# Package 'PBSmapping'

July 2, 2014

**Version** 2.67.60

**Date** 2014-03-27

**Title** Mapping Fisheries Data and Spatial Analysis Tools

**Author**
Jon T. Schnute <schnutej-dfo@shaw.ca>,Nicholas Boers <boersn@macewan.ca>,Rowan Haigh <rowan.haigh@dfo-mpo
son, Paul Wessel, Franklin Antonio

**Maintainer** Rowan Haigh <rowan.haigh@dfo-mpo.gc.ca>

**Depends** R (>= 2.15.0)

**Suggests** foreign, maptools, deldir

**Description** This software has evolved from fisheries research conducted at the
Pacific Biological Station (PBS) in Nanaimo, British Columbia, Canada. It
extends the R language to include two-dimensional plotting features similar
to those commonly available in a Geographic Information System (GIS).
Embedded C code speeds algorithms from computational geometry, such as
finding polygons that contain specified point events or converting between
longitude-latitude and Universal Transverse Mercator (UTM) coordinates.
Additionally, we include C++ code developed by Angus Johnson for the Clipper
library. PBSmapping also includes data for a global shoreline and other
data sets in the public domain.
The R directory '.../library/PBSmapping/doc' offers a complete user's
guide PBSmapping-UG.pdf, which should be consulted to use all functions in
the package effectively.

**License** GPL (>= 2)

**URL** http://code.google.com/p/pbs-mapping/,http://code.google.com/p/pbs-mapx/,http:
//www.angusj.com/delphi/clipper.php

**NeedsCompilation** yes

**X-CRAN-Comment** Earlier versions of this package have been removed as
the license stated conflicted with conditions in the source code.

**Repository** CRAN

**Date/Publication** 2014-04-05 07:58:59

# R topics documented:

---

`addBubbles` *Add Bubbles to Maps*

---

## Description

Add bubbles proportional to some `EventData`'s `Z` column (e.g., catch or effort) to an existing plot, where each unique `EID` describes a bubble.

## Usage

```
addBubbles(events, type=c("perceptual","surface","volume"),
   z.max=NULL, min.size = 0, max.size=0.8, symbol.zero="+",
   symbol.fg=rgb(0,0,0,0.6), symbol.bg=rgb(0,0,0,0.3),
   legend.pos="bottomleft", legend.breaks=NULL,
   show.actual=FALSE, legend.type=c("nested","horiz","vert"),
   legend.title="Abundance", legend.cex=0.8, ...)
```

## Arguments

| | |
|---|---|
| `events` | EventData to use (*required*). |
| `type` | scaling option for bubbles where `"perceptual"` emphasizes large z-values, `"volume"` emphasizes small z-values, and `"surface"` lies in between. |
| `z.max` | maximum value for z (default = `max(events$Z)`); determines the largest bubble; keeps the same legend for different maps. |

| | |
|---|---|
| min.size | minimum size (inches) for a bubble representing min(events$Z). The legend may not actually include a bubble of this size because the calculated legend.breaks does not include the min(events$Z). |
| max.size | maximum size (inches) for a bubble representing z.max. A legend bubble may exceed this size when show.actual is FALSE (on account of using pretty(...)). |
| symbol.zero | symbol to represent z-values equal to 0. |
| symbol.fg | bubble outline (border) colour. |
| symbol.bg | bubble interior (fill) colour. If a vector, the first element represents min(legend.breaks) and the last element represents max(legend.breaks); colours are interpolated for values of events$Z between those boundaries. For values outside of those boundaries, interiors remain unfilled. |
| legend.pos | position for the legend. |
| legend.breaks | break values for categorizing the z-values. The automatic method should work if zeroes are present; otherwise, you can specify your own break values for the legend. If a single number, specifies the number of breaks; if a vector, specifies the breaks. |
| show.actual | logical; if FALSE, legend values are obtained using pretty(...), and consequently, the largest bubble may be larger than z.max. If TRUE, the largest bubble in the legend will correspond to z.max. |
| legend.type | display format for legend. |
| legend.title | title for legend. |
| legend.cex | size of legend text. |
| ... | additional arguments for points function that plots zero-value symbols. |

## Details

Modified from (and for the legend, strongly inspired by) Tanimura et al. (2006) by Denis Chabot to work with **PBSmapping**.

Furthermore, Chabot's modifications make it possible to draw several maps with bubbles that all have the same scale (instead of each bubble plot having a scale that depends on the maximum z-value for that plot). This is done by making z.max equal to the largest z-value from all maps that will be plotted.

The user can also add a legend in one of four corners (see legend) or at a specific c(X,Y) position. If legend.pos is NULL, no legend is drawn.

## Author(s)

Denis Chabot, Maurice Lamontagne Institute, Fisheries and Oceans Canada, Mont-Joli QC

## References

Tanimura, S., Kuroiwa, C., and Mizota, T. (2006) Proportional symbol mapping in R. *Journal of Statistical Software* **15**(5).

## See Also

addPolys, surveyData

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- common code for both examples below
  data(nepacLL,surveyData,envir=.PBSmapEnv)
  surveyData$Z <- surveyData$catch

  #--- plot a version that only varies the size
  plotMap(nepacLL, xlim=c(-131.8,-127.2), ylim=c(50.5,52.7),
    col="gainsboro",plt=c(.08,.99,.08,.99))
  addBubbles(surveyData, symbol.bg=rgb(.9,.5,0,.6),
    legend.type="nested", symbol.zero="+", col="grey")

  #--- plot a version that uses different symbol colours
  plotMap(nepacLL, xlim=c(-131.8,-127.2), ylim=c(50.5,52.7),
  col="gainsboro",plt=c(.08,.99,.08,.99))
  subset <- surveyData[surveyData$Z <= 1000, ]
  addBubbles(subset, symbol.bg=c("red", "yellow", "green"),
    legend.type="horiz", legend.breaks=pretty(range(subset$Z), n=11),
    symbol.zero=FALSE, col="grey", min.size=0.1, max.size=0.4)
  par(oldpar)
})
```

---

# addLabels *Add Labels to an Existing Plot*

---

## Description

Add the label column of data to the existing plot.

## Usage

```
addLabels (data, xlim = NULL, ylim = NULL, polyProps = NULL,
           placement = "DATA", polys = NULL, rollup = 3,
           cex = NULL, col = NULL, font = NULL, ...)
```

## Arguments

| | |
|---|---|
| data | EventData or PolyData to add (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| polyProps | PolyData specifying which labels to plot and their properties. par parameters passed as direct arguments supersede these data. |

| | |
|---|---|
| placement | one of "DATA", "CENTROID", "MEAN_RANGE", or "MEAN_XY". |
| polys | PolySet to use for calculating label placement. |
| rollup | level of detail at which to process polys, and it should match that in data. 1 = PIDs only, 2 = outer contours only, and 3 = no roll-up. |
| cex | vector describing character expansion factors (cycled by EID or PID). |
| col | vector describing colours (cycled by EID or PID). |
| font | vector describing fonts (cycled by EID or PID). |
| ... | additional par parameters for the text function. |

## Details

If data is EventData, it must minimally contain the columns EID, X, Y, and label. Since the EID column does not match a column in polys, set placement = "DATA". The function plots each label at its corresponding X/Y coordinate.

If data is PolyData, it must minimally contain the columns PID and label. If it also contains X and Y columns, set placement = "DATA" to plot labels at those coordinates. Otherwise, set placement to one of "CENTROID", "MEAN_RANGE", or "MEAN_XY". When placement != "DATA", supply a PolySet polys. Using this PolySet, the function calculates a centroid, mean range, or mean X/Y coordinate for each polygon, and then links those PolyData with data by PID/SID to determine label coordinates.

If data contains both PID and EID columns, the function assumes it is PolyData and ignores the EID column.

For additional help on the arguments cex, col, and font, please see par.

## Value

EventData or PolyData with X and Y columns that can subsequently reproduce the labels on the plot. Modify this data frame to tweak label positions.

## See Also

addPoints, calcCentroid, calcMidRange, calcSummary, EventData, plotPoints, PolyData.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample PolyData to label Vancouver Island
  labelData <- data.frame(PID=33, label="Vancouver Island");
  #--- load data
  if (!is.null(version$language) && (version$language == "R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot the map
  plotMap(nepacLL,xlim=c(-129,-122.6),ylim=c(48,51.1),col="lemonchiffon")
  #--- add the labels
  addLabels(labelData,placement="CENTROID",polys=nepacLL,cex=1.2,col=2,font=2)
  par(oldpar)
})
```

addLines *Add a PolySet to an Existing Plot as Polylines*

## Description

Add a PolySet to an existing plot, where each unique (PID, SID) describes a polyline.

## Usage

```
addLines (polys, xlim = NULL, ylim = NULL,
    polyProps = NULL, lty = NULL, col = NULL, arrows = FALSE, ...)
```

## Arguments

| | |
|---|---|
| polys | PolySet to add (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| polyProps | PolyData specifying which polylines to plot and their properties. par parameters passed as direct arguments supersede these data. |
| lty | vector of line types (cycled by PID). |
| col | vector of colours (cycled by PID). |
| arrows | Boolean value; if TRUE, add arrows using the arrows function and consider the arguments angle, length, and code. |
| ... | additional par parameters for the lines function. |

## Details

The plotting routine does not connect the last vertex of each discrete polyline to the first vertex of that polyline. It clips polys to xlim and ylim before plotting.

For additional help on the arguments lty and col, please see par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## See Also

calcLength, clipLines, closePolys, convLP, fixBound, fixPOS, locatePolys, plotLines, thinPolys, thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  polys <- as.PolySet(polys, projection=1)
  #--- plot the PolySet
  plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), projection=1)
  #--- add the PolySet to the plot (in a different style)
  addLines(polys, lwd=5, col=3)
  par(oldpar)
})
```

---

# addPoints *Add EventData/PolyData to an Existing Plot as Points*

---

## Description

Add EventData/PolyData to an existing plot, where each unique EID describes a point.

## Usage

```
addPoints (data, xlim = NULL, ylim = NULL, polyProps = NULL,
           cex = NULL, col = NULL, pch = NULL, ...)
```

## Arguments

| | |
|---|---|
| data | EventData or PolyData to add (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| polyProps | PolyData specifying which points to plot and their properties. par parameters passed as direct arguments supersede these data. |
| cex | vector describing character expansion factors (cycled by EID or PID). |
| col | vector describing colours (cycled by EID or PID). |
| pch | vector describing plotting characters (cycled by EID or PID). |
| ... | additional par parameters for the points function. |

## Details

This function clips data to xlim and ylim before plotting. It only adds PolyData containing X and Y columns.

For additional help on the arguments cex, col, and pch, please see par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## See Also

combineEvents, convDP, findPolys, locateEvents, plotPoints.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,surveyData,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL, xlim=c(-136, -125), ylim=c(48, 57))
  #--- add events
  addPoints(surveyData, col=1:7)
  par(oldpar)
})
```

---

# addPolys *Add a PolySet to an Existing Plot as Polygons*

---

## Description

Add a PolySet to an existing plot, where each unique (PID, SID) describes a polygon.

## Usage

```
addPolys (polys, xlim = NULL, ylim = NULL, polyProps = NULL,
          border = NULL, lty = NULL, col = NULL, colHoles = NULL,
          density = NA, angle = NULL, ...)
```

## Arguments

| | |
|---|---|
| polys | PolySet to add (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| polyProps | PolyData specifying which polygons to plot and their properties. par parameters passed as direct arguments supersede these data. |
| border | vector describing edge colours (cycled by PID). |
| lty | vector describing line types (cycled by PID). |
| col | vector describing fill colours (cycled by PID). |
| colHoles | vector describing hole colours (cycled by PID). The default, NULL, should be used in most cases as it renders holes transparent. colHoles is designed solely to eliminate retrace lines when images are converted to PDF format. If colHoles is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If NA is specified, only outer polygons are drawn, consequently filling holes. |

| | |
|---|---|
| density | vector describing shading line densities (lines per inch, cycled by PID). |
| angle | vector describing shading line angles (degrees, cycled by PID). |
| ... | additional par parameters for the polygon function. |

## Details

The plotting routine connects the last vertex of each discrete polygon to the first vertex of that polygon. It supports both borders (border, lty) and fills (col, density, angle). It clips polys to xlim and ylim before plotting.

For additional help on the arguments border, lty, col, density, and angle, please see polygon and par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## See Also

addLabels, addStipples, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotLines, plotMap, plotPoints, plotPolys, thinPolys, thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  polys <- as.PolySet(polys, projection=1)
  #--- plot the PolySet
  plotPolys(polys,xlim=c(-.5,1.5),ylim=c(-.5,1.5),density=0,projection=1)
  #--- add the PolySet to the plot (in a different style)
  addPolys(polys,col="green",border="blue",lwd=3)
  par(oldpar)
})
```

---

# addStipples *Add Stipples to an Existing Plot*

---

## Description

Add stipples to an existing plot.

## Usage

```
addStipples (polys, xlim=NULL, ylim=NULL, polyProps=NULL,
   side=1, density=1, distance=4, ...)
```

### Arguments

| | |
|---|---|
| polys | PolySet that provides the stipple boundaries (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| polyProps | PolyData specifying which polygons to stipple and their properties. par parameters passed as direct arguments supersede these data. |
| side | one of -1, 0, or 1, corresponding to outside, both sides, or inside, respectively. |
| density | density of points, relative to the default. |
| distance | distance to offset points, measured as a percentage of the absolute difference in xlim. |
| ... | additional par parameters for the points function. |

### Details

This function locates stipples based on the PolySet polys and does not stipple degenerate lines.

### Value

PolyData consisting of the PolyProps used to create the plot.

### See Also

addPoints, addPolys, plotMap, plotPoints, plotPolys, points, PolySet.

### Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL,xlim=c(-128.66,-122.83),ylim=c(48.00,51.16))
  #--- add stippling
  addStipples(nepacLL,col="purple",pch=20,cex=0.25,distance=2)
  par(oldpar)
})
```

---

## appendPolys *Append a Two-Column Matrix to a PolySet*

---

### Description

Append a two-column matrix to a PolySet, assigning PID and possibly SID values automatically or as specified in its arguments.

