Package ‘monographaR’

May 9, 2026

Version 1.3.1

Date 2024-02-01

Title Taxonomic Monographs Tools

Author Marcelo Reginato

Maintainer Marcelo Reginato <`reginatobio@yahoo.com.br`>

Depends R (>= 4.0.0)

Suggests knitr

Imports circular, png, raster, rmarkdown, rnaturalearth, rpart, sf, shiny, shinydashboard, shinythemes, shinyWidgets, sp, terra

Description Contains functions intended to facilitate the production of plant taxonomic monographs. The package includes functions to convert tables into taxonomic descriptions, lists of collectors, examined specimens, identification keys (dichotomous and interactive), and can generate a monograph skeleton. Additionally, wrapper functions to batch the production of phenology histograms and distributional and diversity maps are also available.

VignetteBuilder knitr

License GPL (>= 2)

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2024-02-08 13:00:03 UTC

Contents

`buildMonograph` *Build and export a monograph skeleton (draft)*

Description

This function will generate a MS-Word or html file with a monograph skeleton (draft)

Usage

```
buildMonograph(headings, tableToDescription.data, examinedSpecimens.data = NULL,
collectorList.data = NULL, output = "Word", title = "Taxonomic treatment",
open = TRUE)
```

Arguments

`headings`	data.frame
`tableToDescription.data`	data.frame
`examinedSpecimens.data`	data.frame (optional)
`collectorList.data`	data.frame (optional)
`output`	"Word" or "html"
`title`	character
`open`	logical

Details

This function wraps around the functions tableToDescription, examinedSpecimens and collectorList generating a monograph draft in MS-Word or html format. The resulting monograph skeleton will include the taxonomic heading, the description, comments and examined specimens list for all species found in the input tables, and it will append the collector list in the end of the file. It requires four tables as input. Three of them are the same tables used for `tableToDescription`, `collectorList`, and `examinedSpecimens` functions. The additional input table should have three columns: species, taxonomic heading and comments. The examinedSpecimens.data and collectorList.data tables are optional. It uses functions of the `rmarkdown` package to export the output file.

Value

Exports a file (MS-Word or html).

Author(s)

Marcelo Reginato

See Also

`rmarkdown`

Examples

```
data(monographaR_examples)
monographaR_examples$taxonomic_headings -> taxonomic.headings
monographaR_examples$collectorList -> col.d
monographaR_examples$examinedSpecimens -> exam.d
monographaR_examples$tableToDescription -> desc.d
desc.d[,-1] -> desc.d

### buildMonograph(headings=taxonomic.headings,
###                collectorList.data = col.d,
###                examinedSpecimens.data = exam.d,
###                tableToDescription.data = desc.d,
###                output = "Word", title="Monograph skeleton")
```

`codeGap` *Code gap*

Description

This function takes a numeric vector (or a data.frame with two columns including min and max values of a sample) and tries to find breaks in the distribution (gaps), if any gap is found it returns a coded character based on that.

Usage

```
codeGap(x, n = NULL, max.states = NULL, poly.sep = "/", gap.size = NULL)
```

Arguments

`x`	integer/numeric or a two column data.frame (min and max)
`n`	integer, desired number of states (if NULL the function will try to suggest a number)
`max.states`	integer, the maximum possible number of states
`poly.sep`	character, to indicate polymorphic states (if any)
`gap.size`	numeric, the number that should be considered as a "gap"

Details

If n = NULL the function will try to find the best scenario of states (n) based on the number of polymophic samples in the resulting classification. In large data sets it will be a good idea to constrain the search using (e.g., max.states=10). This coding tries to replicate the coding traditionally used in taxonomy.

Value

list, including: dat = data.frame including the original value and the coded value (state) polymorphic = the number of polymorphic samples (if n=NULL, it returns for all tested scenarios) dist = a histogram of the data distribution

Author(s)

