
          border = "lightskyblue3", lwd = 1, ...)

AddLakes(categ = 1:2, col = "lightskyblue1", border = "lightskyblue3",
        lwd = 1, ...)

AddRivers(categ = 1:5, col = "lightskyblue3", ...)
```

Arguments

categ	category of the lakes (1, 2) and rivers (1:5) . 1 are the biggest waters, 2, resp. 5 the smallest ones.
lakes	the category for the lakes
rivers	the category for the rivers
col	color of the lakes, defaults to "lightskyblue1"
border	bordercolor of the lakes, defaults to "lightskyblue3"
lwd	linewidth of border
...	the dots are passed to the plot command

Details

AddWaters() is a wrapper with sensible defaults. If the color is not provided it will be set to a less intense tint of the border.

Lakes are defined in the original files:

- 00_TOPO/K4_seenyymmdd/c_shp/k4seenyymmdd11_ch2007Poly.shp
- 00_TOPO/K4_seenyymmdd/c_shp/k4seenyymmdd22_ch2007Poly.shp

Rivers are defined in the files:

- 00_TOPO/K4_flusyyyymmdd/c_shp/k4flusyyyymmdd11_ch2007.shp
- 00_TOPO/K4_flusyyyymmdd/c_shp/k4flusyyyymmdd22_ch2007.shp
- 00_TOPO/K4_flusyyyymmdd/c_shp/k4flusyyyymmdd33_ch2007.shp
- 00_TOPO/K4_flusyyyymmdd/c_shp/k4flusyyyymmdd44_ch2007.shp
- 00_TOPO/K4_flusyyyymmdd/c_shp/k4flusyyyymmdd55_ch2007.shp

For accessing the meta data, we can simply use

```
lake <- GetMap("see1.map")
head(lake)
```

Value

None

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
try({
  PlotKant()
  AddWaters(lakes=1, rivers=1, border="grey")
})

try({
  PlotKant()
  AddLakes(categ=1)      # adds the lakes of category 1 to the map
  AddRivers(categ=1:3)   # adds the rivers of category 1:3 to the map
})
```

BfSStamp

Stamp the Current Plot

Description

Stamp the current plot in the lower right corner with the copyright of the BfS-maps:

"Kartengrundlage: (c) BFS, ThemaKart, 20xx"

This copyright is mandatory for all maps in public publications. The default coordinates are chosen by default in the bottomright corner of a Swiss map, but can be redefined by user.

Usage

```
BfSStamp(xy = NULL,
          year_n = getOption("bfsMaps.year", Year(Today()))),
          txt = NULL, cex = 0.6, adj = c(1,0), ...)
```

Arguments

<code>xy</code>	the coordinates for the text to be placed.
<code>year_n</code>	the year for the compulsory BFS copyright message.
<code>txt</code>	the text to be used.
<code>cex</code>	the character extension for the text (default is 0.6)
<code>adj</code>	one or two values in [0, 1] which specify the x (and optionally y) adjustment ('justification') of the labels, with 0 for left/bottom, 1 for right/top, and 0.5 for centered. On most devices values outside [0, 1] will also work. See below.
<code>...</code>	the dots are passed to the function <code>text()</code>

Details

The default value for the year can be entered as option in the .Rprofile file. `bfsMaps.year=2022` would set the default to 2022.

Value

None

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[Stamp\(\)](#)

CombinePolygons

Combine Multiple Polygons to One Spatial Polygon

Description

The function combines polygons to one single spatial polygon object, according to the ID vector that specifies which input polygons belong to which output polygon.

Usage

```
CombinePolygons(map, g)

CombineKant(id, g, map = GetMap("kant.map"))
CombinePolg(id, g, map = GetMap("polg.map"))
```

Arguments

map	the map containing the regions to be combined.
id	the id of the cantons or communities to be aggregated.
g	a vector defining the assignment of the elements to the output polygons to be created. It may contain NA values for input objects not included in the union.