## Usage

```
appendPolys (polys, mat, PID = NULL, SID = NULL, isHole = FALSE)
```

## Arguments

| | |
|---|---|
| polys | existing PolySet; if NULL, creates a new PolySet (*required*). |
| mat | two-column matrix to append (*required*). |
| PID | new polygon's PID. |
| SID | new polygon's SID. |
| isHole | Boolean value; if TRUE, mat represents a hole. |

## Details

If the PID argument is NULL, the appended polygon's PID will be one greater than the maximum within polys (if defined); otherwise, it will be 1.

If polys contains an SID column and the SID argument equals NULL, this function uses the next available SID for the corresponding PID.

If polys does not contain an SID column and the caller passes an SID argument, all existing polygons will receive an SID of 1. The new polygon's SID will match the SID argument.

If isHole = TRUE, the polygon's POS values will appropriately represent a hole (reverse order of POS).

If (PID, SID) already exists in the PolySet, the function will issue a warning and duplicate those identifiers.

## Value

PolySet containing mat appended to polys. The function retains attributes from polys.

## See Also

addPolys, clipPolys, closePolys, convLP, fixBound, fixPOS, joinPolys, plotMap, plotPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create two simple matrices
  a <- matrix(data=c(0,0,1,0,1,1,0,1),ncol=2,byrow=TRUE);
  b <- matrix(data=c(2,2,3,2,3,3,2,3), ncol=2,byrow=TRUE);
  #--- build a PolySet from them
  polys <- appendPolys(NULL, a);
  polys <- appendPolys(polys, b);
  #--- print the result
  print (polys);
})
```

bcBathymetry *Data: Bathymetry Spanning British Columbia's Coast*

## Description

Bathymetry data spanning British Columbia's coast.

## Usage

```
data(bcBathymetry)
```

## Format

Three-element list: `x` = vector of horizontal grid line locations, `y` = vector of vertical grid line locations, `z` = (`x` by `y`) matrix containing water depths measured in meters. Positive values indicate distance below sea level and negative values above it.

`contour` and `contourLines` expect data in this format. `convCP` converts the output from `contourLines` into a PolySet.

## Note

In R, the data must be loaded using the `data` function.

## Source

Bathymetry data acquired from the Scripps Institution of Oceanography at the University of San Diego.

Using their online form, we requested bathymetry data for the complete `nepacLL` region. At forty megabytes, the data were not suitable for distribution in our mapping package. Therefore, we reduced the data to the range $-140° \leq x \leq -122°$ and $47° \leq y \leq 61°$.

## References

Smith, W.H.F. and Sandwell, D.T. (1997) Global seafloor topography from satellite altimetry and ship depth soundings. *Science* **277**, 1957–1962.
http://topex.ucsd.edu/WWW_html/mar_topo.html

## See Also

`contour`, `contourLines`, `convCP`, `nepacLL`, `nepacLLhigh`.

`calcArea` *Calculate the Areas of Polygons*

## Description

Calculate the areas of polygons found in a PolySet.

## Usage

```
calcArea (polys, rollup = 3)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to use. |
| `rollup` | level of detail in the results; 1 = PIDs only, by summing all the polygons with the same PID, 2 = outer contours only, by subtracting holes from their parent, and 3 = no roll-up. |

## Details

If `rollup` equals 1, the results contain an area for each unique `PID` only. When it equals 2, they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain areas for each unique (`PID`, `SID`).

Outer polygons have positive areas and inner polygons negative areas. When polygons are rolled up, the routine sums the positive and negative areas and consequently accounts for holes.

If the PolySet's `projection` attribute equals `"LL"`, the function projects the PolySet in UTM first. If the PolySet's `zone` attribute exists, it uses it for the conversion. Otherwise, it computes the mean longitude and uses that value to determine the zone. The longitude range of zone $i$ is $-186 + 6i^\circ < x \leq -180 + 6i^\circ$.

## Value

PolyData with columns `PID`, `SID` (*may be missing*), and `area`. If the projection equals `"LL"` or `"UTM"`, the units of area are square kilometres.

## See Also

`calcCentroid`, `calcLength`, `calcMidRange`, `calcSummary`, `locatePolys`.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language == "R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- convert LL to UTM so calculation makes sense
  attr(nepacLL, "zone") <- 9
```

```
  nepacUTM <- convUL(nepacLL)
  #--- calculate and print the areas
  print(calcArea(nepacUTM))
})
```

---

`calcCentroid` *Calculate the Centroids of Polygons*

---

## Description

Calculate the centroids of polygons found in a PolySet.

## Usage

```
calcCentroid (polys, rollup = 3)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to use. |
| `rollup` | level of detail in the results; `1` = `PID`s only, `2` = outer contours only, and `3` = no roll-up. When `rollup` equals `1` and `2`, the function appropriately adjusts for polygons with holes. |

## Details

If `rollup` equals `1`, the results contain a centroid for each unique `PID` only. When it equals `2`, they contain entries for outer contours only. Finally, setting it to `3` prevents roll-up, and they contain a centroid for each unique (`PID`, `SID`).

## Value

PolyData with columns `PID`, `SID` (*may be missing*), `X`, and `Y`.

## See Also

`calcArea`, `calcLength`, `calcMidRange`, `calcSummary`, `locateEvents`, `locatePolys`.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcCentroid(nepacLL[is.element(nepacLL$PID,c(33,39,47)),]))
})
```

calcConvexHull *Calculate the Convex Hull for a Set of Points*

## Description

Calculate the convex hull for a set of points.

## Usage

```
calcConvexHull (xydata, keepExtra=FALSE)
```

## Arguments

| | |
|---|---|
| xydata | a data frame with columns X and Y containing spatial coordinates. |
| keepExtra | logical: if TRUE, retain any additional columns from the input data frame xydata. |

## Details

This routine uses the function chull() in the package grDevices. By default, it ignores all columns other than X and Y; however, the user can choose to retain additional columns in xydata by specifying keepExtra=TRUE.

## Value

PolySet with columns PID, POS, X, Y, and additional columns in xydata if keepExtra=TRUE.

## See Also

addPoints, addPolys, calcArea, calcCentroid, calcMidRange, calcSummary, locateEvents, plotMap, plotPoints, plotPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  data(surveyData,envir=.PBSmapEnv)
  #--- plot the convex hull, and then plot the points
  plotMap(calcConvexHull(surveyData),col="moccasin")
  addPoints(surveyData,col="blue",pch=17,cex=.6)
  par(oldpar)
})
```

calcLength *Calculate the Length of Polylines*

## Description

Calculate the length of polylines found in a PolySet.

## Usage

```
calcLength (polys, rollup = 3, close = FALSE)
```

## Arguments

| | |
|---|---|
| polys | PolySet to use. |
| rollup | level of detail in the results; 1 = PIDs only, summing the lengths of each SID within each PID, and 3 = no roll-up. Note: rollup 2 has no meaning in this function and, if specified, will be reset to 3. |
| close | Boolean value; if TRUE, include the distance between each polygon's last and first vertex, if necessary. |

## Details

If rollup equals 1, the results contain an entry for each unique PID only. Setting it to 3 prevents roll-up, and they contain an entry for each unique (PID, SID).

If the projection attribute equals "LL", this routine uses Great Circle distances to compute the surface length of each polyline. In doing so, the algorithm simplifies Earth to a sphere.

If the projection attribute equals "UTM" or 1, this routine uses Pythagoras' Theorem to calculate lengths.

## Value

PolyData with columns PID, SID (*may be missing*), and length. If projection equals "UTM" or "LL", lengths are in kilometres. Otherwise, lengths are in the same unit as the input PolySet.

## See Also

calcArea, calcCentroid, calcMidRange, calcSummary, locatePolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate the perimeter of Vancouver Island
  print(calcLength(nepacLL[nepacLL$PID==33, ]))
})
```

calcMidRange *Calculate the Midpoint of the X/Y Ranges of Polygons*

## Description

Calculate the midpoint of the X/Y ranges of polygons found in a PolySet.

## Usage

```
calcMidRange (polys, rollup = 3)
```

## Arguments

| | |
|---|---|
| polys | PolySet to use. |
| rollup | level of detail in the results; 1 = PIDs only, 2 = outer contours only, and 3 = no roll-up. |

## Details

If rollup equals 1, the results contain a mean range for each unique PID only. When it equals 2, they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain a mean range for each unique (PID, SID).

## Value

PolyData with columns PID, SID (*may be missing*), X, and Y.

## See Also

calcArea, calcCentroid, calcLength, calcSummary.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcMidRange(nepacLL[is.element(nepacLL$PID,c(33,39,47)),]))
})
```

calcSummary *Apply Functions to Polygons in a PolySet*

## Description

Apply functions to polygons in a PolySet.

## Usage

```
calcSummary (polys, rollup = 3, FUN, ...)
```

## Arguments

| | |
|---|---|
| polys | PolySet to use. |
| rollup | level of detail in the results; 1 = PIDs only, by removing the SID column, and then passing each PID into FUN, 2 = outer contours only, by making hole SIDs equal to their parent's SID, and then passing each (PID, SID) into FUN, and 3 = no roll-up. |
| FUN | the function to apply; it must accept a vector and return a vector or scalar. |
| ... | optional arguments for FUN. |

## Details

If rollup equals 1, the results contain an entry for each unique PID only. When it equals 2, they contain entries for outer contours only. Finally, setting it to 3 prevents roll-up, and they contain an entry for each unique (PID, SID).

## Value

PolyData with columns PID, SID (*may be missing*), X, and Y. If FUN returns a vector of length greater than 1 (say $n$), names the columns X1, X2, ..., X$n$ and Y1, Y2, ..., Y$n$.

## See Also

calcArea, calcCentroid, calcConvexHull, calcLength, calcMidRange, combineEvents, findPolys, locateEvents, locatePolys, makeGrid, makeProps.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate and print the centroids for several polygons
  print(calcSummary(nepacLL[is.element(nepacLL$PID,c(33,39,47)),],
    rollup=3, FUN=mean))
})
```

calcVoronoi *Calculate the Voronoi (Dirichlet) Tesselation for a Set of Points*

## Description

Calculate the Voronoi (Dirichlet) tesselation for a set of points.

## Usage

```
calcVoronoi (xydata, xlim = NULL, ylim = NULL, eps = 1e-09, frac = 0.0001)
```

## Arguments

| | |
|---|---|
| xydata | a data frame with columns X and Y containing the points. |
| xlim | range of X-coordinates; a bounding box for the coordinates. |
| ylim | range of Y-coordinates; a bounding box for the coordinates. |
| eps | the value of epsilon used in testing whether a quantity is zero. |
| frac | used to detect duplicate input points, which meet the condition $\|x1 - x2\| <$ frac $\times$ (xmax $-$ xmin)and $\|y1 - y2\| <$ frac $\times$ (ymax $-$ ymin). |

## Details

This routine ignores all columns other than X and Y.

If the user leaves xlim and ylim unspecified, the function defaults to the range of the data with each extent expanded by ten percent of the range.

This function sets the attribute projection to 1 and the attribute zone to NULL as it assumes this projection in its calculations.

## Value

PolySet with columns PID, POS, X, and Y.

## See Also

addPoints, addPolys, calcArea, calcCentroid, calcConvexHull, calcMidRange, calcSummary, locateEvents, plotMap, plotPoints, plotPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData
  events <- as.EventData(data.frame(
    EID=1:200, X=rnorm(200), Y=rnorm(200)), projection=1)
  #--- calculate the Voronoi tesselation
  polys <- calcVoronoi(events)
  #--- create PolyData to color it based on area
```

```
  polyData <- calcArea(polys)
  names(polyData)[is.element(names(polyData), "area")] <- "Z"
  colSeq <- seq(0.4, 0.95, length=4)
  polyData <- makeProps(polyData,
    breaks=quantile(polyData$Z,c(0,.25,.5,.75,1)),
    propName="col", propVals=rgb(colSeq,colSeq,colSeq))
  #--- plot the tesselation
  plotMap(polys, polyProps=polyData)
  #--- plot the points
  addPoints(events, pch=19)
  par(oldpar)
})
```

---

`clipLines` *Clip a PolySet as Polylines*

---

## Description

Clip a PolySet, where each unique (`PID`, `SID`) describes a polyline.