Marcelo Reginato

Examples

```
c(NA, 1:5, 15:20, 25:42, 49:60, 68:90) -> x
data.frame(x,x=x+2) -> x2

codeGap(x, n=3, max.states = 5) -> code1
code1$dat

### check the distribution

na.omit(unique(code1$dat$state)) -> b
cols <- sort(rep(rainbow(length(b)),2))
as.numeric(unlist(strsplit(b, "-"))) -> b
plot(code1$dist)
abline(v=b, lty="dashed", col=cols, lwd=2)

### estimate "n"

codeGap(x, n=NULL, max.states = NULL) -> code1
code1$dat
```

```
plot(code1$dist)

### check the distribution

na.omit(unique(code1$dat$state)) -> b
cols <- sort(rep(rainbow(length(b)),2))
as.numeric(unlist(strsplit(b, "-"))) -> b
plot(code1$dist)
abline(v=b, lty="dashed", col=cols, lwd=2)

### ranges

codeGap(x2, n=NULL, max.states =3 , gap.size=2) -> code1
code1$dat
unique(code1$dat$state)

### check the distribution

na.omit(unique(code1$dat$state)) -> b
cols <- sort(rep(rainbow(length(b)),2))
as.numeric(unlist(strsplit(b, "-"))) -> b
plot(code1$dist)
abline(v=b, lty="dashed", col=cols, lwd=2)
```

`codeMatrix` *Code character matrix*

Description

This is a wrapper function to codeGap, which accepts a data.frame of characters values including ranges (1-10).

Usage

```
codeMatrix(dat, n = NULL, max.states = NULL, poly.sep = "/", gap.size = NULL)
```

Arguments

`dat`	data.frame
`n`	integer, see `codeGap`
`max.states`	integer, see `codeGap`
`poly.sep`	character, see `codeGap`
`gap.size`	integer, see `codeGap`

Value

data.frame

Author(s)

Marcelo Reginato

See Also

`codeGap`

Examples

```
data(monographaR_examples)
monographaR_examples$dichoKey_q -> dat
rownames(dat) <- dat[,1]
dat[,-1] -> dat

codeMatrix(dat, max.states = 3) -> mat
head(mat)
```

`collectorList` *Generates a collector list*

Description

This function will generate a txt file with a collector list for all species in data.

Usage

```
collectorList(data = data, filename = "collector_list.txt",
paragraphs = TRUE)
```

Arguments

`data`	data.frame
`filename`	character
`paragraphs`	logical

Details

It requires a data.frame with five columns, ordered as species, collector name, collector number, herbarium acronym and herbarium number. Herbarium columns are only used if any collector number is missing (NA). Thus, if there is no missing values in collector number, then the herbarium columns might be empty.

Value

Exports a txt file.

Author(s)

Marcelo Reginato

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$collectorList -> data
head(data)

## running the function, it will print in the terminal the output.
## To export a txt, place a ## name in the filename argument
## (i.e., filename = "myoutput.txt")

collectorList(data, filename = "", paragraphs = TRUE)

## or a second option

collectorList(data, filename = "", paragraphs = FALSE)
```

`dataKey` *Prepares a data.frame suitable for buildKey*

Description

This function takes a data.frame with polymorphic and/or missing data (NA) characters and prepares it to use in the function buildKey.

Usage

```
dataKey(dat, poly.sep = "/", return.summary = T)
```

Arguments

`dat`	data.frame
`poly.sep`	character
`return.summary`	boolean

Details

This function takes a data.frame with polymorphic characters and prepares it to use in the function buildKey.

Value

data.frame

Author(s)

Marcelo Reginato

See Also

dichoKey

Examples

```
data(monographaR_examples)

monographaR_examples$dichoKey -> dat
colnames(dat)[1] <- "species"

### Prepare matrix (polymorphic)

dataKey(dat, poly.sep = "/") -> dat.k
dat.k$dat -> dat.p
```

`dichoKey` *Dichotomous key*

Description

This function generates a dichotomous key

Usage

```
dichoKey(dat, cost = NULL, clean.characters = TRUE, cp = 0)
```

Arguments

`dat`	data.frame
`cost`	numeric
`clean.characters`	boolean
`cp`	numeric

Details

This is a wrapper function based on the rpart function of the rpart package. It modifies the output of rpart to a more "taxonomic" dichotomous key.

Value

list

Author(s)

Marcelo Reginato

See Also

`rpart` `dataKey`

Examples

```
data(monographaR_examples)

monographaR_examples$dichoKey -> dat
colnames(dat)[1] <- "species"

### Prepare matrix (polymorphic)

dataKey(dat, poly.sep = "/") -> dat.k
dat.k$dat -> dat.p

### key

dichoKey(dat.p) -> key
length(key$unresolved)

# Export

#cat(key$key, file="Pleiochiton_key.txt")

### key with costs

dat.k$summary
((dat.k$summary)+1) -> dat.c
colnames(dat.p)
dat.c[2] <- max(dat.c)
dat.c[11] <- max(dat.c)

dichoKey(dat.p, dat.c, cp=0) -> key.c
length(key.c$unresolved)

