Package ‘RgoogleMaps’

November 6, 2023

Type Package
Title Overlays on Static Maps
Version 1.5.1
Date 2023-10-31
Depends R (>= 2.10)
Imports graphics, stats, utils, grDevices, methods, png
Suggests PBSmapping, RColorBrewer, leaflet, jpeg, RCurl
Author Markus Loecher
Maintainer Markus Loecher <markus.loecher@gmail.com>
Description Serves two purposes: (i) Provide a comfortable R interface to query the Google server for static maps, and (ii) Use the map as a background image to overlay plots within R. This requires proper coordinate scaling.
License GPL
LazyLoad yes
Repository CRAN
NeedsCompilation no
Date/Publication 2023-11-06 12:20:02 UTC

R topics documented:

AddAlpha ......................................................... 2
ColorMap .......................................................... 4
columbus ......................................................... 5
degreeAxis ....................................................... 7
genStaticMap ..................................................... 8
geosphere_mercator ............................................. 10
GetBingMap ...................................................... 11
AddAlpha

__add alpha level to color that lacks one__

**Description**

add alpha level to color that lacks one

**Usage**

\[
\text{AddAlpha}\left(\text{plotclr, alpha = 0.5, verbose = 0}\right)
\]
AddAlpha

Arguments
plotclr    color to be modified
alpha      alpha level
verbose    level of verbosity

Value
modified color with alpha value

Author(s)
Markus Loecher

Examples

```r
#example:

#require(RColorBrewer)

if (requireNamespace("RColorBrewer", quietly = TRUE)) {

plotclr <- RColorBrewer::brewer.pal(8,"YlOrRd")

plotclr = AddAlpha(plotclr,0.5)

} else {

print("package RColorBrewer must be installed for this example")

}
```
ColorMap

Plot Levels of a Variable in a Colour-Coded Map

Description

Plot Levels of a Variable in a Colour-Coded Map

Usage

ColorMap(values, map = NULL, polys = NULL, log = FALSE,

    nclr = 7, include.legend = list(TRUE), round = 3,

    brks = NULL, legend = NULL, location = "topright",

    rev = FALSE, alpha = 0.5, GRAY = FALSE, palette = c("YlOrRd",

        "RdYlGn", "Spectral")[1], textInPolys = NULL,

    ...)}

Arguments

values variable to plot
map map object
polys an object of class SpatialPolygons (See SpatialPolygons-class
log boolean of whether to plot values on log scale
nclr number of colour-levels to use
include.legend boolean of whether to include legend
round number of digits to round to in legend
brks if desired, pre-specified breaks for legend
legend if desired, a pre-specified legend
location location of legend
rev boolean of whether to reverse colour scheme (darker colours for smaller values)
alpha alpha value of colors
GRAY boolean: if TRUE, use gray scale instead
palette palette to choose from RColorBrewer
textInPolys text to be displayed inside polygons. This can be a column names for values
... extra args to pass to PlotPolysOnStaticMap
Author(s)
Markus Loecher

Examples

```r
if (0){

data("NYleukemia", envir = environment())

population <- NYleukemia$data$population

cases <- NYleukemia$data$cases

mapNY <- GetMap(center=c(lat=42.67456,lon=-76.00365), destfile = "NYstate.png",

maptype = "mobile", zoom=9)

ColorMap(100*cases/population, mapNY, NYleukemia$spatial.polygon, add = FALSE,

alpha = 0.35, log = TRUE, location = "topleft")
}

#ColorMap(100*cases/population, map=NULL, NYleukemia$spatial.polygon)
```

Description
The `columbus` data frame has 49 rows and 22 columns. Unit of analysis: 49 neighbourhoods in Columbus, OH, 1980 data. In addition the data set includes a polylist object `polys` with the boundaries of the neighbourhoods, a matrix of polygon centroids `coords`, and `col.gal.nb`, the neighbours list from an original GAL-format file. The matrix `bbs` is DEPRECATED, but retained for other packages using this data set.
Usage

data(columbus)

Format

This data frame contains the following columns:

- **AREA**: computed by ArcView
- **PERIMETER**: computed by ArcView
- **COLUMBUS**: internal polygon ID (ignore)
- **COLUMBUS.I**: another internal polygon ID (ignore)
- **POLYID**: yet another polygon ID
- **NEIG**: neighborhood id value (1-49); conforms to id value used in Spatial Econometrics book.
- **HOVAL**: housing value (in $1,000)
- **INC**: household income (in $1,000)
- **CRIME**: residential burglaries and vehicle thefts per thousand households in the neighborhood
- **OPEN**: open space in neighborhood
- **PLUMB**: percentage housing units without plumbing
- **DISCBD**: distance to CBD
- **X**: x coordinate (in arbitrary digitizing units, not polygon coordinates)
- **Y**: y coordinate (in arbitrary digitizing units, not polygon coordinates)
- **NSA**: north-south dummy (North=1)
- **NSB**: north-south dummy (North=1)
- **EW**: east-west dummy (East=1)
- **CP**: core-periphery dummy (Core=1)
- **THOUS**: constant=1,000
- **NEIGNO**: NEIG+1,000, alternative neighborhood id value

Details

The row names of columbus and the region.id attribute of polys are set to columbus$NEIGNO.

Note

All source data files prepared by Luc Anselin, Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign.

Source

degreeAxis

Examples

```r
#library(maptools)
#columbus <- readShapePoly(system.file("etc/shapes/columbus.shp",
 # package="spdep")[[1]])
#col.gal.nb <- read.gal(system.file("etc/weights/columbus.gal",
 # package="spdep")[[1]])
```

---

<table>
<thead>
<tr>
<th>degreeAxis</th>
<th>axis with degrees</th>
</tr>
</thead>
</table>

Description

add an axis with degree labels

Usage

degreeAxis(side, at = NULL, labels, MyMap, ...)

Arguments

- `side` integer; see `axis`
- `at` numeric; if missing, `axTicks` is called for nice values; see `axis`
- `labels` character; if omitted labels are constructed with degree symbols, ending in N/S/E/W; in case of negative degrees, sign is reversed and S or W is added; see `axis`
- `MyMap` optional map object to be passed
- `...` optional arguments to `axis`

Value

axis is plotted on current graph

Note

decimal degrees are used if variation is small, instead of minutes and seconds

Author(s)

Markus Loecher
Examples

```r
xy = cbind(x = 2 * runif(100) - 1, y = 2 * runif(100) - 1)

plot(xy, xlim = c(-1, 1), ylim = c(-1, 1))

degreeAxis(1)

degreeAxis(2, at = c(-1, -0.5, 0, 0.5, 1))
```

genStaticMap generates a "static map" from map tiles by "stitching" them together

Description

necessary because the Google static maps API requires a key now

Usage

genStaticMap(center, zoom = 15, size = c(640, 640),

destfile = tempfile("staticMap", fileext = ".png"),

type = c("google", "google-m", "google-s", "osm",

"osm-hot", "stamen-toner", "stamen-terrain",

"stamen-watercolor")[1], urlBase = "http://mt1.google.com/vt/lyrs=m",

tileDir = "/tmp/", tileExt = ".png", verbose = 0,

...)
genStaticMap

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>optional center</td>
</tr>
<tr>
<td>zoom</td>
<td>zoom</td>
</tr>
<tr>
<td>size</td>
<td>size (in pixels) of &quot;stitched&quot; map</td>
</tr>
<tr>
<td>destfile</td>
<td>File to load the map image from or save to, depending on NEWMAP.</td>
</tr>
<tr>
<td>type</td>
<td>choice of tile server</td>
</tr>
<tr>
<td>tileDir</td>
<td>map tiles can be stored in a local directory, e.g. &quot;~/.mapTiles/Google/&quot;</td>
</tr>
<tr>
<td>tileExt</td>
<td>image type of tile</td>
</tr>
<tr>
<td>verbose</td>
<td>level of verbosity</td>
</tr>
<tr>
<td>...</td>
<td>further arguments to be passed to FUN</td>
</tr>
</tbody>
</table>

Value

list with tiles

Author(s)

Markus Loecher

Examples

```r
if (0){

  lat = c(40.702147, 40.718217, 40.711614);

  lon = c(-74.012318, -74.015794, -73.998284);

  center = c(mean(lat), mean(lon));

  zoom <- min(MaxZoom(range(lat), range(lon)));

  bb=qbbox(lat,lon)
```

mt = GetMapTiles(latR = bb$latR, lonR = bb$lonR, zoom = zoom, verbose = 1)

PlotOnMapTiles(mt, lat = lat, lon = lon, pch = 20, col = c('red', 'blue', 'green'), cex = 2)

mt = GetMapTiles(latR = bb$latR, lonR = bb$lonR, zoom = zoom, tileDir = '~/mapTiles/Google/')

PlotOnMapTiles(mt, lat = lat, lon = lon, pch = 20, col = c('red', 'blue', 'green'), cex = 2)

}
GetBingMap  
*download a static map from the Microsoft map tile server*

---

**Description**

Query the Google server for a static map tile, defined primarily by its center and zoom. Many additional arguments allow the user to customize the map tile.