Value

Returns an aggregated spatial polygons object named with the aggregated IDs values in their sorting order; see the ID values of the output object to view the order.

Author(s)

Juerg Guggenbuehl, Andri Signorell <andri@signorell.net>

See Also

[st_union](#)

Examples

```
require(DescTools)

try( {

  # Representation of the language areas in CH combined via cantons
  # by majority per canton
  tkt <- table(d.bfsrg$kt_c, d.bfsrg$sprgeb_x)
  grp <- unique(d.bfsrg$sprgeb_x)[apply(tkt, 1, which.max)]

  # combine and plot cantons
  plot(CombineKant(rownames(tkt), grp), col=SetAlpha(c(horange, hyellow, hecru), 0.8),
       border="grey40", main="Languages in CH")

  # copyright is mandatory for these SFSO maps
  BfSStamp()
  # waters make the maps more realistic ...
  AddLakes(col = "grey80", border = "grey40")

})
```

d.bfsrg

Swiss Federal Statistical Office (SFSO) Spatial Divisions

Description

The Swiss Federal Statistical Office (SFSO) produces, publishes and maintains various spatial divisions for Switzerland. A dataset for the year 2022 is part of the package.

Granularity is the community level.

Usage

```
data("d.bfsrg")
```

Format

A data frame with 2148 observations on the following 27 variables.

`gem_id` community id, a numeric vector

`gemeinde_x` community name, factor with the names of the communities as levels (to ensure the correct order, if needed).

`kt_c` canton id, numeric vector

`kt_x` canton abbreviation, a factor with levels ZH BE LU UR SZ OW NW GL ZG FR SO BS BL SH AR AI SG GR AG TG TI VD VS NE GE JU

`kt_bez_x` a factor with levels Zuerich Bern Luzern Uri Schwyz Obwalden Nidwalden Glarus Zug Fribourg Solothurn Basel-Stadt Basel-Landschaft Schaffhausen Appenzell Ausserrhoden Appenzell Innerrhoden St. Gallen Graubunden Aargau Thurgau Ticino Vaud Wallis Neuchatel Geneve Jura

`bez_k_c` a numeric vector

`bez_k_x` a factor with levels Bezirk Affoltern Bezirk Andelfingen Bezirk Buelach Bezirk Dielsdorf Bezirk Hinwil ...

`greg_c` a numeric vector

`greg_x` a factor with levels Region lemanique Espace Mittelland Nordwestschweiz Zuerich Ostschweiz Zentralschweiz Ticino

`aggl_c` a numeric vector

`aggl_x` a factor with levels keine Agglomerationsgemeinde und keine Kerngemeinde ausserhalb von Agglomerationen Winterthur Zuerich Bern

`aggl_grp_c` a numeric vector

`aggl_grp_x` a factor with levels keine Agglomerationszugehoerigkeit >= 500000 Einwohner/innen 250000 - 499999 Einwohner/innen 100000 - 249999 Einwohner/innen

`stadt_char_c` a numeric vector

`stadt_char_x` a factor with levels Laendliche Gemeinde ohne staedtischen Charakter Agglomerationskerngemeinde (Kernstadt) Agglomerationskerngemeinde (Hauptkern) Agglomerationskerngemeinde (Nebenkern) Agglomerationsguertelgemeinde Mehrfach orientierte Gemeinde Kerngemeinde ausserhalb Agglomerationen

`stadtland_c` a numeric vector

`stadtland_x` a factor with levels stadt agгло land

`gem_typ9_c` a numeric vector

`gem_typ9_x` a factor with 9 levels

`gem_typ25_c` a numeric vector

`gem_typ25_x` a factor with 25 levels, definig types of communities

`degurba_c` a numeric vector

`degurba_x` a factor with levels dense intermediate thin

sprgeb_c a numeric vector
 sprgeb_x a factor with levels d f i r
 preg_c a numeric vector containing the category of the premium region (0:3).
 preg_x a factor with levels ZH1, ZH2, ZH3, BE1, BE2, ...

