

Package ‘maptools’

September 18, 2009

Version 0.7-26

Date 2009-09-17

Title Tools for reading and handling spatial objects

Encoding latin1

Author Nicholas J. Lewin-Koh and Roger Bivand, contributions by Edzer J. Pebesma, Eric Archer, Adrian Baddeley, Hans-Jörg Bibiko, Stéphane Dray, David Forrest, Patrick Giraudoux, Duncan Golicher, Virgilio Gómez Rubio, Patrick Hausmann, Thomas Jagger, Sebastian P. Luque, Don MacQueen, Andrew Niccolai and Tom Short

Maintainer Roger Bivand <Roger.Bivand@nhh.no>

Depends R (>= 2.4), foreign (>= 0.8), sp (>= 0.9-7), methods

Suggests spatstat, PBSmapping, maps, gpclib, RArcInfo

Description Set of tools for manipulating and reading geographic data, in particular ESRI shapefiles; C code used from shapelib. It includes binary access to GSHHS shoreline files. The package also provides interface wrappers for exchanging spatial objects with packages such as PBSmapping, spatstat, maps, RArcInfo, Stata tmap, WinBUGS, Mondrian, and others.

License GPL (>= 2)

Repository CRAN

Date/Publication 2009-09-18 20:25:13

R topics documented:

as.ppp	3
checkPolygonsHoles	5
ContourLines2SLDF	6
dotsInPolys	8
elide-methods	9
gcDestination	11
get.Pcent	12

getKMLcoordinates	13
GE_SpatialGrid	14
gpcholes	16
gzAzimuth	17
kmlLine	18
kmlOverlay	20
kmlPolygon	21
Map2poly	23
map2SpatialPolygons	25
maptools	27
nowrapRecenter	27
pal2SpatialPolygons	28
plot.Map	30
plot.polylist	31
pointLabel	33
ppp-class	35
read.shape	35
readAsciiGrid	37
readGPS	38
readShapeLines	40
readShapePoints	41
readShapePoly	42
readShapeSpatial	44
readSplus	45
Rgshhs	47
sp2Mondrian	49
sp2tmap	50
sp2WB	51
SpatialLines2PolySet	52
spCbind-methods	53
spChFIDs-methods	54
spRbind-methods	55
subset.polylist	56
sun-methods	57
symbolsInPolys	59
unionSpatialPolygons	61
write.linelistShape	62
write.pointShape	63
write.polylistShape	65
wrld_simpl	66

as.ppp

*coercion between sp objects and spatstat objects***Description**

S4-style as() coercion works between objects of S4 sp classes to spatstat S3 classes; direct function calls may also be used.

Usage

```
as.SpatialPoints.ppp(from)
as.SpatialPointsDataFrame.ppp(from)
as.SpatialGridDataFrame.ppp(from)
as.SpatialGridDataFrame.im(from)
as.psp.Line(from, ..., window=NULL, marks=NULL, fatal)
as.psp.Lines(from, ..., window=NULL, marks=NULL, fatal)
as.psp.SpatialLines(from, ..., window=NULL, marks=NULL, fatal)
as.psp.SpatialLinesDataFrame(from, ..., window=NULL, marks=NULL, fatal)
as.SpatialPolygons.tess(x)
as.SpatialPolygons.owin(x)
```

Arguments

from, x	object to coerce from
...	other arguments to be passed through
window	window as defined in the spatstat package
marks	marks as defined in the spatstat package
fatal	formal coercion argument

Methods

```
coerce signature(from = "SpatialPoints", to = "ppp")
coerce signature(from = "SpatialPointsDataFrame", to = "ppp")
coerce signature(from = "Line", to = "psp")
coerce signature(from = "Lines", to = "psp")
coerce signature(from = "SpatialLines", to = "psp")
coerce signature(from = "SpatialLinesDataFrame", to = "psp")
coerce signature(from = "SpatialGridDataFrame", to = "ppp")
coerce signature(from = "SpatialPolygons", to = "owin")
coerce signature(from = "SpatialPixelsDataFrame", to = "owin")
coerce signature(from = "SpatialGridDataFrame", to = "owin")
coerce signature(from = "SpatialGridDataFrame", to = "im")
```

```

coerce signature(from = "im", to = "SpatialGridDataFrame")
coerce signature(from = "ppp", to = "SpatialGridDataFrame")
coerce signature(from = "ppp", to = "SpatialPointsDataFrame")
coerce signature(from = "ppp", to = "SpatialPoints")
coerce signature(from = "owin", to = "SpatialPolygons")
coerce signature(from = "tess", to = "SpatialPolygons")

```

Note

When coercing a `SpatialPolygons` object to an `owin` object, full topology checking is enabled by default. To avoid checking, set `spatstat.options(checkpolygons=FALSE)` (from `spatstat` (1.14-6)). To perform the checking later, `owinpolycheck(W, verbose=TRUE)`.

Author(s)

Edzer Pebesma (e.pebesma@geo.uu.nl), Roger Bivand

Examples

```

library(spatstat)
data(meuse)
coordinates(meuse) = ~x+y
zn1 <- as(meuse["zinc"], "ppp")
zn1
plot(zn1)
as(as(meuse, "SpatialPoints"), "ppp")
data(meuse.grid)
gridded(meuse.grid) = ~x+y
mg_owin <- as(meuse.grid, "owin")
zn1a <- ppp(x=zn1$x, y=zn1$y, marks=zn1$marks, window=mg_owin)
zn1a
plot(zn1a)
rev_ppp_SP <- as.SpatialPoints.ppp(zn1a)
summary(rev_ppp_SP)
rev_ppp_SPDF <- as.SpatialPointsDataFrame.ppp(zn1a)
summary(rev_ppp_SPDF)
rev_ppp_SGDF <- as.SpatialGridDataFrame.ppp(zn1a)
summary(rev_ppp_SGDF)
data(meuse.riv)
mr <- Line(meuse.riv)
mr_psp <- as(mr, "psp")
mr_psp
plot(mr_psp)
mg_owin <- as(as(meuse.grid["ffreq"], "SpatialPixelsDataFrame"), "owin")
mg_owin
ho_sp <- SpatialPolygons(list(Polygons(list(Polygon(cbind(c(0,1,1,0,0),
  c(0,0,1,1,0))), Polygon(cbind(c(0.6,0.4,0.4,0.6,0.6),
  c(0.2,0.2,0.4,0.4,0.2))), hole=TRUE)), ID="ho"))
plot(ho_sp, col="red", pbg="pink")
ho <- as(ho_sp, "owin")

```

```
plot(ho)
pp <- runifpoint(500, win=ho)
plot(pp)
ho_orig <- owin(poly=list(list(x=c(0,1,1,0), y=c(0,0,1,1)),
  list(x=c(0.6,0.4,0.4,0.6), y=c(0.2,0.2,0.4,0.4))))
identical(ho, ho_orig)
ho_sp1 <- as(ho, "SpatialPolygons")
all.equal(ho_sp, ho_sp1, check.attributes=FALSE)
A <- tess(xgrid=0:4,ygrid=0:4)
A_sp <- as(A, "SpatialPolygons")
plot(A_sp)
text(coordinates(A_sp), sapply(slot(A_sp, "polygons"), slot, "ID"), cex=0.6)
mg_dist <- meuse.grid["dist"]
fullgrid(mg_dist) <- TRUE
image(mg_dist, axes=TRUE)
mg_im <- as(mg_dist, "im")
plot(mg_im)
mg2 <- as.SpatialGridDataFrame.im(mg_im)
image(mg2, axes=TRUE)
```

checkPolygonsHoles *Check holes in Polygons objects*

Description

The function checks holes in Polygons objects, using an intersection between a gpclib package `gpc.poly` object with one or more polygon contours and its bounding box to set the hole flag. The function will set single polygon contours to `hole=FALSE`, and if multiple polygon contours are holes, will set them `TRUE`.

Usage

```
checkPolygonsHoles(x)
```

Arguments

`x` An Polygons object as defined in package `sp`

Value

An Polygons object re-created from the input object.

Author(s)

Roger Bivand

Examples

```

library(sp)
library(gpclib)
nc1 <- readShapePoly(system.file("shapes/sids.shp", package="mapproj")[[1],
  proj4string=CRS("+proj=longlat +ellps=clrk66"))
pl <- slot(nc1, "polygons")
sapply(slot(pl[[4]], "Polygons"), function(x) slot(x, "hole"))
pl[[4]] <- Polygons(list(slot(pl[[4]], "Polygons")[[1]],
  Polygon(slot(slot(pl[[4]], "Polygons")[[2]], "coords"), hole=TRUE),
  slot(pl[[4]], "Polygons")[[3]]), slot(pl[[4]], "ID"))
sapply(slot(pl[[4]], "Polygons"), function(x) slot(x, "hole"))
pl_new <- lapply(pl, checkPolygonsHoles)
sapply(slot(pl_new[[4]], "Polygons"), function(x) slot(x, "hole"))
srs <- slot(slot(pl[[1]], "Polygons")[[1]], "coords")
hle2 <- structure(c(-81.64093, -81.38380, -81.34165, -81.66833, -81.64093,
  36.57865, 36.57234, 36.47603, 36.47894, 36.57865), .Dim = as.integer(c(5, 2)))
hle3 <- structure(c(-81.47759, -81.39118, -81.38486, -81.46705, -81.47759,
  36.56289, 36.55659, 36.49907, 36.50380, 36.56289), .Dim = as.integer(c(5, 2)))
x <- Polygons(list(Polygon(srs), Polygon(hle2), Polygon(hle3)),
  ID=slot(pl[[1]], "ID"))
sapply(slot(x, "Polygons"), function(x) slot(x, "hole"))
res <- checkPolygonsHoles(x)
sapply(slot(res, "Polygons"), function(x) slot(x, "hole"))
## Not run:
opar <- par(mfrow=c(1,2))
SPx <- SpatialPolygons(list(x))
plot(SPx)
text(coordinates(SPx),
  labels=sapply(slot(x, "Polygons"), function(i) slot(i, "hole")), cex=0.6)
title(xlab="Hole slot values before checking")
SPres <- SpatialPolygons(list(res))
plot(SPres)
text(coordinates(SPres),
  labels=sapply(slot(res, "Polygons"), function(i) slot(i, "hole")), cex=0.6)
title(xlab="Hole slot values after checking")
par(opar)
## End(Not run)

```

ContourLines2SLDF *Converter functions to build SpatialLinesDataFrame objects*

Description

These functions show how to build converters to `SpatialLinesDataFrame` objects: `ArcObj2SLDF` from the list returned by the `get.arcdata` function in the `RArcInfo` package; `ContourLines2SLDF` from the list returned by the `contourLines` function in the `graphics` package (here the data frame is just the contour levels, with one `Lines` object made up of at least one `Line` object per level); and `MapGen2SL` reads a file in "Mapgen" format into a `SpatialLines` object.

Usage

```
ArcObj2SLDF(arc, proj4string=CRS(as.character(NA)), IDs)
ContourLines2SLDF(cL, proj4string=CRS(as.character(NA)))
MapGen2SL(file, proj4string=CRS(as.character(NA)))
```

Arguments

arc	a list returned by the <code>get.arcdata</code> function in the <code>RArcInfo</code> package
IDs	vector of unique character identifiers; if not given, suitable defaults will be used, and the same values inserted as data slot row names
cL	a list returned by the <code>contourLines</code> function in the <code>graphics</code> package
proj4string	Object of class "CRS"; see CRS-class
file	filename of a file containing a Mapgen line data set

Value

A `SpatialLinesDataFrame` object

Note

Coastlines of varying resolution may be chosen online and downloaded in "Mapgen" text format from <http://www.ngdc.noaa.gov/mgg/shorelines/shorelines.html>, most conveniently using the interactive selection tool, but please note the 500,000 point limit on downloads, which is easy to exceed.

Author(s)

Roger Bivand; Edzer Pebesma

See Also

[SpatialLines-class](#)

Examples

```
#data(co37_d90_arc) # retrieved as:
# library(RArcInfo)
# fl <- "http://www.census.gov/geo/cob/bdy/co/co90e00/co37_d90_e00.zip"
# download.file(fl, "co37_d90_e00.zip")
# e00 <- zip.file.extract("co37_d90.e00", "co37_d90_e00.zip")
# e00toavc(e00, "ncar")
# arc <- get.arcdata(".", "ncar")
#res <- arcobj2SLDF(arc)
#plot(res)
#invisible(title(""))
res <- ContourLines2SLDF(contourLines(volcano))
plot(res, col=terrain.colors(nrow(as(res, "data.frame"))))
title("Volcano contours as SpatialLines")
```

`dotsInPolys` *Put dots in polygons*

Description

Make point coordinates for a dot density map

Usage

```
dotsInPolys(pl, x, f = "random", offset, compatible = FALSE)
```

Arguments

<code>pl</code>	list of polygons of class <code>polylist</code> , or an object of class <code>SpatialPolygons</code> or <code>SpatialPolygonsDataFrame</code>
<code>x</code>	integer vector of counts of same length as <code>pl</code> for dots
<code>f</code>	type of sampling used to place points in polygons, either "random" or "regular"
<code>offset</code>	for regular sampling only: the offset (position) of the regular grid; if not set, <code>c(0.5, 0.5)</code> , that is the returned grid is not random
<code>compatible</code>	what to return, if TRUE a a list of matrices of point coordinates, one matrix for each member of <code>pl</code> , if false a <code>SpatialPointsDataFrame</code> with polygon ID values

Details

With `f="random"`, the dots are placed in the polygon at random, `f="regular"` - in a grid pattern (number of dots not guaranteed to be the same as the count). When the polygon is made up of more than one part, the dots will be placed in proportion to the relative areas of the clockwise rings (anticlockwise are taken as holes). From `maptools` release 0.5-2, correction is made for holes in the placing of the dots, but depends on hole values being correctly set, which they often are not. The wrapper package `sppgc` may be used to check holes, see the `dont.run` section of the example.

Value

If `compatible=TRUE`, the function returns a list of matrices of point coordinates, one matrix for each member of `pl`. If `x[i]` is zero, the list element is `NULL`, and can be tested when plotting - see the examples. If `compatible=FALSE` (default), it returns a `SpatialPointsDataFrame` with polygon ID values as the only column in the data slot.

Note

Waller and Gotway (2004) *Applied Spatial Statistics for Public Health Data* (Wiley, Hoboken, NJ) explicitly warn that care is needed in plotting and interpreting dot density maps (pp. 81-83)

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also[spsample](#)**Examples**

```
x <- read.shape(system.file("shapes/sids.shp", package="mapprools")[1])
ncpolys <- Map2poly(x)
try1 <- dotsInPolys(ncpolys, as.integer(x$att.data$SID74), compatible=TRUE)
plot(ncpolys)
xx <- lapply(try1, function(x) {if (!is.null(x)) points(as.matrix(rbind(x)),
  pch=18, col="red")}) # added as.matrix(rbind(x)), thanks to Nicholas Grassly
try2 <- dotsInPolys(ncpolys, as.integer(x$att.data$SID74), f="regular", compatible=TRUE)
plot(ncpolys)
xx <- lapply(try2, function(x) {if (!is.null(x)) points(as.matrix(rbind(x)),
  pch=18, col="red")})
nc_SP <- readShapePoly(system.file("shapes/sids.shp", package="mapprools")[1],
  proj4string=CRS("+proj=longlat +ellps=clrk66"))
## Not run:
library(spgpc)
pls <- slot(nc_SP, "polygons")
pls_new <- lapply(pls, checkPolygonsHoles)
nc_SP <- SpatialPolygonsDataFrame(SpatialPolygons(pls_new,
  proj4string=CRS(proj4string(nc_SP))), data=as(nc_SP, "data.frame"))
## End(Not run)
try1 <- dotsInPolys(nc_SP, as.integer(nc_SP$SID74))
plot(nc_SP, axes=TRUE)
plot(try1, add=TRUE, pch=18, col="red")
try2 <- dotsInPolys(nc_SP, as.integer(nc_SP$SID74), f="regular")
plot(nc_SP, axes=TRUE)
plot(try2, add=TRUE, pch=18, col="red")
```

 elide-methods

Methods for Function elide in Package 'mapprools'

Description

Methods for function `elide` to translate and disguise coordinate placing in the real world.