## Usage

```
clipLines (polys, xlim, ylim, keepExtra = FALSE)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to clip. |
| `xlim` | range of X-coordinates. |
| `ylim` | range of Y-coordinates. |
| `keepExtra` | Boolean value; if `TRUE`, tries to carry forward any non-standard columns into the result. |

## Details

For each discrete polyline, the function does not connect vertices 1 and N. It recalculates the `POS` values for each vertex, saving the old values in a column named `oldPOS`. For new vertices, it sets `oldPOS` to `NA`.

## Value

PolySet containing the input data, with some points added or removed. A new column `oldPOS` records the original `POS` value for each vertex.

## See Also

`clipPolys`, `fixBound`.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a triangle to clip
  polys <- data.frame(PID=rep(1, 3), POS=1:3, X=c(0,1,0), Y=c(0,0.5,1))
  #--- clip the triangle in the X direction, and plot the results
  plotLines(clipLines(polys, xlim=c(0,.75), ylim=range(polys[, "Y"])))
  par(oldpar)
})
```

---

`clipPolys` *Clip a PolySet as Polygons*

---

## Description

Clip a PolySet, where each unique (`PID`, `SID`) describes a polygon.

## Usage

```
clipPolys (polys, xlim, ylim, keepExtra = FALSE)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to clip. |
| `xlim` | range of X-coordinates. |
| `ylim` | range of Y-coordinates. |
| `keepExtra` | Boolean value; if `TRUE`, tries to carry forward any non-standard columns into the result. |

## Details

For each discrete polygon, the function connects vertices 1 and N. It recalculates the `POS` values for each vertex, saving the old values in a column named `oldPOS`. For new vertices, it sets `oldPOS` to `NA`.

## Value

PolySet containing the input data, with some points added or removed. A new column `oldPOS` records the original `POS` value for each vertex.

## See Also

`clipLines`, `fixBound`.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a triangle that will be clipped
  polys <- data.frame(PID=rep(1, 3), POS=1:3, X=c(0,1,.5), Y=c(0,0,1))
  #--- clip the triangle in the X direction, and plot the results
  plotPolys(clipPolys(polys,xlim=c(0,.75),ylim=range(polys[,"Y"])),col=2)
  par(oldpar)
})
```

---

`closePolys` *Close a PolySet*

---

## Description

Close a PolySet of polylines to form polygons.

## Usage

```
closePolys (polys)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to close. |

## Details

Generally, run `fixBound` before this function. The ranges of a PolySet's `X` and `Y` columns define the boundary. For each discrete polygon, this function determines if the first and last points lie on a boundary. If both points lie on the same boundary, it adds no points. However, if they lie on different boundaries, it may add one or two corners to the polygon.

When the boundaries are adjacent, one corner will be added as follows:

- top boundary + left boundary implies add top-left corner;
- top boundary + right boundary implies add top-right corner;
- bottom boundary + left boundary implies add bottom-left corner;
- bottom boundary + right boundary implies add bottom-right corner.

When the boundaries are opposite, it first adds the corner closest to a starting or ending polygon vertex. This determines a side (left-right or bottom-top) that connects the opposite boundaries. Then, it adds the other corner of that side to close the polygon.

## Value

PolySet identical to `polys`, except for possible additional corner points.

## See Also

fixBound, fixPOS.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- 4 corners
  polys <- data.frame(
    PID = c(1, 1, 2, 2, 3, 3, 4, 4),
    POS = c(1, 2, 1, 2, 1, 2, 1, 2),
    X   = c(0, 1, 2, 3, 0, 1, 2, 3),
    Y   = c(1, 0, 0, 1, 2, 3, 3, 2))
  plotPolys(closePolys(polys), col=2)

  #--- 2 corners and 1 opposite
  polys <- data.frame(
    PID = c(1, 1, 2, 2, 3, 3, 3),
    POS = c(1, 2, 1, 2, 1, 2, 3),
    X   = c(0, 1, 0, 1, 5, 6, 1.5),
    Y   = c(1, 0, 2, 3, 0, 1.5, 3))
  plotPolys(closePolys(polys), col=2)
  par(oldpar)
})
```

---

combineEvents *Combine Measurements of Events*

---

## Description

Combine measurements associated with events that occur in the same polygon.

## Usage

```
combineEvents (events, locs, FUN, ..., bdryOK = TRUE)
```

## Arguments

| | |
|---|---|
| events | EventData with at least four columns (EID, X, Y, Z). |
| locs | LocationSet usually resulting from a call to findPolys. |
| FUN | a function that produces a scalar from a vector (e.g., mean, sum). |
| ... | optional arguments for FUN. |
| bdryOK | Boolean value; if TRUE, include boundary points. |

### Details

This function combines measurements associated with events that occur in the same polygon. Each event (EID) has a corresponding measurement Z. The locs data frame (usually output from findPolys) places events within polygons. Thus, each polygon (PID, SID) determines a set of events within it, and a corresponding vector of measurements Zv. The function returns FUN(Zv), a summary of measurements within each polygon.

### Value

PolyData with columns PID, SID (*if in* locs), and Z.

### See Also

findCells, findPolys, locateEvents, locatePolys, makeGrid, makeProps.

### Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create an EventData data frame: let each event have Z = 1
  events <- data.frame(EID=1:10, X=1:10, Y=1:10, Z=rep(1, 10))
  #--- example output from findPolys where 1 event occurred in the first
  #--- polygon, 3 in the second, and 6 in the third
  locs <- data.frame(EID=1:10,PID=c(rep(1,1),rep(2,3),rep(3,6)),Bdry=rep(0,10))
  #--- sum the Z column of the events in each polygon, and print the result
  print(combineEvents(events=events, locs=locs, FUN=sum))
})
```

---

## combinePolys *Combine Several Polygons into a Single Polygon*

---

### Description

Combine several polygons into a single polygon by modifying the PID and SID indices.

### Usage

```
combinePolys (polys)
```

### Arguments

| | |
|---|---|
| polys | PolySet with one or more polygons, each with possibly several components/holes. |

### Details

This function accepts a PolySet containing one or more polygons (PIDs), each with one or more components or holes (SIDs). The SID column need not exist in the input. The function combines these polygons into a single polygon by simply renumbering the PID and SID indices. The resulting PolySet contains a single PID (with the value 1) and uses the SID value to differentiate between polygons, their components, and holes.

## Value

PolySet, possibly with the addition of an SID column if it did not already exist. The function may also reorder columns such that PID, SID, POS, X and Y appear first, in that order.

## See Also

dividePolys

---

convCP *Convert Contour Lines into a PolySet*

---

## Description

Convert output from contourLines into a PolySet.

## Usage

```
convCP (data, projection = NULL, zone = NULL)
```

## Arguments

| | |
|---|---|
| data | contour line data, often from the contourLines function. |
| projection | optional projection attribute to add to the PolySet. |
| zone | optional zone attribute to add to the PolySet. |

## Details

data contains a list as described below. The contourLines function create a list suitable for the data argument.

A three-element list describes each contour. The named elements in this list include the scalar level, the vector x, and the vector y. Vectors x and y must have equal lengths. A higher-level list (data) contains one or more of these contours lists.

## Value

A list with two named elements PolySet and PolyData. The PolySet element contains a PolySet representation of the contour lines. The PolyData element links each contour line (PID, SID) with a level.

## See Also

contour, contourLines, convLP, makeTopography.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample data for the contourLines() function
  x <- seq(-0.5, 0.8, length=50);  y <- x
  z <- outer(x, y, FUN = function(x,y) { sin(2*pi*(x^2+y^2))^2; } )
  data <- contourLines(x, y, z, levels=c(0.2, 0.8))
  #--- pass that sample data into convCP()
  result <- convCP(data)
  #--- plot the result
  plotLines(result$PolySet, projection=1)
  print(result$PolyData)
  par(oldpar)
})
```

---

convDP *Convert EventData/PolyData into a PolySet*

---

## Description

Convert EventData/PolyData into a PolySet.

## Usage

```
convDP (data, xColumns, yColumns)
```

## Arguments

| | |
|---|---|
| data | PolyData or EventData. |
| xColumns | vector of X-column names. |
| yColumns | vector of Y-column names. |

## Details

This function expects data to contain several X- and Y-columns. For example, consider data with columns x1, y1, x2, and y2. Suppose xColumns = c("x1", "x2") and yColumns = c("y1", "y2"). The result will contain nrow(data) polygons. Each one will have two vertices, (x1, y1) and (x2, y2) and POS values 1 and 2, respectively. If data includes an SID column, so will the result.

If data contains an EID and not a PID column, the function uses the EIDs as PIDs.

If data contains both PID and EID columns, the function assumes it is PolyData and ignores the EID column.

## Value

PolySet with the same PIDs as those given in data. If data has an SID column, the result will include it.

## See Also

addPoints, plotPoints.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create sample PolyData
  polyData <- data.frame(PID=c(1, 2, 3),
     x1=c(1, 3, 5), y1=c(1, 3, 2),
     x2=c(1, 4, 5), y2=c(2, 4, 1),
     x3=c(2, 4, 6), y3=c(2, 3, 1))
  #--- print PolyData
  print(polyData)
  #--- make a PolySet from PolyData
  polys <- convDP(polyData,
    xColumns=c("x1", "x2", "x3"),
    yColumns=c("y1", "y2", "y3"))
  #--- print and plot the PolySet
  print(polys)
  plotLines(polys, xlim=c(0,7), ylim=c(0,5), col=2)
  par(oldpar)
})
```

---

# convLP *Convert Polylines into a Polygon*

---

## Description

Convert two polylines into a polygon.

## Usage

```
convLP (polyA, polyB, reverse = TRUE)
```

## Arguments

| | |
|---|---|
| polyA | PolySet containing a polyline. |
| polyB | PolySet containing a polyline. |
| reverse | Boolean value; if TRUE, reverse polyB's vertices. |

## Details

The resulting PolySet contains all the vertices from polyA in their original order. If reverse = TRUE, this function appends the vertices from polyB in the reverse order (nrow(polyB):1). Otherwise, it appends them in their original order. The PID column equals the PID of polyA. No SID column appears in the result. The resulting polygon is an exterior boundary.

## Value

PolySet with a single PID that is the same as polyA. The result contains all the vertices in polyA and polyB. It has the same projection and zone attributes as those in the input PolySets. If an input PolySet's attributes equal NULL, the function uses the other PolySet's. If the PolySet attributes conflict, the result's attribute equals NULL.

## See Also

addLines, appendPolys, closePolys, convCP, joinPolys, plotLines.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create two polylines
  polyline1 <- data.frame(PID=rep(1,2),POS=1:2,X=c(1,4),Y=c(1,4))
  polyline2 <- data.frame(PID=rep(1,2),POS=1:2,X=c(2,5),Y=c(1,4))
  #--- create two plots to demonstrate the effect of `reverse'
  par(mfrow=c(2, 1))
  plotPolys(convLP(polyline1, polyline2, reverse=TRUE), col=2)
  plotPolys(convLP(polyline1, polyline2, reverse=FALSE), col=3)
  par(oldpar)
})
```

---

# convUL *Convert Coordinates between UTM and Lon/Lat*

---

## Description

Convert coordinates between UTM and Lon/Lat.

## Usage

```
convUL (xydata, km=TRUE, southern=NULL)
```

## Arguments

| | |
|---|---|
| xydata | data frame with columns X and Y. |
| km | Boolean value; if TRUE, UTM coordinates within xydata are in kilometres; otherwise, metres. |
| southern | Boolean value; if TRUE, forces conversions from UTM to longitude/latitude to produce coordinates within the southern hemisphere. For conversions from UTM, this argument defaults to FALSE. For conversions from LL, the function determines southern from xydata. |

## Details

The object xydata must possess a projection attribute that identifies the current projection. If the data frame contains UTM coordinates, it must also have a zone attribute equal to a number between 1 and 60 (inclusive). If it contains geographic (longitude/latitude) coordinates and the zone attribute is missing, the function computes the mean longitude and uses that value to determine the zone. The longitude range of zone $i$ is $-186 + 6i^\circ < x \leq -180 + 6i^\circ$.

This function converts the X and Y columns of xydata from "LL" to "UTM" or vice-versa. If the data span more than **one** zone to the right or left of the intended central zone, the underlying algorithm may produce erroneous results. This limitation means that the user should use the most central zone of the mapped region, or allow the function to determine the central zone when converting from geographic to UTM coordinates. After the conversion, this routine adjusts the data frame's attributes accordingly.

## Value

A data frame identical to xydata, except that the X and Y columns contain the results of the conversion, and the projection attribute matches the new projection.

## Author(s)

Nicholas Boers, Dept. of Computer Science, Grant MacEwan University, Edmonton AB

## References

Ordnance Survey. (2010) A guide to coordinate systems in Great Britain. *Report D00659* **(v2.1)**. Southampton, UK.
http://www.ordnancesurvey.co.uk/oswebsite/gps/docs/A_Guide_to_Coordinate_Systems_in_Great_Britain.pdf.