Examples

```
head(kt <- unique(d.bfsrg[,c("kt_c", "kt_x", "kt_bez_x")][order(d.bfsrg$kt_c),]))
head(bezk <- unique(d.bfsrg[,c("bezk_c", "bezk_x", "kt_x")][order(d.bfsrg$bezk_c),]))
```

DownloadBfSMaps	<i>Helps to Get the Map Data</i>
-----------------	----------------------------------

Description

Helperfunction to download the mapdata and unzip them into a user defined location.

Usage

```
DownloadBfSMaps(url = "https://dam-api.bfs.admin.ch/hub/api/dam/assets/21245514/master",
  path = paste0(path.expand("~"), "/MapData"))
```

Arguments

url the url for the data (might outdate and must then be redefined).
 path the path where the data are to be installed.

Details

It is convenient to not be forced to download the data by oneself. This function can be helpful.

Value

the option entry is returned invisibly

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
## Not run:

DownloadBfsMaps()
# enter the returned option
options(bfsMaps.base="*** your map folder ***")

library(bfsMaps)
PlotKant("ZH", "blue")

## End(Not run)
```

GetMap

*Load a Map***Description**

GetMap directly looks up the path of a map based on a shortcut name or number, loads the map from this location and returns the object.

Usage

```
GetMap(name_x, basedir = getOption("bfsMaps.base",
  default = file.path(find.package("bfsMaps"), "extdata")),
  crs = 2056)
```

Arguments

name_x	the name of a map, currently supported are "kant.map", "bezk.map", "polg.map", "greg.map", "ch.map" or any number referring to a row in the 'maps.csv' file.
basedir	the root directory for the maps to reside. bfsMaps by default looks for the map-files in its install location in the extdata directory. The basedir can be set as an option too.
crs	Coordinate reference system (default 2056) for the map, if not set to NA the map will be transformed to this crs.

Details

Loading the cards no longer causes performance problems and can thus be performed directly.

Value

the map object

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
try( {

  # use map containing Swiss metropolitan regions
  mymap <- GetMap("metr.map")$geometry
  PlotCH()
  plot(mymap, col=Pal("Helsana"), add=TRUE, border=NA)

})
```

kt	<i>Abbreviations for Swiss Cantons</i>
----	--

Description

Abbreviations for Swiss Cantons in the correct order of BfS-ID. The motivation to define this constant is, that the ids in the official definition do not follow the alphabetic order of the canton names.

Usage

```
kt
```

Format

The format is: Factor w/ 26 levels "ZH","BE","LU",...: 1 2 3 4 5 6 7 8 9 10 ...

Neighbours	<i>Find All Neighbours of a Regional Object</i>
------------	---

Description

Finding all directly adjacent neighbours of a regional unit is not trivial. For a list of regional units, this function searches for the corresponding Neighbours and returns the results as a list.

Usage

```
Neighbours(map, id = NULL)
```

Arguments

map	the name of the map
id	vector of ids for which the Neighbours are to be found. When it's left to NULL (default), the neighbours for all the polygons of the map will be returned.

Value

A list of vectors of ids for the neighbours of each region in the map.

Author(s)

Andri Signorell <andri@signorell.net>

Examples

```
try( {

nbs <- Neighbours(GetMap("kant.map"), kt_id <- 18)
PlotKant(c(kt_id, nbs), col=c("red", rep("green", length(nbs))) )

# works as well for communities and for vector of ids
nbs <- Neighbours(GetMap("polg.map"), polg_id <- c(3851, 3352))
PlotPolg(c(polg_id, unlist(nbs)),
         col=c(rep("red", 2), rep("green", length(unlist(nbs)))))

})
```

Plot Swiss Regions

Plot Swiss Regions

Description

The function plots a map of Switzerland overlayed with different types of regions. Included are greater regions ('Grossregionen'), MS regions ('mobilité spatiale'), cantons, districts and political communities. The single regions can be given a defined color, whereas the color need not be defined for all.

