Package 'rangeMapper'

June 30, 2018

Version 0.3-4

Title A Platform for the Study of Macro-Ecology of Life History Traits

Depends R (>= 3.0.0), RSQLite(>= 1.0.0)

Imports methods, sp, rgdal, rgeos, raster, maptools, gridExtra, lattice, ggplot2, RColorBrewer, classInt, magrittr, data.table, foreach

Suggests shiny (>= 0.8.0), doParallel, testthat, knitr, rmarkdown, MASS

Description Tools for easy generation of (life-history) traits maps based on species range (extent-of-occurrence) maps.

License GPL (>= 2)

URL https://github.com/valcu/rangeMapper

RoxygenNote 6.0.1

Collate 'AAA.R' 'AllClasses.R' 'GUI.R' 'MapCanvas-methods.R' 'MapExport-methods.R' 'MapFetch-methods.R' 'MapPlot-methods.R' 'MapProcess-methods.R' 'MapRemove-methods.R' 'MapSave-methods.R' 'MapShow-methods.R' 'projectINI-methods.R' 'assemblageFetch-methods.R' 'bioSave-methods.R' 'metadataUpdate-methods.R' 'rangeFiles-methods.R' 'rangeMapper-package.R' 'rangeMapper-ramp.R' 'style.R' 'utils-spatial.R' 'utils.R'

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author Mihai Valcu [aut, cre], James Dale [aut], Joan Maspons [ctb]

Maintainer Mihai Valcu <valcu@orn.mpg.de>

Repository CRAN

Date/Publication 2018-06-30 21:10:43 UTC

R topics documented:

`as.rmap.frame` *Convert data.table to rmap.frame*

Description

Convert data.table to rmap.frame

Usage

```
as.rmap.frame(x, ...)

## S3 method for class 'data.table'
as.rmap.frame(x, p4s, gridSize, bbox)
```

Arguments

x	a data.table
...	extra arguments
p4s	proj4string
gridSize	grid size
bbox	global bounding box, a list with x and y

Value

an rmap.frame object which inherits from data.table

---

assemblageFetch *Retrieves the species set of an arbitrary canvas cell*

Description

assemblageFetch retrieves the species set of an arbitrary canvas cell optionally with the associated life history data

Usage

```
assemblageFetch(object, xy, BIO)

## S4 method for signature 'rangeMap,SpatialPoints,missing'
assemblageFetch(object, xy)

## S4 method for signature 'rangeMap,SpatialPoints,character'
assemblageFetch(object, xy, BIO)
```

Arguments

object	A connection object.
xy	A SpatialPoints object.
BIO	The name of the BIO_table containing species life-history data.

Value

A data.frame containing the bioid (e.g. species names), the canvas id and optionally any associated life history data contained in the BIO_table table.

Examples

```
require(rangeMapper)
require(rgdal)

projName = "wrens.sqlite"
projLoc = paste(tempdir(), projName, sep = .Platform$file.sep)

dbcon = rangeMap.start(file = projName,dir = tempdir() , overwrite = TRUE)
f = system.file(package = "rangeMapper", "extdata", "wrens", "vector_combined")
r = readOGR(f, "wrens", verbose = FALSE)
global.bbox.save(con = dbcon, bbox = r)
gridSize.save(dbcon, gridSize = 3)
canvas.save(dbcon)
data(wrens)
bio.save(con = dbcon, loc = wrens ,  ID = "sci_name")
processRanges(spdf = r, con =  dbcon, ID = "sci_name")
rangeMap.save(dbcon)

sr = rangeMap.fetch(dbcon)
image(sr, axes = TRUE); grid()

p = list(x = -76.39, y = 9.26)
# or use locator:  p =  locator(1)

xy = SpatialPoints( do.call(cbind, p), proj4string = CRS(proj4string(r)) )
af = assemblageFetch(rangeMap(projLoc) , xy)
points(p, col = 4, cex = 2)
print(af)

af = assemblageFetch(rangeMap(projLoc) , xy, "wrens")
print(af[, c(1, 4, 6:8)])
```

---

bio.save *Import 'BIO' tables to a* rangeMapper *project.*

Description

Import tables (e.g. life history data) to an active rangeMapper project.

Usage

```
bio.save(con, loc, tableName, ...)

bio.merge(con, tableName, ...)

metadata2bio(con, ...)
```

Arguments

con	an sqlite connection pointing to a valid rangeMapper project.
loc	file location or data.frame name.
tableName	if missing, the name of the file or data.frame is used
...	arguments to pass to the corresponding methods: e.g. the ID, the column corresponding to the names of the range files

Value

a 'BIO' table is created in the corresponding rangeMapper project.