Usage

```
elide(obj, ...)
```

Arguments

<code>obj</code>	object to be elided
<code>...</code>	other arguments:
	bb if NULL, uses bounding box of object, otherwise the given bounding box

shift values to shift the coordinates of the input object; this is made ineffective by the scale argument

reflect reverse coordinate axes

scale if NULL, coordinates not scaled; if TRUE, the longer dimension is scaled to lie within [0,1] and aspect maintained; if a scalar, the output range of [0,1] is multiplied by scale

flip translate coordinates on the main diagonal

rotate default 0, rotate angle degrees clockwise around center

center default NULL, if not NULL, the rotation center, numeric of length two

unitsq logical, default FALSE, if TRUE and scale TRUE, impose unit square bounding box (currently only points)

Value

The methods return objects of the input class object with elided coordinates; the coordinate reference system is not set. Note that if the input coordinates or centroids are in the data slot data.frame of the input object, they should be removed before the use of these methods, otherwise they will betray the input positions.

Methods

obj = "SpatialPoints" elides object

obj = "SpatialPointsDataFrame" elides object

obj = "SpatialLines" elides object

obj = "SpatialLinesDataFrame" elides object

obj = "SpatialPolygons" elides object

obj = "SpatialPolygonsDataFrame" elides object

Note

Rotation code kindly contributed by Don MacQueen

Examples

```
data(meuse)
coordinates(meuse) <- c("x", "y")
proj4string(meuse) <- CRS("+init=epsg:28992")
data(meuse.riv)
river_polygon <- Polygons(list(Polygon(meuse.riv)), ID="meuse")
rivers <- SpatialPolygons(list(river_polygon))
proj4string(rivers) <- CRS("+init=epsg:28992")
rivers1 <- elide(rivers, reflect=c(TRUE, TRUE), scale=TRUE)
meusel <- elide(meuse, bb=bbox(rivers), reflect=c(TRUE, TRUE), scale=TRUE)
opar <- par(mfrow=c(1,2))
plot(rivers, axes=TRUE)
plot(meuse, add=TRUE)
plot(rivers1, axes=TRUE)
plot(meusel, add=TRUE)
```

```

par(opar)
meusel <- elide(meuse, shift=c(10000, -10000))
bbox(meuse)
bbox(meusel)
rivers1 <- elide(rivers, shift=c(10000, -10000))
bbox(rivers)
bbox(rivers1)
meusel <- elide(meuse, rotate=-30, center=apply(bbox(meuse), 1, mean))
bbox(meuse)
bbox(meusel)
plot(meusel, axes=TRUE)

```

gcDestination *Find destination in geographical coordinates*

Description

Find the destination in geographical coordinates at distance `dist` and for the given bearing from the starting point given by `lon` and `lat`.

Usage

```

gcDestination(lon, lat, bearing, dist, dist.units = "km",
             model = NULL, Vincenty = FALSE)

```

Arguments

<code>lon</code>	longitude (Eastings) in decimal degrees (either scalar or vector)
<code>lat</code>	latitude (Northings) in decimal degrees (either scalar or vector)
<code>bearing</code>	bearing from 0 to 360 degrees (either scalar or vector)
<code>dist</code>	distance travelled (scalar)
<code>dist.units</code>	units of distance "km" (kilometers), "nm" (nautical miles), "mi" (statute miles)
<code>model</code>	choice of ellipsoid model ("WGS84", "GRS80", "Airy", "International", "Clarke", "GRS67")
<code>Vincenty</code>	logical flag, default FALSE

Details

The bearing argument may be a vector when `lon` and `lat` are scalar, representing a single point.

Value

A matrix of decimal degree coordinates with Eastings in the first column and Northings in the second column.

Author(s)

Eric Archer and Roger Bivand

References

<http://www.movable-type.co.uk/scripts/latlong.html#ellipsoid>,
<http://williams.best.vwh.net/avform.htm>,
<http://www.movable-type.co.uk/scripts/latlong-vincenty-direct.html>,
Original reference http://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf:
Vincenty, T. 1975. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey Review 22(176):88-93

See Also

[gzAzimuth](#)

Examples

```
data(state)
res <- gcDestination(state.center$x, state.center$y, 45, 250, "km")
plot(state.center$x, state.center$y, asp=1, pch=16)
arrows(state.center$x, state.center$y, res[,1], res[,2], length=0.05)
l1list <- vector(mode="list", length=length(state.center$x))
for (i in seq(along=l1list)) l1list[[i]] <- gcDestination(state.center$x[i],
  state.center$y[i], seq(0, 360, 5), 250, "km")
plot(state.center$x, state.center$y, asp=1, pch=3)
n11 <- lapply(l1list, lines)
```

get.Pcent

Polygon centroids

Description

return the centroids of a map of polygons

Usage

```
get.Pcent(theMap)
```

Arguments

theMap a Map object returned by read.shape()

Value

a matrix of centroids.

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand (Roger.Bivand@nhh.no)

Examples

```
x <- read.shape(system.file("shapes/sids.shp", package="mapproj")[1])
get.Pcent(x)
```

getKMLcoordinates *Get a list of coordinates out of a KML file*

Description

This function parses a KML file to get the content of <coordinates> tags and returns a list of matrices representing the longitude-latitude or if ignoreAltitude is FALSE the longitude-latitude-altitude coordinates of a KML geometry.

Usage

```
coords <- getKMLcoordinates(kmlfile, ignoreAltitude=FALSE)
```

Arguments

kmlfile connection object or a character string of the KML file
ignoreAltitude if set to TRUE the altitude values of a KML points will be ignored

Value

coords is a list of matrices representing the longitude-latitude or if ignoreAltitude is FALSE the longitude-latitude-altitude coordinates

Author(s)

Hans-J. Bibiko

See Also

[kmlPolygon](#), [kmlLine](#)

Examples

```
data(wrld_simpl)
## creates a KML file containing the polygons of South Africa (plus hole)
sw <- slot(wrld_simpl[wrld_simpl$NAME=="South Africa",], "polygons")[[1]]
tf <- tempfile()
kmlPolygon(sw, kmlfile=tf, name="South Africa", col="#df0000aa", lwd=5,
  border=4, kmlname="R Test",
  kmldescription="This is <b>only</b> a <a href='http://www.r-project.org'>R</a> test.")
zz <- getKMLcoordinates(tf, ignoreAltitude=TRUE)
str(zz)
```

 GE_SpatialGrid

Create SpatialGrid for PNG output to GE

Description

The function sets up metadata in the form of a `SpatialGrid` object for defining the size and placing of a PNG image overlay in Google Earth. The internal function `Sobj_SpatialGrid` can also be called to build a grid for arbitrary `Spatial*` objects.

Usage

```
GE_SpatialGrid(obj, asp = NA, maxPixels = 600)
Sobj_SpatialGrid(obj, asp=1, maxDim=100, n=NULL)
```

Arguments

<code>obj</code>	a <code>Spatial*</code> object
<code>asp</code>	if NA, will be set to the latitude corrected value
<code>maxPixels</code>	the maximum dimension of the output PNG
<code>maxDim</code>	the maximum dimension of the output grid; ignored if <code>n</code> not NULL
<code>n</code>	if not NULL, the minimum number of cells in the returned grid

Details

The function is used together with `kmlOverlay` to wrap around the opening of a PNG graphics device, plotting code, and the closing of the device. The computed values take account of the adjustment of the actual data bounding box to an integer number of rows and columns in the image file.

The approach may be used as an alternative to writing PNG files from `SpatialGrid` and `SpatialPixel` objects in `rgdal` using `writeGDAL`, and to writing KML files using `writeOGR` for vector data objects. The output PNG files are likely to be very much smaller than large vector data KML files, and hinder the retrieval of exact positional information.

Note that the geometries should be in geographical coordinates with datum WGS84 for export to KML.

Value

returns an S3 object of class GE_SG with components:

height	Integer raster height for png call
width	Integer raster width for png call
SG	a SpatialGrid object with the grid topology of the output PNG
asp	the aspect value used
xlim	xlim taken from SG
ylim	ylim taken from SG

Author(s)

Duncan Golicher, David Forrest and Roger Bivand

See Also

[kmlOverlay](#)

Examples

```
opt_exask <- options(example.ask=FALSE)
qk <- SpatialPointsDataFrame(quakes[, c(2:1)], quakes)
summary(Sobj_SpatialGrid(qk)$SG)
t2 <- Sobj_SpatialGrid(qk, n=10000)$SG
summary(t2)
prod(slot(slot(t2, "grid"), "cells.dim"))
proj4string(qk) <- CRS("+proj=longlat")
tf <- tempfile()
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")
plot(qk, xlim=SGqk$xlim, ylim=SGqk$ylim, setParUsrBB=TRUE)
dev.off()
kmlOverlay(SGqk, paste(tf, ".kml", sep=""), paste(tf, ".png", sep=""))
## Not run:
qk0 <- quakes
qk0$long <- ifelse(qk0$long <= 180, qk0$long, qk0$long-360)
qk0a <- SpatialPointsDataFrame(qk0[, c(2:1)], qk0)
proj4string(qk0a) <- CRS("+proj=longlat")
writeOGR(qk0a, paste(tf, "v.kml", sep=""), "Quakes", "KML")
system(paste("googleearth ", tf, ".kml", sep=""))
## End(Not run)
options(example.ask=opt_exask)
```

gpcholes

Hisaji Ono's lake/hole problem

Description

How to plot polygons with holes - holes are encoded by coordinates going anticlockwise, and overplotting is avoided by re-ordering the order in which polygons are plotted.

Usage

```
data(gpcholes)
```

Format

The format is class "polylist".

Details

"Date: Tue, 11 May 2004 12:54:20 +0900 From: Hisaji ONO To: r-help

I've tried to create a polygon with one hole by gpplib using following example script.

```
holepoly <- read.polyfile(system.file("poly-ex/hole-poly.txt", package="gpplib"), nohole = FALSE)
area.poly(holepoly) plot(holepoly, poly.args=list(col="red", border="blue"))
```

And I noticed plot function couldn't draw polygons with holes correctly.

Does anyone know how to solve this situation?"

(hole1pl has reversed the y component of polygon 1, to make its ring direction clockwise, hole2pl reverses the order of the two polygons in holepoly1@pts)

Source

Data file included in "gpplib" package.

Examples

```
data(gpcholes)
plot(hole2pl, col="red", pbg="white", border="blue", forcefill=FALSE)
plot(hole1pl, col="red", pbg="white", border="blue", forcefill=FALSE)
```

`gzAzimuth`*Find azimuth for geographical coordinates*

Description

The function finds azimuth values for geographical coordinates given as decimal degrees from the `from` coordinates to the `to` coordinate. In function `trackAzimuth`, the azimuth values are found between successive rows of the input coordinate matrix.

Usage

```
gzAzimuth(from, to, type = "snyder_sphere")
trackAzimuth(track, type="snyder_sphere")
```

Arguments

<code>from</code>	a two column matrix of geographical coordinates given as decimal degrees (longitude first)
<code>track</code>	a two column matrix of geographical coordinates given as decimal degrees (longitude first)
<code>to</code>	a one row, two column matrix or two element vector of geographical coordinates given as decimal degrees (longitude first)
<code>type</code>	default is "snyder_sphere", otherwise "abdali"; the results should be identical with slightly less trigonometry in "abdali"

Details

The azimuth is calculated on the sphere, using the formulae given by Snyder (1987, p. 30) and Abdali (1997, p. 17). The examples use data taken from Abdali (p. 17–18). There is a very interesting discussion of the centrality of azimuth-finding in the development of mathematics and mathematical geography in Abdali's paper. Among others, al-Khwarizmi was an important contributor. As Abdali puts it, "This is a veritable who's who of medieval science" (p. 3).

Value

values in decimal degrees - zero is North - of the azimuth from the `from` coordinates to the `to` coordinate.

Author(s)

Roger Bivand, with contributions by Sebastian Luque

References

Snyder JP (1987) Map projections - a working manual, USGS Professional Paper 1395; Abdali SK (1997) "The Correct Qibla", <http://patriot.net/users/abdali/ftp/qibla.pdf>

Examples

```

name <- c("Mecca", "Anchorage", "Washington")
long <- c(39.823333, -149.883333, -77.0166667)
lat <- c(21.423333, 61.2166667, 38.9)
x <- cbind(long, lat)
rownames(x) <- name
crib <- c(-9.098363, 56.575960)
r1 <- gZimuth(x[2:3,], x[1,])
r1
all.equal(r1, crib)
r2 <- gZimuth(x[2:3,], x[1,], type="abdali")
r2
all.equal(r2, crib)
trackAzimuth(x)

```

kmlLine

Create and write a KML file on the basis of a given Lines object

Description

The function is used to create and write a KML file on the basis of a given Lines object (a list of Line objects) for the usage in Google Earth resp. Google Maps.

Usage

```

kmlLine(obj=NULL, kmlfile=NULL,
        name="R Line", description="", col=NULL, visibility=1, lwd=1,
        kmlname="", kmldescription="")
x <- kmlLine(obj=NULL,
            name="R Line", description="", col=NULL, visibility=1, lwd=1,
            kmlname="", kmldescription="")
y <- kmlLine(kmlname="", kmldescription="")

```

Arguments

obj	a Lines or SpatialLinesDataFrame object
kmlfile	if not NULL the name as character string of the kml file to be written
name	the name of the KML line
description	the description of the KML line (HTML tags allowed)
col	the stroke color (see also Color Specification) of the KML line
visibility	if set to 1 or TRUE specifies that the KML line should be visible after loading
lwd	the stroke width for the KML line
kmlname	the name of the KML layer
kmldescription	the description of the KML layer (HTML tags allowed)

Details

The function is used to convert a given `Lines` object (a list of `Line` objects) or the first `Lines` object listed in a passed `SpatialLinesDataFrame` object into KML line(s). If `kmlfile` is not `NULL` the result will be written into that file. If `kmlfile` is `NULL` the generated KML lines will be returned (see also `value`).

For a passed `Lines` object the function generates a `<Style>` tag whereby its `id` attribute is set to the passed object's `ID`.

Note that the geometries should be in geographical coordinates with datum WGS84.

The resulting KML line will be embedded in `<Placemark><MultiGeometry><LineString>`.

Value

`x` is a list with the elements `style` and `content` containing the generated lines of the KML file as character vectors if `kmlfile` is `NULL`.

`y` is a list with the elements `header` and `footer` representing the KML file's header resp. footer if `obj` is `NULL`.

Color Specification

The following color specifications are allowed: `'red'`, `2`, or as hex code `'#RRGGBB'` resp. `'#RRGGBBAA'` for passing the alpha value.