# Export

#cat(key.c$key, file="Pleiochiton_key_costs.txt")
```

`dichoKey.app` *Dichotomous key (shiny app)*

Description

This is a shiny app that generates a dichotomous key

Usage

```
dichoKey.app()
```

Details

A shiny app that runs the function "dichoKey". It allows the user to easily subset the data (taxa and characters), as well as change the cost values of the characters.

Value

NULL

Author(s)

Marcelo Reginato

See Also

`rpart` `dataKey` `dichoKey`

Examples

```
if (interactive()) {

dichoKey.app()

}
```

`examinedSpecimens` *Generates an examined specimens list*

Description

This function will generate a txt file with an examined specimens list.

Usage

```
examinedSpecimens(data, filename = "examined.txt")
```

Arguments

`data`	data.frame
`filename`	character

Details

It requires a data.frame with eight columns, ordered as: species, collector name, collector number, herbarium acronym, herbarium number, country, state and municipality.

Value

Exports a txt file.

Author(s)

Marcelo Reginato

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$examinedSpecimens -> data
head(data)

## running the function, it will print in the terminal the output.
## To export a txt, place a name in the filename argument
## (i.e., filename = "myoutput.txt")

examinedSpecimens(data, filename = "")
```

`interactiveKey` *Interactive key (shiny app)*

Description

Generates code to build an interactive key (shiny app)

Usage

```
interactiveKey(dat = NULL, txt.labels = NULL, poly.sep = "/",
taxa.in.italics = TRUE, theme = "lumen", about.first = FALSE)
```

Arguments

`dat`	data.frame
`txt.labels`	data.frame
`poly.sep`	character
`taxa.in.italics`	logical
`theme`	character
`about.first`	logical, whether the tab "about" is the first or the last

Details

See vignette

Value

It generates 6 files (app.R, server.R, ui.R, about.Rmd, Dat_characters.csv, and Dat_matrix.csv)

Author(s)

Marcelo Reginato

See Also

`interactiveKeyLabels` `themes`

Examples

```
## see vignette
data(monographaR_examples)
monographaR_examples$interactiveKey
```

`interactiveKeyLabels` *Text labels for interactiveKey*

Description

It generates a data.frame with txt labels required to use the function interactiveKey

Usage

```
interactiveKeyLabels(taxon = "species", language = "english")
```

Arguments

`taxon`	character, "species", "genus" or "family"
`language`	character, "english" or "portuguese"

Details

It generates a data.frame with txt labels required to use the function interactiveKey. You can choose what rank will be identified in the key (species, genus or family), and the language (english and portuguese so far). You can further modify the fields prior using it with the interactiveKey function.

Value

data.frame

Author(s)

Marcelo Reginato

See Also

`interactiveKey`

Examples

```
### standard labels species in english

interactiveKeyLabels(taxon = "species", language = "english") -> labs.spp.eng
labs.spp.eng

### standard labels families in portuguese

interactiveKeyLabels(taxon = "family", language = "portuguese") -> labs.fam.por
labs.fam.por
```

`keyCode` *keyCode*

Description

Internal code (interactiveKey)

`mapBatch` *Generates map in batch mode*

Description

This wrapper function will export maps for all species in data.

Usage

```
mapBatch(data, zoom = T, margin = 0.1, axes = T, shape = NULL,
export = "pdf", raster = NULL, RGB = NULL, points.col = "black",
points.border = "gray50", points.cex = 1, shape.col = "white",
shape.border = "black", raster.col = rev(gray.colors(65, start = 0, end = 1)),
raster.legend = F, hillshade = F, width = 8, height = 8,
image.resolution = 100, figure.number = T, title = T, box = T,
add.minimap = F, minimap.shape = NULL, minimap.shape.col = "white",
minimap.shape.border = "gray50", minimap.pos = "topleft",
minimap.add.points = T, minimap.points.col = "black",
minimap.points.border = "gray50", minimap.points.cex = 1,
minimap.extent = NULL, minimap.rect.fill = NA, minimap.rect.border = NULL,
maxpixels = 1e+05, ...)
```

