

# Package ‘RgoogleMaps’

January 2, 2012

**Type** Package

**Title** Overlays on Google map tiles in R

**Version** 1.1.9.15

**Date** 2011-08-02

**Depends** R (>= 1.4.0), graphics, stats, utils, png

**Suggests** grid, ReadImages, PBSmapping, sp

**Enhances** rgdal

**Author** Markus Loecher

**Maintainer** ‘Markus Loecher, Sense Networks’ <markus.loecher@gmail.com>

**Description** This package 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. NOTE: To do anything but downloading static map tiles, RgoogleMaps needs EITHER png OR ReadImages OR rimage installed ! png (default) is your package if you prefer png file format and ReadImages or rimage if you prefer jpg format. In the latter cases, you will also need the libjpeg library installed.

**License** GPL

**LazyLoad** yes

**Repository** CRAN

**Date/Publication** 2011-08-02 10:32:13

**R topics documented:**

RgoogleMaps-package	2
GetMap	4
GetMap.bbox	6
GetMap.OSM	8
LatLon2XY	9
LatLon2XY.centered	10
MapBackground	11
MaxZoom	12
myplot.imagematrix	13
mypolygon	13
PlotArrowsOnStaticMap	14
PlotOnStaticMap	15
PlotPolysOnStaticMap	17
qbbox	18
ReadMapTile	20
RGB2GRAY	20
SPGDF2matrix	21
TextOnStaticMap	22
Tile2R	23
updateusr	24
XY2LatLon	25
<b>Index</b>	<b>27</b>

---

RgoogleMaps-package     *Utilities to enable overlays on Google Maps*

---

**Description**

This package 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.

**Details**

Package:	RgoogleMaps
Type:	Package
Version:	1.1.9.15
Date:	2011-08-02
License:	GPL
LazyLoad:	yes

The two main functions in the package are `GetMap`, which queries the Google server for a map, and `PlotOnStaticMap`, which enables the actual overlay plots in R.

The examples below give a bit of support code for using the functions in this package. Details on using the functions themselves can be found in the documentation for those functions.

NOTE: To do anything but downloading static map tiles, RgoogleMaps needs EITHER png OR ReadImages installed ! png is your package if you prefer png file format and ReadImages/rimage if you prefer jpg format. In the latter cases, you will also need the libjpeg library installed.

### Author(s)

"Markus Loecher" <markus.loecher@gmail.com>

### Examples

#This section contains examples that will execute once you obtain an API key:

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

```
## Not run: MyMap <- GetMap(markers = '40.702147,-74.015794,blues%7C40.711614,-74.012318,greeng%7C40.718217,-73.998284')
```

```
## Not run: tmp <- PlotOnStaticMap(MyMap, lat = c(40.702147, 40.711614, 40.718217), lon = c(-74.015794, -74.012318, -73.998284))
#Of course, one can do without the markers and/or just pass a bounding box:
```

```
## Not run: MyMap <- GetMap(center=c(40.714728,-73.99867), zoom=14, destfile="MyTile2.png", matype="mobile")
```

#The function qbbox() basically computes a bounding box for the given lat,lon points with a few additional options s

```
## Not run: 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]));
```

```
## End(Not run)
```

```
## Not run: MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile="MyTile3.png", matype="satellite")
```

#The main function that overlays points (can easily be extended to lines or any other object) is PlotOnStaticMap().

#The following simple sequence of calls serves to test the coincidence of the markers placed by Google and the corre

#Define the markers:

```
## Not run: 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('','',''));
```

```
## End(Not run)
```

#get the bounding box:

```
## Not run: bb <- qbbox(lat = mymarkers[, "lat"], lon = mymarkers[, "lon"])
```

#download the map:

```
## Not run: MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile="DC.png", GRAYSCALE=T,
  markers = mymarkers);
```

```
## End(Not run)
```

#determine the max zoon, so that all points fit on the plot (not necessary in this case):

```

## Not run: zoom <- min(MaxZoom(latrange=bb$latR,lonrange=bb$lonR));

#plot:

## Not run: png("OverlayTest.png",640,640);

## Not run: tmp <- PlotOnStaticMap(MyMap,lat = mymarkers["lat"], lon = mymarkers["lon"],
                                cex=1.5,pch=20,col=c('blue', 'green', 'red'), add=F);
## End(Not run)

## Not run: tmp <- PlotOnStaticMap(MyMap,lat = mymarkers["lat"], lon = mymarkers["lon"],
                                col=c('purple'), add=T, FUN = lines, lwd = 2)
## End(Not run)

## Not run: dev.off()

```

---

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.