Author(s)

Hans-J. Bibiko

See Also

[kmlOverlay](#), [kmlPolygon](#), [Line](#)

Examples

```
xx <- readShapeSpatial(system.file("shapes/fylk-val-11.shp",
  package="maptools")[1], proj4string=CRS("+proj=longlat"))
out <- sapply(slot(xx, "lines"), function(x) { kmlLine(x,
  name=slot(x, "ID"), col="blue", lwd=1.5,
  description=paste("river:", slot(x, "ID"))) })
tf <- tempfile()
kmlFile <- file(tf, "w")
tf
cat(kmlLine(kmlname="R Test", kmldescription="<i>Hello</i>")$header,
  file=kmlFile, sep="\n")
cat(unlist(out["style",]), file=kmlFile, sep="\n")
cat(unlist(out["content",]), file=kmlFile, sep="\n")
cat(kmlLine()$footer, file=kmlFile, sep="\n")
close(kmlFile)
```

`kmlOverlay`*Create and write KML file for PNG image overlay*

Description

The function is used to create and write a KML file for a PNG image overlay for Google Earth.

Usage

```
kmlOverlay(obj, kmlfile = NULL, imagefile = NULL, name = "R image")
```

Arguments

<code>obj</code>	a <code>GE_SG</code> object from <code>GE_SpatialGrid</code>
<code>kmlfile</code>	if not <code>NULL</code> the name of the kml file to be written
<code>imagefile</code>	the name of the PNG file containing the image - this should be either relative (same directory as kml file) or absolute (fully qualified)
<code>name</code>	the name used to describe the image overlay in GE

Details

The function is used together with `GE_SpatialGrid` to wrap around the opening of a PNG graphics device, plotting code, and the closing of the device. The computed values take account of the adjustment of the actual data bounding box to an integer number of rows and columns in the image file.

The approach may be used as an alternative to writing PNG files from `SpatialGrid` and `SpatialPixel` objects in `rgdal` using `writeGDAL`, and to writing KML files using `writeOGR` for vector data objects. The output PNG files are likely to be very much smaller than large vector data KML files, and hinder the retrieval of exact positional information.

Note that the geometries should be in geographical coordinates with datum WGS84.

Value

`x` is a character vector containing the generated lines of the kml file

Author(s)

Duncan Goulicher, David Forrest and Roger Bivand

See Also

[GE_SpatialGrid](#)

Examples

```

opt_exask <- options(example.ask=FALSE)
qk <- SpatialPointsDataFrame(quakes[, c(2:1)], quakes)
proj4string(qk) <- CRS("+proj=longlat")
tf <- tempfile()
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")
plot(qk, xlim=SGqk$xlim, ylim=SGqk$ylim, setParUsrBB=TRUE)
dev.off()
kmlOverlay(SGqk, paste(tf, ".kml", sep=""), paste(tf, ".png", sep=""))
## Not run:
library(rgdal)
qk0 <- quakes
qk0$long <- ifelse(qk0$long <= 180, qk0$long, qk0$long-360)
qk0a <- SpatialPointsDataFrame(qk0[, c(2:1)], qk0)
proj4string(qk0a) <- CRS("+proj=longlat")
writeOGR(qk0a, paste(tf, "v.kml", sep=""), "Quakes", "KML")
system(paste("googleearth ", tf, ".kml", sep=""))
## End(Not run)
options(example.ask=opt_exask)

```

kmlPolygon

Create and write a KML file on the basis of a given Polygons object

Description

The function is used to create and write a KML file on the basis of a given Polygons object (a list of Polygon objects) for the usage in Google Earth resp. Google Maps.

Usage

```

kmlPolygon(obj=NULL, kmlfile=NULL,
           name="R Polygon", description="", col=NULL, visibility=1, lwd=1, border=1,
           kmlname="", kmldescription="")
x <- kmlPolygon(obj=NULL,
               name="R Polygon", description="", col=NULL, visibility=1, lwd=1, border=1,
               kmlname="", kmldescription="")
y <- kmlPolygon(kmlname="", kmldescription="")

```

Arguments

obj	a Polygons or SpatialPolygonsDataFrame object
kmlfile	if not NULL the name as character string of the kml file to be written
name	the name of the KML polygon
description	the description of the KML polygon (HTML tags allowed)

<code>col</code>	the fill color (see also Color Specification) of the KML polygon
<code>visibility</code>	if set to 1 or TRUE specifies that the KML polygon should be visible after loading
<code>lwd</code>	the stroke width for the KML polygon
<code>border</code>	the stroke color (see also Color Specification) for the KML polygon
<code>kmlname</code>	the name of the KML layer
<code>kmldescription</code>	the description of the KML layer (HTML tags allowed)

Details

The function is used to convert a given `Polygons` object (a list of `Polygon` objects) or the first `Polygons` object listed in a passed `SpatialPolygonsDataFrame` object into KML polygon. If `kmlfile` is not `NULL` the result will be written into that file. If `kmlfile` is `NULL` the generated KML lines will be returned (see also `value`).

The conversion can also handle polygons which are marked as holes inside of the `Polygons` object if these holes are listed right after that polygon in which these holes appear. That implies that a given plot order set in the `Polygons` object will **not** be considered.

For a passed `Polygons` object the function generates a `<Style>` tag whereby its `id` attribute is set to the passed object's ID.

Note that the geometries should be in geographical coordinates with datum WGS84.

The resulting KML polygon will be embedded in `<Placemark><MultiGeometry><Polygon>`.

Value

`x` is a list with the elements `style` and `content` containing the generated lines of the KML file as character vectors if `kmlfile` is `NULL`.

`y` is a list with the elements `header` and `footer` representing the KML file's header resp. footer if `obj` is `NULL` (see second example).

Color Specification

The following color specifications are allowed: `'red'`, 2, or as hex code `'#RRGGBB'` resp. `'#RRGGBBAA'` for passing the alpha value.

Author(s)

Hans-J. Bibiko

See Also

[kmlOverlay](#), [kmlLine](#), [SpatialPolygons](#)

Examples

```

data(wrld_simpl)
## creates a KML file containing the polygons of South Africa (plus hole)
sw <- slot(wrld_simpl[wrld_simpl$NAME=="South Africa",], "polygons")[[1]]
tf <- tempfile()
kmlPolygon(sw, kmlfile=tf, name="South Africa", col="#df000aa", lwd=5,
  border=4, kmlname="R Test",
  kmldescription="This is <b>only</b> a <a href='http://www.r-project.org'>R</a> test.")
tf

## creates a KML file containing the polygons of South Africa, Switzerland, and Canada
sw <- wrld_simpl[wrld_simpl$NAME %in% c("South Africa", "Switzerland", "Canada"),]
out <- sapply(slot(sw, "polygons"), function(x) { kmlPolygon(x,
  name=as(sw, "data.frame")[slot(x, "ID"), "NAME"],
  col="red", lwd=1.5, border='black',
  description=paste("ISO3:", slot(x, "ID"))) })
tf <- tempfile()
kmlFile <- file(tf, "w")
tf
cat(kmlPolygon(kmlname="R Test", kmldescription="<i>Hello</i>")$header,
  file=kmlFile, sep="\n")
cat(unlist(out["style",]), file=kmlFile, sep="\n")
cat(unlist(out["content",]), file=kmlFile, sep="\n")
cat(kmlPolygon()$footer, file=kmlFile, sep="\n")
close(kmlFile)

```

Description

Map2poly() is a function to make imported GIS vector polygons into a "polylist" object from a "Map" object. Map2bbs() retrieves polygon bounding boxes; analogous functions for the "shapefiles" package are shape2poly() and shape2bbs() (thanks to Stéphane Dray for his contribution); convert.pl() serves to convert the deprecated "multipart" "polylist" format to an NA-separated format now used by plot() for polylist objects and poly2nb(). *2lines() and *2points() do the same as *2poly() for shapefiles of types 3=lines and 1=points; polygons are type=5.

Usage

```

Map2poly1(Map, region.id = NULL, raw=TRUE)
Map2poly(Map, region.id = NULL, quiet=TRUE)
Map2lines(Map)
Map2points(Map)
Map2bbs(Map)
MapShapeIds(Map)
shape2poly(shape, region.id = NULL)
shape2lines(shape)

```

```

shape2points(shape)
shape2bbs(shape)
convert.pl(pl)

```

Arguments

Map	object of class "Map", imported from a GIS polygon vector file
shape	list returned by <code>read.shapefile()</code> , imported from a GIS polygon vector file using the package "shapefiles"
region.id	character vector of region ids to be added to the neighbours list as attribute <code>region.id</code>
raw	default TRUE <i>until next release</i> , if TRUE, do not run sanity check for ring directions implying holes in no surrounding polygon
quiet	if TRUE, suppress reports of ring direction changes
pl	list of old-style multipart polygons of class <code>polylist</code>

Details

From release 0.4-12, `Map2poly` replaces the plotting order heuristic with plotting only in strict decreasing order of top-level (multi)polygon area. In multipolygons, the components are plotted in decreasing area order. The multipolygons themselves are assigned their area sum for plotting order. Ring directions are all set to clockwise - very many shapefiles have been observed with quite unrealistic ring orders, so respecting input ring orders seems worse than imposing uniformity. The previous plot order assignment mechanism is retained using `Map2poly1`. From release 0.4-1, `Map2poly` tries to determine a plotting order for multiple parts of a single polygon object, and for lists of polygon objects to avoid overplotting. It also tries to respect the ESRI-stipulated rule that lakes are anti-clockwise and islands are clockwise: <http://shapelib.maptools.org/dl/shapefile.pdf>: "The order of vertices or orientation for a ring indicates which side of the ring is the interior of the polygon. The neighborhood to the right of an observer walking along the ring in vertex order is the neighborhood inside the polygon. Vertices of rings defining holes in polygons are in a counterclockwise direction. Vertices for a single, ringed polygon are, therefore, always in clockwise order" (p.8). If `raw=FALSE`, an additional attempt is made to enforce the final stipulation, where the polygon's bounding box does not fall wholly inside the bounding box of another polygon. This may generate warnings indicating when this has taken place, indicating an unexpected ring direction in the imported shapefile.

Value

`Map2poly`, `shape2poly`, `convert.pl` return `polylist` objects, lists of polygon boundary coordinates (divided by NA where the polygon object is represented by more than one polygon) with many attributes; `Map2lines`, `shape2lines` return `lineslist` objects; `Map2points`, `shape2points` return $(n \times 2)$ matrices; and `Map2bbs`, `shape2bbs` return bounding box matrixes, $c(x1, y1, x2, y2)$ with coordinates for the lower left corner and upper right corner.

Author(s)

Roger Bivand and Stéphane Dray, (Roger.Bivand@nhh.no)

See Also

[read.shape, shapefiles](#)

Examples

```
try2 <- read.shape(system.file("shapes/columbus.shp", package="mapproj")[1])
mappolys <- Map2poly(try2, as.character(try2$att.data$NEIGNO), quiet=FALSE)
plot(mappolys)
title(main="Polygons for Columbus OH from mapproj package")
mappolys <- Map2poly1(try2, as.character(try2$att.data$NEIGNO), raw=FALSE)
plot(mappolys)
title(main="Polygons for Columbus OH from mapproj package")
try3 <- read.shape(system.file("shapes/baltim.shp", package="mapproj")[1])
baltpts <- Map2points(try3)
xylims <- attr(baltpts, "maplim")
plot(xylims$x, xylims$y, asp=1, type='n', xlab="", ylab="")
points(baltpts)
title(main="Baltimore points from mapproj package")
try4 <- read.shape(system.file("shapes/fylk-val.shp", package="mapproj")[1])
fylk.val <- Map2lines(try4)
xylims <- attr(fylk.val, "maplim")
plot(xylims$x, xylims$y, asp=1, type='n', xlab="", ylab="")
for (i in 1:length(fylk.val)) lines(fylk.val[[i]])
title(main="Norwegian river centrelines from mapproj package")
```

map2SpatialPolygons

Convert map objects to sp classes

Description

These functions may be used to convert map objects returned by the map function in the maps package to suitable objects defined in the sp package. In the examples below, arguments are shown for retrieving first polygons by name, then lines by window.

Usage

```
map2SpatialPolygons(map, IDs, proj4string = CRS(as.character(NA)))
map2SpatialLines(map, IDs=NULL, proj4string = CRS(as.character(NA)))
pruneMap(map, xlim=NULL, ylim=NULL)
```

Arguments

map	a map object defined in the maps package and returned by the map function
IDs	Unique character ID values for each output Polygons object; the input IDs can be an integer or character vector with duplicates, where the duplicates will be combined as a single output Polygons object
proj4string	Object of class "CRS"; holding a valid proj4 string

`xlim, ylim` limits for pruning a map object - should only be used for lines, because polygons will not be closed

Value

`map2SpatialPolygons` returns a `SpatialPolygons` object and `map2SpatialLines` returns a `SpatialLines` object (objects defined in the `sp` package); `pruneMap` returns a modified map object defined in the `maps` package

Note

As the examples show, retrieval by name should be checked to see whether a window is not also needed: the "norway" polygons include "Norway:Bouvet Island", which is in the South Atlantic. Here, the `IDs` argument is set uniformly to "Norway" for all the component polygons, so that the output object contains a single `Polygons` object with multiple component `Polygon` objects. When retrieving by window, pruning may be needed on lines which are included because they begin within the window; `interior=FALSE` is used to remove country boundaries in this case.

Author(s)

Roger Bivand

See Also

[map](#)

Examples

```
library(maps)
nor_coast_poly <- map("world", "norway", fill=TRUE, col="transparent",
  plot=FALSE)
range(nor_coast_poly$x, na.rm=TRUE)
range(nor_coast_poly$y, na.rm=TRUE)
nor_coast_poly <- map("world", "norway", fill=TRUE, col="transparent",
  plot=FALSE, ylim=c(58,72))
nor_coast_poly$names
IDs <- sapply(strsplit(nor_coast_poly$names, ":"), function(x) x[1])
nor_coast_poly_sp <- map2SpatialPolygons(nor_coast_poly, IDs=IDs,
  proj4string=CRS("+proj=longlat +datum=wgs84"))
sapply(slot(nor_coast_poly_sp, "polygons"),
  function(x) length(slot(x, "Polygons")))
plot(nor_coast_poly_sp, col="grey", axes=TRUE)
nor_coast_lines <- map("world", interior=FALSE, plot=FALSE, xlim=c(4,32),
  ylim=c(58,72))
plot(nor_coast_lines, type="l")
nor_coast_lines <- pruneMap(nor_coast_lines, xlim=c(4,32), ylim=c(58,72))
lines(nor_coast_lines, col="red")
nor_coast_lines_sp <- map2SpatialLines(nor_coast_lines,
  proj4string=CRS("+proj=longlat +datum=wgs84"))
plot(nor_coast_poly_sp, col="grey", axes=TRUE)
plot(nor_coast_lines_sp, col="blue", add=TRUE)
```

`maptools`*Report version information and changes*

Description

Access version information and changes

Usage

```
maptools(changes = FALSE)
```

Arguments

`changes` If TRUE, reports changes, most recent last

Value

Invisibly returns the change list, with members version and changes

Author(s)

Roger Bivand

Examples

```
maptools(changes=TRUE)
```

`nowrapRecenter`*Break polygons at meridian for recentering*

Description

When recentering a world map, say to change an "Atlantic" view with longitude range -180 to 180, to a "Pacific" view, with longitude range 0 to 360, polygons crossed by the new offset, here 0/360, need to be clipped into left and right sub.polygons to avoid horizontal scratches across the map. The `nowrapSpatialPolygons` function performs this operation using polygon intersection, and `nowrapRecenter` recenters the output `SpatialPolygons` object.