Arguments

`data`	data.frame
`zoom`	logical
`margin`	numeric
`axes`	logical
`shape`	a single or a list of spatial shape objects
`export`	"pdf", "jpeg" or "tiff"
`raster`	a raster object
`RGB`	a raster stack object (with three layers)
`points.col`	character
`points.border`	character
`points.cex`	numeric
`shape.col`	character
`shape.border`	character
`raster.col`	character (a vector of colors)
`raster.legend`	logical
`hillshade`	logical
`width`	numeric (in inches)
`height`	numeric (in inches)
`image.resolution`	numeric
`figure.number`	logical
`title`	logical
`box`	logical
`add.minimap`	logical
`minimap.shape`	a spatial shape object
`minimap.shape.col`	character (color)
`minimap.shape.border`	character (color)
`minimap.pos`	"topleft", "topright", "bottomleft" or "bottomright"
`minimap.add.points`	logical
`minimap.points.col`	character (color)
`minimap.points.border`	character (color)
`minimap.points.cex`	numeric

`minimap.extent`	numeric (x1, x2, y1, y2)
`minimap.rect.fill`	character (color)
`minimap.rect.border`	character (color)
`maxpixels`	numeric
`...`	additional arguments for plotting the extra shapes

Details

The function has three output options: a single pdf with all maps (export = "pdf") or individual image files for each species (export = "tiff" or "jpeg"). It requires a data.frame with three columns, ordered as: species, longitude and latitude. If zoom = TRUE, the function will set the limits of the plot using the distribution of each species plus the margin (relative value). If zoom = FALSE, the function will use the distribution of the whole data to set the limits (all maps will have the same limits). Colors can be changed with the arguments points.col, shape.col, shape.border, while the size of the points can be changed with points.cex. A raster layer can be provided (elevation for instance), and the colors of the raster are controlled by raster.col. The user can provide a single or a list of shape files, otherwise the `rnaturalearth` map is used.

Value

Exports a pdf or image files.

Author(s)

Marcelo Reginato

See Also

`raster`

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$map_data -> data
head(data)

## running the function

# mapBatch(data , type=”simple”, zoom=T, margin=0.2, points.col=”black”,
# points.border=”white”, shape.col=”gray90”, points.cex=1.5, shape.border
# = ”gray90”, export=”pdf”)
```

`mapDiversity` *Diversity heatmap*

Description

This function will generate a diversity heatmap using presence/absence of species on grid cells.

Usage

```
mapDiversity(data, resolution = 1, plot = T, plot.with.grid = T,
col=rev(terrain.colors(55)), alpha=0.8, export = F, legend = T,
filename = "diversity_map")
```

Arguments

`data`	data.frame
`resolution`	numeric, size of the grid cells (degrees)
`plot`	logical
`plot.with.grid`	logical, whether or not to add a grid to the plot
`col`	character, a vector of colors
`alpha`	numerical, controls color transparency (0-1)
`export`	logical
`legend`	logical
`filename`	character

Details

It requires a data.frame with three columns, ordered as: species, longitude and latitude. The function will plot and return a raster object. The resolution of the grid can be changed by the argument "resolution" (in degrees). It uses functions of the package raster.

Value

A raster object.

Author(s)

Marcelo Reginato

See Also

`raster`

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$map_data -> data
head(data)

## running the function

mapDiversity(data , resolution=1, plot=TRUE, plot.with.grid=TRUE)

## Without the grid borders

mapDiversity(data , resolution=1, plot=TRUE, plot.with.grid=FALSE)

## Changing colors

mapDiversity(data , resolution=1, plot=TRUE, col=gray.colors(55))

## Changing transparency

mapDiversity(data , resolution=1, plot=TRUE, alpha=0.5)

## The function returns a raster object

mapDiversity(data , resolution=1, plot=TRUE, alpha=0.5, plot.with.grid=FALSE) -> my.div.raster
my.div.raster
# plot(my.div.raster)
# writeRaster(my.div.raster, "mydivraster.tif")
```

`mapPhenology` *Phenology heatmap*

Description

This function will generate phenology maps across time (month, week, etc..).