## See Also

closePolys, fixBound.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data
  data(nepacLL,envir=.PBSmapEnv)
  #--- set the zone attribute
  #--- use a zone that is most central to the mapped region
  attr(nepacLL, "zone") <- 6
  #--- convert and plot the result
  nepacUTM <- convUL(nepacLL)
  plotMap(nepacUTM)
  par(oldpar)
})
```

## dividePolys *Divide a Single Polygon into Several Polygons*

### Description

Divide a single polygon (with several outer-contour components) into several polygons, a polygon for each outer contour, by modifying the PID and SID indices.

### Usage

```
dividePolys (polys)
```

### Arguments

| | |
|---|---|
| polys | PolySet with one or more polygons, each with possibly several components/holes. |

### Details

Given the input PolySet, this function renumbers the PID and SID indices so that each outer contour has a unique PID and is followed by all of its holes, identifying them with SIDs greater than one.

### Value

PolySet, possibly with the addition of an SID column if it did not already exist. The function may also reorder columns such that PID, SID, POS, X and Y appear first, in that order.

### See Also

combinePolys.

## EventData *EventData Objects*

### Description

PBS Mapping functions that expect EventData will accept properly formatted data frames in their place (see 'Details').

as.EventData attempts to coerce a data frame to an object with class EventData.

is.EventData returns TRUE if its argument is of class EventData.

### Usage

```
as.EventData(x, projection = NULL, zone = NULL)
is.EventData(x, fullValidation = TRUE)
```

### Arguments

| | |
|---|---|
| x | data frame to be coerced or tested. |
| projection | optional projection attribute to add to EventData, possibly overwriting an existing attribute. |
| zone | optional zone attribute to add to EventData, possibly overwriting an existing attribute. |
| fullValidation | Boolean value; if TRUE, fully test x. |

### Details

We define EventData as a data frame with at least three fields named (EID, X, Y). Conceptually, an EventData object describes events that take place at specific points (X, Y) in two-dimensional space. Additional fields specify measurements associated with these events. For example, in a fishery context EventData could describe fishing events associated with trawl tows, based on the fields:

- EID - fishing event (tow) identification number;
- X, Y - fishing location;
- Duration - length of time for the tow;
- Depth - average depth of the tow;
- Catch - biomass captured.

Like PolyData, EventData can have attributes projection and zone, which may be absent. Inserting the string "EventData" as the class attribute's first element alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

### Value

The as.EventData method returns an object with classes "EventData" and "data.frame", in that order.

### See Also

LocationSet, PolyData, PolySet.

---

## extractPolyData *Extract PolyData from a PolySet*

---

### Description

Extract PolyData from a PolySet. Columns for the PolyData include those other than PID, SID, POS, oldPOS, X, and Y.

### Usage

```
extractPolyData (polys)
```

### Arguments

| | |
|---|---|
| polys | PolySet to use. |

### Details

This function identifies the PolySet's extra columns and determines if those columns contain unique values for each (PID, SID). Where they do, the (PID, SID) will appear in the PolyData output with that unique value. Where they do not, the extra column will contain NAs for that (PID, SID).

### Value

PolyData with columns PID, SID, and any extra columns.

### See Also

makeProps, PolyData, PolySet.

### Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolySet with an extra column
  polys <- data.frame(PID = c(rep(1, 10), rep(2, 10)),
    POS = c(1:10, 1:10),
    X = c(rep(1, 10), rep(1, 10)),
    Y = c(rep(1, 10), rep(1, 10)),
    colour = (c(rep("green", 10), rep("red", 10))))
  #--- extract the PolyData
  print(extractPolyData(polys))
})
```

---

## findCells *Find the Grid Cells that Contain Events*

---

### Description

Find the grid cells in a PolySet that contain events specified in EventData. Similar to findPolys, except this function requires a PolySet resulting from makeGrid. This restriction allows this function to calculate the result with greater efficiency.

### Usage

```
findCells (events, polys)
```

### Arguments

| | |
|---|---|
| events | EventData to use. |
| polys | PolySet to use. |

## Details

The resulting data frame, a LocationSet, contains the columns EID, PID, SID (*if in* polys), and Bdry, where an event (EID) occurs in a polygon (PID, SID). The Boolean variable Bdry indicates whether an event lies on a polygon's edge. Note that if an event lies properly outside of all the polygons, then a record with (EID, PID, SID) does not occur in the output. It may happen, however, that an event occurs in multiple polygons (i.e., on two or more boundaries). Thus, the same EID can occur more than once in the output.

## Value

LocationSet that links events with polygons.

## See Also

combineEvents, findPolys, locateEvents, locatePolys, LocationSet, makeGrid.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData: points in a diagonal line
  events <- data.frame(EID=1:11, X=seq(0, 2, length=11),
     Y=seq(0, 2, length=11))
  events <- as.EventData(events, projection=1);
  #--- create a PolySet (a grid)
  polys <- makeGrid (x=seq(0, 2, by=0.50), y=seq(0, 2, by=0.50), projection=1)
  #--- show a picture
  plotPolys(polys, xlim=range(polys$X)+c(-0.1, 0.1),
    ylim=range(polys$Y)+c(-0.1, 0.1), projection=1)
  addPoints(events, col=2)
  #--- run findCells and print the results
  fc <- findCells(events, polys)
  fc <- fc[order(fc$EID, fc$PID, fc$SID), ]
  fc$label <- paste(fc$PID, fc$SID, sep=", ")
  print (fc)
  #--- add labels to the graph
  addLabels(as.PolyData(fc[!duplicated(paste(fc$PID,fc$SID)), ],
    projection=1), placement="CENTROID",
    polys=as.PolySet(polys, projection=1), col=4)
  par(oldpar)
})
```

---

# findPolys *Find the Polygons that Contain Events*

---

## Description

Find the polygons in a PolySet that contain events specified in EventData.

## Usage

```
findPolys (events, polys, maxRows = 1e+05)
```

## Arguments

| | |
|---|---|
| events | EventData to use. |
| polys | PolySet to use. |
| maxRows | estimated maximum number of rows in the output LocationSet. |

## Details

The resulting data frame, a LocationSet, contains the columns EID, PID, SID (*if in* polys), and Bdry, where an event (EID) occurs in a polygon (PID, SID) and SID does not correspond to an inner boundary. The Boolean variable Bdry indicates whether an event lies on a polygon's edge. Note that if an event lies properly outside of all the polygons, then a record with (EID, PID, SID) does not occur in the output. It may happen, however, that an event occurs in multiple polygons. Thus, the same EID can occur more than once in the output.

## Value

LocationSet that links events with polygons.

## See Also

combineEvents, findCells, locateEvents, locatePolys, LocationSet, makeGrid.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create some EventData: a column of points at X = 0.5
  events <- data.frame(EID=1:10, X=.5, Y=seq(0, 2, length=10))
  events <- as.EventData(events, projection=1)
  #--- create a PolySet: two squares with the second above the first
  polys <- data.frame(PID=c(rep(1, 4), rep(2, 4)), POS=c(1:4, 1:4),
    X=c(0, 1, 1, 0, 0, 1, 1, 0),
    Y=c(0, 0, 1, 1, 1, 1, 2, 2))
  polys <- as.PolySet(polys, projection=1)
  #--- show a picture
  plotPolys(polys, xlim=range(polys$X)+c(-0.1, 0.1),
    ylim=range(polys$Y)+c(-0.1, 0.1), projection=1);
  addPoints(events, col=2);
  #--- run findPolys and print the results
  print(findPolys(events, polys))
  par(oldpar)
})
```

`fixBound` *Fix the Boundary Points of a PolySet*

## Description

The ranges of a PolySet's `X` and `Y` columns define its boundary. This function fixes a PolySet's vertices by moving vertices near a boundary to the actual boundary.

## Usage

```
fixBound (polys, tol)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to fix. |
| `tol` | vector (length 1 or 2) specifying a percentage of the ranges to use in defining *near* to a boundary. If `tol` has two elements, the first specifies the tolerance for the x-axis and the second the y-axis. If it has only one element, the function uses the same tolerance for both axes. |

## Details

When moving vertices to a boundary, the function moves them strictly horizontally or vertically, as appropriate.

## Value

PolySet identical to the input, except for possible changes in the `X` and `Y` columns.

## See Also

`closePolys`, `fixPOS`, `isConvex`, `isIntersecting`, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- set up a long horizontal and long vertical line to extend the plot's
  #--- limits, and then try fixing the bounds of a line in the top-left
  #--- corner and a line in the bottom-right corner
  polys <- data.frame(PID=c(1, 1, 2, 2,  3, 3, 4, 4),
    POS=c(1, 2, 1, 2,  1, 2, 1, 2),
    X = c(0, 10, 5, 5, 0.1, 4.9, 5.1, 9.9),
    Y = c(5, 5, 0, 10, 5.1, 9.9, 0.1, 4.9))
  polys <- fixBound(polys, tol=0.0100001)
  plotLines(polys)
  par(oldpar)
})
```

fixPOS *Fix the POS Column of a PolySet*

## Description

Fix the POS column of a PolySet by recalculating it using sequential integers.

## Usage

```
fixPOS (polys, exteriorCCW = NA)
```

## Arguments

| | |
|---|---|
| polys | PolySet to fix. |
| exteriorCCW | Boolean value; if TRUE, orders exterior polygon vertices in a counter-clockwise direction. If FALSE, orders them in a clockwise direction. If NA, maintains their original order. |

## Details

This function recalculates the POS values of each (PID, SID) as either 1 to N or N to 1, depending on the order of POS (ascending or descending) in the input data. POS values in the input must be properly ordered (ascending or descending), but they may contain fractional values. For example, POS = 2.5 might correspond to a point manually added between POS = 2 and POS = 3. If exteriorCCW = NA, all other columns remain unchanged. Otherwise, it orders the X and Y columns according to exteriorCCW.

## Value

PolySet with the same columns as the input, except for possible changes to the POS, X, and Y columns.

## See Also

closePolys, fixBound, isConvex, isIntersecting, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolySet with broken POS numbering
  polys <- data.frame(PID = c(rep(1, 10), rep(2, 10)),
    POS = c(seq(2, 10, length = 10), seq(10, 2, length = 10)),
    X = c(rep(1, 10), rep(1, 10)),
    Y = c(rep(1, 10), rep(1, 10)))
  #--- fix the POS numbering
  polys <- fixPOS(polys)
  #--- print the results
  print(polys)
})
```

## importEvents *Import EventData from a Text File*

### Description

Import a text file and convert into EventData.

### Usage

```
importEvents(EventData, projection=NULL, zone=NULL)
```

### Arguments

| | |
|---|---|
| EventData | filename of EventData text file. |
| projection | optional projection attribute to add to EventData. |
| zone | optional zone attribute to add to EventData. |

### Value

An imported EventData.

### See Also

importPolys, importLocs, importGSHHS, importShapefile

## importGSHHS *Import Data from a GSHHS Database*

### Description

Import data from a GSHHS database and convert data into a PolySet with a PolyData attribute.

### Usage

```
importGSHHS(gshhsDB, xlim, ylim, maxLevel=4, n=0)
```

### Arguments

| | |
|---|---|
| gshhsDB | path name to binary GSHHS database. If unspecified, looks for gshhs_f.b in the root of the **PBSmapping** library directory. |
| xlim | range of X-coordinates (for clipping). The range should be between 0 and 360. |
| ylim | range of Y-coordinates (for clipping). |
| maxLevel | maximum level of polygons to import: 1 (land), 2 (lakes on land), 3 (islands in lakes), or 4 (ponds on islands); ignored when importing lines. |
| n | minimum number of vertices that must exist in a line/polygon in order for it to be imported. |

## Details

This routine requires a binary GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database file. The GSHHS database has been released in the public domain and may be downloaded from
http://www.soest.hawaii.edu/pwessel/gshhg/.
At the time of writing, the most recent database is gshhg-bin-2.2.2.zip.

The database gshhg-bin-2.2.2.zip contains geographical coordinates for shorelines (gshhs), rivers (wbd_rivers), and borders (wdb_borders). The latter two come from World DataBank II (WDBII):
http://meta.wikimedia.org/wiki/Geographical_data#CIA_World_DataBank_II_and_derivates
The five resolutions available are:
full (f), high (h), intermediate (i), low (l), and coarse (c).

This routine returns a PolySet object with an associated PolyData attribute. The attribute contains four fields: (a) PID, (b) SID, (c) Level, and (d) Source. Each record corresponds to a line/polygon in the PolySet. The Level indicates the line's/polygon's level (1=land, 2=lake, 3=island, 4=pond). The Source identifies the data source (1=WVS, 0=CIA (WDBII)).

## Value

A PolySet with a PolyData attribute.

## Author(s)

Nicholas Boers, Dept. of Computer Science, Grant MacEwan University, Edmonton AB

## See Also

importEvents, importLocs, importPolys, importShapefile

## Examples

```
## Not run:
pbsfun = function(ex=1) {
  switch(ex, {
  #--- EXAMPLE 1
  #--- set some limits appropriate for a map of Canada
  limits <- list(x = c(216.0486, 307.1274), y = c(42.87209, 77.35183))
  #--- extract data from the GSHHS binary files; you will need to download
  #--- these files from http://www.soest.hawaii.edu/pwessel/gshhg/
  #--- and place them in an appropriate location
  polys <- importGSHHS ("./gshhg-bin-2.2.2/gshhs_l.b",
                  xlim=limits$x, limits$y, maxLevel=4)
  rivers <- importGSHHS ("./gshhg-bin-2.2.2/wdb_rivers_i.b",
                        xlim=limits$x, limits$y)
  borders <- importGSHHS ("./gshhg-bin-2.2.2/wdb_borders_i.b",
                         xlim=limits$x, limits$y)
  #--- create a PNG for the output
  png ("./Canada.png", width=1600, height=1200, pointsize=24)
  #--- plot the polygons, river, and then borders
  plotMap (polys, plt=c(.05,.99,.075,.99), col="moccasin", bg="skyblue")
```