**Usage**

\[
\text{GetBingMap}(\text{center} = c(\text{lat} = 42, \text{lon} = -76), \text{mapArea} = c(45.219, \-122.325, 47.61, -122.107), \text{size} = c(640, 640), \\
\text{destfile}, \text{zoom} = 12, \text{markers}, \text{path} = "", \text{maptype} = c("Road", "Aerial", "AerialWithLabels")[1], \text{format} = c("png", "gif", "jpg", "jpg-baseline", "png8", "png32")[1], \\
\text{extraURL} = "", \text{RETURNIMAGE} = \text{TRUE}, \text{GRAYSCALE} = \text{FALSE}, \\
\text{NEWMAP} = \text{TRUE}, \text{SCALE} = 1, \text{apiKey} = \text{NULL}, \text{verbose} = 0)
\]

**Arguments**

- **center**: optional center (lat first,lon second)
- **mapArea**: A rectangular area specified as a bounding box (ll,ur). Required when a center point or set of route points are not specified
- **size**: desired size of the map tile image. defaults to maximum size returned by the Google server, which is 640x640 pixels
- **destfile**: File to load the map image from or save to, depending on NEWMAP.
- **zoom**: Google maps zoom level.
- **markers**: (optional) defines one or more markers to attach to the image at specified locations. This parameter takes a string of marker definitions separated by the pipe character (|)
GetBingMap

path (optional) defines a single path of two or more connected points to overlay on the image at specified locations. This parameter takes a string of point definitions separated by the pipe character (I)

maptype defines the type of map to construct. See https://msdn.microsoft.com/en-us/library/ff701724.aspx

format (optional) defines the format of the resulting image. By default, the Static Maps API creates GIF images. There are several possible formats including GIF, JPEG and PNG types. Which format you use depends on how you intend to present the image. JPEG typically provides greater compression, while GIF and PNG provide greater detail. This version supports only PNG.

extraURL custom URL suffix

RETURNIMAGE return image yes/no default: TRUE

GRAYSCALE Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY

NEWMAP if TRUE, query the Google server and save to destfile, if FALSE load from destfile.

SCALE use the API's scale parameter to return higher-resolution map images. The scale value is multiplied with the size to determine the actual output size of the image in pixels, without changing the coverage area of the map

apiKey optional API key (allows for higher rate of downloads)

verbose level of verbosity

Value

map structure or URL used to download the tile.

Note

Note that size is in order (lon, lat)

Author(s)

Markus Loecher

See Also

GetMap.bbox

Examples

if (0){

    #for bing maps you will need your own API key,

apiKey = scan("bingAPIkey.txt",what="")

map1=GetBingMap(center=c(47.619048,-122.35384),zoom=15,apiKey=apiKey,
                  verbose=1, destfile="Seattle.png")

PlotOnStaticMap(map1)

m="&pp=47.620495,-122.34931;21;AA&pp=47.619385,-122.351485;;AB&pp=47.616295,-122.3556;22"

map2=GetBingMap(center=c(47.619048,-122.35384),zoom=15,markers=m,apiKey=apiKey,
                 verbose=1, destfile="Seattle2.png")

PlotOnStaticMap(map2, lat=c(47.620495,47.619385,47.616295),
                 lon=c(-122.34931,-122.351485,-122.3556))

m="&pp=49.28273,-123.12074;22&pp=44.05207,-123.08675;22"

map3= GetBingMap(center=c(47.677006,-122.125526),zoom=6,markers=m,apiKey=apiKey,
                  verbose=1, destfile="Seattle2.png")

#plotmap(map=map3)
m = cbind.data.frame(lat = c(49.28273, 44.05207), lon = c(-123.12074, -123.08675), col = c(3:4))

PlotOnStaticMap(map3, lat = m$lat, lon = m$lon, col = m$col, pch = 19)

# overlay traffic:

# Get a map with Road imagery and traffic flow based on a query.

# This example gets a map with road imagery based on a query result Bellevue, Washington.

# Traffic flow is also included on the map.

# http://dev.virtualearth.net/REST/V1/Imagery/Map/Road/Bellevue%20Washington

# ?mapLayer=TrafficFlow&key=BingMapsKey

# note that we are using the extraURL argument to pass any extra parameters:

map4 = GetBingMap(center = "Bellevue%20Washington", zoom = 12, extraURL = "&mapLayer=TrafficFlow",
                   apiKey = apiKey, verbose = 1, destfile = "BellevueTraffic.png")

PlotOnStaticMap(map4)

# Get a map with Road imagery that displays a route.

# This example gets a map with road imagery that displays a driving

# route between the cities of Seattle and Redmond in Washington State.
#note that we are using the extraURL argument to pass any extra parameters:

#http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/Routes

#?wp.0=Seattle,WA;64;1&wp.1=Redmond,WA;66;2&key=BingMapsKey

map5 = GetBingMap(center="Bellevue%20Washington", zoom=8,

    extraURL="&Routes?wp.0=Seattle,WA;64;1&wp.1=Redmond,WA;66;2",

    apiKey=apiKey, verbose=1, destfile="Seattle2Redmond.png")

PlotOnStaticMap(map5)

}

---

**getGeoCode**  
**geocoding utility**

**Description**

Geocode your data using R, JSON and OSM or Google Maps’ Geocoding APIs

**Usage**

getGeoCode(gcStr, API = c("osm", "google")[1], JSON = FALSE,

    verbose = 0)

**Arguments**

- **gcStr**: address to geocode
- **API**: which API to use. see https://nominatim.org/release-docs/develop/api/Search/ and http://allthingsr.blogspot.de/2012/01/geocode-your-data-using-r-json-and.html
- **JSON**: use the JSON protocol. If FALSE, we do not have to load additional libraries
- **verbose**: level of verbosity
Value

returns lat/lon for address

Author(s)

Markus Loecher

Examples

```r
if (0){

  getGeoCode("1600 Amphitheatre Parkway, Mountain View, CA")

  getGeoCode("Brooklyn")

  #You can run this on the entire column of a data frame or a data table:

  DF = cbind.data.frame(address=c("Berlin,Germany", "Princeton,NJ",
                              "cadillac+mountain+acadia+national+park"), lat = NA, lon = NA)

  DF <- with(DF, data.frame(address, t(sapply(DF$address, getGeoCode)))))
}
```

GetMap

*download a static map from the Google server*

Description

Query the Google server for a static map tile, defined primarily by its center and zoom. Many additional arguments allow the user to customize the map tile.

documentation at https://developers.google.com/maps/documentation/staticmaps/
GetMap

Usage

GetMap(center = c(lat = 42, lon = -76), size = c(640, 640), destfile = tempfile("staticMap", fileext = ".png"),

zoom = 12, markers, path = ",", span, frame, hl,

sensor = "true", matype = c("roadmap", "mobile",

"satellite", "terrain", "hybrid", "mapmaker-roadmap",

"mapmaker-hybrid")[2], format = c("gif", "jpg",

"jpg-baseline", "png8", "png32")[5], extraURL = "",

RETURNIMAGE = TRUE, GRAYSCALE = FALSE, NEWMAP = TRUE,

SCALE = 1, API_console_key, type = c("google",

"google-m", "google-s", "osm", "osm-hot", "stamen-toner",

"stamen-terrain", "stamen-watercolor")[1],

urlBase = "http://mt1.google.com/vt/lyrs=m", tileDir = "/tmp/",

verbose = 0)

Arguments

center optional center (lat first, lon second)

size desired size of the map tile image. defaults to maximum size returned by the Google server, which is 640x640 pixels

destfile File to load the map image from or save to, depending on NEWMAP.