The vegetational area is the spatial area where people live, excluding mountains and further uninhabitable area. The vegetational area can be drawn over an already existing map.

Usage

```
PlotGreg(id = NULL, col = NA, pbgr = "white", main = "",
         border = "grey", lwd = 1, col.vf = NA, border.vf = NA, labels = NULL,
         tmtxt = TRUE, add = FALSE, map_x = "greg.map", ...)

PlotKant(id = NULL, col = NA, pbgr = "white", main = "",
         border = "grey", lwd = 1, col.vf = NA, border.vf = NA, labels = NULL,
         tmtxt = TRUE, add = FALSE, map_x = "kant.map", ...)

PlotMSRe(id = NULL, col = NA, pbgr = "white", main = "",
         border = "grey", lwd = 1, col.vf = NA, border.vf = NA, labels = NULL,
         tmtxt = TRUE, add = FALSE, map_x = "msre.map", ...)

PlotBezkr(id = NULL, col = NA, pbgr = "white", main = "",
         border = "grey", lwd = 1, col.vf = NA, border.vf = NA, labels = NULL,
         tmtxt = TRUE, add = FALSE, map_x = "bezkr.map", ...)

PlotPolg(id = NULL, col = NA, pbgr = "white", main = "",
```

```
border = "grey", lwd = 1, col.vf = NA, border.vf = NA, labels = NULL,
tmtxt = TRUE, add = FALSE, map_x = "polg.map", ...)
```

Arguments

<code>id</code>	vector of region ids. All types of regions can be addressed via their numeric ID, cantons can additionally be identified with their abbreviation: "AG", "AI", "AR", "BE", "BL", "BS", "FR", "GE", "GL", "GR", "JU", "LU", "NE", "NW", "OW", "SG", "SH", "SO", "SZ", "TG", "TI", "UR", "VD", "VS", "ZG", "ZH"
<code>col</code>	vector of colors, defining the colors of the region area.
<code>pbg</code>	color for the plot background.
<code>main</code>	main title in the plot.
<code>border</code>	vector of colors for region borders. Default is "grey30".
<code>lwd</code>	linewidth for region borders.
<code>col.vf</code>	vector of colors for the vegetational. If set to NA (default) the vegetational area will not be drawn.
<code>border.vf</code>	color of borders for the vegetational area. If set to NA (default) the borders of the vegetational area will not be drawn.
<code>labels</code>	optional labels to be placed in the map, by default the centroids of the map is used for that.
<code>tmtxt</code>	logical, should the copyright text be displayed. Default is TRUE.
<code>add</code>	default FALSE; if TRUE, add to existing plot.
<code>map_x</code>	the name of the path of a map to be used. This is convenient, if we want to plot a newer map with the logic of this function.
<code>...</code>	the dots are passed to the plot command.

Details

The different functions all use the same core code, but use different default maps. The default maps are named: "greg.map", "msre.map", "kant.map", "bez.k.map" and "polg.map".

PlotGreg uses a map for Swiss regions (Grossregionen), as defined in `greg.map@data`. The regions are:

- 1 Region lémanique
- 2 Espace Mittelland
- 3 Nordwestschweiz
- 4 Zürich
- 5 Ostschweiz
- 6 Zentralschweiz
- 7 Ticino

The list of MS regions with names and ids can be found in `d.bfsrg`:

```
unique(d.bfsrg[, c("ms_reg_c", "ms_reg_x", "ms_typ_c", "ms_typ_x")])
```

The abbreviations of the cantons are compiled in the variable `kt`. More details can be extracted from

```
unique(d.bfsrg[,c("kt_c", "kt_x", "kt_bez_x")])
```

Districts (german: 'Bezirke') are associations of communities. The district id internally consists of the canton nr (1 or 2 digits) and a 2-digits 'Bezirk-nr'. So is 'Hinwil' with the district nr '51' the 5th district in Zurich (canton '1').