```
  addLines (rivers, col="lightblue")
  addLines (borders, col="red")
  #--- close the output file
  dev.off ()
  },{
  #--- EXAMPLE 2
  #--- clip out Manitoulin Island area which includes all four levels
  polys <- importGSHHS ("./gshhg-bin-2.2.2/gshhs_f.b",
                        xlim=c(276, 279), ylim=c(45.3, 46.5), maxLevel=4)
  #--- plot the map and add a label
  plotMap (polys, col="beige", bg="lightblue");
  text (-82.08, 45.706, "Manitoulin Isl")
  })
  invisible()
}
pbsfun(1); pbsfun(2)

## End(Not run)
```

---

`importLocs` *Import LocationSet from a text file*

---

## Description

Import a text file and convert into a LocationSet.

## Usage

```
importLocs(LocationSet)
```

## Arguments

| | |
|---|---|
| `LocationSet` | filename of LocationSet text file. |

## Value

An imported LocationSet.

## See Also

`importPolys`, `importEvents`, `importGSHHS`, `importShapefile`

## importPolys *Import PolySet from a text file*

### Description

Import a text file and convert into a PolySet with optional PolyData attribute.

### Usage

```
importPolys(PolySet, PolyData=NULL, projection=NULL, zone=NULL)
```

### Arguments

| | |
|---|---|
| PolySet | filename of PolySet text file. |
| PolyData | optional filename of PolyData text file. |
| projection | optional projection attribute to add to EventData. |
| zone | optional zone attribute to add to EventData. |

### Value

An imported PolySet with optional PolyData attribute.

### See Also

importEvents, importLocs, importGSHHS, importShapefile

## importShapefile *Import an ESRI Shapefile*

### Description

Import an ESRI shapefile (.shp) into either a PolySet or EventData.

### Usage

```
importShapefile (fn, readDBF=TRUE, projection=NULL, zone=NULL,
     placeholes=FALSE, minverts=3)
```

## Arguments

| | |
|---|---|
| fn | file name of the shapefile to import; specifying the extension is optional. |
| readDBF | Boolean value; if TRUE, it also imports the .dbf (a database containing the feature attributes) associated with the shapefile. |
| projection | optional projection attribute to override the internally derived value. |
| zone | optional zone attribute to override the default value of NULL. |
| placeholes | logical: if TRUE then for every PID identify solids and holes, and place holes under appropriate solids. |
| minverts | minimum number of vertices required for a polygon representing a hole to be retained (does not affect solids). |

## Details

This routine imports an ESRI shapefile (.shp) into either a PolySet or EventData, depending on the type of shapefile. It supports types 1 (Point), 3 (PolyLine), and 5 (Polygon) and imports type 1 into EventData and types 3 and 5 into a PolySet. In addition to the shapefile (.shp), it requires the related index file (.shx).

If a database containing feature attributes (.dbf) exists, it also imports this database by default. For EventData, it binds the database columns to the EventData object. For a PolySet, it saves the database in a PolyData object and attaches that object to the PolySet in an attribute named "PolyData".

If a .prj file exists, this information is attached as an attribute. If the first 3 characters are 'GEO', then a geographic projection is assumed and projection="LL". If the first 4 characters are 'PROJ', and 'UTM' occurs elsewhere in the string, then the Universal Transverse Mercator projection is assumed and projection="UTM". Otherwise, projection=1.

If an .xml file exists, this information is attached as an attribute.

Shapes of numeric shape type 5 exported from **ArcView** in geographic projection identify solids as polygons with vertices following a clockwise path and holes as polygons that follow a counterclockwise path. Unfortuantely, either the export from **ArcView** or the import using a C-routine from the package **maptools** often does not report solids followed by their holes. We employ a new R function placeHoles to do this for us. Ideally, this routine should be rendered in C, but for now we use this function if the user sets the argument placeholes=TRUE. Depending on the size and complexity of your shapefile, the computation may take a while.

## Value

For points, EventData with columns EID, X, and Y, possibly with other columns from the attribute database. For polylines and polygons, a PolySet with columns PID, SID, POS, X, Y and attribute projection. Other attributes that may or may not be attached: parent.child (boolean vector from original input), shpType (numeric shape type: 1, 3, or 5), prj (projection information from .prj file, xml (metadata from an .xml file), PolyData (data from the attribute database .dbf), and zone (UTM zone).

## See Also

importGSHHS, importEvents, importLocs, importPolys, placeHoles
In the package **sp**, see the function point.in.polygon

---

isConvex *Determine Whether Polygons are Convex*

---

## Description

Determine whether polygons found in a PolySet are convex.

## Usage

```
isConvex (polys)
```

## Arguments

| | |
|---|---|
| polys | PolySet to use. |

## Details

Convex polygons do not self-intersect. In a convex polygon, only the first and last vertices may share the same coordinates (i.e., the polygons are optionally closed).

The function does not give special consideration to holes. It returns a value for each unique (PID, SID), regardless of whether a contour represents a hole.

## Value

PolyData with columns PID, SID (*may be missing*), and convex. Column convex contains Boolean values.

## See Also

isIntersecting, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- calculate then print the polygons that are convex
  p <- isConvex(nepacLL);
  #--- nepacLL actually contains no convex polygons
  print(p[p$convex,])
})
```

`isIntersecting` *Determine Whether Polygons are Self-Intersecting*

## Description

Determine whether polygons found in a PolySet are self-intersecting.

## Usage

```
isIntersecting (polys, numericResult = FALSE)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to use. |
| `numericResult` | Boolean value; if `TRUE`, returns the number of intersections. |

## Details

When `numericResult = TRUE`, this function counts intersections as the algorithm processes them. It counts certain types (i.e., those involving vertices and those where an edge retraces over an edge) more than once.

The function does not give special consideration to holes. It returns a value for each unique (`PID`, `SID`), regardless of whether a contour represents a hole.

## Value

PolyData with columns `PID`, `SID` (*may be missing*), and `intersecting`. If `numericResult` is `TRUE`, `intersecting` contains the number of intersections. Otherwise, it contains a Boolean value.

## See Also

`isConvex`, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
  data(nepacLL,envir=.PBSmapEnv)
  #--- calculate then print the polygons that are self-intersecting
  p <- isIntersecting(nepacLL, numericResult = FALSE)
  print(p[p$intersecting,])
})
```

joinPolys *Join One or Two PolySets using a Logic Operation*

## Description

Join one or two PolySets using a logic operation.

## Usage

```
joinPolys(polysA,polysB=NULL,operation="INT")
```

## Arguments

| | |
|---|---|
| polysA | PolySet to join. |
| polysB | optional second PolySet with which to join. |
| operation | one of "DIFF", "INT", "UNION", or "XOR", representing difference, intersection, union, and exclusive-or, respectively. |

## Details

This function now (Mar. 2013) interfaces with the Clipper library (http://www.angusj.com/delphi/clipper.php) developed by Angus Johnson. Previously, it interfaced with the General Polygon Clipper library (http://www.cs.man.ac.uk/aig/staff/alan/software/) by Alan Murta at the University of Manchester. We keep this historic reference to GPC because joinPolys remains faithful to Murta's definition of a generic polygon, which we describe below.

Murta (2004) defines a *generic polygon* (or *polygon set*) as zero or more disjoint boundaries of arbitrary configuration. He relates a *boundary* to a contour, where each may be convex, concave or self-intersecting. In a PolySet, the polygons associated with each unique PID loosely correspond to a generic polygon, as they can represent both inner and outer boundaries. Our use of the term *generic polygon* includes the restrictions imposed by a PolySet. For example, the polygons for a given PID cannot be arranged arbitrarily.

If polysB is NULL, this function sequentially applies the operation between the generic polygons in polysA. For example, suppose polysA contains three generic polygons (A, B, C). The function outputs the PolySet containing ((A op B) op C).

If polysB is not NULL, this function applies operation between each generic polygon in polysA and each one in polysB. For example, suppose polysA contains two generic polygons (A, B) and polysB contains two generic polygons (C, D). The function's output is the concatenation of A *op* C, B *op* C, A *op* D, B *op* D, with PIDs 1 to 4, respectively. Generally there are $n$ times $m$ comparisons, where $n$ = number of polygons in polysA and $m$ = number of polygons in polysB. If polysB contains only one generic polygon, the function maintains the PIDs from polysA. It also maintains them when polysA contains only one generic polygon and the operation is difference. Otherwise, if polysA contains only one generic polygon, it maintains the PIDs from polysB.

## Value

If polysB is NULL, the resulting PolySet contains a single generic polygon (one PID), possibly with several components (SIDs). The function recalculates the PID and SID columns.

If polysB is not NULL, the resulting PolySet contains one or more generic polygons (PIDs), each with possibly several components (SIDs). The function recalculates the SID column, and depending on the input, it may recalculate the PID column.

## References

Murta, A. (2004) *A General Polygon Clipping Library*. Accessed: July 29, 2004.
http://www.cs.man.ac.uk/aig/staff/alan/software/gpc.html

## See Also

addPolys, appendPolys, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotMap, plotPoints, thickenPolys, thinPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- create a triangle to use in clipping
  polysB <- data.frame(PID=rep(1, 3), POS=1:3,
    X=c(-127.5, -124.5, -125.6), Y = c(49.2, 50.3, 48.6))
  #--- intersect nepacLL with the single polygon, and plot the result
  plotMap(joinPolys(nepacLL, polysB), col=5)
  #--- add nepacLL in a different line type to emphasize the intersection
  addPolys(nepacLL, border=2, lty=8, density=0)
  par(oldpar)
})
```

---

# locateEvents *Locate Events on the Current Plot*

---

## Description

Locate events on the current plot (using the locator function).

## Usage

```
locateEvents (EID, n = 512, type = "p", ...)
```

### Arguments

| | |
|---|---|
| EID | vector of event IDs (*optional*). |
| n | maximum number of events to locate. |
| type | one of "n", "p", "l", or "o". If "p" or "o", then the points are plotted; if "l" or "o", then the points are joined by lines. |
| ... | additional par parameters for the locator function. |

### Details

This function allows its user to define events with mouse clicks on the current plot via the locator function. The arguments n and type are the usual parameters of the locator function. If EID is not missing, then n = length(EID).

On exit from locator, suppose the user defined *m* events. If EID was missing, then the output data frame will contain *m* events. However, if EID exists, then the output data frame will contain length(EID) events, and both X and Y will be NA for events EID[(*m*+1):n]. The na.omit function can remove rows with NAs.

### Value

EventData with columns EID, X, and Y, and projection attribute equal to the map's projection. The function does not set the zone attribute.

### See Also

addPoints, combineEvents, convDP, EventData, findCells, findPolys, plotPoints.

### Examples

```
#--- define five events on the current plot, numbering them 10 to 14
## Not run: events <- locateEvents(EID = 10:14)
```

---

## locatePolys *Locate Polygons on the Current Plot*

---

### Description

Locate polygons on the current plot (using the locator function).

### Usage

```
locatePolys (pdata, n = 512, type = "o", ...)
```

### Arguments

| | |
|---|---|
| pdata | PolyData (*optional*) with columns PID and SID (*optional*), with two more optional columns n and type. |
| n | maximum number of points to locate. |
| type | one of "n", "p", "l", or "o". If "p" or "o", then the points are plotted; if "l" or "o", then the points are joined by lines. |
| ... | additional par parameters for the locator function. |

### Details

This function allows its user to define polygons with mouse clicks on the current plot via the locator function. The arguments n and type are the usual parameters for the locator function, but the user can specify them for each individual (PID, SID) in a pdata object.

If a pdata object exists, the function ignores columns other than PID, SID, n, and type. If pdata includes n, then an outer boundary has n > 0 and an inner boundary has n < 0.

On exit from locator, suppose the user defined $m$ vertices for a given polygon. For that polygon, the X and Y columns will contain NAs where POS = ($m$+1):n for outer-boundaries and POS = (|n|-$m$):1 for inner-boundaries. The na.omit function can remove rows with NAs.

If a pdata object does not exist, the output contains only one polygon with a PID equal to 1. One inner-boundary polygon (POS goes from n to 1) can be generated by supplying a negative n.

If type = "o" or type = "l", the function draws a line connecting the last and first vertices.

### Value

PolySet with projection attribute equal to the map's projection. The function does not set the zone attribute.

### See Also

addPolys, appendPolys, clipPolys, closePolys, findCells, findPolys, fixPOS, joinPolys, plotMap, plotPolys, thickenPolys, thinPolys.

### Examples

```
#--- define one polygon with up to 5 vertices on the current plot
## Not run: polys <- locatePolys(n = 5)
```

---

## LocationSet *LocationSet Objects*

---

### Description

PBS Mapping functions that expect LocationSet's will accept properly formatted data frames in their place (see 'Details').

as.LocationSet attempts to coerce a data frame to an object with class LocationSet.

is.LocationSet returns TRUE if its argument is of class LocationSet.

### Usage