Usage

```
mapPhenology(data, resolution = 1, time.range = c(1:12), label = "Month",
binary = T, by_species = F, plot = T, col = rev(heat.colors(12)),
alpha = 0.8, mfrow = c(4, 3), legend = T, pdf = F, height = 11,
width = 8.5, filename = "mapPhenology.pdf")
```

Arguments

`data`	data.frame
`resolution`	numeric (degrees)

`time.range`	numeric (vector of months, weeks, etc...)
`label`	character ("Month", "Week")
`binary`	logical
`by_species`	logical
`plot`	logical
`col`	character (vector of colors)
`alpha`	numeric (0-1)
`mfrow`	numeric
`legend`	logical
`pdf`	logical
`height`	numerical
`width`	numerical
`filename`	character

Details

This wrapper function will generate heatmaps of phenology across a time range. The default is to produce 12 heatmaps plotted on a single plate. This can be changed with the argument time.range, where any numerical range can be provided (representing weeks for instance). The argument mfrow controls the plate layout. It requires a data.frame with four columns, ordered as: species, longitude, latitude and phenology. The phenology column should be numeric (i.e., the number of the month, week or day the specimen was collected with flower/fruit). It is possible to change the resolution of the resulting rasters. The function can produce presence/absence heatmaps (if binary = T) or abundance heatmaps (if binary = F). The abundance values are relative (divided by the maximum abundance observed across all rasters). The function returns a RasterStack that can be exported or used in customized plots. To export a pdf, set "pdf=TRUE". The function wraps around functions of the raster package.

Value

RasterStack

Author(s)

Marcelo Reginato

See Also

`raster`

Examples

```
### load the example data

data(monographaR_examples)
monographaR_examples$mapPhenology -> data
```

```
head(data) ## check the first rows

### running the function

# mapPhenology(data, binary=FALSE, by_species=FALSE, legend=FALSE)

### changing the colors

# mapPhenology(data, binary=FALSE, by_species=FALSE, legend=FALSE, col=rev(terrain.colors(55)))

### exporting raster

# require(raster)
# mapPhenology(data, binary=FALSE, by_species=FALSE, legend=FALSE) -> myphenorasters
# plot(myphenorasters[[1]]) ## plot first month
# writeRaster(myphenorasters[[2]], "pheno_month2.asc") ## exporting 2nd month

### making an GIF animation

# require(animation)
# saveGIF(
#  {mapPhenology(data, binary=F, resolution=0.5, by_species=F, legend=F, mfrow=c(1,1))},
#  movie.name="phenology.gif", interval=0.5, ani.width=600, ani.height=600
# )
```

`mapTable` *Generates a presence/absence matrix of species on grids or countries*

Description

This function will generate a presence/absence matrix based on a grid (if type="grid") or on countries (if type="countries").

Usage

```
mapTable(data, type = "grid", resolution = 1, pres.abs = TRUE,
write.output = FALSE, layer = NULL)
```

Arguments

`data`	data.frame
`type`	"grid", "countries" or "user"
`resolution`	numeric (degrees)
`pres.abs`	logical
`write.output`	logical
`layer`	Spatial DataFrame object

Details

It requires a data.frame with three columns, ordered as: species, longitude and latitude. The resolution of the grid can be changed by the argument "resolution" (in degrees). If type = "user", a layer to intersect the points and create the matrix should be supplied (a Spatial DataFrame object). It uses functions of the package raster, sf and terra. If pres.abs = F the returned matrix will have "x" instead of 0 and 1.

Value

list, with a matrix and grid (if type="grid), or a matrix (if type="countries").

Author(s)

Marcelo Reginato

See Also

`raster`

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$map_data -> data
head(data)

## running the function with grid

map.table <- mapTable(data, type="grid", resolution=3,
write.output=FALSE)

map.table$table
t(map.table$table)

map.table$grid -> grid

### load world map
library(rnaturalearth)
library(sf)
library(sp)
ne_countries(type="countries", returnclass = "sv") -> wrld_simpl
st_as_sf(wrld_simpl) -> wrld_simpl
as_Spatial(wrld_simpl) -> wrld_simpl

### plot
plot(grid, border="white")
plot(wrld_simpl, add=TRUE)
plot(grid, add=TRUE)
raster::text(grid, grid@data$layer, cex=1)
```

`monographaR` *Tools for taxonomic monographs.*

Description

monographaR contains functions intended to facilitate the production of plant taxonomic monographs. The package includes functions to convert tables into taxonomic descriptions, lists of collectors, examined specimens, identificatio keys (dichotomous and interactive), and can generate a monograph skeleton. Additionally, wrapper functions to batch the production of phenology histograms and distributional and diversity maps are also available.

Details

Package:	monographaR
Type:	Package
Version:	1.3.1
Date:	2024-02-01
License:	GPL (>= 2)

Author(s)

Marcelo Reginato

Maintainer: Marcelo Reginato <reginatobio@yahoo.com.br>

References

Reginato, M. (2016) monographaR: an R package to facilitate the production of plant taxonomic monographs. Brittonia 68(2): 212-216.

See Also

`circular` `raster` `sp` `rmarkdown` `rpart`

`monographaR_examples` *Input data examples*

Description

Input table examples. Seven data.frames are listed in this example data set. See help files of the functions for details.

Examples