The list of all districts is given in `d.bfsrg`:

```
unique(d.bfsrg[, c("bezirk_c", "kt_c", "bezirk_x", "kt_x")])
```

The list of all political communities is given in `d.bfsrg`:

```
d.bfsrg[,c("bfs_nr", "gemeinde_name_x", "kt_x")]
```

All the regions can also be accessed and plotted by manually loading the maps and use the generic `plot` function.

```
cant <- GetMap("kant.map")
plot(cant)
```

There are also dedicated maps for all regions, which contain only the coordinates of the regions' centroids. They can be accessed using according mapname with the extension `.pnt`, e.g. for the cantons `GetMap("kant.pnt")`.

To simplify the description, the function returns the center coordinates. These can then be used with the function `text()`.

Value

A list containing x and y components which are the centroids of the plotted spatial units.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[PlotCH](#), [d.bfsrg](#)

Examples

```
# Note:
# The examples can not be run without having the map data installed before!
try( {

# define the ids for the cantons and the according colors
PlotKant(id=c("GR", "ZH", "VS"), col=c("lightgrey", "lightblue", "lightsalmon"))

require(DescTools)
# get some percentage values...
some_p <- c(AG=0.48, AI=0.47, AR=0.4, BE=0.48, BL=0.44, BS=0.4, FR=0.48, GE=0.28, GL=0.51,
            GR=0.4, JU=0.61, LU=0.49, NE=0.54, NW=0.43, OW=0.58, SG=0.45, SH=0.36, SO=0.45,
            SZ=0.39, TG=0.47, TI=0.46, UR=0.4, VD=0.46, VS=0.45, ZG=0.41, ZH=0.41)
```

```

# and a color ramp from white to hred
cols <- colorRampPalette(c("white", hred))(100)

PlotKant(id=names(some_p), col=FindColor(some_p, cols=cols), main="ECO in CH")
ColorLegend(x="left", inset=-0.01, cols=cols,
            labels=formatC((seq(0, 1, .2)), digits=2, format="f"),
            width=12000, frame="grey", cex=0.8 )

# greater regions
PlotGreg(col=colorRampPalette(c("blue", "white", "red"), space = "rgb")(7),
        main="Greater Regions CH")

PlotGreg(id = c(2,4,7), col = c("bisque","darkolivegreen1","khaki"),
        main="Espace Mittelland, Zurich und Ticino")
AddLakes(col="grey90", border="darkgrey")

xy <- sf::st_coordinates((greg.pnt <- GetMap("greg.pnt"))$geometry[c(2,4,7)])
text(xy[,1], xy[,2], greg.pnt$name[c(2,4,7)], col="black")

# plot the districts
bezk.map <- GetMap("bezk.map")
head(bezk.map)

PlotBezk(id=311:316, col=colorRampPalette(c("red", "white", "blue"), space = "rgb")(5))

PlotBezk(id=bezk.map[[1]], col=rainbow(147), main="Districts in CH")

cols <- c(y=rgb(255,247,174,max=255), o=rgb(251,208,124,max=255),
        v=rgb(228,201,224,max=255), b=rgb(211,230,246,max=255),
        g=rgb(215,233,205,max=255), r=rgb(244,182,156,max=255),
        p=rgb(255,248,236,max=255))

# display MS regions
# start with a cantons map

# start with a cantons map
PlotKant(1:26,col=cols[c("g","g","o","r","v","b","y","g","y","o",
                        "v","o","y","v","y","v","o","y","r","b",
                        "v","y","b","r","v","b")],
        border="grey20", lwd=1, pbg=cols["p"],
        main="106 MS-Regions")

# add the MS regions borders
xy <- PlotMSRe(add=TRUE, border="grey60")

# reoutline the cantons, as they have been overplotted in the step before
PlotKant(1:26, add=TRUE, border="grey30", lwd=1)

# add the waters
AddLakes(1:2, col=rgb(235, 247, 253, max=255), border=rgb(0,166,235, max=255))
AddRivers(1:5, col=rgb(0, 166, 235, max=255))