## Usage

```

GetMap(center, zoom = 12, markers, path = "", span, frame, hl, sensor = "true",
        maptypes = c("roadmap", "mobile", "satellite", "terrain", "hybrid", "mapmaker-roadmap", "mapmaker-hybrid"),
        format = c("gif", "jpg", "jpg-baseline", "png8", "png32")[5],
        size = c(640, 640), destfile = "MyTile.png", RETURNIMAGE = TRUE, GRAYSCALE = FALSE, verbose = 1)

```

## Arguments

center	lat/lon center of map
zoom	Google maps (integer) zoom level. Zoom levels between 0 (the lowest zoom level, in which the entire world can be seen on one map) to 19 (the highest zoom level, down to individual buildings) are possible within the normal maps view.
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 JPEG and PNG.
size	desired size of the map tile image. defaults to maximum size returned by the Gogle server, which is 640x640 pixels
destfile	File to save the map image to.
RETURNIMAGE	return image yes/no default: TRUE
GRAYSCALE	Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see <a href="#">RGB2GRAY</a>
verbose	level of verbosity

### Details

The optional markers can be passed either as a string, such as `#GetMap(markers = '40.702147,-74.015794,blues%7C40.702147,-74.015794&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&sensor=false` or as a dataframe: `'GetMap(markers = cbind(lat = c(40.70214, 40.71161), lon = c(-74.01`

### Value

URL used to download the tile.

### Note

Note: The Google Static Maps API does NOT requires a Maps API key any longer !! You can but do not have to sign up for a free API key at <http://code.google.com/apis/maps/signup.html>.

**Author(s)**

Markus Loecher <markus.loecher@gmail.com>

**References**

<http://code.google.com/apis/maps/documentation/staticmaps/>

**See Also**

[GetMap.bbox](#)

**Examples**

```

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();
## Not run: MyMap <- GetMap(center=center, zoom=zoom,markers = '&markers=color:blue|label:S|40.702147,-74.015794'
#Note that in the presence of markers one often needs to add some extra padding to the latitude range to accomodate

#add a path, i.e. polyline:
## Not run: MyMap <- GetMap(path = "&path=color:0x0000ff|weight:5|40.737102,-73.990318|40.749825,-73.987963|40.749825,-73.987963|40.737102,-73.990318"

#The example below defines a polygonal area within Manhattan, passed a series of intersections as locations:
# \dontrun{MyMap <- GetMap(path = "&path=color:0x00000000|weight:5|fillcolor:0xFFFF0033|8th+Avenue+%26+34th+S"

#note that since the path string is just appended to the URL you can "abuse" the path argument to pass anything to t
#The following example displays a map of Brooklyn where local roads have been changed to bright green and the resid
## Not run: GetMap(center='Brooklyn', zoom=12, matype = "roadmap", path = "&style=feature:road.local|element:g

#In the last example we set RETURNIMAGE to FALSE which is a useful feature in general if neither png nor ReadImages
#In the following example we let the Static Maps API determine the correct center and zoom level implicitly, based
## Not run: MyMap <- GetMap(markers = '&markers=color:blue|label:S|40.702147,-74.015794&markers=color:green|lab

```

---

GetMap.bbox

*download a static map from the Google server*

---

**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**

```
GetMap.bbox(lonR, latR, center, size = c(640, 640), destfile = "MyTile.png",  
            MINIMUMSIZE = FALSE, RETURNIMAGE = TRUE, GRAYSCALE = FALSE,  
            NEWMAP = TRUE, zoom, verbose = 1, ...)
```

**Arguments**

lonR	longitude range
latR	latitude range
center	optional center
size	desired size of the map tile image. defaults to maximum size returned by the Gogle server, which is 640x640 pixels
destfile	File to load the map image from or save to, depending on NEWMAP.
MINIMUMSIZE	reduce the size of the map to its minimum size that still fits the lat/lon ranges ? default: FALSE
RETURNIMAGE	return image yes/no default: TRUE
GRAYSCALE	Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see <a href="#">RGB2GRAY</a>
NEWMAP	if TRUE, query the Google server and save to destfile, if FALSE load from destfile.
zoom	Google maps zoom level. optional
verbose	level of verbosity
...	extra arguments to <a href="#">GetMap</a>

**Value**

map tile

**Note**

To handle png file formats, you will need the package png installed, for jpeg the package ReadImages.