Examples

```
require(rangeMapper)
require(rgdal)
wd = setwd(tempdir())
r = readOGR(system.file(package = "rangeMapper",
"extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)
dbcon = rangeMap.start(file = "wrens.sqlite", overwrite = TRUE,
dir = tempdir() )
global.bbox.save(con = dbcon, bbox = r)
gridSize.save(dbcon, gridSize = 2)
canvas.save(dbcon)
processRanges(spdf = r, con =  dbcon, ID = "sci_name" )

# Upload BIO tables
data(wrens)
Troglodytes  = wrens[grep("Troglodytes", wrens$sci_name), c(2, 5)]
bio.save(con = dbcon, loc = Troglodytes,  ID = "sci_name")

setwd(wd)


## Not run:
require(rangeMapper)
require(rgdal)
wd = setwd(tempdir())
r = readOGR(system.file(package = "rangeMapper",
  "extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)
dbcon = rangeMap.start(file = "wrens.sqlite", overwrite = TRUE,
dir = tempdir() )
global.bbox.save(con = dbcon, bbox = r)
gridSize.save(dbcon, gridSize = 2)
canvas.save(dbcon)
processRanges(spdf = r, con =  dbcon, ID = "sci_name", metadata = rangeTraits() )

wrensPath = system.file(package = "rangeMapper", "data", "wrens.csv")
bio.save(con = dbcon, loc = wrensPath,  ID = "sci_name")
bio.merge(dbcon, "wrensNew")
metadata2bio(dbcon)
```

```
setwd(wd)


## End(Not run)
```

---

canvas.save *Project's canvas*

Description

The canvas is a regular grid of a given resolution. Each range map is overlayed onto the canvas and the results saved to project.

Usage

```
canvas.save(con)

canvas.fetch(con)
```

Arguments

con	An sqlite connection pointing to a valid rangeMapper project.

Value

canvas.fetch Returns a SpatialPixelsDataFrame object.

Note

The method canvasSave() fails if grid.size was not set and if the canvas was already constructed for the given project.

See Also

rangeMap.save.
gridSize.save

Examples

```
wd = tempdir()
dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = wd)
global.bbox.save(con = dbcon)
gridSize.save(dbcon, gridSize = 2)
canvas.save(dbcon)
cnv = canvas.fetch(dbcon)
summary(cnv)
plot(cnv, col = 'grey', axes = TRUE)
```

`global.bbox.save` *Global bounding box*

Description

Computes, sets or retrieves the global spatial bounding box.

Usage

```
global.bbox.save(con, ...)

global.bbox.fetch(con)
```

Arguments

`con`	An `SQLiteConnection` object pointing to a `rangeMapper` project
`...`	Arguments to pass to the corresponding methods: *bbox* can be a `character` vector; the path to the range files directory *bbox* can also be an object inheriting from `Spatial` *p4s* an object of class `CRS`

Details

`global.bbox.save` saves the *global bounding box* and the *proj4* string to the sqlite database.
`global.bbox.fetch` retrieves the *global bounding box* as a `SpatialPolygonsDataFrame`.

Note

If *bbox* is a `character` vector then the corresponding method calls `rangeMapBbox` with `checkProj = TRUE` which requires all ranges to have the same *proj4* argument.
If *p4s* is set then the *bbox* will be set with that *p4s* string else the *p4s* will be identical with the *proj4* string of the range files.
If *bbox* and *p4s* are missing then an unprojected global bounding box is set.

Examples

```
require(rangeMapper)
wd = tempdir()

f= system.file(package = "rangeMapper", "extdata", "wrens", "vector")

# Using default values for both bbox and p4s
dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = wd )
global.bbox.save(con = dbcon)
bbox0 = global.bbox.fetch(dbcon)

plot(bbox0, axes = TRUE)
```

gridSize.save *Save or retrieve the grid size from an* rangeMapper *project.*

Description

Save or retrieve the grid size from the active sqlite database.

Usage

```
gridSize.save(con, ...)

gridSize.fetch(con)
```

Arguments

con	A connection pointing to a valid rangeMapper project.
...	gridSize: A numeric vector of one unit length. See notes.

