Package 'RgoogleMaps'

January 27, 2015

Type Package

Title Overlays on Google map tiles in R

Version 1.2.0.7

Date 2015-01-20

Depends R (>= 2.10)

Imports graphics, stats, utils, png, RJSONIO

Suggests PBSmapping, maptools, sp, rgdal, loa, RColorBrewer

Author Markus Loecher

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

NeedsCompilation no

Repository CRAN

Date/Publication 2015-01-21 17:14:49

R topics documented:

RgoogleMaps-package 2
AddAlpha 3
bubbleMap 4
ColorMap 6
columbus 8
degreeAxis 9
DF2SpatialPointsDataFrame 10
getGeoCode 12
GetMap 13
GetMap.bbox 18

GetMap.OSM 20
IdentifyPoints 22
LatLon2XY 23
LatLon2XY.centered 24
MapBackground 25
MaxZoom 26
mypolygon 26
NYleukemia 27
pennLC 28
PlotArrowsOnStaticMap 29
PlotOnStaticMap 30
PlotPolysOnStaticMap 32
qbbox 36
ReadMapTile 38
RGB2GRAY 38
SpatialToPBS 39
TextOnStaticMap 41
Tile2R 43
updateusr 44
XY2LatLon 46

Index **49**

`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:	1.2.0.7
Date:	2015-01-20
Depends:	R (>= 2.10)
Imports:	graphics, stats, utils, png, RJSONIO
Suggests:	PBSmapping, maptools, sp, rgdal, loa, RColorBrewer
Author:	Markus Loecher
Maintainer:	Markus Loecher <markus.loecher@gmail.com>
License:	GPL
LazyLoad:	yes

Author(s)

Markus Loecher

AddAlpha *add alpha level to color that lacks one*

Description

add alpha level to color that lacks one

Usage

```
AddAlpha(plotclr, alpha = 0.5, verbose = 0)
```

Arguments

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

Value

modified color with alpha value

Author(s)

Markus Loecher

Examples

```
#example:

#require(RColorBrewer)

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

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

```
  plotclr = AddAlpha(plotclr,0.5)

} else {

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

}
```

---

bubbleMap *Create a bubble plot of spatial data on Google Maps*

Description

This function creates a bubble plot of spatial
data, with options for bicolour residual plots.

Usage

```
bubbleMap(SP, coords = c("x", "y"), crs = sp::CRS("+proj=longlat +datum=WGS84"),

    map, filename = "", zcol = 1, max.radius = 100, key.entries,

    do.sqrt = TRUE, colPalette = NULL, strokeColor = "#FFAA00",

    alpha = 0.7, strokeWeight = 1, LEGEND = TRUE, legendLoc = "topleft",

    verbose = 0)
```

Arguments

SP	object of class data.frame or SpatialPointsDataFrame-class with associated coordinate reference systems
coords	names of coordinate columns
crs	coordinate reference systems

map	map object; if missing map is downloaded from server
filename	filename to save the map under, IF map object not given
zcol	variable column name, or column number after removing spatial coordinates from x@data: 1 refers to the first non-coordinate column
max.radius	value for largest circle (the plotting symbols) in metre, circumcircle of triange or quadrangle (square)
key.entries	value for largest circle (the plotting symbols) in metre, circumcircle of triange or quadrangle (square)
do.sqrt	logical; if TRUE the plotting symbol area (sqrt(diameter)) is proportional to the value of the z-variable; if FALSE, the symbol size (diameter) is proportional to the z-variable
colPalette	colours to be used to fill plotting symbols; numeric vector of same size like key.entries colours to be used to fill features depending on attribute
strokeColor	the color to draw the border of circle (the plotting symbols)
alpha	the fill opacity between 0.0 and 1.0
strokeWeight	the stroke width in pixels
LEGEND	logical; if TRUE add bubbleLegend
legendLoc	the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by legend
verbose	level of verbosity

Value

##

map structure or URL used to download the tile.

Author(s)

Markus Loecher

Examples

```
data(lat.lon.meuse, package="loa", envir = environment())



map <- GetMap(center=c(lat=50.97494,lon=5.743606), zoom=13,

       size=c(480,480),destfile = file.path(tempdir(),"meuse.png"),
```

```
          maptype="mobile", SCALE = 1);


par(cex=1.5)

bubbleMap(lat.lon.meuse, coords = c("longitude","latitude"), map=map,

      zcol='zinc', key.entries = 100+ 100 * 2^(0:4));
```

ColorMap *Plot Levels of a Variable in a Colour-Coded Map*

Description

Plot Levels of a Variable in a Colour-Coded Map

Usage