```
data(monographaR_examples)

names(monographaR_examples)

head(monographaR_examples$colletorList)
head(monographaR_examples$examinedSpecimens)
head(monographaR_examples$phenoHist)
head(monographaR_examples$tableToDescription)
head(monographaR_examples$map_data)
head(monographaR_examples$mapPhenology)
head(monographaR_examples$taxonomic_headings)
```

`phenoHist` *Circular histograms of phenology*

Description

This wrapper function will generate circular histograms of phenology, using functions of the package circular.

Usage

```
phenoHist(data = data, mfrow = c(1, 1), shrink = 1.2, axis.cex =
1.5, title.cex = 1.5, pdf = F, height=11, width=8.5,
filename = "phenology.pdf", flower = "Flower", fruit = "Fruit",
both = "Both", flower.col = NULL, flower.border = "black",
fruit.col = "darkgray", fruit.border = "darkgray", mar=c(2,2,2,2))
```

Arguments

`data`	data.frame
`mfrow`	numeric, (nrow, ncol)
`shrink`	numeric

`axis.cex`	numeric
`title.cex`	numeric
`pdf`	logical
`height`	numeric
`width`	numeric
`filename`	character
`flower`	character (how is the flower indicated in data, if missing place "missing")
`fruit`	character (how is the fruit indicated in data, if missing place "missing")
`both`	character (how is the both indicated in data, if missing place "missing")
`flower.col`	character (color of flower bars)
`flower.border`	character (color of flower border bars)
`fruit.col`	character (color of fruit bars)
`fruit.border`	character (color of fruit border bars)
`mar`	numeric (plot margins, vector of 4 values)

Details

It requires a data.frame with three columns, ordered as: species, month and phenology. The month column should be numeric (month number), while the phenology column must have these values: "Flower", "Fruit" and/or "Both". If any of these are missing is possible to indicate in the "flower", "fruit" and "both" arguments (both="missing"). The function will plot the bars indicating flower observations in white, and fruits in gray by default (is possible to change it with the "flower.col", "flower.border", "fruit.col" and "fruit.border" arguments). The size of the bar corresponds to number of observations. The arguments "shrink", "axis.cex" and "title.cex" control sizes, while the "mfrow" changes the number of histograms plotted at the same page (rows, columns).

Value

Exports a pdf file.

Author(s)

Marcelo Reginato

See Also

`circular`

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$phenoHist -> data
head(data)
```

```
## running the function

phenoHist(data, mfrow=c(2,2), shrink=1.2, axis.cex=1.5, title.cex=1.5,
pdf=FALSE)

## changing the color

phenoHist(data, mfrow=c(2,2), shrink=1.2, axis.cex=1.5, title.cex=1.5,
pdf=FALSE, flower.col=rgb(0.2,1,0.2, 0.5), flower.border=rgb(0.2,1,0.2,
0.5), fruit.col="darkgreen", fruit.border="black")

## plotting only flower (if "fruit" and/or "both" information are
## missing for instance)

phenoHist(data, mfrow=c(2,2), shrink=1.2, axis.cex=1.5, title.cex=1.5,
pdf=FALSE, fruit="missing", both="missing", flower.col="red",
flower.border="darkgray")
```

`tableToDescription` *Generates species descriptions*

Description

This function will generate a txt file with species descriptions.

Usage

```
tableToDescription(data, filename = "species_descriptions.txt")
```

Arguments

`data`	data.frame
`filename`	character

Details

It requires a data.frame where the first three columns are the character description, putative complement and the character to use as separator (i.e., words that will remain constant across descriptions). The character description and/or the complement might be empty. The remaining columns are the species with their respective character states, where each row is a character. The function accepts any number of species and/or characters.

Value

Exports a txt file

Author(s)

Marcelo Reginato

Examples

```
## loading the example data

data(monographaR_examples)
monographaR_examples$tableToDescription -> data
head(data)

## the first column is just an identifier for the characters, we need to
## remove it before running the analysis

data[,-1] -> data

## running the function, it will print in the terminal the output.
## To export a txt, place a name in the filename argument
## (i.e., filename = "myoutput.txt")

tableToDescription(data, filename = "")
```

Index