zoom Google maps zoom level.
markers (optional) defines one or more markers to attach to the image at specified locations. This parameter takes a string of marker definitions separated by the pipe character (|)

path (optional) defines a single path of two or more connected points to overlay on the image at specified locations. This parameter takes a string of point definitions separated by the pipe character (|)

span (optional) defines a minimum viewport for the map image expressed as a latitude and longitude pair. The static map service takes this value and produces a map of the proper zoom level to include the entire provided span value from the map’s center point. Note that the resulting map may include larger bounds for either latitude or longitude depending on the rectangular dimensions of the map. If zoom is specified, span is ignored

frame (optional) specifies that the resulting image should be framed with a colored blue border. The frame consists of a 5 pixel, 55% opacity blue border.

hl (optional) defines the language to use for display of labels on map tiles. Note that this parameter is only supported for some country tiles; if the specific language requested is not supported for the tile set, then the default language for that tile set will be used.

sensor specifies whether the application requesting the static map is using a sensor to determine the user’s location. This parameter is now required for all static map requests.

maptype defines the type of map to construct. There are several possible maptype values, including satellite, terrain, hybrid, and mobile.

format (optional) defines the format of the resulting image. By default, the Static Maps API creates GIF images. There are several possible formats including GIF, JPEG and PNG types. Which format you use depends on how you intend to present the image. JPEG typically provides greater compression, while GIF and PNG provide greater detail. This version supports only PNG.

extraURL custom URL suffix

RETURNIMAGE return image yes/no default: TRUE

GRAYSCALE Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY

NEWMAP if TRUE, query the Google server and save to destfile, if FALSE load from destfile.

SCALE use the API’s scale parameter to return higher-resolution map images. The scale value is multiplied with the size to determine the actual output size of the image in pixels, without changing the coverage area of the map

API_console_key API key (formerly optional, now mandatory). If missing, the function "stitches" a static map from map tiles

type choice of tile server


tileDir map tiles can be stored in a local directory, e.g. "~/mapTiles/Google/"

verbose level of verbosity
**GetMap**

**Value**

map structure or URL used to download the tile.

**Note**

Note that size is in order (lon, lat)

**Author(s)**

Markus Loecher

**See Also**

GetMap.bbox

**Examples**

```r
if (0){#takes too long to run for CRAN check

lat = c(40.702147,40.718217,40.711614);

lon = c(-74.012318,-74.015794,-73.998284);

center = c(mean(lat), mean(lon));

zoom <- min(MaxZoom(range(lat), range(lon)));

#this overhead is taken care of implicitly by GetMap.bbox();

markers = paste0("&markers=color:blue|label:S|40.702147,-74.015794&markers=color:",

"green|label:G|40.711614,-74.012318&markers=color:red|color:red|",

"label:C|40.718217,-73.998284")

myMap <- GetMap(center=center, zoom=zoom, markers=markers);
```
#Note that in the presence of markers one often needs to add some extra padding to the
#latitude range to accommodate the extent of the top most marker

#add a path, i.e. polyline:

myMap <- GetMap(center=center, zoom=zoom,

    path = paste0("&path=color:0x0000ff|weight:5|40.737102,-73.990318|",

    "40.749825,-73.987963|40.752946,-73.987384|40.755823,-73.986397")

);

#use implicit geo coding

BrooklynMap <- GetMap(center="Brooklyn", zoom=13)

PlotOnStaticMap(BrooklynMap)

#use implicit geo coding and display labels in Korean:

BrooklynMap <- GetMap(center="Brooklyn", zoom=13, hl="ko")

PlotOnStaticMap(BrooklynMap)

#no highways

ManHatMap <- GetMap(center="Lower Manhattan", zoom=14,

    extraURL="&style=feature:road.highway|visibility:off";

)
GetMap

destfile = "LowerManhattan.png")

PlotOnStaticMap(ManHatMap)

#reload the map without a new download:

ManHatMap <- GetMap(destfile = "LowerManhattan.png", NEWMAP=FALSE)

PlotOnStaticMap(ManHatMap)

#The example below defines a polygonal area within Manhattan, passed a series of
#intersections as locations:

#myMap <- GetMap(path = paste0("&path=color:0x00000000|weight:5|fillcolor:0xFFFF0033|",,
# "8th+Avenue+%26+34th+St,New+York,NY|8th+Avenue+%26+42nd+St,New+York,NY|",,
# "Park+Ave+%26+42nd+St,New+York,NY,NY|Park+Ave+%26+34th+St,New+York,NY, NY"),,
# destfile = "MyTile3a.png");

#note that since the path string is just appended to the URL you can "abuse" the path
#argument to pass anything to the query, e.g. the style parameter:

#The following example displays a map of Brooklyn where local roads have been changed
#to bright green and the residential areas have been changed to black:
GetMap

```r
# myMap <- GetMap(center="Brooklyn", zoom=12, maptype = "roadmap",

#path = paste0("&style=feature:road.local|element:geometry|hue:0x00ff00|",

# "saturation:100&style=feature:landscape|element:geometry|lightness:-100"),

# sensor='false', destfile = "MyTile4.png", RETURNIMAGE = FALSE);

#In the last example we set RETURNIMAGE to FALSE which is a useful feature in general
# if png is not installed. In that cases, the images can still be fetched

#and saved but not read into R.

#In the following example we let the Static Maps API determine the correct center and
#zoom level implicitly, based on evaluation of the position of the markers.

#However, to be of use within R we do need to know the values for zoom and
#center explicitly, so it is better practice to compute them ourselves and
#pass them as arguments, in which case meta information on the map tile can be saved as well.

#myMap <- GetMap(markers = paste0("&markers=color:blue|label:S|40.702147,-74.015794&",

# "markers=color:green|label:G|40.711614,-74.012318&markers=color:red|",
```
Description

Wrapper function for `GetMap`. Query the Google server for a static map tile, defined primarily by its lat/lon range and/or center and/or zoom.

Multiple additional arguments allow the user to customize the map tile.

Usage

```r
GetMap.bbox(lonR, latR, center, size = c(640, 640),
             destfile = "MyTile.png", MINIMUMSIZE = FALSE, RETURNIMAGE = TRUE,
             GRAYSCALE = FALSE, NEWMAP = TRUE, zoom, verbose = 0,
             SCALE = 1, type = c("google", "google-m", "google-s",
                                "osm", "osm-hot", "stamen-toner", "stamen-terrain",
                                "stamen-watercolor")[1], urlBase = "http://mt1.google.com/vt/lyrs=m",
             tileDir = "/tmp/", ...)```

Arguments

- `lonR` longitude range
- `latR` latitude range
- `center` optional center
size

destfile

MINIMUMSIZE

RETURNIMAGE

GRAYSCALE

NEWMAP

zoom

verbose

SCALE

type

urlBase

tileDir

extra arguments to GetMap

Value

map tile

Author(s)

Markus Loecher

Examples

```r
if (0){

mymarkers <- cbind.data.frame(lat = c(38.898648,38.889112, 38.880940),
lon = c(-77.037692, -77.050273, -77.03660), size = c('tiny','tiny','tiny'),
col = c('blue', 'green', 'red'), char = c(' ',' ',' '));

##get the bounding box:
```
GetMapTiles

bb <- qbbox(lat = mymarkers[, "lat"], lon = mymarkers[, "lon"]);

##download the map:

MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.png", GRAYSCALE = TRUE,

markers = mymarkers);

##The function qbbox() basically computes a bounding box for the given lat,lon
#points with a few additional options such as quantile boxes, additional buffers, etc.
bb <- qbbox(c(40.702147, 40.711614, 40.718217), c(-74.015794, -74.012318, -73.998284),

TYPE = "all", margin = list(m=rep(5,4), TYPE = c("perc", "abs")[1]));

##download the map:

MyMap <- GetMap.bbox(bb$slonR, bb$slatR, destfile = "MyTile3.png", maptype = "satellite")

}
Example OSM: http://a.tile.openstreetmap.org/10/549/335.png
Also see https://wiki.openstreetmap.org/wiki/Tile_servers
Example Google mobile: http://mt1.google.com/vt/lyrs=m&x=1325&y=3143&z=13
Example Google satellite: http://mt1.google.com/vt/lyrs=s&x=1325&y=3143&z=13

Usage