```

```

# ... and finally add labels
text(x=xy$x, y=xy$y, GetMap("msre.map")$id, cex=0.6)

# plot political communities

# plot only the first 10 elements
PlotPolg(id=1:10,
         col=colorRampPalette(c("red", "white", "blue"), space = "rgb")(10))

# plot all communities
PlotPolg(id=(map <- GetMap("polg.map"))$id, col=rainbow(nrow(map)),
         main="Political communities in CH")

})

```

PlotCH

Plot a Map of Switzerland

Description

Simple map plot of Switzerland following the borders valid since 1848.

Usage

```

PlotCH(col = NA, main = "", col.vf = NA,
       border = "grey", border.vf = NA, lwd = 1,
       tmtxt = TRUE, add = FALSE, ...)

```

Arguments

<code>col</code>	vector of colors, defining the colors of the cantons. Note: NAs are recoded as white.
<code>main</code>	main title in the plot.
<code>col.vf</code>	defines a color for the vegetational area ("Vegetationsflaeche"). If NA only the total area is used.
<code>border</code>	color of map border. Default is "grey".
<code>border.vf</code>	color of borders for the vegetational area. Default is "grey".
<code>lwd</code>	linewidth for the border. Default is <code>par("lwd")</code> .
<code>tmtxt</code>	logical, should the copyright text be displayed. Default is TRUE.
<code>add</code>	default FALSE; if TRUE, add to existing plot.
<code>...</code>	the dots are passed to the plot command.

Details

The list of all cantons and their ids is given by d.bfsrg:

```
cantons <- unique(d.bfsrg[,c("kt_c", "kt_x", "kt_bez_x")])
```

Value

A list containing x and y component of the centroid of the plotted spatial unit.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[PlotGreg](#), [PlotKant](#), [PlotBez](#), [PlotPolg](#), [d.bfsrg](#)

Examples

```
try( {

  PlotCH(col="lightgrey")
  AddLakes()

  # use the result to add a semitransparent label
  xy <- PlotCH(col.vf = "grey90", col="grey75", border="grey50", border.vf = NA)
  AddLakes()
  AddRivers()
  PlotCH(add=TRUE, col=NA)
  BoxedText(x=xy$x, y=xy$y, labels = "Visit\n Switzerland", cex=3, txt.col = "grey40",
            col=SetAlpha("white", 0.6), border=NA, ypad=0.5)

  # waving flag ...
  PlotCH(col="red", main="Switzerland")
  sw <- 15000;
  xc <- 2671975;
  yc <- 1200600;

  ccol <- rgb(1,1,1,0.85)
  rect(xleft=xc-sw, ytop=yc-sw, xright=xc+sw, ybottom=yc+sw, col=ccol, border=NA)
  rect(xleft=(xc-2*sw)-sw, ytop=yc-sw, xright=(xc-2*sw)+sw, ybottom=yc+sw, col=ccol, border=NA)
  rect(xleft=(xc+2*sw)-sw, ytop=yc-sw, xright=(xc+2*sw)+sw, ybottom=yc+sw, col=ccol, border=NA)
  rect(xleft=xc-sw, ytop=(yc-2*sw)-sw, xright=xc+sw, ybottom=(yc-2*sw)+sw, col=ccol, border=NA)
  rect(xleft=xc-sw, ytop=(yc+2*sw)-sw, xright=xc+sw, ybottom=(yc+2*sw)+sw, col=ccol, border=NA)

  # using panel.first ensures that the borders are not hidden by waters
  PlotCH(col=NA, lwd=2, panel.first=AddLakes())

})
```

PlotMapDot*Plot a Map and a Dotplot*

Description

Prepares the layout to plot a map and a dotplot side by side.