```
as.LocationSet(x)
is.LocationSet(x, fullValidation = TRUE)
```

### Arguments

| | |
|---|---|
| x | data frame to be coerced or tested. |
| fullValidation | Boolean value; if TRUE, fully test x. |

### Details

A PolySet can define regional boundaries for drawing a map, and EventData can give event points on the map. Which events occur in which regions? Our function findPolys resolves this problem. The output lies in a LocationSet, a data frame with three or four columns (EID, PID, SID, Bdry), where SID may be missing. One row in a LocationSet means that the event EID occurs in the polygon (PID, SID). The boundary (Bdry) field specifies whether (Bdry=T) or not (Bdry=F) the event lies on the polygon boundary. If SID refers to an inner polygon boundary, then EID occurs in (PID, SID) only if Bdry=T. An event may occur in multiple polygons. Thus, the same EID can occur in multiple records. If an EID does not fall in any (PID, SID), or if it falls within a hole, it does not occur in the output LocationSet. Inserting the string "LocationSet" as the first element of a LocationSet's class attribute alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

### Value

The as.LocationSet method returns an object with classes "LocationSet" and "data.frame", in that order.

### See Also

EventData, PolyData, PolySet.

---

## makeGrid *Make a Grid of Polygons*

---

### Description

Make a grid of polygons, using PIDs and SIDs according to the input arguments.

### Usage

```
makeGrid(x,y,byrow=TRUE,addSID=TRUE,projection=NULL,zone=NULL)
```

## Arguments

| | |
|---|---|
| x | vector of X-coordinates (of length $m$). |
| y | vector of Y-coordinates (of length $n$). |
| byrow | Boolean value; if TRUE, increment PID along X. |
| addSID | Boolean value; if TRUE, include an SID column in the resulting PolySet. |
| projection | optional projection attribute to add to the PolySet. |
| zone | optional zone attribute to add to the PolySet. |

## Details

This function makes a grid of polygons, labeling them according to byrow and addSID. In the following description, the variables $i$ and $j$ indicate column and row numbers, respectively, where the lower-left cell of the grid is (1, 1).

- byrow = TRUE and addSID = FALSE implies PID = $i + (j-1) \times (m-1)$
- byrow = FALSE and addSID = FALSE implies PID = $j + (i-1) \times (n-1)$
- byrow = TRUE and addSID = TRUE implies PID = $i$, SID = $j$
- byrow = FALSE and addSID = TRUE implies PID = $j$, SID = $i$

## Value

PolySet with columns PID, SID (*if* addSID = TRUE), POS, X, and Y. The PolySet is a set of rectangular grid cells with vertices:
$(x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1})$.

## See Also

addPolys, clipPolys, combineEvents, findCells, findPolys, PolySet, thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- make a 10 x 10 grid
  polyGrid <- makeGrid(x=0:10, y=0:10)
  #--- plot the grid
  plotPolys(polyGrid, density=0, projection=1)
  par(oldpar)
})
```

makeProps *Make Polygon Properties*

## Description

Append a column for a polygon property (e.g., `border` or `lty`) to PolyData based on measurements in the PolyData's `Z` column.

## Usage

```
makeProps(pdata,breaks,propName="col",propVals=1:(length(breaks)-1))
```

## Arguments

| | |
|---|---|
| `pdata` | PolyData with a `Z` column. |
| `breaks` | either a vector of cut points or a scalar denoting the number of intervals that `Z` is to be cut into. |
| `propName` | name of the new column to append to `pdata`. |
| `propVals` | vector of values to associate with `Z` breaks. |

## Details

This function acts like the `cut` function to produce PolyData suitable for the `polyProps` plotting argument (see `addLabels`, `addLines`, `addPoints`, `addPolys`, `addStipples`, `plotLines`, `plotMap`,`plotPoints`, and `plotPolys`). The `Z` column of `pdata` is equivalent to the data vector `x` of the `cut` function.

## Value

PolyData with the same columns as `pdata` plus an additional column `propName`.

## See Also

`addLabels`, `addLines`, `addPoints`, `addPolys`, `addStipples`, `plotLines`, `plotMap`, `plotPoints`, `plotPolys`, PolyData, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- create a PolyData object
  pd <- data.frame(PID=1:10, Z=1:10)

  #--- using 3 intervals, create a column named `col' and populate it with
  #--- the supplied values
  makeProps(pdata=pd, breaks=3, propName="col", propVals=c(1:3))
})
```

makeTopography *Make Topography Data From Freely Available Online Data*

## Description

Make topography data suitable for the contour and contourLines functions using freely available global seafloor topography data.

## Usage

```
makeTopography (dat, digits=2, func=NULL)
```

## Arguments

| | |
|---|---|
| dat | data frame with three optionally-named columns: X, Y, and Z. The columns must appear in that order. |
| digits | integer indicating the precision to be used by the function round on (X,Y) values. |
| func | function to summarize Z if (X,Y) points are duplicated. Defaults to mean() if no function is specified. |

## Details

Data obtained through the acquisition form at http://topex.ucsd.edu/cgi-bin/get_data.cgi is suitable for this function. read.table will import its ASCII files into R/S, creating the data argument for this function.

When creating data for regions with longitude values spanning -180° to 0°, consider subtracting 360 from the result's X coordinates (x).

When creating bathymetry data, consider negating the result's elevations (z) to give depths positive values.

Combinations of (X,Y) do not need to be complete (z[x,y]=NA) or unique (z[x,y]=func(Z[x,y])).

## Value

List with elements x, y, and z. x and y are vectors, while z is a matrix with rownames x and colnames y. contour and contourLines expect data conforming to this list format.

## See Also

graphics::contour, grDevices::contourLines, convCP.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- Example 1: Sample data frame and conversion.
  file <- data.frame(X=c(1,1,2,2),Y=c(3,4,3,4),Z=c(5,6,7,8))
  print(makeTopography(file))

  #--- Example 2: Aleutian Islands bathymetry
  isob <- c(100,500,1000,2500,5000)
  icol <- rgb(0,0,seq(255,100,len=length(isob)),max=255)
  afile <- paste(system.file(package="PBSmapping"),
    "/Extra/aleutian.txt",sep="")
  aleutian <- read.table(afile, header=FALSE, col.names=c("x","y","z"))
  aleutian$x <- aleutian$x - 360
  aleutian$z <- -aleutian$z
  alBathy <- makeTopography(aleutian)
  alCL <- contourLines(alBathy,levels=isob)
  alCP <- convCP(alCL)
  alPoly <- alCP$PolySet
  attr(alPoly,"projection") <- "LL"
  plotMap(alPoly,type="n")
  addLines(alPoly,col=icol)
  data(nepacLL,envir=.PBSmapEnv)
  addPolys(nepacLL,col="gold")
  legend(x="topleft",bty="n",col=icol,lwd=2,legend=as.character(isob))
  par(oldpar)
})
```

---

# nepacLL *Data: Shoreline of the NE Pacific Ocean (Normal Resolution)*

---

## Description

PolySet of polygons for the northeast Pacific Ocean shoreline.

## Usage

```
data(nepacLL)
```

## Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinate, and `Y` = latitude coordinate. Attributes: `projection = "LL"`.

## Note

In R, the data must be loaded using the `data` function.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database gshhs_h.b.
Download from http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html

```
nepacLL <-importGSHHS("gshhs_h.b", xlim=c(-190,-110), ylim=c(34,72),
                      level=1, n=15, xoff=-360)
```

## References

Wessel, P. and Smith, W.H.F. (1996) A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research* **101**, 8741–8743.
http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html

## See Also

Data: nepacLLhigh, worldLL, worldLLhigh, bcBathymetry

importGSHHS, addPolys, clipPolys, plotPolys, plotMap, thickenPolys, thinPolys

---

# nepacLLhigh *Data: Shoreline of the NE Pacific Ocean (High Resolution)*

---

## Description

PolySet of polygons for the northeast Pacific Ocean shoreline.

## Usage

```
data(nepacLLhigh)
```

## Format

Data frame consisting of 4 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, and Y = latitude coordinate. Attributes: projection = "LL".

## Note

In R, the data must be loaded using the data function.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database gshhs_f.b.
Download from http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html

```
nepacLLhigh <-importGSHHS("gshhs_f.b", xlim=c(-190,-110),
                 ylim=c(34,72), level=1, n=0, xoff=-360)
nepacLLhigh <- thinPolys(nepacLLhigh, tol=0.1, filter=3)
```

## References

Wessel, P. and Smith, W.H.F. (1996) A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research* **101**, 8741–8743.
http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html

## See Also

Data: nepacLL, worldLL, worldLLhigh, bcBathymetry

importGSHHS, addPolys, clipPolys, plotPolys, plotMap, thickenPolys, thinPolys

---

PBSmapping *PBS Mapping: Draw Maps and Implement Other GIS Procedures*

---

## Description

This software has evolved from fisheries research conducted at the Pacific Biological Station (PBS) in Nanaimo, British Columbia, Canada. It extends the R language to include two-dimensional plotting features similar to those commonly available in a Geographic Information System (GIS). Embedded C code speeds algorithms from computational geometry, such as finding polygons that contain specified point events or converting between longitude-latitude and Universal Transverse Mercator (UTM) coordinates. It includes data for a global shoreline and other data sets in the public domain.

For a complete user's guide, see the file PBSmapping-UG.pdf in the R directory .../library/PBSmapping/doc.

PBSmapping includes 10 demos that appear as figures in the User's Guide. To see them, run the function .PBSfigs().

More generally, a user can view all demos available from locally installed packages with the function runDemos() in our related (and recommended) package PBSmodelling.

---

PBSprint *Specify Whether to Print Summaries*

---

## Description

Specify whether PBS Mapping should print object summaries or not. If not, data objects are displayed as normal.

## Usage

```
PBSprint
```

### Details

If PBSprint = TRUE, the mapping software will print summaries rather than the data frames for EventData, LocationSet, PolyData, and PolySet objects. If PBSprint = FALSE, it will print the data frames.

This variable's default value is FALSE.

### Value

TRUE or FALSE, depending on the user's preference.

### See Also

summary.

---

## placeHolds *Place Holes Under Correct Solids*

---

### Description

Place secondary polygons (SIDs) identified as holes (counter-clockwise rotation) under SIDs identified as solids (clockwise rotation) if the vertices of the holes lie completely within the vertices of the solids. This operation is performed for each primary polygon (PID).

### Usage

```
placeHolds(polyset, minVerts=3)
```

### Arguments

| | |
|---|---|
| polyset | a valid **PBSmapping** PolySet. |
| minVerts | minimum number of vertices required for a polygon representing a hole to be retained (does not affect solids). |

### Details

The algorith identifies the rotation of each polygon down to the SID level using the **PBSmapping** function .calcOrientation, where output values of 1 = solids (clockwise rotation) and -1 = holes (counter-clockwise rotation). Then for each solid, the function tests whether each hole occurs within the solid. To facilitate computation, the algorithm assumes that once a hole is located in a solid, it will not occur in any other solid. This means that for each successive solid, the number of candidate holes will either decrease or stay the same.

This function makes use of the point.in.polygon function contained in the package **sp**. For each hole vertex, the latter algorithm returns a numeric value:
0 = hole vertex is strictly exterior to the solid;
1 = hole vertex is strictly interior to the solid;
2 = hole vertex lies on the relative interior of an edge of the solid;
3 = hole vertex coincides with a solid vertex.

## Value

Returns the input PolySet where SID holes have been arranged beneath appropriate SID solids for each PID.

## Author(s)

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC.

## References

See copyright notice in point.in.polygon.

## See Also

importShapefile, point.in.polygon

---

# plotLines *Plot a PolySet as Polylines*

---

## Description

Plot a PolySet as polylines.

## Usage

```
plotLines (polys, xlim = NULL, ylim = NULL, projection = FALSE,
           plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
           lty = NULL, col = NULL, bg = 0, axes = TRUE,
           tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| polys | PolySet to plot (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| projection | desired projection when PolySet lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute. |
| plt | four element numeric vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired. |
| polyProps | PolyData specifying which polylines to plot and their properties. par parameters passed as direct arguments supersede these data. |
| lty | vector describing line types (cycled by PID). |
| col | vector describing colours (cycled by PID). |

| | |
|---|---|
| bg | background colour of the plot. |
| axes | Boolean value; if TRUE, plot axes. |
| tckLab | Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tck | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tckMinor | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| ... | additional par parameters, or the arguments main, sub, xlab, or ylab for the title function. |

## Details

This function plots a PolySet, where each unique (PID, SID) describes a polyline. It does not connect each polyline's last vertex to its first. Unlike plotMap, the function ignores the aspect ratio. It clips polys to xlim and ylim before plotting.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = "n" as part of ..., which is equivalent to specifying polys = NULL, but requires a PolySet. In both cases, the function's behaviour changes slightly. To resemble the plot function, it plots the border, labels, and other parts according to par parameters such as col.

For additional help on the arguments lty and col, please see par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, par parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

## See Also

addLines, calcLength, clipLines, closePolys, convLP, fixBound, fixPOS,

locatePolys, thinPolys, thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotLines(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5))
  par(oldpar)
})
```

---

`plotMap` *Plot a PolySet as a Map*

---

## Description

Plot a PolySet as a map, using the correct aspect ratio.