```r
GetMapTiles(center = c(lat = 52.431635, lon = 13.194773),
lonR, latR, nTiles = c(3, 3), zoom = 13, type = c("google",
"google-m", "google-s", "osm", "osm-hot", "stamen-toner",
"stamen-terrain", "stamen-watercolor")[1],
urlBase = "http://mt1.google.com/vt/lyrs=m", tileDir = "/tmp/",
CheckExistingFiles = TRUE, TotalSleep = NULL, tileExt = ".png",
returnTiles = TRUE, verbose = 0)
```

Arguments

- center: optional center (lat first, lon second)
- lonR: longitude range
- latR: latitude range
- nTiles: number of tiles in x and y direction
- zoom: Google maps zoom level
- type: choice of tile server
- tileDir: map tiles can be stored in a local directory, e.g. "~/mapTiles/Google/"
- CheckExistingFiles: logical, if TRUE check if files already exist and only download if not!
- TotalSleep: overall time (in seconds) that one is willing to add in between downloads. This is intended to lower the risk of a server denial. If NULL no call to Sys.sleep is executed
- tileExt: image type of tile
- returnTiles: return tiles in a list?
- verbose: level of verbosity
**Value**

list with important information

**Note**

Note that size is in order (lon, lat)

**Author(s)**

Markus Loecher

**See Also**

GetMap.bbox

**Examples**

```r
if (0){

  # OSM, Ireland
  xlim = c(-7, -3.5)
  ylim = c(51.35, 55.35)

  Dublin = c(lon=-6.266155, lat=53.350140)
  DublinMerc = geosphere_mercator(Dublin)

  ir.osm <- GetMapTiles(lonR=xlim, latR=ylim, zoom=7, verbose=1,
                        type = "osm", tileDir= TRUE)

  map = plotOSM(ir.osm)
}
par("usr") # A vector of the form c(x1, x2, y1, y2)

points(map$bbox$upperLeft, col=2, pch=20)

points(map$bbox$lowerRight, col=2, pch=20)

points(DublinMerc, col=2, pch=1, cex=1.5)

ir.stamenToner <- GetMapTiles(lonR=xlim, latR=ylim, zoom=7, verbose=0,
                              type = "stamen", tileDir=TRUE)

plotOSM(ir.stamenToner)

ir.stamenWater <- GetMapTiles(lonR=xlim, latR=ylim, zoom=7, verbose=1,
                              type = "stamen-watercolor", tileDir=TRUE)

plotOSM(ir.stamenWater)

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

zoom=5

nTiles = prod(NumTiles(lonR=c(-135,-66), latR=c(25,54), zoom=zoom))
us_google_5 = GetMapTiles(lonR=c(-135,-66), latR=c(25,54), zoom=zoom, TotalSleep = 2*nTiles,
                      type = "google", tileDir= TRUE, verbose = TRUE)

PlotOnMapTiles(us_google_5)

wtc_ll = getGeoCode("World Trade Center, NY")

wtc_google_15=GetMapTiles(wtc_ll, zoom=15,nTiles = c(3,3), type = "google",
                      tileDir= TRUE, verbose = 1)

PlotOnMapTiles(wtc_google_15)

wtc_google_16 =GetMapTiles(wtc_ll, zoom=16,nTiles = c(4,4), type = "google",
                      tileDir= TRUE, verbose=1)

PlotOnMapTiles(wtc_google_16)

wtc_stamen=GetMapTiles(wtc_ll, zoom=15,nTiles = c(3,3), verbose=1,
                      type = "stamen-toner", tileDir= TRUE)

PlotOnMapTiles(wtc_stamen)
### combine with leaflet:

#From: http://stackoverflow.com/questions/5050851/

# best-lightweight-web-server-only-static-content-for-windows

# To use Python as a simple web server just change your working directory to the folder with your static content and type

#python -m SimpleHTTPServer 8000, everything in the directory

# will be available at http://localhost:8000/

library(leaflet)

m = leaflet::leaflet() %>%
  addTiles( urlTemplate = "http://localhost:8000/mapTiles/OSM/{z}_{x}_{y}.png")

m = leaflet::leaflet() %>%
  addTiles( urlTemplate = "http://localhost:8000/mapTiles/Google/{z}_{x}_{y}.png")

m = m %>% leaflet::setView(-74.01312, 40.71180, zoom = 16)

m = m %>% leaflet::addMarkers(-74.01312, 40.71180)
GetOsmMap

Query the Open Street Map server for map tiles instead of Google Maps

Description
The querying parameters for Open Street Maps are somewhat different in this version.
Instead of a zoom, center and size, the user supplies a scale parameter and a lat/lon bounding box.
The scale determines the image size.

Usage
GetOsmMap(lonR = c(-74.02132, -73.98622), latR = c(40.69983, 40.72595), scale = 20000, destfile = "MyTile.png",
format = "png", RETURNIMAGE = TRUE, GRAYSCALE = FALSE,
NEWMAP = TRUE, verbose = 1, ...)

Arguments
lonR longitude range
latR latitude range
scale Open Street map scale parameter. The larger this value, the smaller the resulting map tile in memory. There is a balance to be struck between the lat/lon bounding box and the scale parameter.
destfile File to load the map image from or save to, depending on NEWMAP.
format (optional) defines the format of the resulting image.

RETURNIMAGE return image yes/no default: TRUE

GRAYSCALE Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY

NEWMAP if TRUE, query the Google server and save to destfile, if FALSE load from destfile.

verbose level of verbosity,

... extra arguments to be used in future versions

Value

map structure or URL used to download the tile.

Note

The OSM maptile server is frequently too busy to accomodate every request, so patience is warranted.

Author(s)

Markus Loecher

Examples

if (0) {

CologneMap <- GetOsmMap(lonR= c(6.89, 7.09), latR = c(50.87, 51), scale = 150000,

destfile = "Cologne.png");

PlotOnStaticMap(CologneMap, mar=rep(4,4), NEWMAP = FALSE, TrueProj = FALSE, axes= TRUE);

PrincetonMap <- GetOsmMap(lonR= c(-74.67102, -74.63943), latR = c(40.33804,40.3556),

scale = 12500, destfile = "Princeton.png");

png("PrincetonWithAxes.png", 1004, 732)
IdentifyPoints

identify points by clicking on map

Description
The user can try to identify lat/lon pairs on the map by clicking on them

Usage
IdentifyPoints(MyMap, n = 1, verbose = 0)

Arguments
- MyMap: map object
- n: the maximum number of points to locate.
- verbose: level of verbosity

Value
the lat/lon coordinates of the chosen points are returned

Author(s)
Markus Loecher

Examples

#The first step naturally will be to download a static map from the Google server. A simple example:

#Identify points:

#IdentifyPoints(MyMap,5)
Description

The incidents data frame has 5000 rows and 16 columns. These are 5000 random rows from the 2012 crime data recorded in San Francisco.

Usage

data(incidents)

Format

This data frame contains the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncidntNum</td>
<td>incident number assigned by the police</td>
</tr>
<tr>
<td>Category</td>
<td>Category of crime</td>
</tr>
<tr>
<td>Descript</td>
<td>longer description</td>
</tr>
<tr>
<td>DayOfWeek</td>
<td>day of week</td>
</tr>
<tr>
<td>Date</td>
<td>date</td>
</tr>
<tr>
<td>Time</td>
<td>time of day formatted as hh:mm</td>
</tr>
<tr>
<td>PdDistrict</td>
<td>police district</td>
</tr>
<tr>
<td>Resolution</td>
<td>was the crime resolved?</td>
</tr>
<tr>
<td>Location</td>
<td>location as address</td>
</tr>
<tr>
<td>lon</td>
<td>longitude</td>
</tr>
<tr>
<td>lat</td>
<td>latitude</td>
</tr>
<tr>
<td>violent</td>
<td>violent flag</td>
</tr>
<tr>
<td>HrOfDay</td>
<td>hour of day as 2-digit integer</td>
</tr>
<tr>
<td>TimeOfDay</td>
<td>hour of day as decimal number</td>
</tr>
<tr>
<td>HourOfWeek</td>
<td>hour of week as decimal number between 0-168</td>
</tr>
<tr>
<td>censusBlock</td>
<td>ID of census block</td>
</tr>
</tbody>
</table>

Details

crime data recorded in San Francisco
LatLon2XY

Source
URL https://data.sfgov.org/

Examples

```r
data(incidents)
table(incidents$Category)
```

LatLon2XY computes the coordinate transformation from lat/lon to map tile coordinates.