**Author(s)**

Markus Loecher <markus.loecher@gmail.com>

**See Also**

[GetMap](#)

## Examples

```

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:

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

#download the map:

## Not run: MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.png", GRAYSCALE =T,
  markers = mymarkers);
## End(Not run)
#The function qbbox() basically computes a bounding box for the given lat,lon points with a few additional options s

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:
## Not run: MyMap <- GetMap.bbox(bb$lonR, bb$latR,destfile = "MyTile3.png", maptype = "satellite")

```

---

GetMap.OSM

*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

```
GetMap.OSM(lonR = c(-74.02132, -73.98622), latR = c(40.69983, 40.72595), scale = 20000, destfile = "MyT
```

## 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 <a href="#">RGB2GRAY</a>
NEWMAP	if TRUE, query the Google server and save to destfile, if FALSE load from destfile.
verbose	level of verbosity
...	extra arguments to <a href="#">GetMap</a>

**Note**

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

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

**Examples**

```
## Not run:
CologneMap <- GetMap.OSM(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 <- GetMap.OSM(lonR= c(-74.67102, -74.63943), latR = c(40.33804,40.3556), scale = 12500, destfile = "Princeton.png");
png("PrincetonWithAxes.png", 1004, 732)
  PlotOnStaticMap(PrincetonMap, axes = TRUE, mar = rep(4,4));
  dev.off()

## End(Not run)
```

---

LatLon2XY	<i>computes the coordinate transformation from lat/lon to map tile coordinates</i>
-----------	--

---

**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. Thanks to Neil Young (see [http://groups.google.com/group/Google-Maps-API/browse\\_thread/thread/d2103ac29e95696f?hl=en](http://groups.google.com/group/Google-Maps-API/browse_thread/thread/d2103ac29e95696f?hl=en)) for providing the formulae used.

**Usage**

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

**Arguments**

lat	vector of latitude values
lon	vector of longitude values
zoom	integer zoom level for Google Maps

**Value**

A list with values	
Tile	integer numbers specifying the tile
Coords	pixel coordinate within the Tile

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

**See Also**

[LatLon2XY.centered](#)

**Examples**

```
LatLon2XY(38.45, -122.375, 11)
```

---

LatLon2XY.centered	<i>computes the centered coordinate transformation from lat/lon to map tile coordinates</i>
--------------------	---

---

**Description**

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

**Usage**

```
LatLon2XY.centered(MyMap, lat, lon, zoom)
```

**Arguments**

MyMap	map object
lat	latitude values to transform
lon	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

newX                transformed longitude

newY                transformed latitude

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

**See Also**

[LatLon2XY Tile2R](#)

---

MapBackground	<i>get static Map from the Google server</i>
---------------	--

---

**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	vector latitude values for the map to contain
lon	vector of longitude values for the map to contain
destfile	File to load the map image from or save to, depending on the value of NEWMAP.
NEWMAP	get new map from teh Google server or load it from existing file
myTile	map tile from previous downloads
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 <a href="#">RGB2GRAY</a>
mar	outer margin in plot; if you want to see axes, change the default
PLOT	if TRUE, leave the plotting to <a href="#">PlotOnStaticMap</a> , highly recommended
verbose	level of verbosity
...	further arguments to be passed to <a href="#">GetMap.bbox</a>

**Value**

list containing the map tile

**Note**

To handle png file formats, you will need the package `rgdal` installed, for jpeg the package `ReadImages`.

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

**Examples**

```
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random,
##--or do help(data=index) for the standard data sets.
```

---

MaxZoom	<i>computes the maximum zoom level which will contain the given lat/lon range</i>
---------	---

---

**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	size of the map tile, defaults to 640x640

**Value**

zoom level

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

---

myplot.imagematrix      *Plotting an imagematrix object*

---

### Description

This function outputs an imagematrix object as an image.