Value

list("gridSize.fetch")
: Returns a numeric vector of one unit length containing the grid size previously saved by gridSize.save

Note

If gridSize is not given the default grid size is computed based on the bounding box as the range of the smallest axis /100.

Examples

```
wd = tempdir()
dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = wd )
global.bbox.save(con = dbcon)
gridSize.save(dbcon, gridSize = 2)

dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = wd )
global.bbox.save(con = dbcon)
gridSize.save(dbcon)
gridSize.fetch(dbcon) #default grid size value
```

metadata.update *Updates metadata table*

Description

Updates metadata_table of a rangeMapper project *after* importing ranges with processRanges.

Usage

```
metadata.update(rangeMap, FUN, name, map, overwrite = FALSE, ...)
```

Arguments

rangeMap	A rangeMap object.
FUN	Function used to aggregate the map values corresponding to each range
name	The name of the new metadata_table field containing the variable computed by FUN
map	Single-band SpatialGridDataFrame object
overwrite	If set to TRUE the the values of the field are replaced
...	extra arguments (e.g. na.rm = TRUE) to be passed to FUN.

Value

NULL.

Note

In order to compute taxa-level metadata which are not dependent on the project's resolution use processRanges with a metadata argument. See rangeTraits for more details.
The method can be extended to work with raster or vector objects (e.g. lines, polygons, points) using overlaying functions in the package raster and rgeos respectively.

Examples

```
require(rangeMapper)
require(rgdal)
# data
spdf = readOGR(system.file(package = "rangeMapper",
"extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)
rloc = system.file(package = "rangeMapper", "extdata",
"etopo1", "etopo1_Americas.tif")
r = readGDAL(rloc, output.dim = c(50, 50))
spdf = spTransform(spdf, CRS(proj4string(r)) )

# the project
dbcon = rangeMap.start(file = "wrens.sqlite", overwrite = TRUE,
```

```
dir = tempdir() )
rmap = new("rangeMap", CON = dbcon)
global.bbox.save(con = dbcon, bbox = spdf )
gridSize.save(dbcon, gridSize = 300000)
canvas.save(dbcon)
processRanges(spdf = spdf, con =  dbcon, ID = "sci_name" )

# metadata.update
metadata.update (rmap,
 FUN = function(x, ...) {
  res = diff(range(x, ...))
  if( !is.finite(res)) res = 0
  res
  },
name = 'AltitudeRange', map = r, na.rm = TRUE, overwrite = TRUE)
# plot
mr = dbGetQuery(dbcon, 'select * from metadata_ranges')
maxRangeSp = mr[mr$AltitudeRange== max(mr$AltitudeRange), 'bioid']
image(r)
plot(rangeFetch(rmap, maxRangeSp), add = TRUE, border = 4, lwd = 3)
title(main = maxRangeSp)
```

---

palette_rangemap *A few color palettes*

Description

A few color palettes

Usage

```
palette_rangemap(set = "set1")
```

Arguments

set	set type (currently one set only)

---

plot,rmap.frame,missing-method *Plot a rmap.frame*

Description

Plot a rmap.frame

Usage

```
## S4 method for signature 'rmap.frame,missing'
plot(x, colours = palette_rangemap("set1"),
  outlierDetector, boundary, boundaryCol = 1, boundarySize = 0.5, ...)
```

Arguments

x	a rmap.frame object.
colours	a vector of colours to pass to scale_fill_gradientn.
outlierDetector	a function used to detect ouliers. Should return lower and upper limits of non-outliers.
boundary	a Spatial* object which can be fortifyed.
boundaryCol	boundary color, see geom_polygon.
boundarySize	boundary size, geom_polygon.
...	further arguments to pass to arrangeGrob.

Value

a ggplot object for one map or a gtable in case of more than one map.

See Also

plot,SpatialPixelsRangeMap,missing-method

Examples

```
require(rangeMapper)
breding_ranges = rgdal::readOGR(system.file(package = "rangeMapper",
     "extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)[1:70, ]
data(wrens)
d = subset(wrens, select = c('sci_name', 'body_mass', 'clutch_size') )
con = ramp("wrens.sqlite", gridSize = 4, spdf = breding_ranges, biotab = d, ID = "sci_name",
            FUN = "median", overwrite = TRUE)
m = rangeMap.fetch(con, c('median_body_mass', 'median_clutch_size'), spatial = FALSE)
plot(m, ncol = 2)

wrens_boundary = rgeos::gUnionCascaded(breding_ranges)
plot(m, ncol = 2, boundary = wrens_boundary)

## Not run:
if(require(extremevalues))
plot(m, ncol = 2, outlierDetector = function(x) getOutliersI(x)$limit)

## End(Not run)
```

plot,SpatialPixelsRangeMap,missing-method
Plot a SpatialPixelsRangeMap

Description

This is a wrapper around spplot

Usage

```
## S4 method for signature 'SpatialPixelsRangeMap,missing'
plot(x,
  colorpalette = brewer.pal.get("Spectral")[11:1], ncols = 20,
  scales = FALSE, style = "equal", ...)
```