```
ColorMap(values, map = NULL, polys = NULL, log = FALSE, nclr = 7,

    include.legend = list(TRUE), round = 3, brks = NULL, legend = NULL,

    location = "topright", rev = FALSE, alpha = 0.5, GRAY = FALSE,

    palette = c("YlOrRd", "RdYlGn", "Spectral")[1], textInPolys = NULL,

    ...)
```

Arguments

values	variable to plot
map	map object
polys	an object of class SpatialPolygons (See SpatialPolygons-class

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

Author(s)

Markus Loecher

Examples

```
if (interactive()){

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

population <- NYleukemia$data$population

cases <- NYleukemia$data$cases

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

                maptype = "mobile", zoom=9)

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

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

}
```

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

`columbus` *Columbus OH spatial analysis data set*

Description

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

Usage

```
data(columbus)
```

Format

This data frame contains the following columns:

AREA computed by ArcView

PERIMETER computed by ArcView

COLUMBUS_ internal polygon ID (ignore)

COLUMBUS_I another internal polygon ID (ignore)

POLYID yet another polygon ID

NEIG neighborhood id value (1-49); conforms to id value used in Spatial Econometrics book.

HOVAL housing value (in \$1,000)

INC household income (in \$1,000)

CRIME residential burglaries and vehicle thefts per thousand households in the neighborhood

OPEN open space in neighborhood

PLUMB percentage housing units without plumbing

DISCBD distance to CBD

X x coordinate (in arbitrary digitizing units, not polygon coordinates)

Y y coordinate (in arbitrary digitizing units, not polygon coordinates)

NSA north-south dummy (North=1)

NSB north-south dummy (North=1)
EW east-west dummy (East=1)
CP core-periphery dummy (Core=1)
THOUS constant=1,000
NEIGNO NEIG+1,000, alternative neighborhood id value

Details

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

Note

All source data files prepared by Luc Anselin, Spatial Analysis Laboratory, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, http://sal.agecon.uiuc.edu/datasets/columbus.zip.

Source

Anselin, Luc. 1988. Spatial econometrics: methods and models. Dordrecht: Kluwer Academic, Table 12.1 p. 189.

Examples

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

---

degreeAxis *axis with degrees*

Description

add an axis with degree labels

Usage

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

Arguments

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

Value

axis is plotted on current graph

Note

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

Author(s)

Markus Loecher

Examples

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

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

degreeAxis(1)

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

DF2SpatialPointsDataFrame

change data.frame to SpatialPointsDataFrame

Description

This function modifies an object of class data.frame to one of class SpatialPointsDataFrame

Usage

```
DF2SpatialPointsDataFrame(x, coords = c("x", "y"), crs = sp::CRS("+init=epsg:28992"))
```

Arguments

x	data frame to be converted
coords	which columns are coordinates
crs	projection scheme

Value

the new object of class SpatialPointsDataFrame

Author(s)

Markus Loecher

Examples

```
if (requireNamespace("sp", quietly = TRUE)) {

  data("meuse", package = "sp", envir = environment())

  meuseSP = DF2SpatialPointsDataFrame(meuse)



  sp::plot(meuseSP, asp = 1, cex = 4 * meuse$zinc/max(meuse$zinc),

       pch = 1, col = as.numeric(meuse$ffreq)+1 )

  data("meuse.riv", package = "sp", envir = environment())

  lines(meuse.riv)

} else {

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

}
```

getGeoCode *geocoding utility*

Description

Geocode your data using, R, JSON and Google Maps' Geocoding APIs
see http://allthingsr.blogspot.de/2012/01/geocode-your-data-using-r-json-and.html
and

Usage

```
getGeoCode(gcStr, verbose = 0)
```

Arguments

gcStr	adddress to geocode
verbose	level of verbosity

Value

returns lat/lon for address

Author(s)

Markus Loecher

Examples

```
getGeoCode("Brooklyn")

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

DF = cbind.data.frame(address=c("Berlin,Germany", "Princeton,NJ",

          "cadillac+mountain+acadia+national+park"), lat = NA, lon = NA)