Description

The function LatLon2XY(lat,lon,zoom) computes the coordinate transformation from lat/lon to map tile coordinates given a zoom level.

It returns the tile coordinates as well as the pixel coordinates within the Tile itself.

Usage

```r
LatLon2XY(lat, lon, zoom)
```

Arguments

- `lat`: latitude values to transform
- `lon`: longitude values to transform
- `zoom`: zoom level. lat, lon, zoom

Value

A list with values

- `Tile`: integer numbers specifying the tile
- `Coords`: pixel coordinate within the Tile

Note

The fractional part times 256 is the pixel coordinate within the Tile itself.

Author(s)

Markus Loecher
Examples

LatLon2XY(38.45, -122.375, 11)

Description

The function \texttt{LatLon2XY\_centered(MyMap, lat, lon, zoom)} computes the coordinate transformation from lat/lon to map tile coordinates given a map object.

Usage

\texttt{LatLon2XY\_centered(MyMap, lat, lon, zoom)}

Arguments

\begin{itemize}
  \item \texttt{MyMap} \hspace{1cm} map object
  \item \texttt{lat} \hspace{1cm} latitude values to transform
  \item \texttt{lon} \hspace{1cm} longitude values to transform
  \item \texttt{zoom} \hspace{1cm} optional zoom level. If missing, taken from \texttt{MyMap}
\end{itemize}

Value

\begin{itemize}
  \item \texttt{newX} \hspace{1cm} transformed longitude
  \item \texttt{newY} \hspace{1cm} transformed latitude
\end{itemize}

Author(s)

Markus Loecher

See Also

\texttt{LatLon2XY Tile2R}
MapBackground

get static Map from the Google server

Description

get static Map from the Google server

Usage

MapBackground(lat, lon, destfile, NEWMAP = TRUE, myTile, zoom = NULL, size = c(640, 640), GRAYSCALE = FALSE, mar = c(0, 0, 0, 0), PLOT = FALSE, verbose = 1, ...)

Arguments

lat center latitude
lon center longitude
destfile File to load the map image from or save to, depending on NEWMAP.
NEWMAP if TRUE, query the Google server and save to destfile, if FALSE load from destfile.
myTile map tile from previous downloads
zoom Google maps zoom level.
size desired size of the map tile image. defaults to maximum size returned by the Google server, which is 640x640 pixels
GRAYSCALE Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY
mar outer margin in plot; if you want to see axes, change the default
PLOT if TRUE, leave the plotting to PlotOnStaticMap, highly recommended
verbose level of verbosity
... further arguments to be passed to GetMap.bbox

Value

list containing the map tile

Author(s)

Markus Loecher
MaxZoom

*computes the maximum zoom level which will contain the given lat/lon range*

**Description**

computes the maximum zoom level which will contain the given lat/lon range

**Usage**

MaxZoom(latrange, lonrange, size = c(640, 640))

**Arguments**

- latrange: range of latitude values
- lonrange: range of longitude values
- size: desired size of the map tile image. defaults to maximum size returned by the Gogle server, which is 640x640 pixels

**Value**

zoom level

**Author(s)**

Markus Loecher

mypolygon

*simple wrapper function to plot colored polygons*

**Description**

same as polygon, except the value for color is taken from the 1st element of the extra column `col`

**Usage**

mypolygon(x, ...)

**Arguments**

- x: matrix containing columns X,Y,col
- ...: extra arguments passed to polygon

**Author(s)**

Markus Loecher
NumTiles

computes the necessary number of tiles from a bounding box and a zoom level

Description

computes the necessary number of tiles from a bounding box and a zoom level

Usage

NumTiles(lonR, latR, zoom = 13, CheckExistingFiles = TRUE,

tileExt = ".png", tileDir = "~/mapTiles/OSM/",

verbose = 0)

Arguments

lonR longitude range
latR latitude range
zoom zoom level
CheckExistingFiles logical, if TRUE check if files already exist and only download if not!
tileExt image type of tile
tileDir map tiles are stored in a local directory, e.g. "~/mapTiles/Google/"
verbose level of verbosity

Value
tuple with number of tiles for lon and lat extent

Author(s)
Markus Loecher

Examples

if (0){

#US bounding box:
NYleukemia

Upstate New York Leukemia Data

Description


Usage

data(NYleukemia)

Format

List with 5 items:

- geo
table of the FIPS code, longitude, and latitude of the geographic centroid of each census tract
- data
table of the FIPS code, number of cases, and population of each census tract
- spatial.polygon
object of class SpatialPolygons (See SpatialPolygons-class) containing a map of the study region
- surrounded
row IDs of the 4 census tracts that are completely surrounded by the surrounding census tracts
- surrounding
row IDs of the 4 census tracts that completely surround the surrounded census tracts
Source
http://www.sph.emory.edu/~lwaller/ch4index.htm

References

Examples
if (0) {
  data(NYleukemia)
  population <- NYleukemia$data$population
  cases <- NYleukemia$data$cases
  mapNY <- GetMap(center=c(lon=-76.00365, lat=42.67456), destfile = "NYstate.png", maptype = "mobile", zoom=9)
  ColorMap(100*cases/population, mapNY, NYleukemia$spatial.polygon, add = FALSE, alpha = 0.35, log = TRUE, location = "topleft")
}

osmtile_bbox compute the bounding box of an OpenStreetmap tile

Description
inspired by osmtile from the package OpenStreetmap
returns the Mercator projection bounding box

Usage
osmtile_bbox(x = 61, y = 41, zoom = 7, minim = -20037508)

Arguments
  x         x tile coordinate
  y         x tile coordinate
  zoom      zoom level
  minim     parameter for OSM projection

Value
  bounding box, Mercator projection

Author(s)
  Markus Loecher
pennLC Pennsylvania Lung Cancer

Description

County-level (n=67) population/case data for lung cancer in Pennsylvania in 2002, stratified on race (white vs non-white), gender and age (Under 40, 40-59, 60-69 and 70+). Additionally, county-specific smoking rates.

Usage

data(pennLC)

Format

List of 3 items:

- geo: a table of county IDs, longitude/latitude of the geographic centroid of each county
- data: a table of county IDs, number of cases, population and strata information
- smoking: a table of county IDs and proportion of smokers
- spatial.polygon: an object of class SpatialPolygons (See SpatialPolygons-class)

Source

Population data was obtained from the 2000 decennial census, lung cancer and smoking data were obtained from the Pennsylvania Department of Health website: http://www.dsf.health.state.pa.us/

See Also

NYleukemia

Examples

data(pennLC)
#pennLC$geo
#pennLC$data
#pennLC$smoking

# Map smoking rates in Pennsylvania
#mapvariable(pennLC$smoking[,2], pennLC$spatial.polygon)

PlotArrowsOnStaticMap plots arrows or segments on map
PlotArrowsOnStaticMap

Description

This function plots/overlays arrows or segments on a map.

Usage

PlotArrowsOnStaticMap(MyMap, lat0, lon0, lat1 = lat0,

lon1 = lon0, TrueProj = TRUE, FUN = arrows, add = FALSE,

verbose = 0, ...)

Arguments

MyMap       map image returned from e.g. GetMap()
lat0        latitude values of points FROM which to draw.
lon0        longitude values of points FROM which to draw.
lat1        latitude values of points TO which to draw.
lon1        longitude values of points TO which to draw.
TrueProj    set to FALSE if you are willing to accept some degree of inaccuracy in the
            mapping. In that case, the coordinates of the image are in lat/lon and the user
            can simply overly points/lines/axis without worrying about projections
FUN         , plotting function to use for overlay; typical choices would be arrows and segments
add         start a new plot or add to an existing
verbose     level of verbosity
...          further arguments to be passed to FUN

Value

return value of FUN

Author(s)

Markus Loecher

See Also

PlotOnStaticMap arrows
Examples

```r
if (0) {

  MyMap <- GetMap(center=c(lat=40.7,lon=-74), zoom=11)

  PlotArrowsOnStaticMap(MyMap, lat0=40.69, lon0=-73.9, lat1=40.71, lon1=-74.1, col = 'red')
}
```

### Description

Note the similarity in name to PBSmapping::plotMap

This function is the workhorse of the package RgoogleMaps. It overlays plot on background image of map tile.