Arguments

x	a SpatialPixelsRangeMap.
colorpalette	a color palette.
ncols	number of color classes required, default to 20; argument to be passed to classIntervals.
scales	ff 'FALSE', the default, axes scale are not drawn.
style	class interval style; see classIntervals for more details.
...	any argument that can be passed to see spplot

See Also

plot,rmap.frame,missing-method.

Examples

```
breding_ranges = rgdal::readOGR(system.file(package = "rangeMapper",
     "extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)[1:10, ]
data(wrens)
d = subset(wrens, select = c('sci_name', 'body_size', 'body_mass', 'clutch_size') )
con = ramp("wrens.sqlite", gridSize = 10, spdf = breding_ranges, biotab = d, ID = "sci_name",
            metadata = rangeTraits(), FUN = "median", overwrite = TRUE)
all = rangeMap.fetch(con)
sr = rangeMap.fetch(con, 'species_richness')
plot(sr)
plot(all)
```

processRanges *processRanges*

Description

processRanges

Usage

```
processRanges(con, spdf, dir, ID, metadata)


  ## S4 method for signature
## 'SQLiteConnection,SpatialPolygonsDataFrame,missing,character,missing'
processRanges(con,
  spdf, ID, metadata)


  ## S4 method for signature
## 'SQLiteConnection,SpatialPolygonsDataFrame,missing,character,list'
processRanges(con,
  spdf, ID, metadata)


 ## S4 method for signature 'SQLiteConnection,missing,character,missing,missing'
processRanges(con,
  dir)

## S4 method for signature 'SQLiteConnection,missing,character,missing,list'
processRanges(con,
  dir, metadata)
```

Arguments

con	a connection object.
spdf	SpatialPolygonsDataFrame object containing all the ranges.
dir	ranges file directory where the individual ranges shp files are located. In this case the range ID is the file name.
ID	a character vector of length one. An spdf column name indicating the range ID (e.g. species name).
metadata	a named list of functions. See rangeTraits and metadata.update.

Methods (by class)

- con = SQLiteConnection,spdf = SpatialPolygonsDataFrame,dir = missing,ID = character,metadata = mi
  Method 1: One SpatialPolygonsDataFrame containing all the ranges. No metadata.

- con = SQLiteConnection,spdf = SpatialPolygonsDataFrame,dir = missing,ID = character,metadata = li
  Method 2: One SpatialPolygonsDataFrame containing all the ranges. Metadata are computed.
- con = SQLiteConnection,spdf = missing,dir = character,ID = missing,metadata = missing:
  Method 3: Each range file is a separate shp file. No metadata.
- con = SQLiteConnection,spdf = missing,dir = character,ID = missing,metadata = list:
  Method 4: Each range file is a separate shp file. Metadata are computed.

Note

if a parallel backend is registered with the foreach package then processRanges runs in parallel.

Examples

```
require(rangeMapper)
require(rgdal)
## Not run:
if (require(doParallel) ) {
 cl = makePSOCKcluster(2)
 registerDoParallel(cl)
 }

## End(Not run)

dbcon = rangeMap.start(file = "wrens.sqlite", dir = tempdir(), overwrite = TRUE)
f = system.file(package = "rangeMapper", "extdata", "wrens", "vector_combined")
r = readOGR(f, "wrens", verbose = FALSE)[1:50, ]
global.bbox.save(con = dbcon, bbox = r)
gridSize.save(dbcon, gridSize = 5)
canvas.save(dbcon)
processRanges(con = dbcon, spdf = r, ID = "sci_name", metadata = rangeTraits() )
dbDisconnect(dbcon)

## Not run:
stopCluster(cl)

## End(Not run)
```

---

ramp **ra***nge* **m***apper* **p***ipe line.*

Description

A quick alternative to initiate a project by pipelining several functions.