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

GetMap *download a static map from the Google server*

Description

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

Usage

```
GetMap(center = c(lat = 42, lon = -76), size = c(640, 640), destfile,
    zoom = 12, markers, path = "", span, frame, hl, sensor = "true",
    maptype = c("roadmap", "mobile", "satellite", "terrain",
        "hybrid", "mapmaker-roadmap", "mapmaker-hybrid")[2],
    format = c("gif", "jpg", "jpg-baseline", "png8", "png32")[5],
    RETURNIMAGE = TRUE, GRAYSCALE = FALSE, NEWMAP = TRUE, SCALE = 1,
    API_console_key = NULL, verbose = 0)
```

Arguments

center	optional center (lat first,lon second)
size	desired size of the map tile image. defaults to maximum size returned by the 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 paramater 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 RGB2GRAY
NEWMAP	if TRUE, query the Google server and save to destfile, if FALSE load from destfile.
SCALE	use the API's scale parameter to return higher-resolution map images. The scale value is multiplied with the size to determine the actual output size of the image in pixels, without changing the coverage area of the map
API_console_key	optional API key (allows for higher rate of downloads)
verbose	level of verbosity

Value

map structure or URL used to download the tile.

Note

Note that size is in order (lon, lat)

Author(s)

Markus Loecher

See Also

GetMap.bbox

Examples

```
lat = c(40.702147,40.718217,40.711614);

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

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

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

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

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

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

                 "label:C|40.718217,-73.998284")

MyMap <- GetMap(center=center, zoom=zoom,markers=markers,destfile = "MyTile1.png");

#Note that in the presence of markers one often needs to add some extra padding to the

#latitude range to accomodate the extent of the top most marker



#add a path, i.e. polyline:

MyMap <- GetMap(center=center, zoom=zoom,destfile = "MyTile3.png",

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

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

  #use implicit geo coding

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

  PlotOnStaticMap(BrooklynMap)



  #use implicit geo coding and display labels in Korean:

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

  PlotOnStaticMap(BrooklynMap)



   #The example below defines a polygonal area within Manhattan, passed a series of

  #intersections as locations:

#MyMap <- GetMap(path = paste0("&path=color:0x00000000|weight:5|fillcolor:0xFFFF0033|",

#          "8th+Avenue+%26+34th+St,New+York,NY|8th+Avenue+%26+42nd+St,New+York,NY|",

#          "Park+Ave+%26+42nd+St,New+York,NY,NY|Park+Ave+%26+34th+St,New+York,NY,NY"),

#            destfile = "MyTile3a.png");



  #note that since the path string is just appended to the URL you can "abuse" the path

  #argument to pass anything to the query, e.g. the style parameter:
```

```
#The following example displays a map of Brooklyn where local roads have been changed

#to bright green and the residential areas have been changed to black:

# MyMap <- GetMap(center="Brooklyn", zoom=12, maptype = "roadmap",

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

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

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



 #In the last example we set RETURNIMAGE to FALSE which is a useful feature in general

#if png is not installed. In that cases, the images can still be fetched

#and saved but not read into R.



#In the following example we let the Static Maps API determine the correct center and

#zoom level implicitly, based on evaluation of the position of the markers.

#However, to be of use within R we do need to know the values for zoom and

#center explicitly, so it is better practice to compute them ourselves and

#pass them as arguments, in which case meta information on the map tile can be saved as well.



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

```
#          "markers=color:green|label:G|40.711614,-74.012318&markers=color:red|",
#          "color:red|label:C|40.718217,-73.998284"),
#           destfile = "MyTile1.png",  RETURNIMAGE = FALSE);
```

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 = 0, SCALE = 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 RGB2GRAY

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

Value

map tile

Author(s)

Markus Loecher

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 such as quantile boxes, additional buffers, etc.

bb <- qbbox(c(40.702147,40.711614,40.718217),c(-74.015794,-74.012318,-73.998284),

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

##download the map:

MyMap <- GetMap.bbox(bb$lonR, bb$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 = "MyTile.png", format = "png",

    RETURNIMAGE = TRUE, GRAYSCALE = FALSE, NEWMAP = TRUE, verbose = 1,

    ...)
```

Arguments

lonR	longitude range
latR	latitude range
scale	Open Street map scale parameter. The larger this value, the smaller the resulting map tile in memory. There is a balance to be struck between the lat/lon bounding box and the scale parameter.
destfile	File to load the map image from or save to, depending on NEWMAP.
format	(optional) defines the format of the resulting image.
RETURNIMAGE	return image yes/no default: TRUE
GRAYSCALE	Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY
NEWMAP	if TRUE, query the Google server and save to destfile, if FALSE load from destfile.
verbose	level of verbosity,
...	extra arguments to be used in future versions

Value

map structure or URL used to download the tile.