Usage

```
nowrapRecenter(obj, offset = 0, eps = rep(.Machine$double.eps, 2))  
nowrapSpatialPolygons(obj, offset = 0, eps=rep(.Machine$double.eps, 2))
```

Arguments

obj A SpatialPolygons object
offset offset from the Greenwich meridian
eps vector of two (left and right) fuzz factors to retract the ring from the offset

Value

A SpatialPolygons object

Author(s)

Roger Bivand

See Also

[recenter-methods](#), [nowrapSpatialLines](#)

Examples

```
library(gpclib)
## Not run:
library(maps)
world <- map("world", fill=TRUE, col="transparent", plot=FALSE)
worldSpP <- map2SpatialPolygons(world, world$names, CRS("+proj=longlat"))
worldSpPr <- recenter(worldSpP)
plot(worldSpPr)
title("Pacific view without polygon splitting")
worldSpPnr <- nowrapRecenter(worldSpP)
plot(worldSpPnr)
title("Pacific view with polygon splitting")
## End(Not run)
crds <- matrix(c(-1, 1, 1, -1, 50, 50, 52, 52), ncol=2)
rcrds <- rbind(crds, crds[1,])
SR <- SpatialPolygons(list(Polygons(list(Polygon(rcrds)), ID="r1"),
  proj4string=CRS("+proj=longlat")))
bbox(SR)
SRr <- recenter(SR)
bbox(SRr)
SRnr <- nowrapRecenter(SR)
bbox(SRnr)
```

pal2SpatialPolygons

Making SpatialPolygons objects from RArcInfo input

Description

This function is used in making SpatialPolygons objects from RArcInfo input.

Usage

```
pal2SpatialPolygons(arc, pal, IDs, dropPoly1=TRUE,
  proj4string=CRS(as.character(NA)))
```

Arguments

IDs	Unique character ID values for each output Polygons object; the input IDs can be an integer or character vector with duplicates, where the duplicates will be combined as a single output Polygons object
proj4string	Object of class "CRS"; holding a valid proj4 string
arc	Object returned by <code>get.arcdata</code>
pal	Object returned by <code>get.paldata</code>
dropPoly1	Should the first polygon in the AVC or e00 data be dropped; the first polygon is typically the compound boundary of the whole dataset, and can be detected by looking at the relative lengths of the list components in the second component of pal, which are the numbers of arcs making up the boundary of each polygon

Value

The functions return a SpatialPolygons object

Author(s)

Roger Bivand

Examples

```
nc1 <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1], ID="FIPS")
plot(nc1)
text(coordinates(nc1), labels=sapply(slot(nc1, "polygons"),
  function(i) slot(i, "ID")), cex=0.6)
library(maps)
ncmap <- map("county", "north carolina", fill=TRUE, col="transparent",
  plot=FALSE)
IDs <- sapply(strsplit(ncmap$names, "[,:]"), function(x) x[2])
nc2 <- map2SpatialPolygons(ncmap, IDs)
plot(nc2)
text(coordinates(nc2), labels=sapply(slot(nc2, "polygons"),
  function(i) slot(i, "ID")), cex=0.6)
library(RArcInfo)
td <- tempdir()
tmpcover <- paste(td, "nc", sep="/")
if (!file.exists(tmpcover)) e00toavc(system.file("share/co37_d90.e00",
  package="maptools")[1], tmpcover)
arc <- get.arcdata(td, "nc")
pal <- get.paldata(td, "nc")
pat <- get.tabledata(paste(td, "info", sep="/"), "NC.PAT")
sapply(pat[[2]], function(x) length(x[[1]]))
IDs <- paste(pat$ST[-1], pat$CO[-1], sep="")
```

```
nc3 <- pal2SpatialPolygons(arc, pal, IDs=IDs)
plot(nc3)
text(coordinates(nc3), labels=sapply(slot(nc3, "polygons"),
  function(i) slot(i, "ID")), cex=0.6)
```

plot.Map

Plot a Map object (deprecated)

Description

This function is deprecated. It is difficult to maintain and there are several alternatives, either by converting Map objects to sp class objects or polylist etc. objects. (The function plots a map object directly from the imported shapefile data).

Usage

```
## S3 method for class 'Map':
plot(x, recs, auxvar = NULL, add = FALSE, fg = "gray", ol = "black",
  prbg = NULL, glyph = 16, color='red', type = "q", nclass = 5, ...)
```

Arguments

x	a Map object returned by read.shape()
recs	a vector with the shapes to plot, if missing defaults to all shapes; if fg is not equal to the length of recs, all fg are set to fg[1]
auxvar	if the Map has polygon shapes, a variable from which to derive polygon fill colours using the computed class intervals, must be same length as number of shapes
add	default FALSE, if TRUE add to existing plot
fg	foreground colour, can be used to pass through point and polygon colours, permitting the built-in class interval calculation to be avoided; if fg is not equal to the length of recs, all fg are set to fg[1]
ol	line/boundary colour; boundaries may be removed by setting to NA or "transparent"
prbg	if not NULL, draw a rectangle around the plot in this colour
glyph	points plotted as this pch code
color	base colour for color.ramp(); default "red"
type	default "q" for quantile classes for color.ramp(), can also be "e" for equal intervals
nclass	number of classes for class intervals
...	passed through to plotting functions

Value

If the `auxvar` variable is not used, the function returns `NULL`, otherwise it returns the list constructed by `maptools:::color.ramp()` with components:

<code>ramp</code>	vector of colours
<code>col.class</code>	vector of classes
<code>breaks</code>	class intervals given as argument to <code>cut</code>

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand (Roger.Bivand@nhh.no)

See Also

[read.shape](#), [readShapePoly](#), [readShapeLines](#), [readShapePoints](#), [getinfo.shape](#)

Examples

```
## Not run:
x <- read.shape(system.file("shapes/sids.shp", package="maptools")[1])
plot(x)
nParts <- sapply(x$Shapes, function(x) attr(x, "nPart"))
table(nParts)
cols <- c("azure", "blue", "orange")
fgs <- cols[nParts]
plot(x, fg=fgs)
res <- plot(x, auxvar=x$att.data$BIR74)
str(res)
res <- plot(x, auxvar=x$att.data$BIR74, type="e")
str(res)
## End(Not run)
```

plot.polylist

Plot polygons

Description

A helper function for plotting polygons in a global bounding box, attempts to handle overplotting internally, holes should be coded by coordinates ordered anti-clockwise. Earlier behaviour requires `forcefill=TRUE`; this is now default, but will cease to be so from next release. Changes caused by some imported shapefiles data leading to erroneously plotting lakes/islands, and R `polygon()` overplotting internal polygons. `leglabs` makes character strings from the same break points. The `plot.polylist()` function may be used as a generic S3 method. NOTE! `plotpolys()` DEPRECATED: use generic `plot()` for `polylist` objects function instead.

Usage

```
## S3 method for class 'polylist':
plot(x, col, border = par("fg"), add = FALSE, xlim=NULL,
     ylim=NULL, xlab = "", ylab = "", xpd = NULL, density = NULL, angle = 45,
     pbg=NULL, forcefill=TRUE, ...)
leglabs(vec, under="under", over="over", between="-")
plotpolys(pl, bb, col = NA, border = par("fg"), add = FALSE, xlim=NULL,
          ylim=NULL, ...)
```

Arguments

pl, x	list of polygons of class <code>polylist</code>
bb	matrix of polygon bounding boxes - columns are LL(x), LL(y), UR(x), UR(y); note that this argument may be omitted if the polygon list object has an "maplim" attribute, or if both <code>xlim</code> and <code>ylim</code> arguments are given
col	colours to use for filling the polygons
border	the color to draw the border
add	add to existing plot
xlim, ylim	numeric of length 2, giving the x and y coordinates ranges
xlab, ylab	x and y axis labels, default no label
xpd	(where) should clipping take place?
density	the density of shading lines, in lines per inch
angle	the slope of shading lines, given as an angle in degrees (counter-clockwise)
pbg	colour to be used for hole and background fill, by default NULL (note that <code>par("bg")</code> may be "transparent")
forcefill	default TRUE <i>until next release</i> - fill anyway, if FALSE: believe ring direction as indication of holes/lakes
...	other arguments passed to plot to set the plot window - not passed to polygon
vec	vector of break values
under	character value for under
over	character value for over
between	character value for between

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

[Map2poly](#), [findInterval](#)

Examples

```

try2 <- read.shape(system.file("shapes/columbus.shp", package="mapprools")[1])
mappolys <- Map2poly(try2, as.character(try2$att.data$NEIGNO))
brks <- round(quantile(try2$att.data$CRIME, probs=seq(0,1,0.2)), digits=2)
colours <- c("salmon1", "salmon2", "red3", "brown", "black")
plot(mappolys, col=colours[findInterval(try2$att.data$CRIME, brks,
  all.inside=TRUE)], forcefill=FALSE)
legend(x=c(5.8, 7.1), y=c(13, 14.5), legend=leglabs(brks),
  fill=colours, bty="n")
invisible(title(main=paste("Columbus OH: residential burglaries and vehicle",
  "thefts per thousand households, 1980", sep="\n")))
try3 <- read.shape(system.file("shapes/sids.shp", package="mapprools")[1])
mappolys <- Map2poly(try3, as.character(try3$att.data$FIPSNO))
cols <- c("pink", "orange", "red")
np <- sapply(mappolys, function(x) attr(x, "nParts"))
plot(mappolys, col=cols[np], forcefill=FALSE)
invisible(title(main="White: one part, orange: two part, red: three part polygons"))
plot(mappolys, density=c(10, 20, 30)[np], angle=c(-45, 0, 45)[np],
  axes=FALSE, forcefill=FALSE)

```

pointLabel

*Label placement for points to avoid overlaps***Description**

Use optimization routines to find good locations for point labels without overlaps.

Usage

```

pointLabel(x, y = NULL, labels = seq(along = x), cex = 1,
  method = c("SANN", "GA"),
  allowSmallOverlap = FALSE,
  trace = FALSE,
  doPlot = TRUE,
  ...)

```

Arguments

<code>x</code> , <code>y</code>	as with <code>plot.default</code> , these provide the x and y coordinates for the point labels. Any reasonable way of defining the coordinates is acceptable. See the function <code>xy.coords</code> for details.
<code>labels</code>	as with <code>text</code> , 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 <code>as.character</code> .
<code>cex</code>	numeric character expansion factor as with <code>text</code> .
<code>method</code>	the optimization method, either "SANN" for simulated annealing (the default) or "GA" for a genetic algorithm.

<code>allowSmallOverlap</code>	logical; if TRUE, labels are allowed a small overlap. The overlap allowed is 2% of the diagonal distance of the plot area.
<code>trace</code>	logical; if TRUE, status updates are given as the optimization algorithms progress.
<code>doPlot</code>	logical; if TRUE, the labels are plotted on the existing graph with <code>text</code> .
<code>...</code>	arguments passed along to <code>text</code> to specify labeling parameters such as <code>col</code> .

Details

Eight positions are candidates for label placement, either horizontally, vertically, or diagonally offset from the points. The default position for labels is the top right diagonal relative to the point (considered the preferred label position).

With the default settings, simulating annealing solves faster than the genetic algorithm. It is an open question as to which settles into a global optimum the best (both algorithms have parameters that may be tweaked).

The label positioning problem is NP-hard (nondeterministic polynomial-time hard). Placement becomes difficult and slows considerably with large numbers of points. This function places all labels, whether overlaps occur or not. Some placement algorithms remove labels that overlap.

Note that only `cex` is used to calculate string width and height (using `strwidth` and `strheight`), so passing a different font may corrupt the label dimensions. You could get around this by adjusting the font parameters with `par` prior to running this function.

Value

An `xy` list giving the `x` and `y` positions of the label as would be placed by `text(xy, labels)`.

Author(s)

Tom Short, EPRI, (tshort@epri.com)

References

http://en.wikipedia.org/wiki/Automatic_label_placement

<http://illwww.itl.uni-karlsruhe.de/map-labeling/bibliography/>

<http://www.eecs.harvard.edu/~shieber/Projects/Carto/carto.html>

<http://www.szoraster.com/Cartography/PracticalExperience.htm>

The genetic algorithm code was adapted from the python code at

http://meta.wikimedia.org/wiki/Map_generator.

The simulated annealing code follows the algorithm and guidelines in:

Jon Christensen, Joe Marks, and Stuart Shieber. Placing text labels on maps and diagrams. In Paul Heckbert, editor, Graphics Gems IV, pages 497-504. Academic Press, Boston, MA, 1994.

<http://www.eecs.harvard.edu/~shieber/Biblio/Papers/jc.label.pdf>

See Also

`text`, `thigmophobe.labels` in package **plotrix**

Examples

```
n <- 50
x <- rnorm(n)*10
y <- rnorm(n)*10
plot(x, y, col = "red", pch = 20)
pointLabel(x, y, as.character(round(x,5)), offset = 0, cex = .7)

plot(x, y, col = "red", pch = 20)
pointLabel(x, y, expression(over(alpha, beta[123])), offset = 0, cex = .8)
```

ppp-class

Virtual class "ppp"

Description

Virtual S4 class definition for S3 classes in the spatstat package to allow S4-style coercion to these classes

Objects from the Class

A virtual Class: No objects may be created from it.

Author(s)

Edzer J. Pebesma

read.shape

Read shapefile into Map object

Description

Read shapefile into Map object; the file should be given including its ".shp" extension, and the function will reconstruct the names of the database (dbf) file and the index (shx) file from these.

Usage

```
read.shape(filename, dbf.data = TRUE, verbose=TRUE, repair=FALSE)
getinfo.shape(filename)
## S3 method for class 'shapehead':
print(x, ...)
```

Arguments

<code>filen</code>	name of file with *.shp extension
<code>dbf.data</code>	read DBF data together with shapes, default TRUE
<code>verbose</code>	default TRUE — report type of shapefile and number of shapes
<code>repair</code>	default FALSE: some shapefiles provided by Geolytics Inc. have values of object sizes stored in the *.shx index file that are eight bytes too large, leading the function to try to read past the end of file. If <code>repair=TRUE</code> , an attempt is made to repair the internal values, permitting such files to be read.
<code>x</code>	a shapehead list as returned by <code>getinfo.shape</code>
<code>...</code>	other arguments passed to print

Details

The function calls code from shapelib to read shapefiles, a file format used by ESRI GIS software among others

Value

`read.shape()` returns either a list of shapes of class `ShapeList`, or if `dbf.data = TRUE` a Map object with:

<code>Shapes</code>	a list of shapes of class <code>ShapeList</code> ; both the individual shapes and the list have attributes
<code>att.data</code>	a data frame of data from the associated DBF file; note that the field names are adjusted to use in R using <code>make.names()</code> , and so will permit the underscore character from R release 1.9.0.

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand <Roger.Bivand@nhh.no>; shapelib by Frank Warmerdam

References

<http://shapelib.maptools.org/>

See Also

`plot.Map`, `read.dbf`

Examples

```
x <- read.shape(system.file("shapes/sids.shp", package="maptools")[1])
length(x$Shapes)
unlist(lapply(x$att.data, class))
str(getinfo.shape(system.file("shapes/fylk-val.shp", package="maptools")[1]))
```

readAsciiGrid *read/write to/from (ESRI) asciigrid format*

Description

read/write to/from ESRI asciigrid format; a fuzz factor has been added to writeAsciiGrid to force cell resolution to equality if the difference is less than the square root of machine precision

Usage

```
readAsciiGrid(fname, as.image = FALSE, plot.image = FALSE,
  colname = basename(fname), proj4string = CRS(as.character(NA)),
  dec=options()$OutDec)
writeAsciiGrid(x, fname, attr = 1, na.value = -9999, dec=options()$OutDec, ...)
```

Arguments

fname	file name
as.image	logical; if TRUE, a list is returned, ready to be shown with the image command; if FALSE an object of class SpatialGridDataFrame-class is returned
plot.image	logical; if TRUE, an image of the map is plotted
colname	alternative name for data column if not file basename
proj4string	A CRS object setting the projection arguments of the Spatial Grid returned
dec	decimal point character. This should be a character string containing just one single-byte character — see note below.
x	object of class SpatialGridDataFrame
attr	attribute column; if missing, the first column is taken; a name or a column number may be given
na.value	numeric; value given to missing valued cells in the resulting map
...	arguments passed to write.table , which is used to write the numeric data

Value

readAsciiGrid returns the grid map read; either as an object of class [SpatialGridDataFrame-class](#) or, if as.image is TRUE, as list with components x, y and z.

