

# Package ‘RgoogleMaps’

March 15, 2012

**Type** Package

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

**Version** 1.2.0

**Date** 2012-02-27

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

**Suggests** PBSmapping, maptools

**Author** ‘Markus Loecher, Berlin School of Economics and Law (BSEL)’ <markus.loecher@gmail.com>

**Maintainer** ‘Markus Loecher’ <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.

**License** GPL

**LazyLoad** yes

**Repository** CRAN

**Date/Publication** 2012-03-15 20:25:31

## R topics documented:

RgoogleMaps-package	2
degAxis	3
GetMap	4
GetMap.bbox	6
GetMap.OSM	8
LatLon2XY	9
LatLon2XY.centered	10

MapBackground	11
MaxZoom	12
mypolygon	13
PlotArrowsOnStaticMap	13
PlotOnStaticMap	14
PlotPolysOnStaticMap	16
qbbox	18
ReadMapTile	19
RGB2GRAY	20
TextOnStaticMap	20
Tile2R	22
updateusr	23
XY2LatLon	24

<b>Index</b>	<b>26</b>
--------------	-----------

---

RgoogleMaps-package      *Overlays on Google map tiles in R*

---

## 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
Title:	Overlays on Google map tiles in R
Version:	1.2.0
Date:	2012-02-27
Depends:	R (>= 1.4.0), graphics, stats, utils, png
Suggests:	PBSmapping, maptools
Author:	"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>
Maintainer:	"Markus Loecher" <markus.loecher@gmail.com>
License:	GPL
LazyLoad:	yes
Packaged:	2011-08-02 07:24:03 UTC; mloecher
Repository:	CRAN
Date/Publication:	2011-08-02 10:32:13

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

---

degAxis                      *axis with degrees*

---

**Description**

add an axis with degree labels

**Usage**

```
degAxis(side,  
        at,  
        labels,  
        MyMap,  
        ...)
```

**Arguments**

side	integer; see <a href="#">axis</a>
at	numeric; if missing, <a href="#">axTicks</a> is called for nice values; see <a href="#">axis</a>
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 <a href="#">axis</a>
MyMap	optional map object to be passed
...	optional arguments to <a href="#">axis</a>

**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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**Examples**

```
xy = cbind(x = 2 * runif(100) - 1, y = 2 * runif(100) - 1)  
plot(xy, xlim=c(-1,1), ylim=c(-1,1))  
degAxis(1)  
degAxis(2, at = c(-1,-0.5,0,0.5,1))
```

---

 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,
       size = c(640,
               640),
       destfile = "MyTile.png",
       zoom = 12,
       markers,
       path = "",
       span,
       frame,
       hl,
       sensor = "true",
       maptype = c("roadmap",
                  "mobile",
                  "satellite",
                  "terrain",
                  "hybrid",
                  "mapmaker-roadmap",
                  "mapmaker-hybrid")[4],
       format = c("gif",
                  "jpg",
                  "jpg-baseline",
                  "png8",
                  "png32")[5],
       RETURNIMAGE = TRUE,
       GRAYSCALE = FALSE,
       NEWMAP = TRUE,
       verbose = 1)
```

### Arguments

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.
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.
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

**Value**

map structure or URL used to download the tile.

**Note**

Note that size is in order (lon, lat) !

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**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();
MyMap <- GetMap(center=center, zoom=zoom,markers = "&markers=color:blue|label:S|40.702147,-74.015794&markers=col
#Note that in the presence of markers one often needs to add some extra padding to the latitude range to accomodate t

#add a path, i.e. polyline:
MyMap <- GetMap(path = "&path=color:0x0000ff|weight:5|40.737102,-73.990318|40.749825,-73.987963|40.752946,-73.9

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

#note that since the path string is just appended to the URL you can "abuse" the path argument to pass anything to the
#The following example displays a map of Brooklyn where local roads have been changed to bright green and the residen
# MyMap <- GetMap(center="Brooklyn", zoom=12, maptype = "roadmap", path = "&style=feature:road.local|element:geom

#In the last example we set RETURNIMAGE to FALSE which is a useful feature in general if neither png nor ReadImages a

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

#MyMap <- GetMap(markers = "&markers=color:blue|label:S|40.702147,-74.015794&markers=color:green|label:G|40.711614,-73.998284")