Usage

```
ramp(file, dir = tempdir(), gridSize, spdf, bbox = spdf, ID, biotab,
  metadata, FUN, overwrite = FALSE)
```

Arguments

file	project file name.
dir	project directory.
gridSize	grid resolution (in units previously set by global.bbox.save)
spdf	SpatialPolygonsDataFrame object containing all the ranges.
bbox	the spatial domain of the project (see global.bbox.save)
ID	a character vector of length one. An spdf column name indicating the range ID (e.g. species name).
biotab	character string identifying the 'BIO' table to use.
metadata	a named list of functions. See rangeTraits and metadata.update.
FUN	the function to be applied to each pixel. If FUN is missing then species richness (species count) is computed.
overwrite	logical vector, default to FALSE (the file is kept but all tables are dropped).

Value

an sqlite connection to a rangeMapper project

Note

ramp combines all the functions from rangeMap.start() to processRanges() and rangeMap.save() but is less flexible as compared with a step-by-step project building.

See Also

rangeMap.start global.bbox.save gridSize.save canvas.save processRanges bio.save rangeMap.save

Examples

```
breding_ranges = rgdal::readOGR(system.file(package = "rangeMapper",
     "extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)[1:50, ]
data(wrens)
d = subset(wrens, select = c('sci_name', 'body_size', 'clutch_size') )
con = ramp("wrens.sqlite", gridSize = 15, spdf = breding_ranges, biotab = d, ID = "sci_name",
            metadata = rangeTraits(), FUN = "median", overwrite = TRUE)
m = rangeMap.fetch(con)
dbDisconnect(con)
```

rangeFetch *rangeFetch Range extractor*

Description

Fetch an arbitrary range from a rangeMapper project.

Usage

```
rangeFetch(rangeMap, bioid)
```

Arguments

rangeMap	A rangeMap object.
bioid	A character vector, usually a taxon name, which identifies a range within a given rangeMapper project.

Value

A SpatialPolygons.

Examples

```
wd = setwd(tempdir())
require(rangeMapper)
require(rgdal)
spdf = readOGR(system.file(package = "rangeMapper", "extdata",
"wrens", "vector_combined"), "wrens", verbose = FALSE)
dbcon = rangeMap.start(file = "wrens.sqlite",
overwrite = TRUE, dir = tempdir() )
rmo = rangeMap("wrens.sqlite")
global.bbox.save(con = dbcon, bbox = spdf)
gridSize.save(dbcon, gridSize = 3)
canvas.save(dbcon)
processRanges(spdf = spdf, con =  dbcon, ID = "sci_name" )
rangeMap.save(dbcon)

house_wren = rangeFetch(rmo, "Troglodytes_aedon")
image(rangeMap.fetch(dbcon))
plot(house_wren, add = TRUE, border = 'blue', lwd = 2)
setwd(wd)
```

rangeMap *Initiate/open a new rangeMapper project*

Description

Initiate/open a new rangeMapper project

Usage

```
rangeMap(path)

rangeMap.start(dir, file, overwrite = FALSE)

rangeMap.open(path, verbose = TRUE)
```

Arguments

path	character vector; a path to a valid rangeMapper project.
dir	project directory.
file	project file name.
overwrite	logical vector, default to FALSE (the file is kept but all tables are dropped).
verbose	character vector; if TRUE the project's summary is printed.

Value

rangeMap.start() and rangeMap.open() returns an sqlite connection
rangeMap() returns an object of class rangeMap

See Also

rangeMap.save

Examples

```
td = setwd(tempdir())

dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = tempdir() )
summary(dbcon)

summary(rangeMap("test.sqlite"))

dbcon = rangeMap.open(path = "test.sqlite")
summary(dbcon)
setwd(td)
```

rangeMap.export *Export 'MAP' tables*

Description

Export 'MAP' tables as single-band geotiff files

Usage

```
rangeMap.export(con, dirName = dirName, ...)
```

Arguments

con	An sqlite connection pointing to a valid rangeMapper project.
dirName	The directory name where the 'MAP's will be exported. If missing, the 'MAP's will be exported in project's directory
...	Further arguments to pass to writeGDAL

rangeMap.fetch *rangeMap.fetch*

Description

rangeMap.fetch

Usage

```
rangeMap.fetch(con, maps, spatial = TRUE)
```

Arguments

con	a connection to a valid rangeMapper project.
maps	map(s) name as character vector. If missing then all the maps are returned.
spatial	If TRUE (default) a SpatialPixelsRangeMap is returned, else a rmap.frame.