### Usage

```r
plotmap(lat, lon, map, zoom = NULL, API = c("google",

  "OSM", "bing", "google2")[1], maptype = c("roadmap",

  "mobile", "satellite", "terrain", "hybrid", "mapmaker-roadmap",

  "mapmaker-hybrid")[2], destfile, data, alpha = 1,

  col = 1, apiKey = NULL, verbose = 0, ...)
```

### Arguments

- `lat`  latitude values to be overlaid OR string to be geocoded OR named vector (lat,lon)!
- `lon` longitude values to be overlaid
- `map` optional map object
- `zoom` Google maps zoom level
plotmap

API choice of map tile API
maptype defines the type of map to construct. There are several possible maptype values, including satellite, terrain, hybrid, and mobile.
destfile File to save the map image to
data data to look up variables in
alpha opacity
col plot color
apiKey optional API key (allows for higher rate of downloads for Google); mandatory for Bing maps
verbose level of verbosity
... further arguments to be passed to PlotOnStaticMap

Author(s)
Markus Loecher

Examples

if (0){

#############################Google maps####################################

mapBG1 = plotmap("Brandenburg Gate, Berlin", zoom = 15)

#############################bing maps########################################

# for Bing maps you will need your own API key,

apiKey = scan("bingAPIkey.txt",what="")

mapBG2 = plotmap("Brandenburg Gate, Berlin", zoom = 15, API = "bing", apiKey=apiKey)
latlon <- cbind.data.frame(lat = c(38.898648, 38.889112, 38.880940),

lon = c(-77.037692, -77.050273, -77.03660));

map3 = plotmap(lat = latlon$lat, lon = latlon$lon, API = "bing", apiKey=apiKey,

col = "purple", pch="X",cex=1.5)


#############################OSM maps##########################################

map4 = plotmap(lat = latlon$lat, lon = latlon$lon, API = "OSM", zoom=15,

col = "purple", pch="X",cex=1.5)

}
Usage

PlotOnMapTiles(mt, lat, lon, center, size = c(768, 768), add = FALSE, FUN = points, mar = c(0, 0, 0, 0), verbose = 0, ...)

Arguments

- **mt** list returned by `GetMapTiles`
- **lat** latitude values to be overlaid, if any
- **lon** longitude values to be overlaid, if any
- **center** optional center
- **size** size (in pixels) of "stitched" map
- **add** start a new plot or add to an existing
- **FUN** plotting function to use for overlay; typical choices would be `points` and `lines`
- **mar** outer margin in plot; if you want to see axes, change the default
- **verbose** level of verbosity
- **...** further arguments to be passed to `FUN`

Value

nothing returned

Author(s)

Markus Loecher

Examples

```r
if (0){

lat = c(40.702147, 40.718217, 40.711614);

lon = c(-74.012318, -74.015794, -73.998284);

center = c(mean(lat), mean(lon));
```
```r
zoom <- min(MaxZoom(range(lat), range(lon)));

bb=qbbox(lat,lon)

manhattan_osm = GetMapTiles(latR=bb$latR , lonR=bb$lonR,zoom=zoom,verbose=1)
PlotOnMapTiles(manhattan_osm,lat=lat,lon=lon,pch=20,col=c('red', 'blue', 'green'),cex=2)

manhattan_goo = GetMapTiles(latR=bb$latR , lonR=bb$lonR,zoom=zoom,
  tileDir= TRUE, type="google" )
PlotOnMapTiles(manhattan_goo,lat=lat,lon=lon,pch=20,col=c('red', 'blue', 'green'),cex=2)
```

### Description

This function is the workhorse of the package RgoogleMaps. It overlays plot on background image of map tile.

### Usage

```r
PlotOnStaticMap(MyMap, lat, lon, destfile, zoom = NULL,

  size, GRAYSCALE = FALSE, add = FALSE, FUN = points,
```
mar = c(0, 0, 0, 0), NEWMAP = TRUE, TrueProj = TRUE,

axes = FALSE, atX = NULL, atY = NULL, verbose = 0,

...)  

Arguments

MyMap optional map object
lat latitude values to be overlaid
lon longitude values to be overlaid
destfile File to load the map image from or save to, depending on whether MyMap was passed.
zoom Google maps zoom level. optional if MyMap is passed, required if not.
size desired size of the map tile image. defaults to maximum size returned by the Gogle server, which is 640x640 pixels
GRAYSCALE Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY
add start a new plot or add to an existing
FUN plotting function to use for overlay; typical choices would be points and lines
mar outer margin in plot; if you want to see axes, change the default
NEWMAP load map from file or get it "new" from the static map server
TrueProj set to FALSE if you are willing to accept some degree of inaccuracy in the mapping. In that case, the coordinates of the image are in lat/lon and the user can simply overly points/lines/axis without worrying about projections
axes overlay axes?
atX numeric; position of ticks on x-axis; if missing, axTicks is called for nice values; see axis
atY numeric; position of ticks on y-axis; if missing, axTicks is called for nice values; see axis
verbose level of verbosity
... further arguments to be passed to FUN

Value

the map object is returned via invisible(MyMap)

Author(s)

Markus Loecher
Examples

#The first step naturally will be to download a static map from the Google server. A simple example:

```r
if (0)
{

lat = c(40.702147, 40.711614, 40.718217);

lon = c(-74.015794, -74.012318, -73.998284);

center = c(mean(lat), mean(lon));

zoom <- min(MaxZoom(range(lat), range(lon)));

#this overhead is taken care of implicitly by GetMap.bbox();

MyMap <- GetMap(center=center, zoom=zoom, markers = paste0("&markers=color:blue|label:S|",

"40.702147,-74.015794&markers=color:green|label:G|40.711614,-74.012318&markers=",

"color:red|color:red|label:C|40.718217,-73.998284"), destfile = "MyTile1.png");

tmp <- PlotOnStaticMap(MyMap, lat = lat,

lon = lon,

destfile = "MyTile1.png", cex=1.5,pch=20,

col=c('red', 'blue', 'green'), add=FALSE);

#and add lines:
```
PlotOnStaticMap(MyMap, lat = c(40.702147, 40.711614, 40.718217),
lon = c(-74.015794, -74.012318, -73.998284),
lwd=1.5, col=c('red', 'blue', 'green'), FUN = lines, add=TRUE)

plotOSM

plots OSM map tiles

Description
places tiles on plot

Usage
plotOSM(mt, upperLeft, lowerRight, lat, lon, add = FALSE,
removeMargin = TRUE, verbose = 0, ...)

Arguments
mt list returned by GetMapTiles
upperLeft upperLeft corner in lat/lon of the plot region
lowerRight lowerRight corner in lat/lon of the plot region
lat latitude values to be overlaid, if any
lon longitude values to be overlaid, if any
add Boolean, whether to add to existing plot
removeMargin Boolean, whether to strip margins of plot
verbose level of verbosity
... further arguments to be passed to rasterImage

Value
returns map object invisibly
Author(s)

Markus Loecher

---

plotOSMtile  
*plots a single OSM tile*

---

Description

Adds tile to plot

Usage

```
plotOSMtile(osmtile, zoom, add = TRUE, raster = TRUE,
             verbose = 0, ...)  
```

Arguments

- `osmtile`: tile object
- `zoom`: zoom level
- `add`: Boolean, whether to add to existing plot
- `raster`: Boolean, whether to load raster image
- `verbose`: level of verbosity
- `...`: further arguments to be passed to `rasterImage`

Value

returns nothing

Author(s)

Markus Loecher
PlotPolysOnStaticMap  plots polygons on map

Description
This function plots/overlays polygons on a map. Typically, the polygons originate from a shapefile.

Usage
PlotPolysOnStaticMap(MyMap, polys, col, border = NULL,

  lwd = 0.25, verbose = 0, add = TRUE, textInPolys = NULL,

  ...)