## Usage

```
plotMap (polys, xlim = NULL, ylim = NULL, projection = TRUE,
         plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
         border = NULL, lty = NULL, col = NULL, colHoles = NULL,
         density = NA, angle = NULL, bg = 0, axes = TRUE,
         tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| `polys` | PolySet to plot (*required*). |
| `xlim` | range of X-coordinates. |
| `ylim` | range of Y-coordinates. |
| `projection` | desired projection when PolySet lacks a `projection` attribute; one of `"LL"`, `"UTM"`, or a numeric value. If Boolean, specifies whether to check `polys` for a `projection` attribute. |
| `plt` | four element numeric vector `(x1, x2, y1, y2)` giving the coordinates of the plot region measured as a fraction of the figure region. Set to `NULL` if `mai` in `par` is desired. |
| `polyProps` | PolyData specifying which polygons to plot and their properties. `par` parameters passed as direct arguments supersede these data. |
| `border` | vector describing edge colours (cycled by `PID`). |
| `lty` | vector describing line types (cycled by `PID`). |
| `col` | vector describing fill colours (cycled by `PID`). |

| | |
|---|---|
| colHoles | vector describing hole colours (cycled by PID). The default, NULL, should be used in most cases as it renders holes transparent. colHoles is designed solely to eliminate retrace lines when images are converted to PDF format. If colHoles is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If NA is specified, only outer polygons are drawn, consequently filling holes. |
| density | vector describing shading line densities (lines per inch, cycled by PID). |
| angle | vector describing shading line angles (degrees, cycled by PID). |
| bg | background colour of the plot. |
| axes | Boolean value; if TRUE, plot axes. |
| tckLab | Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tck | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tckMinor | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| ... | additional par parameters, or the arguments main, sub, xlab, or ylab for the title function. |

## Details

This function plots a PolySet, where each unique (PID, SID) describes a polygon. It connects each polygon's last vertex to its first. The function supports both borders (border, lty) and fills (col, density, angle). When supplied with the appropriate arguments, it can draw only borders or only fills . Unlike plotLines and plotPolys, it uses the aspect ratio supplied in the projection attribute of polys. If this attribute is missing, it attempts to use its projection argument. In the absence of both, it uses a default aspect ratio of 1:1. It clips polys to xlim and ylim before plotting.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = "n" as part of ..., which is equivalent to specifying polys = NULL, but requires a PolySet. In both cases, the function's behaviour changes slightly. To resemble the plot function, it plots the border, labels, and other parts according to par parameters such as col.

For additional help on the arguments border, lty, col, density, and angle, please see polygon and par.

## Value

PolyData consisting of the PolyProps used to create the plot.

### Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, par parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

### Author(s)

Nicholas Boers, Dept. of Computer Science, Grant MacEwan University, Edmonton AB

### See Also

addLabels, addPolys, addStipples, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotLines, plotPoints, thinPolys, thickenPolys.

### Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotMap(polys,xlim=c(-.5,1.5),ylim=c(-.5,1.5),density=0,projection=1)
  par(oldpar)
})
```

---

## plotPoints *Plot EventData/PolyData as Points*

---

### Description

Plot EventData/PolyData, where each unique EID or (PID, SID) describes a point.

### Usage

```
plotPoints (data, xlim = NULL, ylim = NULL, projection = FALSE,
            plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
            cex = NULL, col = NULL, pch = NULL, axes = TRUE,
            tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

### Arguments

| | |
|---|---|
| data | EventData or PolyData to plot (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| projection | desired projection when PolySet lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute. |

| | |
|---|---|
| plt | four element numeric vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired. |
| polyProps | PolyData specifying which points to plot and their properties. par parameters passed as direct arguments supersede these data. |
| cex | vector describing character expansion factors (cycled by EID or PID). |
| col | vector describing colours (cycled by EID or PID). |
| pch | vector describing plotting characters (cycled by EID or PID). |
| axes | Boolean value; if TRUE, plot axes. |
| tckLab | Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tck | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tckMinor | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| ... | additional par parameters, or the arguments main, sub, xlab, or ylab for the title function. |

## Details

This function clips data to xlim and ylim before plotting. It only adds PolyData containing X and Y columns.

The function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = "n" as part of ..., which is equivalent to specifying polys = NULL, but requires a PolySet. In both cases, the function's behaviour changes slightly. To resemble the plot function, it plots the border, labels, and other parts according to par parameters such as col.

For additional help on the arguments cex, col, and pch, please see par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, par parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

## See Also

addPoints, combineEvents, convDP, findPolys, locateEvents.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,surveyData,envir=.PBSmapEnv)
  #--- plot a map
  plotMap(nepacLL, xlim=c(-136, -125), ylim=c(48, 57))
  #--- add events
  addPoints(surveyData, col=1:7)
  par(oldpar)
})
```

---

# plotPolys *Plot a PolySet as Polygons*

---

## Description

Plot a PolySet as polygons.

## Usage

```
plotPolys (polys, xlim = NULL, ylim = NULL, projection = FALSE,
           plt = c(0.11, 0.98, 0.12, 0.88), polyProps = NULL,
           border = NULL, lty = NULL, col = NULL, colHoles = NULL,
           density = NA, angle = NULL, bg = 0, axes = TRUE,
           tckLab = TRUE, tck = 0.014, tckMinor = 0.5 * tck, ...)
```

## Arguments

| | |
|---|---|
| polys | PolySet to plot (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |
| projection | desired projection when PolySet lacks a projection attribute; one of "LL", "UTM", or a numeric value. If Boolean, specifies whether to check polys for a projection attribute. |
| plt | four element numeric vector (x1, x2, y1, y2) giving the coordinates of the plot region measured as a fraction of the figure region. Set to NULL if mai in par is desired. |
| polyProps | PolyData specifying which polygons to plot and their properties. par parameters passed as direct arguments supersede these data. |
| border | vector describing edge colours (cycled by PID). |
| lty | vector describing line types (cycled by PID). |
| col | vector describing fill colours (cycled by PID). |

| | |
|---|---|
| colHoles | vector describing hole colours (cycled by PID). The default, NULL, should be used in most cases as it renders holes transparent. colHoles is designed solely to eliminate retrace lines when images are converted to PDF format. If colHoles is specified, underlying information (i.e., previously plotted shapes) will be obliterated. If NA is specified, only outer polygons are drawn, consequently filling holes. |
| density | vector describing shading line densities (lines per inch, cycled by PID). |
| angle | vector describing shading line angles (degrees, cycled by PID). |
| bg | background colour of the plot. |
| axes | Boolean value; if TRUE, plot axes. |
| tckLab | Boolean vector (length 1 or 2); if TRUE, label the major tick marks. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tck | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. If tckLab = TRUE, these tick marks will be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| tckMinor | numeric vector (length 1 or 2) describing the length of tick marks as a fraction of the smallest dimension. These tick marks can not be automatically labelled. If given a two-element vector, the first element describes the tick marks on the x-axis and the second element describes those on the y-axis. |
| ... | additional par parameters, or the arguments main, sub, xlab, or ylab for the title function. |

## Details

This function plots a PolySet, where each unique (PID, SID) describes a polygon. It connects each polygon's last vertex to its first. The function supports both borders (border, lty) and fills (col, density, angle). When supplied with the appropriate arguments, it can draw only borders or only fills. Unlike plotMap, it ignores the aspect ratio. It clips polys to xlim and ylim before plotting.

This function creates a blank plot when polys equals NULL. In this case, the user must supply both xlim and ylim arguments. Alternatively, it accepts the argument type = "n" as part of ..., which is equivalent to specifying polys = NULL, but requires a PolySet. In both cases, the function's behaviour changes slightly. To resemble the plot function, it plots the border, labels, and other parts according to par parameters such as col.

For additional help on the arguments border, lty, col, density, and angle, please see polygon and par.

## Value

PolyData consisting of the PolyProps used to create the plot.

## Note

To satisfy the aspect ratio, this plotting routine resizes the plot region. Consequently, par parameters such as plt, mai, and mar will change. When the function terminates, these changes persist to allow for additions to the plot.

## See Also

addLabels, addPolys, addStipples, clipPolys, closePolys, fixBound, fixPOS, locatePolys, plotLines, plotMap, plotPoints, thinPolys, thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- create a PolySet to plot
  polys <- data.frame(PID=rep(1,4),POS=1:4,X=c(0,1,1,0),Y=c(0,0,1,1))
  #--- plot the PolySet
  plotPolys(polys, xlim=c(-.5,1.5), ylim=c(-.5,1.5), density=0)
  par(oldpar)
})
```

---

# PolyData *PolyData Objects*

---

## Description

PBS Mapping functions that expect PolyData will accept properly formatted data frames in their place (see 'Details').

`as.PolyData` attempts to coerce a data frame to an object with class PolyData.

`is.PolyData` returns `TRUE` if its argument is of class PolyData.

## Usage

```
as.PolyData(x, projection = NULL, zone = NULL)
is.PolyData(x, fullValidation = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | data frame to be coerced or tested. |
| `projection` | optional `projection` attribute to add to PolyData, possibly overwriting an existing attribute. |
| `zone` | optional `zone` attribute to add to PolyData, possibly overwriting an existing attribute. |
| `fullValidation` | Boolean value; if `TRUE`, fully test `x`. |

## Details

We define PolyData as a data frame with a first column named `PID` and (optionally) a second column named `SID`. Unlike a PolySet, where each contour has many records corresponding to the vertices, a PolyData object must have only one record for each `PID` or each (`PID`, `SID`) combination. Conceptually, this object associates data with contours, where the data correspond to additional fields in the data frame. The R/S language conveniently allows data frames to contain fields of various

atomic modes ("logical", "numeric", "complex", "character", and "null"). For example, PolyData with the fields (PID, PName) might assign character names to a set of primary polygons. Additionally, if fields X and Y exist (perhaps representing locations for placing labels), consider adding attributes zone and projection. Inserting the string "PolyData" as the class attribute's first element alters the behaviour of some functions, including print (if PBSprint is TRUE) and summary.

Our software particularly uses PolyData to set various plotting characteristics. Consistent with graphical parameters used by the R/S functions lines and polygon, column names can specify graphical properties:

- lty - line type in drawing the border and/or shading lines;
- col - line or fill colour;
- border - border colour;
- density - density of shading lines;
- angle - angle of shading lines.

When drawing polylines (as opposed to closed polygons), only lty and col have meaning.

## Value

The as.PolyData method returns an object with classes "PolyData" and "data.frame", in that order.

## See Also

EventData, LocationSet, PolySet.

---

PolySet *PolySet Objects*

---

## Description

PBS Mapping functions that expect PolySet's will accept properly formatted data frames in their place (see 'Details').

as.PolySet attempts to coerce a data frame to an object with class PolySet.

is.PolySet returns TRUE if its argument is of class PolySet.

## Usage

```
as.PolySet(x, projection = NULL, zone = NULL)
is.PolySet(x, fullValidation = TRUE)
```

## Arguments

| | |
|---|---|
| x | data frame to be coerced or tested. |
| projection | optional projection attribute to add to the PolySet, possibly overwriting an existing attribute. |
| zone | optional zone attribute to add to the PolySet, possibly overwriting an existing attribute. |
| fullValidation | Boolean value; if TRUE, fully test x. |

## Details

In our software, a PolySet data frame defines a collection of polygonal contours (i.e., line segments joined at vertices), based on four or five numerical fields:

- PID - the primary identification number for a contour;
- SID - optional, the secondary identification number for a contour;
- POS - the position number associated with a vertex;
- X - the horizontal coordinate at a vertex;
- Y - the vertical coordinate at a vertex.

The simplest PolySet lacks an SID column, and each PID corresponds to a different contour. By analogy with a child's "follow the dots" game, the POS field enumerates the vertices to be connected by straight lines. Coordinates (X, Y) specify the location of each vertex. Thus, in familiar mathematical notation, a contour consists of *n* points $(x_i, y_i)$ with $i = 1, ..., n$, where *i* corresponds to the POS index. A PolySet has two potential interpretations. The first associates a line segment with each successive pair of points from 1 to *n*, giving a *polyline* (in GIS terminology) composed of the sequential segments. The second includes a final line segment joining points *n* and 1, thus giving a *polygon*.

The secondary ID field allows us to define regions as composites of polygons. From this point of view, each primary ID identifies a collection of polygons distinguished by secondary IDs. For example, a single management area (PID) might consist of two fishing areas, each defined by a unique SID. A secondary polygon can also correspond to an inner boundary, like the hole in a doughnut. We adopt the convention that POS goes from 1 to *n* along an outer boundary, but from *n* to 1 along an inner boundary, regardless of rotational direction. This contrasts with other GIS software, such as ArcView (ESRI 1996), in which outer and inner boundaries correspond to clockwise and counter-clockwise directions, respectively.

The SID field in a PolySet with secondary IDs must have integer values that appear in ascending order for a given PID. Furthermore, inner boundaries must follow the outer boundary that encloses them. The POS field for each contour (PID, SID) must similarly appear as integers in strictly increasing or decreasing order, for outer and inner boundaries respectively. If the POS field erroneously contains floating-point numbers, fixPOS can renumber them as sequential integers, thus simplifying the insertion of a new point, such as point 3.5 between points 3 and 4.

A PolySet can have a projection attribute, which may be missing, that specifies a map projection. In the current version of PBS Mapping, projection can have character values "LL" or "UTM", referring to "Longitude-Latitude" and "Universal Transverse Mercator". We explain these projections more completely below. If projection is numeric, it specifies the aspect ratio *r*, the number of *x* units per *y* unit. Thus, *r* units of *x* on the graph occupy the same distance as one unit of *y*. Another

optional attribute zone specifies the UTM zone (if projection="UTM") or the preferred zone for conversion from Longitude-Latitude (if projection="LL").

A data frame's class attribute by default contains the string "data.frame". Inserting the string "PolySet" as the class vector's first element alters the behaviour of some functions. For example, the summary function will print details specific to a PolySet. Also, when PBSprint is TRUE, the print function will display a PolySet's summary rather than the contents of the data frame.