### Usage

```
myplot.imagematrix(x,y,z, ...)
```

### Arguments

x	optional x coordinates
y	optional y coordinates
z	target image
...	plotting options

### Examples

```
## Not run:  
data(logo)  
plot(z=logo, main="plot(logo)")  
  
## End(Not run)
```

---

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 <a href="#">polygon</a>

### Author(s)

Markus Loecher, Sense Networks <markus@sensenetworks.com>

---

PlotArrowsOnStaticMap *plots arrows or segments on map*

---

### 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 = F
```

### Arguments

MyMap	map image returned from e.g. GetMap()
lat0, lon0	coordinates of points from which to draw.
lat1, lon1	coordinates 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 <a href="#">arrows</a> and <a href="#">segments</a>
add	add to existing map ?
verbose	level of verbosity
...	further arguments passed to <a href="#">arrows</a> or <a href="#">segments</a>

### Author(s)

Markus Loecher, Sense Networks <markus@sensenetworks.com>

### See Also

[PlotOnStaticMap mypolygon](#)

### Examples

```
## Not run:

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

shp=importShapefile(shpFile,projection="LL");
bb <- qbbox(lat = shp[,"Y"], lon = shp[,"X"]);
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.jpg");
PlotPolysOnStaticMap(MyMap, shp, lwd=.5, col = rgb(0.25,0.25,0.25,0.025), add = F);
```

```

#North Carolina SIDS data set:
shpFile <- system.file("shapes/sids.shp", package="mapprools");
shp=importShapefile(shpFile,projection="LL");
bb <- qbbox(lat = shp[, "Y"], lon = shp[, "X"]);
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "SIDS.jpg");
#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=.5, col = shp[, "col"], add = F);

#compare the accuracy of this plot to a Google Map overlay:
library(mapprools);
qk <- SpatialPointsDataFrame(as.data.frame(shp[, c("X", "Y")]), as.data.frame(shp[, c("X", "Y")]))
proj4string(qk) <- CRS("+proj=longlat");
tf <- "NC.counties";
SGqk <- GE_SpatialGrid(qk)
png(file=paste(tf, ".png", sep=""), width=SGqk$width, height=SGqk$height,
bg="transparent")
par(mar=c(0,0,0,0), xaxs="i", yaxs="i");par(mai = rep(0,4))
plotPolys(shp, plt=NULL)
dev.off()
kmlOverlay(SGqk, paste(tf, ".kml", sep=""), paste(tf, ".png", sep=""));
#This kml file can now be inspected in Google Earth or Google Maps

#or choose an aspect ratio that corresponds better to North Carolina's elongated shape:
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "SIDS.jpg", size = c(640, 320), zoom = 7);
PlotPolysOnStaticMap(MyMap, shp, lwd=.5, col = shp[, "col"], add = F);

## End(Not run)

```

---

PlotOnStaticMap

*overlays plot on background image of map tile*


---

## Description

This function is the workhorse of the package RgoogleMaps. It

## Usage

```

PlotOnStaticMap(MyMap, lat, lon, destfile, zoom=NULL, size = c(640,640), GRAYSCALE = FALSE,
add=FALSE, FUN = points, mar=c(0,0,0,0), NEWMAP = TRUE, TrueProj = TRUE, axes= FALSE, verbose

```

**Arguments**

MyMap	optional map object to be passed
lat	vector latitude values to be overlaid
lon	vector of 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 <a href="#">RGB2GRAY</a>
add	start a new plot or add to an existing
FUN	plotting function to use for overlay; typical choices would be <a href="#">points</a> and <a href="#">lines</a>
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 ? Note that if the package sp is installed, those axis will show labels in degrees.
verbose	level of verbosity
...	further arguments to be passed to FUN

**Details**

Note: To handle png file formats, you will need the package png installed, for jpeg the package ReadImages.

**Value**

the map object is returned via invisible(MyMap)

**Note**

Credit for the implementation of the png file reading goes to both Gabor Grothendieck (<http://www.mail-archive.com/r-sig-geo@stat.math.ethz.ch/msg04640.html>) and Michael Sumner (<http://finzi.psych.upenn.edu/R/Rhelp02a/archive/94605.html>). Credit for the implementation of the jpg file reading goes to Greg Snow: <https://stat.ethz.ch/pipermail/r-sig-geo/2009-March/005229.html>

**Author(s)**

Markus Loecher <markus.loecher@gmail.com>

**See Also**[GetMap](#) [GetMap.bbox](#)**Examples**

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