Note

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

Author(s)

Markus Loecher

Examples

```
if (interactive()) {

  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()

}
```

`IdentifyPoints` *identify points by clicking on map*

Description

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

Usage

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

Arguments

`MyMap`	map object
`n`	the maximum number of points to locate.
`verbose`	level of verbosity

Value

the lat/lon coordinates of the chosen points are returned

Author(s)

Markus Loecher

Examples

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



#identifiy points:

#IdentifyPoints(MyMap,5)
```

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) 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

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

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, mar = c(0, 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 RGB2GRAY
mar	outer margin in plot; if you want to see axes, change the default
PLOT	if TRUE, leave the plotting to PlotOnStaticMap, highly recommended
verbose	level of verbosity
...	further arguments to be passed to GetMap.bbox

Value

list containing the map tile

Author(s)

Markus Loecher

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

Description

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

Usage

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

Arguments

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

Value

zoom level

Author(s)

Markus Loecher

`mypolygon` *simple wrapper function to plot colored polygons*

Description

same as polygon, execept the value for color is taken from the 1st element of the exra column 'col'

Usage

```
mypolygon(x, ...)
```

Arguments

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

Author(s)

Markus Loecher

NYleukemia *Upstate New York Leukemia Data*

Description

Census tract level (n=281) leukemia data for the 8 counties in upstate New York from 1978-1982, paired with population data from the 1980 census.

Usage

```
data(NYleukemia)
```

Format

List with 5 items:

geo	table of the FIPS code, longitude, and latitude of the geographic centroid of each census tract
data	table of the FIPS code, number of cases, and population of each census tract
spatial.polygon	object of class SpatialPolygons (See SpatialPolygons-class) containing a map of the study region
surrounded	row IDs of the 4 census tracts that are completely surrounded by the surrounding census tracts
surrounding	row IDs of the 4 census tracts that completely surround the surrounded census tracts

Source

http://www.sph.emory.edu/~lwaller/ch4index.htm

References

Turnbull, B. W. et al (1990) Monitoring for clusters of disease: application to leukemia incidence in upstate New York *American Journal of Epidemiology*, **132**, 136–143

Examples

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

pennLC *Pennsylvania Lung Cancer*

Description

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

Usage

```
data(pennLC)
```

Format

List of 3 items:

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

Source

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

See Also

NYleukemia

Examples

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

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

PlotArrowsOnStaticMap *plots arrows or segments on map*

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

Value

return value of FUN

Author(s)

Markus Loecher

See Also

PlotOnStaticMap arrows

Examples

```
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,

    GRAYSCALE = FALSE, add = FALSE, FUN = points, mar = c(0,

        0, 0, 0), NEWMAP = TRUE, TrueProj = TRUE, axes = FALSE,

    atX = NULL, atY = NULL, verbose = 0, ...)
```

Arguments

`MyMap`	optional map object
`lat`	latitude values to be overlaid
`lon`	longitude values to be overlaid
`destfile`	File to load the map image from or save to, depending on whether `MyMap` was passed.
`zoom`	Google maps zoom level. optional if `MyMap` is passed, required if not.
`size`	desired size of the map tile image. defaults to maximum size returned by the Gogle server, which is 640x640 pixels
`GRAYSCALE`	Boolean toggle; if TRUE the colored map tile is rendered into a black & white image, see RGB2GRAY

add	start a new plot or add to an existing
FUN	plotting function to use for overlay; typical choices would be points and lines
mar	outer margin in plot; if you want to see axes, change the default
NEWMAP	load map from file or get it "new" from the static map server
TrueProj	set to FALSE if you are willing to accept some degree of inaccuracy in the mapping. In that case, the coordinates of the image are in lat/lon and the user can simply overly points/lines/axis without worrying about projections
axes	overlay axes ?
atX	numeric; position of ticks on x-axis; if missing, axTicks is called for nice values; see axis
atY	numeric; position of ticks on y-axis; if missing, axTicks is called for nice values; see axis
verbose	level of verbosity
...	further arguments to be passed to FUN

Value

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

Author(s)

Markus Loecher

Examples

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


  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 = paste0("&markers=color:blue|label:S|",
       "40.702147,-74.015794&markers=color:green|label:G|40.711614,-74.012318&markers=",
        "color:red|color:red|label:C|40.718217,-73.998284"), destfile = "MyTile1.png");


 tmp <- PlotOnStaticMap(MyMap, lat = c(40.702147,40.711614,40.718217),
                        lon = c(-74.015794,-74.012318,-73.998284),
                        destfile = "MyTile1.png", cex=1.5,pch=20,
                        col=c('red', 'blue', 'green'), add=FALSE);
 #and add lines:
 PlotOnStaticMap(MyMap, lat = c(40.702147,40.711614,40.718217),
                 lon = c(-74.015794,-74.012318,-73.998284),
                 lwd=1.5,col=c('red', 'blue', 'green'), FUN = lines, add=TRUE)
```