Usage

```
PlotMapDot(mar = c(5.1,4.1,0,1), oma = c(0,0,5,0), widths = c(2,0.8))
```

Arguments

mar	defines the plot margins.
oma	defines the outer margins. We use that for displaying a title.
widths	a vector of values for the widths of two columns, the first for the map, the second for dotplot. Relative widths are specified with numeric values. Absolute widths (in centimetres) are specified with the lcm() function. Default is c(2, 0.8).

Value

None

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[layout](#)

Examples

```
require(DescTools)

try( {

yes_p <- c(ZH=0.465, BE=0.417, LU=0.376, UR=0.308, SZ=0.276,
          OW=0.273, NW=0.277, GL=0.324, ZG=0.344, FR=0.469, SO=0.352,
          BS=0.602, BL=0.414, SH=0.457, AR=0.325, AI=0.24, SG=0.365,
          GR=0.325, AG=0.347, TG=0.321, TI=0.446, VD=0.532, VS=0.329,
          NE=0.562, GE=0.601, JU=0.532)

PlotMapDot()
cols <- colorRampPalette( colors=c("red","white","green"), space = "rgb")(10)
PlotKant(id=names(yes_p),
        col=FindColor(yes_p, cols=cols, min.x=0, max.x=1 ), main="",
```

```

        labels=TRUE)

ColorLegend(x="left", width=10000,
            labels=paste(seq(0, 100, 10), "%", sep=""),
            cols=cols, cex=0.8, adj=c(1,0.5), frame="grey")

x <- Sort(yes_p, decreasing=TRUE)

opt <- DescToolsOptions(stamp=NULL)
PlotDot(x, labels=gettextf("%s (%s)", names(x), Format(x, fmt="%", digits=1)),
        cex=0.9, xlim=c(0,1))
abline(v=0.5, col="grey")

title(main="Volksinitiative 'Mehr bezahlbare Wohnungen'
       Abstimmung vom 09.02.2020", outer=TRUE)
DescToolsOptions(opt)

# reset the layout
layout(1)

})

```

PlotPremReg

Plot Premium Regions CH

Description

Plot premium regions in Switzerland.

Usage

```

PlotPremReg(id = NULL, col = NA, pbg = "white", main = "",
            border = "grey", lwd = 1,
            labels = NULL, tmtxt = TRUE, add = FALSE, ...)

```

Arguments

<code>id</code>	vector of region ids. The premium regions can be addressed via their their abbreviation: "AG0", "AI0", "AR0", "BE1", ...
<code>col</code>	vector of colors, defining the colors of the region area.
<code>pbg</code>	color for the plot background.
<code>main</code>	main title in the plot.
<code>border</code>	vector of colors for region borders. Default is "grey30".
<code>lwd</code>	linewidth for region borders.
<code>labels</code>	optional labels to be placed in the map, by default the centroids of the map is used for that.
<code>tmtxt</code>	logical, should the copyright text be displayed. Default is TRUE.
<code>add</code>	default FALSE; if TRUE, add to existing plot.
<code>...</code>	the dots are passed to the plot command.

Value

A list containing x and y components which are the centroids of the plotted spatial units.

Author(s)

Andri Signorell <andri@signorell.net>

See Also

[PlotCH](#), [d.bfsrg](#)