Note

In ArcGIS 8, it was not in general necessary to set the dec argument; it is not necessary in a mixed environment with ArcView 3.2 (R writes and ArcView reads "."), but inter-operation with ArcGIS 9 requires care because the defaults used by ArcGIS seem to be misleading, and it may be necessary to override what appear to be platform defaults by setting the argument.

Author(s)

Edzer J. Pebesma, e.pebesma@geo.uu.nl

See Also

[image](#), [image](#)

Examples

```
x <- readAsciiGrid(system.file("grids/test.ag", package="mapproj")[1])
summary(x)
image(x)
xp <- as(x, "SpatialPixelsDataFrame")
abline(h=332000, lwd=3)
xpS <- xp[coordinates(xp)[,2] < 332000,]
summary(xpS)
xS <- as(xpS, "SpatialGridDataFrame")
summary(xS)
tmpfl <- paste(tempdir(), "testS.ag", sep="/")
writeAsciiGrid(xS, tmpfl)
axS <- readAsciiGrid(tmpfl)
opar <- par(mfrow=c(1,2))
image(xS, main="before export")
image(axS, main="after import")
par(opar)
unlink(tmpfl)
```

readGPS

GPSbabel read interface

Description

The function reads a data frame from an attached GPS using the external program `gpsbabel`. The columns of the data frame need to be identified by hand because different GPS order NMEA data in different ways, and the columns should be converted to the correct classes by hand. Once the specifics of a particular GPS are identified, and ways of cleaning erroneous locations are found, the conversion of the output data frame into a usable one may be automated.

Usage

```
readGPS(i = "garmin", f = "usb:", type="w", invisible=TRUE, ...)
```

Arguments

<code>i</code>	INTYPE: a supported file type, default "garmin"
<code>f</code>	INFILE: the appropriate device interface, default "usb:", on Windows for serial interfaces commonly "com4:" or similar
<code>type</code>	"w" waypoints, or "t" track, or others provided in <code>gpsbabel</code>

invisible Under Windows, do not open an extra window
 ... arguments passed through to read.table

Details

The function just wraps: `gpsbabel -i INTYPE -f INFILE -o tabsep -F -` in `system()`, and reads the returned character vector of lines into a data frame. On some systems, INFILE may not be readable by ordinary users without extra configuration. The `gpsbabel` program must be present and on the user's PATH for the function to work. Typically, for a given GPS, the user will have to experiment first to find a set of data-cleaning tricks that work, but from then on they should be repeatable.

Value

A data frame of waypoint values

Author(s)

Patrick Giraudoux and Roger Bivand

References

<http://www.gpsbabel.org>

Examples

```
## Not run:
b1 <- readGPS(f="usb:")
str(b1)
b2 <- b1[1:172,]
wp0 <- b2[,c(2,3,4,8,9,19)]
str(wp0)
wp0$long <- wp0$V9
wp0$lat <- as.numeric(as.character(wp0$V8))
wp0$id <- as.character(wp0$V2)
wp0$alt <- as.numeric(substring(as.character(wp0$V19), 1,
  (nchar(as.character(wp0$V19))-1)))
wp0$time <- as.POSIXct(strptime(paste(as.character(wp0$V3),
  as.character(wp0$V4)), format="%d-%b-%y %H:%M:%S"))
str(wp0)
wp1 <- wp0[,-(1:6)]
str(wp1)
summary(wp1)
## End(Not run)
```

readShapeLines *Read arc shape files into SpatialLinesDataFrame objects*

Description

The `readShapeLines` function reads data from an arc/line shapefile into a `SpatialLinesDataFrame` object; the shapefile may be of type polygon, but for just plotting for example coastlines, a `SpatialLines` object is sufficient. The `writeLinesShape` function writes data from a `SpatialLinesDataFrame` object to a shapefile.

Usage

```
readShapeLines(fn, proj4string=CRS(as.character(NA)), verbose=FALSE,
  repair=FALSE)
writeLinesShape(x, fn, factor2char = TRUE, max_nchar=254)
```

Arguments

<code>fn</code>	shapefile layer name, when writing omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
<code>proj4string</code>	Object of class CRS; holding a valid proj4 string
<code>verbose</code>	default TRUE - report type of shapefile and number of shapes
<code>repair</code>	default FALSE: some shapefiles provided by Geolytics Inc. have values of object sizes stored in the *.shx index file that are eight bytes too large, leading the function to try to read past the end of file. If <code>repair=TRUE</code> , an attempt is made to repair the internal values, permitting such files to be read.
<code>x</code>	a <code>SpatialLinesDataFrame</code> object
<code>factor2char</code>	logical, default TRUE, convert factor columns to character
<code>max_nchar</code>	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

The `shpID` values of the shapefile will be used as `Lines ID` values; when writing shapefiles, the object data slot rownames are added to the DBF file as column `SL_ID`.

Value

a `SpatialLinesDataFrame` object

Author(s)

Roger Bivand

Examples

```

xx <- readShapeLines(system.file("shapes/fylk-val.shp", package="mapprools")[1],
  proj4string=CRS("+proj=utm +zone=33 +datum=WGS84"))
plot(xx, col="blue")
summary(xx)
xxx <- xx[xx$LENGTH > 30000,]
plot(xxx, col="red", add=TRUE)
tmpfl <- paste(tempdir(), "xxline", sep="/")
writeLinesShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
axx <- readShapeLines(tmpfl, proj4string=CRS("+proj=utm +zone=33 +datum=WGS84"))
plot(xxx, col="black", lwd=4)
plot(axx, col="yellow", lwd=1, add=TRUE)
unlink(paste(tmpfl, ".*", sep=""))
xx <- readShapeLines(system.file("shapes/sids.shp", package="mapprools")[1],
  proj4string=CRS("+proj=longlat +datum=NAD27"))
plot(xx, col="blue")

```

readShapePoints *Read points shape files into SpatialPointsDataFrame objects*

Description

The `readShapePoints` reads data from a points shapefile into a `SpatialPointsDataFrame` object. The `writePointsShape` function writes data from a `SpatialPointsDataFrame` object to a shapefile. Both reading and writing can be carried out for 2D and 3D point coordinates.

Usage

```

readShapePoints(fn, proj4string = CRS(as.character(NA)), verbose = FALSE,
  repair=FALSE)
writePointsShape(x, fn, factor2char = TRUE, max_nchar=254)

```

Arguments

<code>fn</code>	shapefile layer name, when writing omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
<code>proj4string</code>	Object of class CRS; holding a valid proj4 string
<code>verbose</code>	default TRUE - report type of shapefile and number of shapes
<code>repair</code>	default FALSE: some shapefiles provided by Geolytics Inc. have values of object sizes stored in the *.shx index file that are eight bytes too large, leading the function to try to read past the end of file. If <code>repair=TRUE</code> , an attempt is made to repair the internal values, permitting such files to be read.
<code>x</code>	a <code>SpatialPointsDataFrame</code> object
<code>factor2char</code>	logical, default TRUE, convert factor columns to character
<code>max_nchar</code>	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Value

a SpatialPointsDataFrame object

Author(s)

Roger Bivand

Examples

```
library(maptools)
xx <- readShapePoints(system.file("shapes/baltim.shp", package="maptools")[1])
plot(xx)
summary(xx)
xxx <- xx[xx$PRICE < 40,]
tmpfl <- paste(tempdir(), "xxpts", sep="/")
writePointsShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
axx <- readShapePoints(tmpfl)
plot(axx, col="red", add=TRUE)
unlink(paste(tmpfl, ".*", sep=""))
xx <- readShapePoints(system.file("shapes/pointZ.shp", package="maptools")[1])
dimensions(xx)
plot(xx)
summary(xx)
```

readShapePoly

Read polygon shape files into SpatialPolygonsDataFrame objects

Description

The readShapePoly reads data from a polygon shapefile into a SpatialPolygonsDataFrame object. The writePolyShape function writes data from a SpatialPolygonsDataFrame object to a shapefile.

Usage

```
readShapePoly(fn, IDvar=NULL, proj4string=CRS(as.character(NA)),
  verbose=FALSE, repair=FALSE, force_ring=FALSE, delete_null_obj=FALSE,
  retrieve_ABS_null=FALSE)
writePolyShape(x, fn, factor2char = TRUE, max_nchar=254)
```

Arguments

fn	shapefile layer name, when writing omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
IDvar	a character string: the name of a column in the shapefile DBF containing the ID values of the shapes - the values will be converted to a character vector
proj4string	Object of class CRS; holding a valid proj4 string

verbose	default TRUE - report type of shapefile and number of shapes
repair	default FALSE: some shapefiles provided by Geolytics Inc. have values of object sizes stored in the *.shx index file that are eight bytes too large, leading the function to try to read past the end of file. If repair=TRUE, an attempt is made to repair the internal values, permitting such files to be read.
force_ring	if TRUE, close unclosed input rings
delete_null_obj	if TRUE, null geometries inserted by ABS will be removed together with their data.frame rows
retrieve_ABS_null	default FALSE, if TRUE and delete_null_obj also TRUE, the function will return a data frame containing the data from any null geometries inserted by ABS
x	a SpatialPolygonsDataFrame object
factor2char	logical, default TRUE, convert factor columns to character
max_nchar	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

If no IDvar argument is given, the shpID values of the shapefile will be used as Polygons ID values; when writing shapefiles, the object data slot rownames are added to the DBF file as column SP_ID.

Value

a SpatialPolygonsDataFrame object

Author(s)

Roger Bivand

Examples

```
library(maptools)
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
plot(xx, border="blue", axes=TRUE, las=1)
text(coordinates(xx), labels=sapply(slot(xx, "polygons"),
  function(i) slot(i, "ID")), cex=0.6)
as(xx, "data.frame")[1:5, 1:6]
xxx <- xx[xx$SID74 < 2,]
plot(xxx, border="red", add=TRUE)
tmpfl <- paste(tempdir(), "xxpoly", sep="/")
writePolyShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
axx <- readShapePoly(tmpfl, proj4string=CRS("+proj=longlat +ellps=clrk66"))
plot(axx, border="black", lwd=4)
plot(axx, border="yellow", lwd=1, add=TRUE)
unlink(paste(tmpfl, ".*", sep=""))
```

readShapeSpatial *Read shape files into Spatial*DataFrame objects*

Description

The readShapeSpatial reads data from a shapefile into a Spatial*DataFrame object. The writeSpatialShape function writes data from a Spatial*DataFrame object to a shapefile.

Usage

```
readShapeSpatial(fn, proj4string=CRS(as.character(NA)),
  verbose=FALSE, repair=FALSE, IDvar=NULL, force_ring=FALSE,
  delete_null_obj=FALSE, retrieve_ABS_null=FALSE)
writeSpatialShape(x, fn, factor2char = TRUE, max_nchar=254)
```

Arguments

fn	shapefile layer name, when writing omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
proj4string	Object of class CRS; holding a valid proj4 string
verbose	default TRUE - report type of shapefile and number of shapes
repair	default FALSE: some shapefiles provided by Geolytics Inc. have values of object sizes stored in the *.shx index file that are eight bytes too large, leading the function to try to read past the end of file. If repair=TRUE, an attempt is made to repair the internal values, permitting such files to be read.
IDvar	a character string: the name of a column in the shapefile DBF containing the ID values of the shapes - the values will be converted to a character vector (Polygons only)
force_ring	if TRUE, close unclosed input rings (Polygons only)
delete_null_obj	if TRUE, null geometries inserted by ABS will be removed together with their data.frame rows (Polygons only)
retrieve_ABS_null	default FALSE, if TRUE and delete_null_obj also TRUE, the function will return a data frame containing the data from any null geometries inserted by ABS (Polygons only)
x	a vector data Spatial*DataFrame object
factor2char	logical, default TRUE, convert factor columns to character
max_nchar	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

If no IDvar argument is given, the shpID values of the shapefile will be used as Polygons ID values; when writing shapefiles, the object data slot rownames are added to the DBF file as column SP_ID.

Value

a Spatial*DataFrame object of a class corresponding to the input shapefile

Author(s)

Roger Bivand

Examples

```
library(maptools)
xx <- readShapeSpatial(system.file("shapes/sids.shp", package="maptools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
summary(xx)
xxx <- xx[xx$SID74 < 2,]
tmpfl <- paste(tempdir(), "xxpoly", sep="/")
writeSpatialShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
unlink(paste(tmpfl, ".*", sep=""))
xx <- readShapeSpatial(system.file("shapes/fylk-val.shp",
  package="maptools")[1], proj4string=CRS("+proj=utm +zone=33 +datum=WGS84"))
summary(xx)
xxx <- xx[xx$LENGTH > 30000,]
plot(xxx, col="red", add=TRUE)
tmpfl <- paste(tempdir(), "xxline", sep="/")
writeSpatialShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
unlink(paste(tmpfl, ".*", sep=""))
xx <- readShapeSpatial(system.file("shapes/baltim.shp", package="maptools")[1])
summary(xx)
xxx <- xx[xx$PRICE < 40,]
tmpfl <- paste(tempdir(), "xxpts", sep="/")
writeSpatialShape(xxx, tmpfl)
getinfo.shape(paste(tmpfl, ".shp", sep=""))
unlink(paste(tmpfl, ".*", sep=""))
```

readSplus

Read exported WinBUGS maps

Description

The function permits an exported WinBUGS map to be read into an **sp** package class SpatialPolygons object.

Usage

```
readSplus(file, proj4string = CRS(as.character(NA)))
```

Arguments

`file` name of file
`proj4string` Object of class "CRS"; holding a valid proj4 string

Value

`readSplus` returns a `SpatialPolygons` object

Note

In the example, taken from the GeoBUGS manual, the smaller part of `area1` has a counter-clockwise ring direction in the data, while other rings are clockwise. This implies that it is a hole, and does not get filled. Errant holes may be filled using `checkPolygonsHoles`. The region labels are stored in the ID slots of the `Polygons` objects.

Author(s)

Virgilio Gomez Rubio <Virgilio.Gomez@uclm.es>

References

<http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/geobugs12manual.pdf>

See Also

[map2SpatialPolygons](#)

Examples

```
geobugs <- readSplus(system.file("share/Splus.map", package="mapprotools"))
plot(geobugs, axes=TRUE, col=1:3)
sapply(slot(geobugs, "polygons"), slot, "ID")
pls <- slot(geobugs, "polygons")
sapply(pls, function(i) sapply(slot(i, "Polygons"), slot, "hole"))
pls1 <- lapply(pls, checkPolygonsHoles)
sapply(pls1, function(i) sapply(slot(i, "Polygons"), slot, "hole"))
plot(SpatialPolygons(pls1), axes=TRUE, col=1:3)
```

`Rgshhs`*Read GSHHS data into sp object*

Description

If the data are polygon data, the function will read GSHHS polygons into SpatialPolygons object for a chosen region, using binary shorelines from Global Self-consistent Hierarchical High-resolution Shorelines, release 1.11 of July 3, 2008 (ftp://ftp.soest.hawaii.edu/pwessel/gshhs/gshhs_1.11.zip). If the data are line data, the borders or river lines will be read into a SpatialLines object. The data are provided in integer form as millionths of decimal degrees. Reading of much earlier versions of the GSHHS binary files will fail with an error message. The netCDF GSHHS files distributed with GMT ≥ 4.2 cannot be read as they are in a very different format.