Value

an object of SpatialPixelsRangeMap or data.table containing the spatial coordinates and proj4 string as an attribute if spatial = FALSE.

rangeMap.save *Save, retrieve and export maps.*

Description

Apply a chosen SQL or function at each grid cell, allowing for complex subsetting at both ID (e.g. species) and pixel (e.g. assemblage) levels.

Usage

```
rangeMap.save(CON, tableName, FUN, biotab, biotrait, subset = list(), path,
  overwrite = FALSE, cl, ...)
```

Arguments

CON	an sqlite connection pointing to a valid rangeMapper project.
tableName	name of the table (quoted) to be added to the sqlite database. the prefix 'MAP' will be appended to tableName prior to saving.
FUN	the function to be applied to each pixel. If FUN is missing then species richness (species count) is computed.
biotab	character string identifying the 'BIO' table to use.
biotrait	character string identifying the ID of the 'BIO' table. see bio.save
subset	a named list. See details
path	path to the raster file(quoted) to be imported to the existing project. raster package is required at this step.
overwrite	if TRUE then the table is removed
cl	the number of cores to use or a cluster object defined with makeCluster in package parallel or makeCluster from snow package.
...	when FUN is a function, ... denotes any extra arguments to be passed to it.

Details

The subset argument accepts a named list. Names refers to 'BIO', 'MAP' and 'metadata_rages' table names while the strings in the list are character strings containing the SQL WHERE clause. The subset can point to either one table type (e.g. list(MAP_species_richness = "species_richness > 500")) or can point to several table types (e.g. list(BIO_lifeHistory = "clutch_size > 4",MAP_meanAltitude = "meanAlti

Any valid SQL expression can be used to build up a subset. See http://www.sqlite.org/lang_expr.html

When using cl parameter you must load the given packages used in FUN by loading the packages inside the function, using '::' or initializing the cluster before calling rangeMap.save (e.g. clusterEvalQ(cl=cl, library(caper))).

Value

TRUE when the MAP was created successfully.

Note

SQL aggregate functions are more efficient then their R counterparts. For simple aggregate functions like mean, median, sd, count it is advisable to use SQL functions rather then R functions.

See Also

metadata.update.

Examples

```
require(rangeMapper)
require(rgdal)
breding_ranges = readOGR(system.file(package = "rangeMapper",
     "extdata", "wrens", "vector_combined"), "wrens", verbose = FALSE)[1:50, ]
breding_ranges = spTransform(breding_ranges,
    CRS("+proj=cea +lon_0=0 +lat_ts=30 +x_0=0 +y_0=0
        +ellps=WGS84 +units=m +no_defs") )
data(wrens)
d = subset(wrens, select = c('sci_name', 'body_size', 'body_mass', 'clutch_size') )

con = ramp("wrens.sqlite", gridSize = 500000, spdf = breding_ranges, biotab = d,
            ID = "sci_name", metadata = rangeTraits(),
            FUN = "median", overwrite = TRUE)


lmSlope = function(formula, data) {
    fm = try(lm(formula, data = data), silent = TRUE)
    if (inherits(fm, "try-error"))
        res = NA else res = coef(fm)[2]
    as.numeric(res)
}

# Subsetting by Species and Assembladge
rangeMap.save(con, FUN = lmSlope, biotab = "biotab", biotrait = "body_mass",
    tableName = "slope_bodyMass_clutchSize", formula = log(body_mass) ~ clutch_size,
    list(MAP_species_richness = "species_richness >= 5",
        BIO_biotab = "body_size > 15"
        ), overwrite = TRUE)

## Not run:
# Import raster maps to the current project
r = system.file(package = "rangeMapper", "extdata", "etopo1", "etopo1_Americas.tif")
rangeMap.save(con, path = r, tableName = "meanAltitude", FUN = mean, overwrite = TRUE)
m = rangeMap.fetch(con, spatial = FALSE)
plot(m)


## End(Not run)
```

`rangeMapper` *rangeMapper: A platform for the study of macroecology of life history traits.*

Description

rangeMapper is a front end platform for the study of macroecology of life history traits at both inter-specific and assemblage levels.

References

Valcu, M., Dale, J. and Kempenaers, B. (2012) rangeMapper: A platform for the study of macroecology of life history traits. 21(9). (DOI: 10.1111/j.1466-8238.2011.00739.x)

`rangeMapProjInfo` *rangeMap file info*

Description

rangeMap file info

Usage

```
rangeMapProjInfo(con)
```

Arguments

`con`	a connection to a rangeMapper project.