PlotPolysOnStaticMap *plots polygons on map*

Description

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

Usage

```
PlotPolysOnStaticMap(MyMap, polys, col, border = NULL, lwd = 0.25,

    verbose = 0, add = TRUE, textInPolys = NULL, ...)
```

Arguments

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

Author(s)

Markus Loecher

See Also

PlotOnStaticMap mypolygon

Examples

```
if (interactive()){

  #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);



#Try an open street map:



mapOSM <- GetMap.OSM(lonR=bb$lonR, latR=bb$latR, scale = 150000, destfile = "DC.png");

PlotPolysOnStaticMap(mapOSM, shp, lwd=.5, col = rgb(0.75,0.25,0.25,0.15), add = F);



#North Carolina SIDS data set:

shpFile <- system.file("shapes/sids.shp", package="maptools");

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(maptools);

qk <- SpatialPointsDataFrame(as.data.frame(shp[, c("X","Y")]), as.data.frame(shp[, c("X","Y")]))

sp::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))

PBSmapping::plotPolys(shp, plt=NULL)

dev.off()
```

```
maptools::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);

}
```

---

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	
margin	

q.lat

q.lon

verbose

Value

latR	latitude range
lonR	longitude range

Author(s)

Markus Loecher

Examples

```
lat = 37.85 + rnorm(100, sd=0.001);

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

#add a few outliers:

lat[1:5] <- lat[1:5] + rnorm(5, sd =.01);

lon[1:5] <- lon[1:5] + rnorm(5, sd =.01);



#range, discarding the upper and lower 10% of the data

qbbox(lat, lon, TYPE = "quantile");

#full range:

qbbox(lat, lon, TYPE = "all");

#add a 10% extra margin on all four sides:

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

ReadMapTile — *Read a bitmap image stored in the PNG format*

Description

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

Usage

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

Arguments

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

Value

map or tile object

Author(s)

Markus Loecher

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

Description

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

Usage

```
RGB2GRAY(myTile)
```

Arguments

myTile	rgb image matrix, usually array with 3 dimensions

Details

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

Value

image tile

Author(s)

Markus Loecher

Examples

```
if (interactive()){

  BrooklynLatLon = getGeoCode("Brooklyn")

  mapBrooklyn <- GetMap(center=BrooklynLatLon, destfile = file.path(tempdir(), "Brooklyn.png"),

                zoom=11, size = c(240,240))

  mapBrooklynBW$myTile = RGB2GRAY(mapBrooklyn$myTile)

  PlotOnStaticMap(mapBrooklynBW)

}
```

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

Description

The PlotPolysOnStaticMap() function currently does not take sp objects directly but instead needs PBSmapping type data.frames. This function converts sp objects into such.

Usage

```
SpatialToPBS(xy, verbose = 0)
```

Arguments

xy	spatial object, such as SpatialPoints, SpatialPolygons, etc..
verbose	level of verbosity

Value

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

Author(s)

Markus Loecher

Examples

```
if (interactive()) {

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

population <- NYleukemia$data$population

cases <- NYleukemia$data$cases

mapNY <- GetMap(center=c(lat=42.67456,lon=-76.00365),

                destfile = file.path(tempdir(),"NYstate.png"),

                maptype = "mobile", zoom=9)

#mapNY=ReadMapTile("NYstate.png")

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

NYpolys = SpatialToPBS(NYleukemia$spatial.polygon)

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

```
  legend("topleft", legend = clrStuff$legend, fill = clrStuff$fill,

         bg = rgb(0.1,0.1,0.1,0.3))

}
```