Usage

```
Rgshhs(fn, xlim = NULL, ylim = NULL, level = 4, minarea = 0, shift=FALSE,
       verbose = TRUE, no.clip = FALSE)
```

Arguments

<code>fn</code>	filename or full path to GSHHS 1.11 file to be read
<code>xlim</code>	longitude limits within 0-360 in most cases, negative longitudes are also found east of the Atlantic, but the Americas are recorded as positive values
<code>ylim</code>	latitude limits
<code>level</code>	maximum GSHHS level to include, defaults to 4 (everything), setting 1 will only retrieve land, no lakes
<code>minarea</code>	minimum area in square km to retrieve, default 0
<code>shift</code>	default FALSE, can be used to shift longitudes > 180 degrees to below zero, beware of artefacts involving unhandled polygon splitting at 180 degrees
<code>verbose</code>	default TRUE, print progress reports
<code>no.clip</code>	default FALSE, if TRUE, do not clip output polygons to bounding box

Details

The package is distributed with the coarse version of the shoreline data, and much more detailed versions may be downloaded from the referenced websites. The data is of high quality, matching the accuracy of SRTM shorelines for the full dataset (but not for inland waterbodies). In general, users will construct study region SpatialPolygons objects, which can then be exported (for example as a shapefile), or used in other R packages (such as PBSmapping). The largest land polygons take considerable time to clip to the study region, certainly many minutes for an extract from the full resolution data file including Eurasia (with Africa) or the Americas. For this reason, do not give up if nothing seems to be happening after the (verbose) message: "Rgshhs: clipping <m> of <n> polygons ..." appears. Clipping the largest polygons in full resolution also needs a good deal of memory

Value

for polygon data, a list with the following components:

polydata	data from the headers of the selected GSHHS polygons
belongs	a matrix showing which polygon belongs to (is included in) which polygon, going from the highest level among the selected polygons down to 1 (land); levels are: 1 land, 2 lake, 3 island_in_lake, 4 pond_in_island_in_lake.
new_belongs	a ragged list of polygon inclusion used for making SP
SP	a SpatialPolygons object; this is the principal output object, and will become the only output object as the package matures
SP	a SpatialLines object

Note

A number of steps are taken in this implementation that are unexpected, print messages, and so require explanation. Following the extraction of polygons intersecting the required region, a check is made to see if Antarctica is present. If it is, a new southern border is imposed at the southern ylim value or -90 if no ylim value is given. When clipping polygons seeming to intersect the required region boundary, it can happen that no polygon is left within the region (for example when the boundaries are overlaid, but also because the min/max polygon values in the header may not agree with the polygon itself (one case observed for a lake west of Groningen). The function then reports a null polygon. Another problem occurs when closed polygons are cut up during the finding of intersections between polygons and the required region boundary. The code in gplib does not close them, so they are closed later, and the closure noted.

Please also note that limitations on the use of gplib are very unclear, as exactly the same GPC code is included in the GPL'ed PBSmapping package on CRAN.

Author(s)

Roger Bivand

References

<http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html>,
<http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html> (only old format - use for information only); data downloaded from
ftp://ftp.soest.hawaii.edu/pwessel/gshhs/gshhs_1.10.zip; Wessel, P., and W. H. F. Smith, A Global Self-consistent, Hierarchical, High-resolution Shoreline Database, J. Geophys. Res., 101, 8741-8743, 1996.

Examples

```
gshhs.c.b <- system.file("share/gshhs.c.b", package="maptools")
NZx <- c(160,180)
NZy <- c(-50,-30)
NZ <- Rgshhs(gshhs.c.b, xlim=NZx, ylim=NZy)
plot(NZ$SP, col="khaki", pbg="azure2", xlim=NZx, ylim=NZy, xaxs="i", yaxs="i", axes=TRUE)
```

```

GLx <- c(265,285)
GLy <- c(40,50)
GL <- Rgshhs(gshhs.c.b, xlim=GLx, ylim=GLy)
plot(GL$SP, col="khaki", pbg="azure2", xlim=GLx, ylim=GLy, xaxs="i", yaxs="i", axes=TRUE)
BNLx <- c(2,8)
BNLy <- c(49,54)
wdb_lines <- system.file("share/wdb_borders_c.b", package="mapprools")
BNLp <- Rgshhs(gshhs.c.b, xlim=BNLx, ylim=BNLy)
BNLl <- Rgshhs(wdb_lines, xlim=BNLx, ylim=BNLy)
plot(BNLp$SP, col="khaki", pbg="azure2", xlim=BNLx, ylim=BNLy, xaxs="i", yaxs="i", axes=TRUE)
lines(BNLl$SP)

```

sp2Mondrian

write map data for Mondrian

Description

The function outputs a SpatialPolygonsDataFrame object to be used by Mondrian

Usage

```
sp2Mondrian(SP, file, new_format=TRUE)
```

Arguments

SP	a SpatialPolygonsDataFrame object
file	file where output is written
new_format	default TRUE, creates a text data file and a separate map file; the old format put both data sets in a single file - the map file is named by inserting "MAP_" into the file= argument after the rightmost directory separator (if any)

Note

At this release, the function writes out a text file with both data and polygon(s) identified as belonging to each row of data.

Author(s)

Patrick Hausmann and Roger Bivand

References

<http://rosuda.org/Mondrian/>

Examples

```
## Not run:
xx <- readShapePoly(system.file("shapes/columbus.shp", package="maptools")[1])
sp2Mondrian(xx, file="columbus1.txt")
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1])
sp2Mondrian(xx, file="sids1.txt")
## End (Not run)
```

sp2tmap

Convert SpatialPolygons object for Stata tmap command

Description

The function converts a SpatialPolygons object for use with the Stata tmap command, by creating a data frame with the required columns.

Usage

```
sp2tmap(SP)
```

Arguments

SP a SpatialPolygons object

Value

a data frame with three columns:

<code>_ID</code>	an integer vector of polygon identifiers in numeric order
<code>_X</code>	numeric x coordinate
<code>_Y</code>	numeric y coordinate

and an `ID_n` attribute with the named polygon identifiers

Author(s)

Roger Bivand

References

<http://www.stata.com/search.cgi?query=tmap>

See Also

[write.dta](#)

Examples

```
## Not run:
xx <- readShapePoly(system.file("shapes/sids.shp", package="mapprojtools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
plot(xx, border="blue", axes=TRUE, las=1)
tmapdf <- sp2tmap(as(xx, "SpatialPolygons"))
write.dta(tmapdf, file="NCmap.dta", version=7)
NCdf <- as(xx, "data.frame")
NCdf$ID_n <- attr(tmapdf, "ID_names")
write.dta(NCdf, file="NC.dta", version=7)
## End(Not run)
```

sp2WB

*Export SpatialPolygons object as S-Plus map for WinBUGS***Description**

The function exports an `sp SpatialPolygons` object into a S-Plus map format to be import by WinBUGS.

Usage

```
sp2WB(map, filename, Xscale = 1, Yscale = Xscale, plotorder = FALSE)
```

Arguments

<code>map</code>	a <code>SpatialPolygons</code> object
<code>filename</code>	file where output is written
<code>Xscale, Yscale</code>	scales to be written in the output file
<code>plotorder</code>	default=FALSE, if TRUE, export polygons in plotting order

Author(s)

Virgilio Gómez Rubio, partly derived from earlier code by Thomas Jagger

References

<http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/geobugs12manual.pdf>

Examples

```
xx <- readShapePoly(system.file("shapes/sids.shp", package="mapprojtools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
plot(xx, border="blue", axes=TRUE, las=1)
tf <- tempfile()
sp2WB(as(xx, "SpatialPolygons"), filename=tf)
xxx <- readSplus(tf, proj4string=CRS("+proj=longlat +ellps=clrk66"))
```

```

all.equal(xxx, as(xx, "SpatialPolygons"), tolerance=.Machine$double.eps^(1/4),
  check.attributes=FALSE)
## Not run:
x <- readAsciiGrid(system.file("grids/test.ag", package="mapprools")[1])
xp <- as(x, "SpatialPixelsDataFrame")
pp <- as.SpatialPolygons.SpatialPixels(xp)
sp2WB(pp, filename="test.map")
## End(Not run)

```

SpatialLines2PolySet

Convert sp line and polygon objects to PBSmapping PolySet objects

Description

Functions `SpatialLines2PolySet` and `SpatialPolygons2PolySet` convert objects of `sp` classes to `PolySet` class objects as defined in the `PBSmapping` package, and `PolySet2SpatialLines` and `PolySet2SpatialPolygons` convert in the opposite direction.

Usage

```

SpatialLines2PolySet (SL)
SpatialPolygons2PolySet (SpP)
PolySet2SpatialLines (PS)
PolySet2SpatialPolygons (PS, close_polys=TRUE)

```

Arguments

<code>SL</code>	a <code>SpatialLines</code> object as defined in the <code>sp</code> package
<code>SpP</code>	a <code>SpatialPolygons</code> object as defined in the <code>sp</code> package
<code>PS</code>	a <code>PolySet</code> object
<code>close_polys</code>	should polygons be closed if open

Value

`PolySet` objects as defined in the `PBSmapping` package

Author(s)

Roger Bivand and Andrew Niccolai

See Also

[PolySet](#), [MapGen2SL](#)

Examples

```

library(PBSmapping)
library(maps)
nor_coast_lines <- map("world", interior=FALSE, plot=FALSE, xlim=c(4,32),
  ylim=c(58,72))
nor_coast_lines <- pruneMap(nor_coast_lines, xlim=c(4,32), ylim=c(58,72))
nor_coast_lines_sp <- map2SpatialLines(nor_coast_lines,
  proj4string=CRS("+proj=longlat +datum=wgs84"))
nor_coast_lines_PS <- SpatialLines2PolySet(nor_coast_lines_sp)
summary(nor_coast_lines_PS)
plotLines(nor_coast_lines_PS)
o3 <- PolySet2SpatialLines(nor_coast_lines_PS)
plot(o3, axes=TRUE)
nor_coast_poly <- map("world", "norway", fill=TRUE, col="transparent",
  plot=FALSE, ylim=c(58,72))
IDs <- sapply(strsplit(nor_coast_poly$names, ":"), function(x) x[1])
nor_coast_poly_sp <- map2SpatialPolygons(nor_coast_poly, IDs=IDs,
  proj4string=CRS("+proj=longlat +datum=wgs84"))
nor_coast_poly_PS <- SpatialPolygons2PolySet(nor_coast_poly_sp)
summary(nor_coast_poly_PS)
plotPolys(nor_coast_poly_PS)
o1 <- PolySet2SpatialPolygons(nor_coast_poly_PS)
plot(o1, axes=TRUE)

```

spCbind-methods *cbind for spatial objects*

Description

spCbind provides cbind-like methods for Spatial*DataFrame objects in addition to the \$, [\leftarrow and [\leftarrow methods already available.

Methods

- obj = "SpatialPointsDataFrame", x = "data.frame"** cbind a data frame to the data slot of a SpatialPointsDataFrame object
- obj = "SpatialPointsDataFrame", x = "vector"** cbind a vector to the data slot of a SpatialPoints-DataFrame object
- obj = "SpatialLinesDataFrame", x = "data.frame"** cbind a data frame to the data slot of a SpatialLinesDataFrame object; the data frame argument must have row names set to the Lines ID values, and should be re-ordered first by matching against a shared key column
- obj = "SpatialLinesDataFrame", x = "vector"** cbind a vector to the data slot of a SpatialLines-DataFrame object
- obj = "SpatialPolygonsDataFrame", x = "data.frame"** cbind a data frame to the data slot of a SpatialPolygonsDataFrame object; the data frame argument must have row names set to the Polygons ID values, and should be re-ordered first by matching against a shared key column
- obj = "SpatialPolygonsDataFrame", x = "vector"** cbind a vector to the data slot of a SpatialPolygonsDataFrame object

Author(s)

Roger Bivand

See Also[spChFIDs-methods](#), [spRbind-methods](#)**Examples**

```
xx <- readShapePoly(system.file("shapes/sids.shp", package="mapprojtools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
xtra <- read.dbf(system.file("share/nc_xtra.dbf", package="mapprojtools")[1])
o <- match(xx$CNTY_ID, xtra$CNTY_ID)
xtral <- xtra[o,]
row.names(xtral) <- xx$FIPSNO
xx1 <- spCbind(xx, xtral)
names(xx1)
identical(xx1$CNTY_ID, xx1$CNTY_ID.1)
```

spChFIDs-methods *change feature IDs in spatial objects*

Description

When the feature IDs need to be changed in `SpatialLines*` or `SpatialPolygons*` objects, these methods may be used. The new IDs should be a character vector of unique IDs of the correct length.

Methods

obj = "`SpatialLines`", **x** = "`character`" replace IDs in a `SpatialLines` object

obj = "`SpatialLinesDataFrame`", **x** = "`character`" replace IDs in a `SpatialLinesDataFrame` object

obj = "`SpatialPolygons`", **x** = "`character`" replace IDs in a `SpatialPolygons` object

obj = "`SpatialPolygonsDataFrame`", **x** = "`character`" replace IDs in a `SpatialPolygonsDataFrame` object

Note

It is usually sensible to keep a copy of the original feature IDs in the object, but this should be done by the user.

Author(s)

Roger Bivand

See Also[spCbind-methods](#), [spRbind-methods](#)

Examples

```
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
row.names(as(xx, "data.frame"))
xx1 <- spChFIDs(xx, as.character(xx$CNTY_ID))
row.names(as(xx1, "data.frame"))
```

spRbind-methods *rbind for spatial objects*

Description

spRbind provides rbind-like methods for Spatial*DataFrame objects

Methods

obj = "SpatialPoints", x = "SpatialPoints" rbind two SpatialPoints objects

obj = "SpatialPointsDataFrame", x = "SpatialPointsDataFrame" rbind two SpatialPointsDataFrame objects

obj = "SpatialLines", x = "SpatialLines" rbind two SpatialLines objects

obj = "SpatialLinesDataFrame", x = "SpatialLinesDataFrame" rbind two SpatialLinesDataFrame objects

obj = "SpatialPolygons", x = "SpatialPolygons" rbind two SpatialPolygons objects

obj = "SpatialPolygonsDataFrame", x = "SpatialPolygonsDataFrame" rbind two SpatialPolygonsDataFrame objects

Author(s)

Roger Bivand

See Also

[spChFIDs-methods](#), [spCbind-methods](#)

Examples

```
xx <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1],
  IDvar="FIPSNO", proj4string=CRS("+proj=longlat +ellps=clrk66"))
summary(xx)
xx$FIPSNO
xx1 <- xx[xx$CNTY_ID < 1982,]
xx2 <- xx[xx$CNTY_ID >= 1982,]
xx3 <- spRbind(xx2, xx1)
summary(xx3)
xx3$FIPSNO
```

subset.polylist *Subset polygon list objects*

Description

The function subsets a polygon list object, also subsetting region ID attributes and also map limits if required.

