

# Package ‘RgoogleMaps’

October 13, 2009

**Type** Package

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

**Version** 1.1.6

**Date** 2009-10-04

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

**Suggests** ReadImages, rgdal

**Author** Markus Loecher, Sense Networks

**Maintainer** Markus Loecher, Sense Networks <markus@sensenetworks.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 rgdal OR ReadImages OR rimage installed ! Such an OR dependency is difficult to express in the Depends field, so I moved those packages to suggested. rgdal 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** 2009-10-13 18:26:47

## R topics documented:

RgoogleMaps-package	2
GetMap	4
GetMap.bbox	6
LatLon2XY	7
LatLon2XY.centered	8

MapBackground . . . . .	9
MaxZoom . . . . .	10
mypolygon . . . . .	11
PlotOnStaticMap . . . . .	11
PlotPolysOnStaticMap . . . . .	13
qbbox . . . . .	13
RGB2GRAY . . . . .	14
SPGDF2matrix . . . . .	15
Tile2R . . . . .	16
updateusr . . . . .	17
XY2LatLon . . . . .	18

<b>Index</b>	<b>20</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:	1.1.6
Date:	2009-10-04
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 1: The Google Static Maps API requires a Maps API key. If you haven't already done so, sign up for a free API key at <http://code.google.com/apis/maps/signup.html>

NOTE 2: To do anything but downloading static map tiles, RgoogleMaps needs EITHER `rgdal` OR `ReadImages` installed ! Such an OR dependency is difficult to express in the 'Depends' field, so I moved both packages to suggested. `rgdal` 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.



```

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

```

## Arguments

key	Google maps API key; NOTE: if missing the function tries to read the key from the file 'API.key.txt' in the user's home directory.
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.
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.
verbose	level of verbosity

### Details

The optional markers can be passed either as a string, such as `'GetMap(markers = '40.702147,-74.015794,blue', 'MyTile1.png')'` or as a dataframe: `'GetMap(markers = cbind.data.frame(lat = c(40.70214, 40.71161), lon = c(-74.01579, -73.9982), size = c('tiny', 'mid'), col = c('blue', 'red'), char = c(' ', 'n')), destfile = "MyTile2.png");'`

### Value

URL used to download the tile.

### Note

Note: The Google Static Maps API requires a Maps API key. If you haven't already done so, sign up for a free API key at <http://code.google.com/apis/maps/signup.html>. Future versions will include the capability to read png tiles using the `rgdal` package. In fact, the code is already built in but commented out.

### Author(s)

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

**References**

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

**See Also**

[GetMap.bbox](#)

**Examples**

```
#The first step naturally will be to download a static map from the Google server. A
## Not run: GetMap(markers = '40.702147,-74.015794,blues%7C40.711614,-74.012318,greeng%7C4
```

---

GetMap.bbox	<i>download a static map from the Google server</i>
-------------	---

---

**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 `rgdal` installed, for jpeg the package `ReadImages`.

**Author(s)**

Markus Loecher, Sense Networks <markus@sensenetworks.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

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", matype = "s

```

---

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

<code>lat</code>	vector of latitude values
<code>lon</code>	vector of longitude values
<code>zoom</code>	integer zoom level for Google Maps

### Value

	A list with values
<code>Tile</code>	integer numbers specifying the tile
<code>Coords</code>	pixel coordinate within the Tile

### Author(s)

Markus Loecher, Sense Networks <[markus@sensenetWORKS.com](mailto:markus@sensenetWORKS.com)>

### See Also

[LatLon2XY.centered](#)

### Examples

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

---

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

---

### 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      *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, 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>
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>

---

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, Sense Networks <markus@sensenetworks.com>

---

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, verbose = 1, ...)
```

### 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>
verbose	level of verbosity
...	further arguments to be passed to FUN

### Details

NOte: To handle png file formats, you will need the package rgdal 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, Sense Networks <markus@sensenetworks.com>

**See Also**

[GetMap](#) [GetMap.bbox](#)

**Examples**

```
#The first step naturally will be to download a static map from the Google server. A simple
## Not run: GetMap(markers = '40.702147,-74.015794,blues%7C40.711614,-74.012318,greeng%7C4
## Not run: tmp <- PlotOnStaticMap(lat = c(40.702147,40.711614,40.718217), lon = c(-74.01
```

---

PlotPolysOnStaticMap  
*plots polygons on map*

---

**Description**

This function plots polygons on a map.

**Usage**

```
PlotPolysOnStaticMap(MyMap, polys, col, verbose = 1, ...)
```

**Arguments**

MyMap	map image returned from e.g. <code>GetMap()</code>
polys	polygons to overlay
col	(optional) vector of colors, one for each polygon
verbose	level of verbosity
...	further arguments passed to <code>PlotOnStaticMap</code>

**Author(s)**

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

**See Also**

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

---

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

```

qbbox(lat, lon, TYPE = "all");
#add a 10
qbbox(lat, lon, margin = list(m = c(10, 10, 10, 10), TYPE = c("perc", "abs")[1]));

```

---

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)
```

**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</code> (as <code>(x, "Spatial")</code> , <code>axes=axes</code> , <code>xlim=xlim</code> , <code>ylim=ylim</code> , <code>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>

---

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**

<code>points</code>	XY coordinates returned by e.g. <a href="#">LatLon2XY</a>
<code>center</code>	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 `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)**

Greg Snow, [greg.snow@imail.org](mailto: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, [17](#)

## \*Topic **dplot**

updateusr, [17](#)

## \*Topic **package**

RgoogleMaps-package, [2](#)

GetMap, [4](#), [6](#), [7](#), [12](#)

GetMap.bbox, [5](#), [6](#), [9](#), [12](#)

image, [16](#)

LatLon2XY, [7](#), [9](#), [16](#), [19](#)

LatLon2XY.centered, [8](#), [8](#)

lines, [12](#)

MapBackground, [9](#)

MaxZoom, [10](#)

mypolygon, [11](#), [13](#)

par, [18](#)

PlotOnStaticMap, [11](#), [13](#)

PlotPolysOnStaticMap, [13](#)

points, [12](#)

polygon, [11](#)

qbbox, [13](#)

RGB2GRAY, [6](#), [9](#), [11](#), [14](#)

RgoogleMaps

(*RgoogleMaps-package*), [2](#)

RgoogleMaps-package, [2](#)

Spatial-class, [16](#)

SpatialGridDataFrame, [15](#)

SPGDF2matrix, [15](#)

Tile2R, [9](#), [16](#), [19](#)

updateusr, [17](#)

XY2LatLon, [18](#)