```
## Not run: MyMap <- GetMap(markers = '40.702147,-74.015794,blues%7C40.711614,-74.012318,greeng%7C40.718217,-73.998284')
## Not run: tmp <- PlotOnStaticMap(MyMap, lat = c(40.702147,40.711614,40.718217), lon = c(-74.015794,-74.012318,-73.998284),
  #and add lines:
  PlotOnStaticMap(MyMap, lat = c(40.702147,40.711614,40.718217), lon = c(-74.015794,-74.012318,-73.998284), lwd=2)
## End(Not run)
```

---

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 = .25, verbose = 0, add=TRUE, ...)
```

**Arguments**

MyMap	map image returned from e.g. <a href="#">GetMap()</a>
polys	polygons to overlay
col	(optional) vector of colors, one for each polygon
border	the color to draw the border. The default, NULL, means to use <a href="#">par("fg")</a> . Use <code>border = NA</code> to omit borders, see <a href="#">polygon</a>
lwd	line width, see <a href="#">par</a>
verbose	level of verbosity
add	start a new plot or add to an existing
...	further arguments passed to <a href="#">PlotOnStaticMap</a>

**Author(s)**

Markus Loecher <[markus.loecher@gmail.com](mailto:markus.loecher@gmail.com)>

**See Also**[PlotOnStaticMap](#) [mypolygon](#)

## Examples

```
## Not run:

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

shp=importShapefile(shpFile,projection="LL");
bb <- qbbox(lat = shp[, "Y"], lon = shp[, "X"]);
MyMap <- GetMap.bbox(bb$lonR, bb$latR, destfile = "DC.png");
PlotPolysOnStaticMap(MyMap, shp, lwd=.5, col = rgb(0.25,0.25,0.25,0.025), add = F);

#North Carolina SIDS data set:
shpFile <- system.file("shapes/sids.shp", package="mapprools");
shp=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=.5, col = shp[, "col"], add = F);

#compare the accuracy of this plot to a Google Map overlay:
library(mapprools);
qk <- SpatialPointsDataFrame(as.data.frame(shp[, c("X", "Y")]), as.data.frame(shp[, c("X", "Y")]))
proj4string(qk) <- CRS("+proj=longlat");
tf <- "NC.counties";
SGqk <- GE_SpatialGrid(qk)
png(file=paste(tf, ".png", sep=""), width=SGqk$width, height=SGqk$height,
bg="transparent")
par(mar=c(0,0,0,0), xaxs="i", yaxs="i");par(mai = rep(0,4))
plotPolys(shp, plt=NULL)
dev.off()
kmlOverlay(SGqk, paste(tf, ".kml", sep=""), paste(tf, ".png", sep=""));
#This kml file can now be inspected in Google Earth or Google Maps

#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=.5, col = shp[, "col"], add = F);

## End(Not run)
```

**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	if 'ALL' return just the ranges, if 'quantile' return the quantiles specified by q.lat and q.lon
margin	list of parameters describing any desired extra margin: <i>m</i> : margins for the four sides; <i>TYPE</i> : percentage or absolute margin ?
q.lat	lower and upper quantile for the longitude range
q.lon	lower and upper quantile for the latitude range
verbose	level of verbosity

**Value**

latR	latitude range
lonR	longitude range

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.com>

**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

*Reading a map tile object*

---

**Description**

This function reads a map tile and its associated metaInfo file as an image.

**Usage**

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

**Arguments**

destfile	image file to read from
METADATA	read MetaInfo as well ?

---

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  $0.30R + 0.59G + 0.11B$

**Value**

gray scale image matrix

**Author(s)**

Markus Loecher

---

 SPGDF2matrix

*image gridded spatial data, or convert to format for image*


---

### Description

This is a modification/ripoff from `as.image.SpatialGridDataFrame()` in package `sp`. It merely serves to convert an object of type `SpatialGridDataFrame` to a matrix

### Usage

```
SPGDF2matrix(x, xcol = 1, ycol = 2, col = heat.colors(12), red = 1, green = 2, blue = 3, axes = FALSE, xlim
```