Value

data.table

rangeOverlay *rangeOverlay*

Description

rangeOverlay

Usage

```
rangeOverlay(spp, canvas, name)
```

Arguments

spp	a SpatialPolygons* object.
canvas	a SpatialPointsDataFrame.
name	a character vector (a bioid)

Value

a data.frame with two columns: id (canvas id) and bioid.

rangeTraits *A container of functions to apply on a* SpatialPolygons *object*

Description

This is a convenience function returning a named list of functions.

Usage

```
rangeTraits(..., use.default = TRUE)
```

Arguments

...	functions, given as myfun = FUN, to apply on a SpatialPolygons object
use.default	If TRUE, the default, the output list contains functions to extract Area, Median, Min and Max extent of the SpatialPolygons object. This option is ignored if no functions are given.

Details

The function returns a named list so any additional functions should be given as rangeTraits(funName1 = FUN1, funName2 = FUN2) where FUN1, FUN2 are SpatialPolygons extractor functions.

Value

Returns a named list containing extractor functions to apply on SpatialPolygons objects.

See Also

processRanges.

Examples

```
summary(rangeTraits(use.default = FALSE))

f = system.file(package = "rangeMapper", "extdata", "wrens", "vector")
troaed = selectShpFiles(f, ogr = TRUE,
polygons.only = TRUE)[71, ] # path to Troglodytes_aedon
require(rgdal)
r = readOGR(troaed$dsn, troaed$layer)

# Beware of the value returned for Area!
sapply(rangeTraits(), function(x) x(r) )

# Define an extra function to compute correct Area
Area2 = function(x) {
x = spTransform(x,
CRS("+proj=cea +lon_0=0 +lat_ts=30 +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_defs")
)

sum(sapply(slot(x, "polygons"), function(x) slot(x, "area") ))
}

sapply(rangeTraits(Area_sqm = Area2), function(x) x(r) )
```

---

rm.rangeMapper *Remove tables from a give project*

Description

Remove tables given prefix attribute or by name

Usage

```
rm.rangeMapper(con, ...)
```

Arguments

con	A valid sqlite connection.
...	Arguments passed to the corresponding methods specifically 'tablePrefix' or 'tableName'

Note

The default 'rm.rangeMapper(con)' will remove all 'MAP' and 'BIO' tables.

`selectShpFiles` *Select (recursively) shape files*

Description

Returns the file path to all '.shp' polygons in a directory.

Usage

```
selectShpFiles(dir, ...)
```

Arguments

`dir`	character string specifying the directory containing .shp files.
`...`	currently ignored

Value

Either a `data.frame` or a character vector is returned.

Note

The function uses `getinfo.shape` to only select polygon files (aka type 5).

Examples

```
f = system.file(package="rangeMapper", "extdata", "wrens", "vector")
res = selectShpFiles(f, ogr = TRUE, polygons.only = TRUE)
head(res)
```

`tables` *Method tables*

Description

Method tables

Usage

```
tables(object)
```

Arguments

`object`	an sqlite connection object.

tables,SQLiteConnection-method *Tables and column names of an sqlite db*

Description

Tables and column names of an sqlite db

Usage

```
## S4 method for signature 'SQLiteConnection'
tables(object)
```

Arguments

object	an sqlite connection object.

Value

data.frame

theme_rangemap *ggplot theme*

Description

A ggplot theme based on theme_bw

Usage

```
theme_rangemap(base_size = 12, base_family = "")
```

Arguments

base_size	base_size
base_family	base_family

View_rmap *rangeMapper browser*

Description

rangeMapper browser

Usage

```
View_rmap(path)
```

Arguments

path	path to a rangeMapper project. If missing a demo project is created on the fly.

Examples

```
## Not run:
View_rmap()

## End(Not run)
```

WKT2SpatialPolygonsDataFrame *Convert WKT polygons to SpatialPolygonsDataFrame*

Description

Convert a data.frame containing WKT polygons to a SpatialPolygonsDataFrame.

Extract vertices from a SpatialPolygonsDataFrame and optionally applies an aggregating function to each Polygon.

Usage

```
WKT2SpatialPolygonsDataFrame(dat, geom, id)

vertices(object, FUN)

## S4 method for signature 'SpatialPolygons'
vertices(object, FUN)
```

Arguments

dat	data.frame
geom	is the name (character vector) of the column in the data.frame containing the geometry.
id	is the name (character vector) of the column in the data.frame identifying the polygon. when id is not unique then polygons are combined using gUnionCascaded.
object	An object.
FUN	A function.