Usage

```
## S3 method for class 'polylist':
subset(x, subset, fit.bbox = TRUE, ...)
```

Arguments

x	a polylist object
subset	a logical vector valued TRUE if the element is to be retained
fit.bbox	if TRUE (default) modifies the maplim attribute to bound the subset
...	other arguments passed through

Value

returns a polylist object, lists of polygon boundary coordinates (divided by NA where the polygon object is represented by more than one polygon);

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

Examples

```
nc <- read.shape(system.file("shapes/sids.shp", package = "maptools")[1])
mappolys <- Map2poly(nc, as.character(nc$att.data$FIPSNO))
submap <- subset(mappolys, nc$att.data$SID74 > 0)
plot(mappolys, col="orange")
plot(submap, add=TRUE, col="lightpink", forcefill=TRUE)
```

Description

Functions for calculating sunrise, sunset, and times of dawn and dusk, with flexibility for the various formal definitions. They use algorithms provided by the National Oceanic & Atmospheric Administration (NOAA).

Usage

```
## S4 method for signature 'SpatialPoints, POSIXct':
crepuscule(crds, dateTime, solarDep, direction=c("dawn", "dusk"),
           POSIXct.out=FALSE)
## S4 method for signature 'matrix, POSIXct':
crepuscule(crds, dateTime,
           proj4string=CRS("+proj=longlat +datum=WGS84"), solarDep,
           direction=c("dawn", "dusk"), POSIXct.out=FALSE)
## S4 method for signature 'SpatialPoints, POSIXct':
sunriset(crds, dateTime, direction=c("sunrise", "sunset"),
         POSIXct.out=FALSE)
## S4 method for signature 'matrix, POSIXct':
sunriset(crds, dateTime,
         proj4string=CRS("+proj=longlat +datum=WGS84"),
         direction=c("sunrise", "sunset"), POSIXct.out=FALSE)
## S4 method for signature 'SpatialPoints, POSIXct':
solarnoon(crds, dateTime, POSIXct.out=FALSE)
## S4 method for signature 'matrix, POSIXct':
solarnoon(crds, dateTime, proj4string=CRS("+proj=longlat +datum=WGS84"),
         POSIXct.out=FALSE)
## S4 method for signature 'SpatialPoints, POSIXct':
solarpos(crds, dateTime, ...)
## S4 method for signature 'matrix, POSIXct':
solarpos(crds, dateTime, proj4string=CRS("+proj=longlat +datum=WGS84"), ...)
```

Arguments

<code>crds</code>	a <code>SpatialPoints</code> or <code>matrix</code> object, containing x and y coordinates (in that order).
<code>dateTime</code>	a <code>POSIXct</code> object with the date and time associated to calculate ephemerides for points given in <code>crds</code> .
<code>solarDep</code>	numeric vector with the angle of the sun below the horizon in degrees.
<code>direction</code>	one of "dawn", "dusk", "sunrise", or "sunset", indicating which ephemerides should be calculated.

`POSIXct.out` logical indicating whether POSIXct output should be included.
`proj4string` string with valid projection string describing the projection of data in `crds`.
`...` other arguments passed through.

Details

NOAA used the reference below to develop their Sunrise/Sunset

<http://www.srrb.noaa.gov/highlights/sunrise/sunrise.html> and Solar Position

<http://www.srrb.noaa.gov/highlights/sunrise/azel.html> Calculators. The algorithms include corrections for atmospheric refraction effects.

Input can consist of one location and at least one POSIXct times, or one POSIXct time and at least one location. `solarDep` is recycled as needed.

Do not use the daylight savings time zone string for supplying `dateTime`, as many OS will not be able to properly set it to standard time when needed.

Value

`crepuscule`, `sunriseset`, and `solarnoon` return a numeric vector with the time of day at which the event occurs, expressed as a fraction, if `POSIXct.out` is `FALSE`; otherwise they return a data frame with both the fraction and the corresponding POSIXct date and time. `solarpos` returns a matrix with the solar azimuth (in degrees from North), and elevation.

Warning

Compared to NOAA's original Javascript code, the sunrise and sunset estimates from this translation may differ by +/- 1 minute, based on tests using selected locations spanning the globe. This translation does not include calculation of prior or next sunrises/sunsets for locations above the Arctic Circle or below the Antarctic Circle.

Note

NOAA notes that "for latitudes greater than 72 degrees N and S, calculations are accurate to within 10 minutes. For latitudes less than +/- 72 degrees accuracy is approximately one minute."

Author(s)

Sebastian P. Luque (spluque@gmail.com), translated from Greg Pelletier's (gpel461@ecy.wa.gov) VBA code (available from <http://www.ecy.wa.gov/programs/eap/models.html>), who in turn translated it from original Javascript code by NOAA (see Details). Roger Bivand (roger.bivand@nhh.no) adapted the code to work with `sp` classes.

References

Meeus, J. (1991) *Astronomical Algorithms*. Willmann-Bell, Inc.

Examples

```
## Location of Helsinki, Finland, in decimal degrees,
## as listed in NOAA's website
hels <- matrix(c(24.97, 60.17), nrow=1)
Hels <- SpatialPoints(hels, proj4string=CRS("+proj=longlat +datum=WGS84"))
d041224 <- as.POSIXct("2004-12-24", tz="EET")
## Astronomical dawn
crepuscule(hels, d041224, solarDep=18, direction="dawn", POSIXct.out=TRUE)
crepuscule(Hels, d041224, solarDep=18, direction="dawn", POSIXct.out=TRUE)
## Nautical dawn
crepuscule(hels, d041224, solarDep=12, direction="dawn", POSIXct.out=TRUE)
crepuscule(Hels, d041224, solarDep=12, direction="dawn", POSIXct.out=TRUE)
## Civil dawn
crepuscule(hels, d041224, solarDep=6, direction="dawn", POSIXct.out=TRUE)
crepuscule(Hels, d041224, solarDep=6, direction="dawn", POSIXct.out=TRUE)
solarnoon(hels, d041224, POSIXct.out=TRUE)
solarnoon(Hels, d041224, POSIXct.out=TRUE)
solarpos(hels, as.POSIXct(Sys.time(), tz="EET"))
solarpos(Hels, as.POSIXct(Sys.time(), tz="EET"))
sunriset(hels, d041224, direction="sunrise", POSIXct.out=TRUE)
sunriset(Hels, d041224, direction="sunrise", POSIXct.out=TRUE)
## Using a sequence of dates
Hels_seq <- seq(from=d041224, length.out=365, by="days")
up <- sunriset(Hels, Hels_seq, direction="sunrise", POSIXct.out=TRUE)
down <- sunriset(Hels, Hels_seq, direction="sunset", POSIXct.out=TRUE)
day_length <- down$time - up$time
plot(Hels_seq, day_length, type="l")

## Using a grid of spatial points for the same point in time
grd <- GridTopology(c(-179,-89), c(1,1), c(359,179))
SP <- SpatialPoints(coordinates(grd),
                    proj4string=CRS("+proj=longlat +datum=WGS84"))
wint <- as.POSIXct("2004-12-21", tz="GMT")
win <- crepuscule(SP, wint, solarDep=6, direction="dawn")
SPDF <- SpatialGridDataFrame(grd,
                             proj4string=CRS("+proj=longlat +datum=WGS84"),
                             data=data.frame(winter=win))
image(SPDF, axes=TRUE, col=cm.colors(40))
```

symbolsInPolys

Place grids of points over polygons

Description

Place grids of points over polygons with chosen density and/or symbols (suggested by Michael Wolf).

Usage

```
symbolsInPolys(pl, dens, symb = "+", compatible = FALSE)
```

Arguments

<code>pl</code>	list of polygons of class <code>polylist</code>
<code>dens</code>	number of symbol plotting points per unit area; either a single numerical value for all polygons, or a numeric vector the same length as <code>pl</code> with values for each polygon
<code>symb</code>	plotting symbol; either a single value for all polygons, or a vector the same length as <code>pl</code> with values for each polygon
<code>compatible</code>	what to return, if TRUE a a list of matrices of point coordinates, one matrix for each member of <code>pl</code> , with a <code>symb</code> attribute, if false a <code>SpatialPointsDataFrame</code> with a <code>symb</code> column

Details

The dots are placed in a grid pattern with the number of points per polygon being polygon area times density (number of dots not guaranteed to be the same as the count). When the polygon is made up of more than one part, the dots will be placed in proportion to the relative areas of the clockwise rings (anticlockwise are taken as holes). From `maptools` release 0.5-2, correction is made for holes in the placing of the dots, but depends on hole values being correctly set, which they often are not. The wrapper package `sppgc` may be used to check holes, see the `dontrun` section of the example.

Value

The function returns a list of matrices of point coordinates, one matrix for each member of `pl`; each matrix has a `symb` attribute that can be used for setting the `pch` argument for plotting. If the count of points for the given density and polygon area is zero, the list element is `NULL`, and can be tested when plotting - see the examples.

Note

Extension to plot pixmaps at the plotting points using `addlogo()` from the `pixmap` package is left as an exercise for the user.

Author(s)

Roger Bivand <Roger.Bivand@nhh.no>

See Also

[spsample](#)

Examples

```
x <- read.shape(system.file("shapes/sids.shp", package="maptools")[1])
ncpolys <- Map2poly(x)
np <- sapply(ncpolys, function(x) attr(x, "nPart"))
syms <- c("-", "+", "x")
try1 <- symbolsInPolys(ncpolys, 100, symb=syms[np], compatible=TRUE)
plot(ncpolys)
```

```

xx <- lapply(try1, function(x) {if (!is.null(x)) points(x, pch=attr(x, "symb"))})
nc_SP <- readShapePoly(system.file("shapes/sids.shp", package="mapproj")[1],
  proj4string=CRS("+proj=longlat +ellps=clrk66"))
## Not run:
library(spgpc)
pls <- slot(nc_SP, "polygons")
pls_new <- lapply(pls, checkPolygonsHoles)
nc_SP <- SpatialPolygonsDataFrame(SpatialPolygons(pls_new,
  proj4string=CRS(proj4string(nc_SP))), data=as(nc_SP, "data.frame"))
## End(Not run)
syms <- c("-", "+", "x")
np <- sapply(slot(nc_SP, "polygons"), function(x) length(slot(x, "Polygons")))
try1 <- symbolsInPolys(nc_SP, 100, symb=syms[np])
plot(nc_SP, axes=TRUE)
plot(try1, add=TRUE, pch=as.character(try1$symb))

```

unionSpatialPolygons

Aggregate Polygons in a SpatialPolygons object

Description

The function aggregates Polygons in a SpatialPolygons object, according to the IDs vector specifying which input Polygons belong to which output Polygons; internal boundaries are dissolved using the `gpcplib` package `union()` function.

Usage

```
unionSpatialPolygons(SpP, IDs, threshold=NULL)
```

Arguments

<code>SpP</code>	A SpatialPolygons object as defined in package <code>sp</code>
<code>IDs</code>	A vector defining the output Polygons objects, equal in length to the length of the polygons slot of <code>SpP</code> ; it may be character, integer, or factor (try <code>table(factor(IDs))</code> for a sanity check)
<code>threshold</code>	if not <code>NULL</code> , an area measure below which slivers will be discarded (some polygons have non-identical boundaries, for instance along rivers, generating slivers on union which are artefacts, not real sub-polygons)

Value

Returns an aggregated SpatialPolygons object named with the aggregated IDs values in their sorting order; see the ID values of the output object to view the order.

Author(s)

Roger Bivand

Examples

```
library(sp)
library(gpclib)
nc1 <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1],
  proj4string=CRS("+proj=longlat +datum=NAD27"))
lps <- coordinates(nc1)
ID <- cut(lps[,1], quantile(lps[,1]), include.lowest=TRUE)
reg4 <- unionSpatialPolygons(nc1, ID)
sapply(slot(reg4, "polygons"), function(i) slot(i, "ID"))
```

```
write.linelistShape
```

Write a arc-type shapefile

Description

The function calls code from shapelib to write an arc-type shapefile from a list of matrices of line coordinates with no NAs.

Usage

```
write.linelistShape(linelist, df, file, factor2char = TRUE,
  strictFilename=FALSE, max_nchar=254)
```

Arguments

linelist	a list of matrices of line coordinates
df	a data frame object
file	a file name of maximum 8 characters, numbers or the underscore symbol to be written, omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
factor2char	logical, default TRUE, convert factor columns to character
strictFilename	if TRUE, impose file basename length limit of 8 characters
max_nchar	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

The function calls code from shapelib to write an arc-type shapefile (both the geometry file with a *.shp extension, the index file with a *.shx extension, and the database file with a *.dbf extension - see [write.dbf](#) for details of the data frame export within this function.

Value

no return value.

Note

From maptools 0.4-10, this function is placed in the user-visible namespace on a trial basis, and reports of any malfunction should be sent to the package maintainer, Roger Bivand (Roger.Bivand@nhh.no). It is likely that this function and its arguments will be changed.

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand; shapelib by Frank Warmerdam

References

<http://shapelib.maptools.org/>

See Also

[write.pointShape](#), [write.dbf](#)

Examples

```
x <- 10 * 1:nrow(volcano)
y <- 10 * 1:ncol(volcano)
line.list <- contourLines(x, y, volcano)
vol.levels <- data.frame(alt=sapply(line.list, function(x) x[[1]]))
vol.ll <- lapply(line.list, function(x) cbind(x$x, x$y))
for (i in seq(along=vol.ll)) {
  attr(vol.ll[[i]], "nParts") <- as.integer(1)
  attr(vol.ll[[i]], "pstart") <- list(as.integer(1),
    as.integer(nrow(vol.ll[[i]])))
}
tmpshp <- paste(tempdir(), "volcano", sep="/")
write.linelistShape(vol.ll, vol.levels, file=tmpshp)
try1 <- readShapeLines(tmpshp)
plot(try1)
```

write.pointShape *Write a point-type shapefile*

Description

The function calls code from shapelib to write a point-type shapefile.

Usage

```
write.pointShape(coordinates, df, file, factor2char = TRUE,
  strictFilename=FALSE, max_nchar=254)
```

Arguments

coordinates	a 2-column numeric matrix of coordinates
df	a data frame object
file	a file name of maximum 8 characters, numbers or the underscore symbol to be written, omitting the extensions *.shp, *.shx and *.dbf, which are added in the function
factor2char	logical, default TRUE, convert factor columns to character
strictFilename	if TRUE, impose file basename length limit of 8 characters
max_nchar	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

The function calls code from shapelib to write a point-type shapefile (both the geometry file with a *.shp extension, the index file with a *.shx extension, and the database file with a *.dbf extension - see [write.dbf](#) for details of the data frame export within this function.

Value

no return value.

Note

From maptools 0.4-7, this function is placed in the user-visible namespace on a trial basis, and reports of any malfunction should be sent to the package maintainer, Roger Bivand (Roger.Bivand@nhh.no). It is likely that this function and its arguments will be changed.

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand; shapelib by Frank Warmerdam

References

<http://shapelib.maptools.org/>

See Also

[write.dbf](#)

Examples

```
balt_orig <- readShapePoints(system.file("shapes/baltim.shp", package="maptools")[1])
plot(balt_orig)
balt_cheap <- balt_orig[balt_orig$PRICE < 40,]
file <- tempfile("")
write.pointShape(coordinates=coordinates(balt_cheap),
  df=as(balt_cheap, "data.frame"), file)
getinfo.shape(paste(file, ".shp", sep=""))
```

```
balt_new <- readShapePoints(paste(file, ".shp", sep=""))
plot(balt_new, col="red", pch=16, add=TRUE)
```

```
write.polylistShape
```

Write a polygon-type shapefile

Description

The function calls code from shapelib to write a polygon-type shapefile from an S3 polylist object.