### Arguments

<code>x</code>	object of class <a href="#">SpatialGridDataFrame</a>
<code>xcol</code>	column number of x-coordinate, in the coordinate matrix
<code>ycol</code>	column number of y-coordinate, in the coordinate matrix
<code>col</code>	a vector of colors
<code>red,green,blue</code>	columns names or numbers given instead of the <code>attr</code> argument when the data represent an image encoded in three colour bands on the 0-255 integer scale; all three columns must be given in this case, and the attribute values will be constructed using function <code>rgb</code>
<code>axes</code>	logical; should coordinate axes be drawn?
<code>xlim</code>	x-axis limits
<code>ylim</code>	y-axis limits
<code>add</code>	logical; if <code>FALSE</code> , the image is added to the plot layout setup by <code>plot(as(x, "Spatial"), axes=axes, xlim=xlim, ylim=ylim, asp=asp)</code> which sets up axes and plotting region; if <code>TRUE</code> , the image is added to the existing plot.
<code>PLOT</code>	plot, yes/no?
<code>...</code>	arguments passed to <a href="#">image</a> , see examples
<code>asp</code>	aspect ratio to be used for plot
<code>setParUsrBB</code>	default <code>FALSE</code> , see <a href="#">Spatial-class</a> for further details

### Author(s)

Markus Loecher, Sense Networks <markus@sensenetworks.com>

---

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 = FALSE)
```

### 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 <a href="#">expression</a> 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 <a href="#">as.character</a> . 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 overlay points/lines/axis without worrying about projections
FUN	overlay function, typical choice would be <a href="#">text</a>
add	add to existing map ?
verbose	level of verbosity
...	further arguments passed to <a href="#">text</a>

### Author(s)

Markus Loecher <markus.loecher@gmail.com>

### See Also

[PlotOnStaticMap text](#)

### Examples

```
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 = '&markers=color:blue|label:S|40.702147,-74.015794&markers=co
```

```
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. <a href="#">LatLon2XY</a>
center	XY coordinates of center returned by e.g. <a href="#">LatLon2XY</a>

### Details

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

### Value

X	X values of proper coordinate system
Y	Y values of proper coordinate system

### Author(s)

Markus Loecher, Sense Networks <markus@sensenetworks.com>

### 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);
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 <code>xy.coords</code> .
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)**

Greg Snow, <greg.snow@imail.org>

**See Also**

[par](#)

**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	<i>computes the centered coordinate transformation from lat/lon to map tile coordinates</i>
-----------	---

---

**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, Sense Networks <markus@sensenetworks.com>

**See Also**

[LatLon2XY](#) [Tile2R](#)

**Examples**

```
#quick test:
```

```
zoom=12;MyMap <- list(40,-120,zoom);
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:
```

```
for (zoom in 2:10){
  cat("zoom: ", zoom, "\n");
  MyMap <- list(40,-120,zoom);
  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

- \*Topic **aplot**
  - updateusr, [24](#)
- \*Topic **dplot**
  - updateusr, [24](#)
- \*Topic **package**
  - RgoogleMaps-package, [2](#)
  
- arrows, [14](#)
- as.character, [22](#)
  
- expression, [22](#)
  
- GetMap, [4](#), [6](#), [7](#), [9](#), [17](#)
- GetMap.bbox, [6](#), [6](#), [11](#), [17](#)
- GetMap.OSM, [8](#)
  
- image, [21](#)
  
- LatLon2XY, [9](#), [11](#), [23](#), [26](#)
- LatLon2XY.centered, [10](#), [10](#)
- lines, [16](#)
  
- MapBackground, [11](#)
- MaxZoom, [12](#)
- myplot.imagematrix, [13](#)
- mypolygon, [13](#), [14](#), [17](#)
  
- par, [17](#), [25](#)
- PlotArrowsOnStaticMap, [14](#)
- PlotOnStaticMap, [11](#), [14](#), [15](#), [17](#), [22](#)
- PlotPolysOnStaticMap, [17](#)
- points, [16](#)
- polygon, [13](#), [17](#)
  
- qbbox, [18](#)
  
- ReadMapTile, [20](#)
- RGB2GRAY, [5](#), [7](#), [9](#), [11](#), [16](#), [20](#)
- RgoogleMaps (RgoogleMaps-package), [2](#)
- RgoogleMaps-package, [2](#)
  
- segments, [14](#)
  
- Spatial-class, [21](#)
- SpatialGridDataFrame, [21](#)
- SPGDF2matrix, [21](#)
  
- text, [22](#)
- TextOnStaticMap, [22](#)
- Tile2R, [11](#), [23](#), [26](#)
  
- updateusr, [24](#)
  
- XY2LatLon, [25](#)