Value

a SpatialPolygonsDataFrame object.

A SpatialPointsDataFrame containing an id column corresponding to each extracted Polygon.

Examples

```
require(rangeMapper)
require(rgeos)

# generate a few random polygons
randPoly = function(mean, sd) {
  writeWKT(
    gConvexHull(
     readWKT(paste("MULTIPOINT (",
             paste(apply(matrix(rnorm(n= 100, mean, sd), ncol = 2), 1,
             paste, collapse = ' '), collapse = ","), ")"))))
}
n = 50
d = data.frame( nam = sample(letters, n, TRUE),
               range = mapply(randPoly, mean = sample(1:2, n, TRUE),
               sd = sample(1:2/5, n, TRUE) ))


X = WKT2SpatialPolygonsDataFrame(d, 'range', 'nam')


dbcon = rangeMap.start(file = "test.sqlite", overwrite = TRUE, dir = tempdir() )
global.bbox.save(con = dbcon, bbox = X)
gridSize.save(dbcon)
canvas.save(dbcon)
processRanges(spdf = X, con =  dbcon, ID = "nam")
rangeMap.save(dbcon)
plot(rangeMap.fetch(dbcon))

require(rangeMapper)
require(rgdal)
f = system.file(package = "rangeMapper", "extdata", "wrens", "vector")
# path to Campylorhynchus_gularis breeding range:
camgul = selectShpFiles(f, ogr = TRUE, polygons.only = TRUE)[6, ]
```

```
r = readOGR(camgul$dsn, camgul$layer)
mp = vertices(r, mean)
v = vertices(r)

plot(r)
points(mp, col = 2, pch = 3, cex = 2)
points(v, pch = 3, cex = .5)
```

---

wrens *Life history data of the New World Wrens*

Description

Life history data (body size, body mass and clutch size) of 84 wren **(Troglodytidae)** species

Format

A data frame with 84 observations on the following 7 variables.

ID_HBW Handbook of the birds of the world ID

body_mass body mass (grams)

body_size body size (cm)

clutch_size mean or modal clutch size

com_name English name; a factor with 84 levels

genus Genus name

sci_name scientific name; a factor with 84 levels

source bibliographic source of each trait (see references)

Details

Taxonomic nomenclature follows (Kroodsma & Brewer, 2005) with the exception of *Donacoblis atricapilla* which has been excluded due to its uncertain taxonomic position.

References

Auer, S.K., Logue, D.M., Bassar, R.D. & Gammon, D.E. (2007) Nesting biology of the Black-bellied Wren (Thryothorus fasciatoventris) in central Panama. Wilson Journal of Ornithology, 119, 71-76.

Dunning, J.B. (2008) CRC handbook of avian body masses, 2nd edn. CRC Press, Boca Raton.

Freeman, B.G. & Greeney, H.F. (2008) First description of the nest, eggs and cooperative breeding behavior in sharpe's wren (Cinnycerthia olivascens). Ornitologia Colombiana, 7, 88-92.

Hron, K., Templ, M. & Filzmoser, P. (2010) Imputation of missing values for compositional data using classical and robust methods. Computational Statistics & Data Analysis, 54, 3095-3107 (function impKNNa using default arguments).

Kroodsma, D.E. & Brewer, D. (2005) Family Troglodytidae (Wrens). Lynx Edicions, Barcelona,

Spain.
Londono, G.A. (2009) Eggs, Nests, and Incubation Behavior of the Moustached Wren (Thryothorus genibarbis) in Manu National Park, Peru. Wilson Journal of Ornithology, 121, 623-627.
Ridgely, R.S., T. F. Allnutt, T. Brooks, D. K. McNicol, D. W. Mehlman, B. E. & Young, a.J.R.Z. (2007) Digital Distribution Maps of the Birds of the Western Hemisphere, version 3.0. NatureServe, Arlington, Virginia, USA.
Vargas-Soriano, J., Ortiz, J.S. & Segura, G.E. (2010) Breeding Phenology and Nesting Success of the Yucatan Wren in the Yucatan Peninsula, Mexico. Wilson Journal of Ornithology, 122, 439-446.

See Also

rangeMap.save.

Examples

```
data(wrens)
plot(body_size ~ body_mass, wrens)
plot(clutch_size ~ log(body_mass), wrens)
```

Index