TextOnStaticMap *plots text on map*

Description

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

Usage

```
TextOnStaticMap(MyMap, lat, lon, labels = seq_along(lat), TrueProj = TRUE,

    FUN = text, add = FALSE, verbose = 0, ...)
```

Arguments

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

Value

return value of FUN

Author(s)

Markus Loecher

Examples

```
lat = c(40.702147,40.718217,40.711614);

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

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

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




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

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

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

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

Tile2R *simple utility to offset and scale XY coordinates with respect to the center*

Description

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

Usage

```
Tile2R(points, center)
```

Arguments

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

Details

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

Value

list with X and Y pixel values

Author(s)

Markus Loecher

Examples

```
latR <- c(34.5,34.9);

 lonR <- c(-100.3, -100);

 lat.center <- 34.7;

 lon.center <- -100.2;

 zoom = 10;

 ll <- LatLon2XY(latR[1], lonR[1], zoom);#lower left corner
```

```
ur <- LatLon2XY(latR[2], lonR[2], zoom );#upper right corner

cr <- LatLon2XY(lat.center, lon.center, zoom );#center

ll.Rcoords <- Tile2R(ll, cr);

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 matricies and `x2` and `y2` are not specified, then `x1` is taken to be the coordinates in the current system and `y1` is the coordinates in the new system.
Currently you need to give the function exactly 2 points in each system. The 2 points cannot have the same x values or y values in either system.

Value

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

Note

Currently you need to give coordinates for exactly 2 points without missing values. Future versions of the function will allow missing values or multiple points.
Note by Markus Loecher: both the source and the documentations were copied from the package TeachingDemos version 2.3

Author(s)

Markus Loecher

Examples

```
tmp <- barplot(1:4)

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

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




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

# quarter
```

```
tmp <- rnorm(100)

hist(tmp)

tmp2 <- par('usr')

xx <- seq(min(tmp), max(tmp), length.out=250)

yy <- dnorm(xx, mean(tmp), sd(tmp))

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

lines(xx,yy)
```

XY2LatLon *computes the centered coordinate transformation from lat/lon to map tile coordinates*

Description

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

Usage

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

Arguments

MyMap	map object
X	latitude values to transform
Y	longitude values to transform
zoom	optional zoom level. If missing, taken from MyMap

Value

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

lon	longitude
lat	latitude

Author(s)

Markus Loecher

See Also

LatLon2XY Tile2R

Examples

```
#quick test:


  zoom=12;MyMap <- list(40,-120,zoom, url="google");

  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, url="google");

  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 **datasets**
columbus, 8
NYleukemia, 27
pennLC, 28
∗Topic **package**
RgoogleMaps-package, 2

AddAlpha, 3
arrows, *29*
as.character, *41*
axis, *9*, *31*
axTicks, *9*, *31*

bbs *(*columbus*)*, 8
bubbleMap, 4

col.gal.nb *(*columbus*)*, 8
ColorMap, 6
columbus, 8
coords *(*columbus*)*, 8

degreeAxis, 9
DF2SpatialPointsDataFrame, 10

expression, *41*

getGeoCode, 12
GetMap, 13, *18, 19*
GetMap.bbox, *15*, 18, *25*
GetMap.OSM, 20

IdentifyPoints, 22

LatLon2XY, 23, *24*, *43*, *47*
LatLon2XY.centered, 24
lines, *31*

MapBackground, 25
MaxZoom, 26
mypolygon, 26, *33*

NYleukemia, 27, *28*

par, *33*
pennLC, 28
PlotArrowsOnStaticMap, 29
PlotOnStaticMap, *25*, *29*, 30, *33*
PlotPolysOnStaticMap, 32
points, *31*
polygon, *26*, *33*
polys *(*columbus*)*, 8
PolySet, *33*

qbbox, 36

ReadMapTile, 38
readPNG, *38*
RGB2GRAY, *14*, *18*, *21*, *25*, *30*, 38
RgoogleMaps *(*RgoogleMaps-package*)*, 2
RgoogleMaps-package, 2

segments, *29*
SpatialPointsDataFrame-class, *4*
SpatialPolygons, *33*
SpatialPolygons-class, *6*, *27, 28*
SpatialToPBS, 39

text, *41*
TextOnStaticMap, 41
Tile2R, *24*, 43, *47*

updateusr, 44

XY2LatLon, 46