Arguments
MyMap          map image returned from e.g. GetMap()
polys          or of class SpatialPolygons from the package sp polygons to overlay; these can be either of class PolySet from the package PB-Smapping
col            (optional) vector of colors, one for each polygon
border        the color to draw the border. The default, NULL, means to use par("fg"). Use border = NA to omit borders, see polygon
lwd            line width, see par
verbose       level of verbosity
add            start a new plot or add to an existing
textInPolys   text to be displayed inside polygons.
...            further arguments passed to PlotOnStaticMap

Author(s)
Markus Loecher

See Also
PlotOnStaticMap mypolygon

Examples

if (0){
  #require(PBSmapping);
  shpFile <- paste(system.file(package = "RgoogleMaps"), "/shapes/bg11_d00.shp", sep = "")
qbbox <- system.file('bg11_d00.shp', package = "RgoogleMaps");

shp <- PBSmapping::importShapefile(shp$projection = "LL");
bb <- qbbox(lat = shp[, "Y"], lon = shp[, "X"]);
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.png");
PlotPolysOnStaticMap(MyMap, shp, lwd=0.5, col = rgb(0.25, 0.25, 0.25, 0.025), add = F);

# Try an open street map:

mapOSM <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.png", type="osm");
PlotPolysOnStaticMap(mapOSM, shp, lwd=0.5, col = rgb(0.75, 0.25, 0.25, 0.15), add = F);

# North Carolina SIDS data set:

shpFile <- system.file("shapes/sids.shp", package="rgooglemaps");
shp <- PBSmapping::importShapefile(shpFile, projection = "LL");
bb <- qbbox(lat = shp[, "Y"], lon = shp[, "X"]);
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "SIDS.png");
# compute regularized SID rate
sid <- 100*attr(shp, "PolyData")$SID74/(attr(shp, "PolyData")$BIR74+500)
b <- as.integer(cut(sid, quantile(sid, seq(0, 1, length=8))));
b[is.na(b)] <- 1;
opal <- col2rgb(grey.colors(7), alpha=TRUE)/255; opal["alpha",] <- 0.2;
shp[, "col"] <- rgb(0.1, 0.1, 0.1, 0.2);
for (i in 1:length(b))
  shp[shp[, "PID"] == i, "col"] <- rgb(opal[1, b[i]], opal[2, b[i]], opal[3, b[i]], opal[4, b[i]]);
PlotPolysOnStaticMap(MyMap, shp, lwd=0.5, col = shp[, "col"], add = F);

# or choose an aspect ratio that corresponds better to North Carolina's elongated shape:
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "SIDS.png", size = c(640, 320), zoom = 7);
PlotPolysOnStaticMap(MyMap, shp, lwd=0.5, col = shp[, "col"], add = F);

qbbox computes bounding box

Description

The function qbbox computes a bounding box for the given lat,lon
points with a few additional options such as quantile boxes, additional margins, etc.

Usage

qbbox(lat, lon, TYPE = c("all", "quantile")[[1]], margin = list(m = c(1, 1, 1, 1), TYPE = c("perc", "abs")[[1]], q.lat = c(0.1, 0.9), q.lon = c(0.1, 0.9), verbose = 0)
Arguments

lat  latitude values
lon  longitude values
TYPE absolute or percentage trimming?
margin relative or absolute margin around the data. Set to NULL if no margin desired.
q.lat latitude quantile trimming, the tails will be trimmed from the bounding box
q.lon longitude quantile trimming,
verbose level of verbosity

Value

latR latitude range
lonR longitude range

Author(s)

Markus Loecher

Examples

lat = 37.85 + rnorm(100, sd=0.001);
lon = -120.47 + rnorm(100, sd=0.001);

#add a few outliers:
lat[1:5] <- lat[1:5] + rnorm(5, sd =.01);
lon[1:5] <- lon[1:5] + rnorm(5, sd =.01);

#range, discarding the upper and lower 10% of the data
qbbox(lat, lon, TYPE = "quantile");

#full range:
qbbox(lat, lon, TYPE = "all");

#add a 10% extra margin on all four sides:
qbbox(lat, lon, margin = list(m = c(10, 10, 10, 10), TYPE = c("perc", "abs")[1]));
**ReadMapTile** read a bitmap image stored in the PNG format

**Description**

Reads an image from a PNG file/content into a raster array.

**Usage**

ReadMapTile(destfile, METADATA = TRUE, native = TRUE)

**Arguments**

- **destfile**: png file to read
- **METADATA**: read MetaInfo as well?
- **native**: determines the image representation - if FALSE then the result is an array, if TRUE then the result is a native raster representation, see readPNG in package png.

**Value**

map or tile object

**Author(s)**

Markus Loecher

---

**RGB2GRAY** translates an RGB image matrix to gray scale

**Description**

This function translates the rgb values of the array myTile into a scalar matrix with just one gray value per pixel.

**Usage**

RGB2GRAY(myTile)

**Arguments**

- **myTile**: rgb image matrix, usually array with 3 dimensions

**Details**

Gray scale intensity defined as 0.30R + 0.59G + 0.11B
**SpatialToPBS**

**Value**
image tile

**Author(s)**
Markus Loecher

**Examples**

```
if (0){

  BrooklynLatLon = getGeoCode("Brooklyn")

  mapBrooklyn <- GetMap(center=BrooklynLatLon, destfile = file.path(tempdir(), "Brooklyn.png"),
                        zoom=11, size = c(240,240))

  mapBrooklynBW$myTile = RGB2GRAY(mapBrooklyn$myTile)

  PlotOnStaticMap(mapBrooklynBW)

  }
```

**SpatialToPBS**  
converts spatial objects as defined in package sp to simpler PBSmapping type dataframes

**Description**
The PlotPolysOnStaticMap() function currently does not take sp objects directly but instead needs PBSmapping type data.frames. This function converts sp objects into such.  
THANKS TO Fabio Priuli for a major bug fix w.r.t. holes in spatial polygons!

**Usage**
```
SpatialToPBS(xy, verbose = 0)
```
Arguments

xy  
spatial object, such as SpatialPoints, SpatialPolygons, etc..

verbose  
level of verbosity

Value

list with elements xy = converted object, bb = bounding box, fun = plot function

Author(s)

Markus Loecher

Examples

if (0) {

  data("NYleukemia", envir = environment())

  population <- NYleukemia$data$population

  cases <- NYleukemia$data$cases

  mapNY <- GetMap(center=c(lat=42.67456,lon=-76.00365),
                  destfile = file.path(tempdir(),"NYstate.png"),
                  maptype = "mobile", zoom=9)

  #mapNY=ReadMapTile("NYstate.png")

  clrStuff=ColorMap(100*cases/population, alpha = 0.35, log = TRUE)

  NYpolys = SpatialToPBS(NYleukemia$spatial.polygon)

  PlotPolysOnStaticMap(mapNY, NYpolys$xy, col = clrStuff$colcode, add = FALSE)

  legend("topleft", legend = clrStuff$legend, fill = clrStuff$fill,
           }
Description

The function sp_bbox computes a bounding box; it was copied from the sp package bbox function

Usage

sp_bbox(obj)

Arguments

obj  object deriving from class "Spatial", or one of classes: "Line", "Lines", "Polygon" or "Polygons", or ANY, which requires obj to be an array with at least two columns

Value

two-column matrix; the first column has the minimum, the second the maximum values; rows represent the spatial dimensions

Author(s)

Roger Bivand

Examples

# just 9 points on a grid:
x <- c(1,1,1,2,2,2,3,3,3)
y <- c(1,2,3,1,2,3,1,2,3)
xy <- cbind(x,y)
sp_bbox(xy)
**TextOnStaticMap**  
*plots text on map*

**Description**  
TextOnStaticMap draws the strings given in the vector labels at the coordinates given by x and y on a map. y may be missing since xy.coords(x,y) is used for construction of the coordinates.

**Usage**  
TextOnStaticMap(MyMap, lat, lon, labels = seq_along(lat),

TrueProj = TRUE, FUN = text, add = FALSE, verbose = 0,

...)