```

---

`GetMap.bbox`*GetMap.bbox*

---

**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 ?
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

### Author(s)

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

### 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:
```

```
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
```

```
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$lonR, bb$latR, destfile = "MyTile3.png", mapytype = "satellite")
```

---

GetMap.OSM	<i>Query the Open Street Map server for map tiles instead of Google Maps</i>
------------	--

---

## 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 = "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 <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 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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**Examples**

```
if (0) {
  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");
  PlotOnStaticMap(PrincetonMap, axes = TRUE, mar = rep(4,4));
  dev.off()
}
```

---

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. 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	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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.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),  
PLOT = FALSE,  
verbose = 1,  
...)
```

**Arguments**

lat	
lon	
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 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

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

---

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(latrangle,
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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

---

mypolygon	<i>simple wrapper function to plot colored polygons</i>
-----------	---

---

**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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

---

PlotArrowsOnStaticMap	<i>plots arrows or segments on map</i>
-----------------------	--

---

**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 overlay 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	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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**See Also**

[PlotOnStaticMap arrows](#)

**Examples**

```
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')
```

---

PlotOnStaticMap      *overlays plot on background image of map tile*

---

**Description**

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

**Usage**

```

PlotOnStaticMap(MyMap,
  lat,
  lon,
  destfile,
  zoom = NULL,
  size = c(640,
  640),
  GRAYSCALE = FALSE,
  add = FALSE,
  FUN = points,
  mar = c(0,
  0,
  0),
  NEWMAP = TRUE,
  TrueProj = TRUE,
  axes = FALSE,
  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 <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 ?
verbose	level of verbosity
...	further arguments to be passed to FUN

**Value**

the map object is returned via `invisible(MyMap)`

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**Examples**

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

```
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();
MyMap <- GetMap(center=center, zoom=zoom,markers = '&markers=color:blue|label:S|40.702147,-74.015794&markers=c

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=
```

---

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,
  ...)
```

**Arguments**

MyMap	map image returned from e.g. <code>GetMap()</code>
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 <code>par("fg")</code> . 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 <code>PlotOnStaticMap</code>

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**See Also**

[PlotOnStaticMap](#) [mypolygon](#)

**Examples**

```

if (0){
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);
}

```

---

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

<code>lat</code>	latitude values
<code>lon</code>	longitude values
<code>TYPE</code>	
<code>margin</code>	
<code>q.lat</code>	
<code>q.lon</code>	
<code>verbose</code>	

**Value**

latR	latitude range
lonR	longitude range

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.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

*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 <a href="#">readPNG</a> in package png.

**Value**

map or tile object

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

---

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**

image tile

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.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 = 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 <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 overly points/lines/axis without worrying about projections
FUN	overlay function, typical choice would be <a href="#">text</a>
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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**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**

list with X and Y pixel values

**Author(s)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.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	<i>Updates the 'usr' coordinates in the current plot.</i>
-----------	---

---

**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)**

"Markus Loecher, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.com>

**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, Berlin School of Economics and Law (BSEL)" <markus.loecher@gmail.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 **package**  
    RgoogleMaps-package, [2](#)

arrows, [14](#)  
as.character, [21](#)  
axis, [3](#)  
axTicks, [3](#)

degAxis, [3](#)

expression, [21](#)

GetMap, [4](#), [6](#), [7](#)  
GetMap.bbox, [6](#), [6](#), [12](#)  
GetMap.OSM, [8](#)

LatLon2XY, [9](#), [11](#), [22](#), [25](#)  
LatLon2XY.centered, [10](#)  
lines, [15](#)

MapBackground, [11](#)  
MaxZoom, [12](#)  
mypolygon, [13](#), [17](#)

par, [17](#)  
PlotArrowsOnStaticMap, [13](#)  
PlotOnStaticMap, [12](#), [14](#), [14](#), [17](#)  
PlotPolysOnStaticMap, [16](#)  
points, [15](#)  
polygon, [13](#), [17](#)

qbbox, [18](#)

ReadMapTile, [19](#)  
readPNG, [19](#)  
RGB2GRAY, [5](#), [7](#), [8](#), [12](#), [15](#), [20](#)  
RgoogleMaps-package, [2](#)

segments, [14](#)

text, [21](#)  
TextOnStaticMap, [20](#)

Tile2R, [11](#), [22](#), [25](#)  
updateusr, [23](#)  
XY2LatLon, [24](#)