Examples

```
# Note:
# The examples can not be run without having the map data installed before!
try( {

preg_x <- sort(unique(d.bfsrg$preg_x))

PlotPremReg(id=preg_x, border="grey60",
            col=c("white", "olivedrab4", "olivedrab3", "olivedrab2")[
              StrVal(preg_x, as.numeric=T)+1],
            main="Prämienregionen CH")

legend(x="topleft", fill=c("white", "olivedrab4", "olivedrab3", "olivedrab2"),
       cex=0.8,
       legend=c("Region 0", "Region 1", "Region 2", "Region 3") )

AddLakes()

# plot all premium regions
# find all regions
d.bfsrg$preg_x <- paste0(d.bfsrg$kt_x, d.bfsrg$preg_c)
preg <- unique(d.bfsrg$preg_x)

cols <- c("white", "darkolivegreen3", "darkolivegreen2", "darkolivegreen1")

PlotPremReg(preg, cols[ZeroIfNA(StrVal(preg, as.numeric = T))+1], labels = NA)
PlotKant(add=TRUE, border="grey55")
AddLakes()

# plot some selected premium regions
PlotPremReg(c("ZH1", "GR2"), c("blue", "yellow"), labels=TRUE)
PlotKant(add=TRUE, border="grey55")
AddLakes()
})
```

SwissLocator

Get the Community, District and Canton of a Located Mappoint

Description

Locate a point in a Switzerland map and get the according community, district and canton.

Usage

```
SwissLocator()
```

Value

For each clicked and identified point the coordinates, the political community, the district and the canton will be returned.

	x	y	bfs_nr	community_x	district_x	kt_x
1014	536281.5	167176.3	2703	Riehen	Kanton Basel-Stadt	BS
1781	616565.2	268959.6	5136	Onsernone	Distretto di Locarno	TI
1962	690861.6	119006.1	5524	Goumoens-la-Ville	District du Gros-de-Vaud	VD

The result will also be stored for later use in the variable `tkart$found`.

Author(s)

Andri Signorell <andri@signorell.net>

tkart

Unlocked Environment for Maps

Description

Loading maps and parsing their structure takes time. In order to avoid to load the maps multiple times in a session they're cached in this special environment after the first load in a session.

Usage

```
tkart
```

Format

The format is: <environment: 0x000001d443d516f8>

Index

- * **aplot**
 - AddWaters, [6](#)
 - BfSStamp, [7](#)
- * **datasets**
 - d.bfsrg, [9](#)
 - kt, [13](#)
- * **hplot**
 - Plot Swiss Regions, [14](#)
 - PlotCH, [18](#)
 - PlotMapDot, [20](#)
- * **iplot**
 - SwissLocator, [23](#)
- * **package**
 - bfsMaps-package, [2](#)
- * **spatial**
 - CombinePolygons, [8](#)
 - Neighbours, [13](#)
 - Plot Swiss Regions, [14](#)
 - SwissLocator, [23](#)
- AddLakes, [2](#)
- AddLakes (AddWaters), [6](#)
- AddRivers, [2](#)
- AddRivers (AddWaters), [6](#)
- AddWaters, [2](#), [6](#)
- bfsMaps (bfsMaps-package), [2](#)
- bfsMaps-package, [2](#)
- BfSStamp, [7](#)
- CombineKant (CombinePolygons), [8](#)
- CombinePolg (CombinePolygons), [8](#)
- CombinePolygons, [8](#)
- d.bfsrg, [9](#), [16](#), [19](#), [22](#)
- DownloadBfsMaps, [11](#)
- GetMap, [12](#)
- kt, [13](#)
- layout, [20](#)
- Neighbours, [13](#)
- Plot Swiss Regions, [14](#)
- PlotBezK, [2](#), [19](#)
- PlotBezK (Plot Swiss Regions), [14](#)
- PlotCH, [16](#), [18](#), [22](#)
- PlotGreg, [2](#), [19](#)
- PlotGreg (Plot Swiss Regions), [14](#)
- PlotKant, [2](#), [19](#)
- PlotKant (Plot Swiss Regions), [14](#)
- PlotMapDot, [20](#)
- PlotMSRe (Plot Swiss Regions), [14](#)
- PlotPolg, [2](#), [19](#)
- PlotPolg (Plot Swiss Regions), [14](#)
- PlotPremReg, [21](#)
- st_union, [9](#)
- Stamp, [8](#)
- SwissLocator, [23](#)
- text, [8](#)
- tkart, [23](#)