**Arguments**

- **MyMap**: map image returned from e.g. GetMap()
- **lat**: latitude where to put text.
- **lon**: longitude where to put text.
- **labels**: a character vector or expression specifying the text to be written. An attempt is made to coerce other language objects (names and calls) to expressions, and vectors and other classed objects to character vectors by as.character. If labels is longer than x and y, the coordinates are recycled to the length of labels.
- **TrueProj**: set to FALSE if you are willing to accept some degree of inaccuracy in the mapping. In that case, the coordinates of the image are in lat/lon and the user can simply overly points/lines/axis without worrying about projections
- **FUN**: overlay function, typical choice would be text
- **add**: start a new plot or add to an existing
- **verbose**: level of verbosity
- **...**: further arguments to be passed to FUN

**Value**

return value of FUN

**Author(s)**

Markus Loecher
Examples

if (0) {

lat = c(40.702147, 40.718217, 40.711614);

lon = c(-74.012318, -74.015794, -73.998284);

center = c(mean(lat), mean(lon));

zoom <- min(MaxZoom(range(lat), range(lon)));

MyMap <- GetMap(center = center, zoom = zoom, markers = paste0("&markers=color:blue|label:S|",

"40.702147,-74.015794&markers=color:green|label:G|40.711614,-74.012318&markers=",

"color:red|color:red|label:C|40.718217,-73.998284"), destfile = "MyTile1.png");

TextOnStaticMap(MyMap, lat = 40.711614, lon = -74.012318, "Some Text", cex = 2, col = 'red')

}

Tile2R

simple utility to offset and scale XY coordinates with respect to the center

Description

simple utility to offset and scale XY coordinates with respect to the center
Usage

Tile2R(points, center)

Arguments

points    XY coordinates returned by e.g. LatLon2XY
center    XY coordinates of center returned by e.g. LatLon2XY

Details

mainly used for shrinking the size of a tile to the minimum size.

Value

list with X and Y pixel values

Author(s)

Markus Loecher

Examples

latR <- c(34.5, 34.9);
lonR <- c(-100.3, -100);
lat.center <- 34.7;
lon.center <- -100.2;
zoom = 10;
ll <- LatLon2XY(latR[1], lonR[1], zoom); # lower left corner
ur <- LatLon2XY(latR[2], lonR[2], zoom); # upper right corner
cr <- LatLon2XY(lat.center, lon.center, zoom); # center
ll.Rcoords <- Tile2R(ll, cr);
updateusr

```
ur.Rcoords <- Tile2R(ur, cr);
```

**updateusr**

*Updates the 'usr' coordinates in the current plot.*

**Description**

For a traditional graphics plot this function will update the 'usr' coordinates by transforming a pair of points from the current usr coordinates to those specified.

**Usage**

```
updateusr(x1, y1 = NULL, x2, y2 = NULL)
```

**Arguments**

- `x1` The x-coords of 2 points in the current 'usr' coordinates, or anything that can be passed to `xy.coords`.
- `y1` The y-coords of 2 points in the current 'usr' coordinates, or an object representing the points in the new 'usr' coordinates.
- `x2` The x-coords for the 2 points in the new coordinates.
- `y2` The y-coords for the 2 points in the new coordinates.

**Details**

Sometimes graphs (in the traditional graphing scheme) end up with usr coordinates different from expected for adding to the plot (for example `barplot` does not center the bars at integers). This function will take 2 points in the current 'usr' coordinates and the desired 'usr' coordinates of the 2 points and transform the user coordinates to make this happen. The updating only shifts and scales the coordinates, it does not do any rotation or warping transforms.

If `x1` and `y1` are lists or matrices and `x2` and `y2` are not specified, then `x1` is taken to be the coordinates in the current system and `y1` is the coordinates in the new system.

Currently you need to give the function exactly 2 points in each system. The 2 points cannot have the same x values or y values in either system.
Value

An invisible list with the previous `usr` coordinates from `par`.

Note

Currently you need to give coordinates for exactly 2 points without missing values. Future versions of the function will allow missing values or multiple points.

Note by Markus Loecher: both the source and the documentations were copied from the package TeachingDemos version 2.3

Author(s)

Markus Loecher

Examples

tmp <- barplot(1:4)

updateusr(tmp[1:2], 0:1, 1:2, 0:1)

lines(1:4, c(1,3,2,2), lwd=3, type='b',col='red')

# update the y-axis to put a reference distribution line in the bottom
# quarter

tmp <- rnorm(100)

hist(tmp)

tmp2 <- par('usr')

xx <- seq(min(tmp), max(tmp), length.out=250)
yy <- dnorm(xx, mean(tmp), sd(tmp))

updateusr( tmp2[1:2], tmp2[3:4], tmp2[1:2], c(0, max(yy)*4) )

lines(xx,yy)

---

**XY2LatLon**

*computes the centered coordinate transformation from lat/lon to map tile coordinates*

### Description
The function `XY2LatLon(MyMap, X,Y, zoom)` computes the coordinate transformation from map tile coordinates to lat/lon given a map object.

### Usage
`XY2LatLon(MyMap, X, Y, zoom)`

### Arguments
- **MyMap** map object
- **X** latitude values to transform
- **Y** longitude values to transform
- **zoom** optional zoom level. If missing, taken from `MyMap`

### Value
properly scaled and centered (with respect to the center of `MyMap`) coordinates

- **lon** longitude
- **lat** latitude

### Author(s)
Markus Loecher

### See Also
- `LatLon2XY`  
- `Tile2R`
Examples

#quick test:

```r
zoom=12;MyMap <- list(40,-120,zoom, url="google", BBOX = list(ll=c(35,-125), ur=c(45,-115)))

LatLon <- c(lat = 40.0123, lon = -120.0123);

Rcoords <- LatLon2XY.centered(MyMap,LatLon["lat"],LatLon["lon"])

newLatLon <- XY2LatLon(MyMap, Rcoords$newX, Rcoords$newY)

max(abs(newLatLon - LatLon));
```

#more systematic:

```r
for (zoom in 2:10){

cat("zoom: ", zoom, "\n");

MyMap <- list(40,-120,zoom, url="google", BBOX = list(ll=c(35,-125), ur=c(45,-115)))

LatLon <- c(lat = runif(1,-80,80), lon = runif(1,-170,170));

Rcoords <- LatLon2XY.centered(MyMap,LatLon["lat"],LatLon["lon"])

newLatLon <- XY2LatLon(MyMap, Rcoords$newX, Rcoords$newY)

if(max(abs(newLatLon - LatLon)) > 0.0001) print(rbind(LatLon, newLatLon));
}
```
## Index

* **datasets**
  - `columbus`, 5
  - `incidents`, 34
  - `NYleukemia`, 40
  - `pennLC`, 42

- `AddAlpha`, 2
- `arrows`, 43
- `as.character`, 60
- `axis`, 7, 49
- `axTicks`, 7, 49

- `bbs (columbus)`, 5
- `col.gal.nb (columbus)`, 5
- `ColorMap`, 4
- `columbus`, 5
- `coords (columbus)`, 5

- `degreeAxis`, 7

- `expression`, 60

- `genStaticMap`, 8
- `geosphere_mercator`, 10
- `GetBingMap`, 11
- `getGeoCode`, 15
- `GetMap`, 16, 23, 24
- `GetMap.bbox`, 12, 19, 23, 27, 37
- `GetMapTiles`, 25
- `GetOsmMap`, 31

- `IdentifyPoints`, 33
- `incidents`, 34

- `LatLon2XY`, 35, 36, 62, 65
- `LatLon2XY.centered`, 36
- `lines`, 47, 49

- `MapBackground`, 37
- `MaxZoom`, 38

- `mypolygon`, 38, 53

- `NumTiles`, 39
- `NYleukemia`, 40, 42

- `osmtile_bbox`, 41

- `par`, 53
- `pennLC`, 42
- `PlotArrowsOnStaticMap`, 42
- `plotmap`, 44
- `PlotOnMapTiles`, 46
- `PlotOnStaticMap`, 37, 43, 48, 53
- `plotOSM`, 51
- `plotOSMtile`, 52
- `PlotPolysOnStaticMap`, 53
- `points`, 47, 49
- `polygon`, 38, 53
- `polys (columbus)`, 5
- `PolySet`, 53

- `qbbox`, 54

- `ReadMapTile`, 56
- `readPNG`, 56

- `RGB2GRAY`, 12, 18, 24, 32, 37, 49, 56

- `segments`, 43
- `sp_bbox`, 59
- `SpatialPolygons`, 53
- `SpatialPolygons-class`, 4, 40, 42
- `SpatialToPBS`, 57
- `Sys.sleep`, 26

- `text`, 60
- `TextOnStaticMap`, 60
- `Tile2R`, 36, 61, 65

- `updateusr`, 63

- `XY2LatLon`, 65