Usage

```
write.polylistShape(polylist, df, file, factor2char = TRUE,
  strictFilename=FALSE, force = TRUE, max_nchar=254)
```

Arguments

<code>polylist</code>	list of polygons of class <code>polylist</code>
<code>df</code>	a data frame object
<code>file</code>	a file name of maximum 8 characters, numbers or the underscore symbol to be written, omitting the extensions <code>*.shp</code> , <code>*.shx</code> and <code>*.dbf</code> , which are added in the function
<code>factor2char</code>	logical, default TRUE, convert factor columns to character
<code>strictFilename</code>	if TRUE, impose file basename length limit of 8 characters
<code>force</code>	default TRUE, to try to force malformed polylist objects to some reasonable form that will not cause both maptools and R to crash. Because polylist objects are old-style rather than new-style classes, it is possible to crash R by trying to write malformed objects. Attempts are made to check for known problems, but using polylist objects rather than <code>sp SpatialPolygons</code> objects is known to be more risky. From release 0.6-14, polylist objects will be given an <code>nDims</code> attribute at this stage to permit the output of constructed 3D objects, and so report a warning, which should be ignored.
<code>max_nchar</code>	default 254, may be set to a higher limit and passed through to the DBF writer, please see Details in write.dbf

Details

The function calls code from shapelib to write a polygon-type shapefile (both the geometry file with a `*.shp` extension, the index file with a `*.shx` extension, and the database file with a `*.dbf` extension - see [write.dbf](#) for details of the data frame export within this function.

Value

no return value.

Note

From mapprools 0.4-7, this function is placed in the user-visible namespace on a trial basis, and reports of any malfunction should be sent to the package maintainer, Roger Bivand (Roger.Bivand@nhh.no). It is likely that this function and its arguments will be changed.

Author(s)

Nicholas J. Lewin-Koh, modified by Roger Bivand; shapelib by Frank Warmerdam

References

<http://shapelib.mapprools.org/>

See Also

[write.pointShape](#), [write.dbf](#)

Examples

```
col_orig <- read.shape(system.file("shapes/columbus.shp", package="mapprools")[1])
mappolys <- Map2poly(col_orig, as.character(col_orig$att.data$NEIGNO))
plot(mappolys)
col_df <- col_orig$att.data
col_cheap <- subset(mappolys, col_df$HOVAL < 34)
col_df_cheap <- subset(col_df, col_df$HOVAL < 34)
file <- tempfile("")
write.polylistShape(col_cheap, col_df_cheap, file)
getinfo.shape(paste(file, ".shp", sep=""))
col_new <- read.shape(paste(file, ".shp", sep=""))
mappolys <- Map2poly(col_new, as.character(col_new$att.data$NEIGNO))
plot(mappolys, border="red", add=TRUE)
```

wrlld_simpl

Simplified world country polygons

Description

The object loaded is a `SpatialPolygonsDataFrame` object containing a slightly modified version of Bjoern Sandvik's improved version of `world_borders.zip - TM_WORLD_BORDERS_SIMPL-0.2.zip` dataset from the Mapping Hacks geodata site. The country Polygons objects and the data slot data frame row numbers have been set to the ISO 3166 three letter codes.

Usage

```
data(wrlld_simpl)
```

Format

The format is: Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots; the data slot contains a data.frame with 246 obs. of 11 variables:

FIPS factor of FIPS country codes

ISO2 factor of ISO 2 character country codes

ISO3 factor of ISO 3 character country codes

UN integer vector of UN country codes

NAME Factor of country names

AREA integer vector of area values

POP2005 integer vector of population in 2005

REGION integer vector of region values

SUBREGION integer vector of subregion values

LON numeric vector of longitude label points

LAT numeric vector of latitude label points

The object is in geographical coordinates using the WGS84 datum.

Source

http://mappinghacks.com/data/TM_WORLD_BORDERS_SIMPL-0.2.zip

Examples

```
data(wrlld_simpl)
plot(wrlld_simpl)
```

Index

- *Topic **aplot**
 - pointLabel, 32
- *Topic **classes**
 - ppp-class, 34
- *Topic **datasets**
 - gpcholes, 15
 - wrld_simpl, 65
- *Topic **manip**
 - sun-methods, 56
- *Topic **methods**
 - elide-methods, 8
 - spCbind-methods, 52
 - spChFIDs-methods, 53
 - spRbind-methods, 54
 - sun-methods, 56
- *Topic **programming**
 - readAsciiGrid, 36
- *Topic **spatial**
 - as.ppp, 2
 - checkPolygonsHoles, 4
 - ContourLines2SLDF, 5
 - dotsInPolys, 7
 - elide-methods, 8
 - gcDestination, 10
 - GE_SpatialGrid, 13
 - get.Pcent, 11
 - getKMLcoordinates, 12
 - gzAzimuth, 16
 - kmlLine, 17
 - kmlOverlay, 19
 - kmlPolygon, 20
 - Map2poly, 22
 - map2SpatialPolygons, 24
 - maptools, 26
 - nowrapRecenter, 26
 - pal2SpatialPolygons, 27
 - plot.Map, 29
 - plot.polylist, 30
 - read.shape, 34
 - readGPS, 37
 - readShapeLines, 39
 - readShapePoints, 40
 - readShapePoly, 41
 - readShapeSpatial, 43
 - readSplus, 44
 - Rgshhs, 46
 - sp2Mondrian, 48
 - sp2tmap, 49
 - sp2WB, 50
 - SpatialLines2PolySet, 51
 - spCbind-methods, 52
 - spChFIDs-methods, 53
 - spRbind-methods, 54
 - subset.polylist, 55
 - symbolsInPolys, 58
 - unionSpatialPolygons, 60
 - write.linelistShape, 61
 - write.pointShape, 62
 - write.polylistShape, 64
- *Topic **utilities**
 - sun-methods, 56
- ArcObj2SLDF (*ContourLines2SLDF*), 5
- as.im.SpatialGridDataFrame
 - (*as.ppp*), 2
- as.owin.SpatialGridDataFrame
 - (*as.ppp*), 2
- as.owin.SpatialPixelsDataFrame
 - (*as.ppp*), 2
- as.owin.SpatialPolygons (*as.ppp*), 2
- as.ppp, 2
- as.ppp.SpatialGridDataFrame
 - (*as.ppp*), 2
- as.ppp.SpatialPoints (*as.ppp*), 2
- as.ppp.SpatialPointsDataFrame
 - (*as.ppp*), 2
- as.psp.Line (*as.ppp*), 2
- as.psp.Lines (*as.ppp*), 2

- as.psp.SpatialLines (*as.ppp*), 2
- as.psp.SpatialLinesDataFrame (*as.ppp*), 2
- as.SpatialGridDataFrame.im (*as.ppp*), 2
- as.SpatialGridDataFrame.ppp (*as.ppp*), 2
- as.SpatialPoints.ppp (*as.ppp*), 2
- as.SpatialPointsDataFrame.ppp (*as.ppp*), 2
- as.SpatialPolygons.owin (*as.ppp*), 2
- as.SpatialPolygons.tess (*as.ppp*), 2
- checkPolygonsHoles, 4, 45
- coerce, im, SpatialGridDataFrame-method (*as.ppp*), 2
- coerce, Line, psp-method (*as.ppp*), 2
- coerce, Lines, psp-method (*as.ppp*), 2
- coerce, owin, SpatialPolygons-method (*as.ppp*), 2
- coerce, ppp, SpatialGridDataFrame-method (*as.ppp*), 2
- coerce, ppp, SpatialPoints-method (*as.ppp*), 2
- coerce, ppp, SpatialPointsDataFrame-method (*as.ppp*), 2
- coerce, SpatialGridDataFrame, im-method (*as.ppp*), 2
- coerce, SpatialGridDataFrame, owin-method (*as.ppp*), 2
- coerce, SpatialLines, psp-method (*as.ppp*), 2
- coerce, SpatialLinesDataFrame, psp-method (*as.ppp*), 2
- coerce, SpatialPixelsDataFrame, owin-method (*as.ppp*), 2
- coerce, SpatialPoints, ppp-method (*as.ppp*), 2
- coerce, SpatialPointsDataFrame, ppp-method (*as.ppp*), 2
- coerce, SpatialPolygons, owin-method (*as.ppp*), 2
- coerce, tess, SpatialPolygons-method (*as.ppp*), 2
- ContourLines2SLDF, 5
- convert.pl (*Map2poly*), 22
- crepuscule (*sun-methods*), 56
- crepuscule, matrix, POSIXct-method (*sun-methods*), 56
- crepuscule, SpatialPoints, POSIXct-method (*sun-methods*), 56
- crepuscule-methods (*sun-methods*), 56
- CRS-class, 6
- dotsInPolys, 7
- elide (*elide-methods*), 8
- elide, SpatialLines-method (*elide-methods*), 8
- elide, SpatialLinesDataFrame-method (*elide-methods*), 8
- elide, SpatialPoints-method (*elide-methods*), 8
- elide, SpatialPointsDataFrame-method (*elide-methods*), 8
- elide, SpatialPolygons-method (*elide-methods*), 8
- elide, SpatialPolygonsDataFrame-method (*elide-methods*), 8
- elide-methods, 8
- findInterval, 31
- geDestination, 10
- GE_SpatialGrid, 13, 19
- get.Pcent, 11
- getinfo.shape, 30
- getinfo.shape (*read.shape*), 34
- getKMLcoordinates, 12
- gpcholes, 15
- gzAzimuth, 11, 16
- hole1pl (*gpcholes*), 15
- hole2pl (*gpcholes*), 15
- im-class (*ppp-class*), 34
- image, 37
- kmlLine, 12, 17, 21
- kmlOverlay, 14, 18, 19, 21
- kmlPolygon, 12, 18, 20
- leglabs (*plot.polylist*), 30
- Line, 18
- map, 25

- Map2bbs (*Map2poly*), 22
- Map2lines (*Map2poly*), 22
- Map2points (*Map2poly*), 22
- Map2poly, 22, 31
- Map2poly1 (*Map2poly*), 22
- map2SpatialLines
 - (*map2SpatialPolygons*), 24
- map2SpatialPolygons, 24, 45
- MapGen2SL, 51
- MapGen2SL (*ContourLines2SLDF*), 5
- MapShapeIds (*Map2poly*), 22
- maptools, 26

- nowrapRecenter, 26
- nowrapSpatialLines, 27
- nowrapSpatialPolygons
 - (*nowrapRecenter*), 26

- owin-class (*ppp-class*), 34

- pal2SpatialPolygons, 27
- plot.Map, 29, 35
- plot.polylist, 30
- plotpolys (*plot.polylist*), 30
- pointLabel, 32
- PolySet, 51
- PolySet2SpatialLines
 - (*SpatialLines2PolySet*), 51
- PolySet2SpatialPolygons
 - (*SpatialLines2PolySet*), 51
- ppp-class, 34
- print.shapehead (*read.shape*), 34
- pruneMap (*map2SpatialPolygons*), 24
- psp-class (*ppp-class*), 34

- read.dbf, 35
- read.shape, 24, 30, 34
- readAsciiGrid, 36
- readGPS, 37
- readShapeLines, 30, 39
- readShapePoints, 30, 40
- readShapePoly, 30, 41
- readShapeSpatial, 43
- readSplus, 44
- recenter-methods, 27
- Rgshhs, 46

- shape2bbs (*Map2poly*), 22
- shape2lines (*Map2poly*), 22
- shape2points (*Map2poly*), 22
- shape2poly (*Map2poly*), 22
- shapefiles, 24
- Sobj_SpatialGrid
 - (*GE_SpatialGrid*), 13
- solarnoon (*sun-methods*), 56
- solarnoon, matrix, POSIXct-method
 - (*sun-methods*), 56
- solarnoon, SpatialPoints, POSIXct-method
 - (*sun-methods*), 56
- solarnoon-methods (*sun-methods*), 56
- solarpos (*sun-methods*), 56
- solarpos, matrix, POSIXct-method
 - (*sun-methods*), 56
- solarpos, SpatialPoints, POSIXct-method
 - (*sun-methods*), 56
- solarpos-methods (*sun-methods*), 56
- sp2Mondrian, 48
- sp2tmap, 49
- sp2WB, 50
- SpatialGridDataFrame, 36
- SpatialGridDataFrame-class, 36
- SpatialLines-class, 6
- SpatialLines2PolySet, 51
- SpatialPolygons, 21
- SpatialPolygons2PolySet
 - (*SpatialLines2PolySet*), 51
- spCbind (*spCbind-methods*), 52
- spCbind, SpatialLinesDataFrame, data.frame-method
 - (*spCbind-methods*), 52
- spCbind, SpatialLinesDataFrame, vector-method
 - (*spCbind-methods*), 52
- spCbind, SpatialPointsDataFrame, data.frame-method
 - (*spCbind-methods*), 52
- spCbind, SpatialPointsDataFrame, vector-method
 - (*spCbind-methods*), 52
- spCbind, SpatialPolygonsDataFrame, data.frame-method
 - (*spCbind-methods*), 52
- spCbind, SpatialPolygonsDataFrame, vector-method
 - (*spCbind-methods*), 52
- spCbind-methods, 53, 54
- spCbind-methods, 52
- spChFIDs (*spChFIDs-methods*), 53
- spChFIDs, SpatialLines, character-method
 - (*spChFIDs-methods*), 53
- spChFIDs, SpatialLinesDataFrame, character-method
 - (*spChFIDs-methods*), 53

spChFIDs, SpatialPolygons, character-method `writePolyShape (readShapePoly)`, 41
 (*spChFIDs-methods*), 53 `writeSpatialShape`
 spChFIDs, SpatialPolygonsDataFrame, character-method `readShapeSpatial`, 43
 (*spChFIDs-methods*), 53 `wrld_simpl`, 65
 spChFIDs-methods, 53, 54
 spChFIDs-methods, 53
 spRbind (*spRbind-methods*), 54
 spRbind, SpatialLines, SpatialLines-method
 (*spRbind-methods*), 54
 spRbind, SpatialLinesDataFrame, SpatialLinesDataFrame-method
 (*spRbind-methods*), 54
 spRbind, SpatialPoints, SpatialPoints-method
 (*spRbind-methods*), 54
 spRbind, SpatialPointsDataFrame, SpatialPointsDataFrame-method
 (*spRbind-methods*), 54
 spRbind, SpatialPolygons, SpatialPolygons-method
 (*spRbind-methods*), 54
 spRbind, SpatialPolygonsDataFrame, SpatialPolygonsDataFrame-method
 (*spRbind-methods*), 54
 spRbind-methods, 53
 spRbind-methods, 54
 spsample, 8, 59
 subset.polylist, 55
 sun-methods, 56
 sunriset (*sun-methods*), 56
 sunriset, matrix, POSIXct-method
 (*sun-methods*), 56
 sunriset, SpatialPoints, POSIXct-method
 (*sun-methods*), 56
 sunriset-methods (*sun-methods*), 56
 symbolsInPolys, 58

 text, 33
 thigmophobe.labels, 33
 trackAzimuth (*gzAzimuth*), 16

 unionSpatialPolygons, 60

 write.dbf, 39, 40, 42, 43, 61–65
 write.dta, 49
 write.linelistShape, 61
 write.pointShape, 62, 62, 65
 write.polylistShape, 64
 write.table, 36
 writeAsciiGrid (*readAsciiGrid*), 36
 writeLinesShape (*readShapeLines*),
 39
 writePointsShape
 (*readShapePoints*), 40