## Value

The as.PolySet method returns an object with classes "PolySet" and "data.frame", in that order.

## References

Environmental Systems Research Institute (ESRI). (1996) *ArcView GIS: The Geographic Information System for Everyone*. ESRI Press, Redlands, California. 340 pp.

## See Also

EventData, LocationSet, PolyData.

---

# print *Print PBS Mapping Objects*

---

## Description

This function displays information about a PBS Mapping object.

summary.EventData, summary.LocationSet, summary.PolyData, and summary.PolySet produce an object with class summary.PBS.

## Usage

```
## S3 method for class 'EventData'
print(x, ...)
## S3 method for class 'LocationSet'
print(x, ...)
## S3 method for class 'PolyData'
print(x, ...)
## S3 method for class 'PolySet'
print(x, ...)
## S3 method for class 'summary.PBS'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | a PBS Mapping object of appropriate class. |
| ... | additional arguments to print. |

## See Also

EventData, LocationSet, PBSprint, PolyData, PolySet, summary.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- change to summary printing style
  PBSprint <- TRUE
  #--- print the PolySet
  print(nepacLL)
})
```

---

# pythagoras *Data: Pythagoras' Theorem Diagram PolySet*

---

## Description

PolySet of shapes to prove Pythagoras' Theorem: $a^2 + b^2 = c^2$.

## Usage

```
data(pythagoras)
```

## Format

4 column data frame: PID = primary polygon ID, POS = position of each vertex within a given polyline, X = X-coordinate, and Y = Y-coordinate. Attributes: projection = 1.

## Note

In R, the data must be loaded using the data function.

## Source

An artificial construct to illustrate the proof of Pythagoras' Theorem using trigonometry.

## See Also

addPolys, plotPolys, plotMap, PolySet.

refocusWorld *Refocus the* worldLL/worldLLhigh *Data Sets*

## Description

Refocus the worldLL/worldLLhigh data sets, e.g., refocus them so that Eastern Canada appears to the west of Western Europe.

## Usage

```
refocusWorld (polys, xlim = NULL, ylim = NULL)
```

## Arguments

| | |
|---|---|
| polys | PolySet with one or more polygons; typically worldLL or worldLLhigh (*required*). |
| xlim | range of X-coordinates. |
| ylim | range of Y-coordinates. |

## Details

This function accepts a PolySet containing one or more polygons with X-coordinates that collectively span approximately 360 degrees. The function effectively joins the PolySet into a cylinder and then splits it at an arbitrary longitude according to the user-specified limits. Modifications in the resulting PolySet are restricted to shifting X-coordinates by +/- multiples of 360 degrees, and instead of clipping polygons, the return value simply omits out-of-range polygons.

## Value

PolySet, likely a subset of the input PolySet, which retains the same PID/SID values.

## Author(s)

Nicholas Boers, Dept. of Computer Science, Grant MacEwan University, Edmonton AB

## See Also

joinPolys

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load appropriate data
  data(worldLL,envir=.PBSmapEnv)
  #--- set limits
  xlim <- c(-100,25)
  ylim <- c(0,90)
```

```
  #--- refocus and plot the world
  polys <- refocusWorld(worldLL, xlim, ylim)
  plotMap(polys, xlim, ylim)
  par(oldpar)
})
```

---

summary *Summarize PBS Mapping Objects*

---

## Description

summary method for PBS Mapping classes.

## Usage

```
## S3 method for class 'EventData'
summary(object, ...)
## S3 method for class 'LocationSet'
summary(object, ...)
## S3 method for class 'PolyData'
summary(object, ...)
## S3 method for class 'PolySet'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | a PBS Mapping object, such as EventData, a LocationSet, PolyData, or a PolySet. |
| ... | further arguments passed to or from other methods. |

## Details

After creating a list of summary statistics, this function assigns the class "summary.PBS" to the output in order to accomplish formatted printing via print.summary.PBS.

## Value

A list of summary statistics.

## See Also

EventData, LocationSet, PBSprint, PolyData, PolySet.

## Examples

```
local(envir=.PBSmapEnv,expr={
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(surveyData,envir=.PBSmapEnv)
  print(summary(surveyData))
})
```

---

`surveyData` *Data: Tow Information from Pacific Ocean Perch Survey*

---

## Description

EventData of Pacific ocean perch (POP) tow information (1966-89).

## Usage

```
data(surveyData)
```

## Format

Data frame consisting of 9 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polygon, `X` = longitude coordinate, `Y` = latitude coordinate, `trip` = trip ID, `tow` = tow number in trip, `catch` = catch of POP (kg), `effort` = tow effort (minutes), `depth` = fishing depth (m), and `year` = year of survey trip. Attributes: `projection = "LL"`, `zone = 9`.

## Note

In R, the data must be loaded using the `data` function.

## Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC).

The POP (*Sebastes alutus*) survey data were extracted from GFBio. The data extraction covers bottom trawl surveys that focus primarily on POP biomass estimation: 1966-89 for the central BC coast and 1970-85 for the west coast of Vancouver Island. Additionally, a 1989 cruise along the entire BC coast concentrated on the collection of biological samples. Schnute et al. (2001) provide a more comprehensive history of POP surveys including the subset of data presented here.

## References

Schnute, J.T., Haigh, R., Krishka, B.A. and Starr, P. (2001) Pacific ocean perch assessment for the west coast of Canada in 2001. *Canadian Science Advisory Secretariat, Research Document* **2001/138**, 90 pp.

## See Also

addPoints, combineEvents, EventData, findPolys, makeGrid, plotPoints.

---

thickenPolys *Thicken a PolySet of Polygons*

---

## Description

Thicken a PolySet, where each unique (PID, SID) describes a polygon.

## Usage

```
thickenPolys (polys, tol = 1, filter = 3, keepOrig = TRUE,
              close = TRUE)
```

## Arguments

| | |
|---|---|
| polys | PolySet to thicken. |
| tol | tolerance (in kilometres when proj is "LL" and "UTM"; otherwise, same units as polys). |
| filter | minimum number of vertices per result polygon. |
| keepOrig | Boolean value; if TRUE, keep the original points in the PolySet. |
| close | Boolean value; if TRUE, create intermediate vertices between each polygon's last and first vertex, if necessary. |

## Details

This function thickens each polygon within polys according to the input arguments.

If keepOrig = TRUE, all of the original vertices appear in the result. It calculates the distance between two sequential original vertices, and if that distance exceeds tol, it adds a sufficient number of vertices spaced evenly between the two original vertices so that the distance between vertices no longer exceeds tol. If close = TRUE, it adds intermediate vertices between the last and first vertices when necessary.

If keepOrig = FALSE, only the first vertex of each polygon is guaranteed to appear in the results. From this first vertex, the algorithm walks the polygon summing the distance between vertices. When this cumulative distance exceeds tol, it adds a vertex on the line segment under inspection. After doing so, it resets the distance sum, and walks the polygon from this new vertex. If close = TRUE, it will walk the line segment from the last vertex to the first.

## Value

PolySet containing the thickened data. The function recalculates the POS values for each polygon.

## See Also

thinPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot Vancouver Island
  plotMap(nepacLL[nepacLL$PID == 33, ])
  #--- calculate a thickened version using a 30 kilometres tolerance,
  #--- without keeping the original points
  p <- thickenPolys(nepacLL[nepacLL$PID == 33, ], tol = 30, keepOrig = FALSE)
  #--- convert the PolySet to EventData by dropping the PID column and
  #--- renaming POS to EID
  p <- p[-1]; names(p)[1] <- "EID"
  #--- convert the now invalid PolySet into a data frame, and then into
  #--- EventData
  p <- as.EventData(as.data.frame(p), projection="LL")
  #--- plot the results
  addPoints(p, col=2, pch=19)
  par(oldpar)
})
```

---

# thinPolys *Thin a PolySet of Polygons*

---

## Description

Thin a PolySet, where each unique (PID, SID) describes a polygon.

## Usage

```
thinPolys (polys, tol = 1, filter = 3)
```

## Arguments

| | |
|---|---|
| polys | PolySet to thin. |
| tol | tolerance (in kilometres when proj is "LL" and "UTM"; otherwise, same units as polys). |
| filter | minimum number of vertices per result polygon. |

## Details

This function executes the Douglas-Peuker line simplification algorithm on each polygon within polys.

## Value

PolySet containing the thinned data. The function recalculates the POS values for each polygon.

## See Also

thickenPolys.

## Examples

```
local(envir=.PBSmapEnv,expr={
  oldpar = par(no.readonly=TRUE)
  #--- load the data (if using R)
  if (!is.null(version$language) && (version$language=="R"))
    data(nepacLL,envir=.PBSmapEnv)
  #--- plot a thinned version of Vancouver Island (3 km tolerance)
  plotMap(thinPolys(nepacLL[nepacLL$PID == 33, ], tol = 3))
  #--- add the original Vancouver Island in a different line type to
  #--- emphasize the difference
  addPolys(nepacLL[nepacLL$PID == 33, ], border=2, lty=8, density=0)
  par(oldpar)
})
```

---

# towData *Data: Tow Information from Longspine Thornyhead Survey*

---

## Description

PolyData of tow information for a longspine thornyhead survey (2001).

## Usage

```
data(towData)
```

## Format

Data frame consisting of 8 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, Y = latitude coordinate, depth = fishing depth (m), effort = tow effort (minutes), distance = tow track distance (km), catch = catch of longspine thornyhead (kg), and year = year of survey. Attributes: projection = "LL", zone = 9.

## Note

In R, the data must be loaded using the data function.

## Source

The GFBio database, maintained at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7), archives catches and related biological data from commercial groundfish fishing trips and research/assessment cruises off the west coast of British Columbia (BC). The longspine thornyhead (*Sebastolobus altivelis*) survey data were extracted from GFBio. Information on the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Effort is time (minutes) from winch lock-up to winch release.

## References

Starr, P.J., Krishka, B.A. and Choromanski, E.M. (2002) Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2421**, 60 pp.

## See Also

`makeProps`, PolyData, `towTracks`.

---

`towTracks` *Data: Tow Track Polylines from Longspine Thornyhead Survey*

---

## Description

PolySet of geo-referenced polyline tow track data from a longspine thornyhead survey (2001).

## Usage

```
data(towTracks)
```

## Format

Data frame consisting of 4 columns: `PID` = primary polygon ID, `POS` = position of each vertex within a given polyline, `X` = longitude coordinate, and `Y` = latitude coordinate. Attributes: `projection = "LL"`, `zone = 9`.

## Note

In R, the data must be loaded using the `data` function.

## Source

The longspine thornyhead (*Sebastolobus altivelis*) tow track spatial coordinates are available at the Pacific Biological Station (Fisheries and Oceans Canada, Nanaimo, BC V9T 6N7). The geo-referenced coordinates of the first 45 tows from the 2001 survey (Starr et al. 2002) are included here. Coordinates are recorded once per minute between winch lock-up and winch release.

## References

Starr, P.J., Krishka, B.A. and Choromanski, E.M. (2002) Trawl survey for thornyhead biomass estimation off the west coast of Vancouver Island, September 15 - October 2, 2001. *Canadian Technical Report of Fisheries and Aquatic Sciences* **2421**, 60 pp.

## See Also

`addLines`, `calcLength`, `clipLines`, `plotLines`, PolySet, `towData`.

worldLL *Data: Shorelines of the World (Normal Resolution)*

## Description

PolySet of polygons for the global shorelines.

## Usage

```
data(worldLL)
```

## Format

Data frame consisting of 4 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, and Y = latitude coordinate. Attributes: projection = "LL".

## Note

In R, the data must be loaded using the data function.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database gshhs_l.b.
Download from http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html

```
worldLL <-importGSHHS("gshhs_l.b", xlim=c(-20,360), ylim=c(-90,90),
                      level=1, n=15, xoff=0)
worldLL <- .fixGSHHSWorld(worldLL)
```

## References

Wessel, P. and Smith, W.H.F. (1996) A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research* **101**, 8741–8743.
http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html

## See Also

Data: worldLLhigh, nepacLL, nepacLLhigh

importGSHHS, addPolys, clipPolys, plotPolys, plotMap, thickenPolys, thinPolys

worldLLhigh *Data: Shorelines of the World (High Resolution)*

## Description

PolySet of polygons for the global shorelines.

## Usage

```
data(worldLLhigh)
```

## Format

Data frame consisting of 4 columns: PID = primary polygon ID, POS = position of each vertex within a given polygon, X = longitude coordinate, and Y = latitude coordinate. Attributes: projection = "LL".

## Note

In R, the data must be loaded using the data function.

## Source

Polygon data from the GSHHS (Global Self-consistent, Hierarchical, High-resolution Shoreline) database gshhs_i.b.
Download from http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html

```
worldLLhigh <-importGSHHS("gshhs_i.b", xlim=c(-20,360),
                 ylim=c(-90,90), level=1, n=15, xoff=0)
worldLLhigh <- .fixGSHHSWorld(worldLLhigh)
```

## References

Wessel, P. and Smith, W.H.F. (1996) A global, self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research* **101**, 8741–8743.
http://www.soest.hawaii.edu/pwessel/pwessel_pubs.html

## See Also

Data: worldLL, nepacLL, nepacLLhigh

importGSHHS, addPolys, clipPolys, plotPolys, plotMap, thickenPolys, thinPolys

# Index