Package 'snippets'

September 26, 2024

Version 0.1-3

Title Code snippets, mostly visualization-related

Author Simon Urbanek <simon.urbanek@r-project.org>

Maintainer Simon Urbanek <simon.urbanek@r-project.org>

Imports png, jpeg

Description This package collects code snippets that I have written for one reason or another and are deemed useful enough to be shared. Most of them are related to visualization.

License MIT

URL http://www.rforge.net/snippets/

NeedsCompilation no

Contents

`add.layout` *Functions designed to create non-overlaping layout of rectangles or labels.*

Description

`add.layout` creates or adds to a layout of non-overlapping rectangles.

`add.labels` is a front-end to `add.layout` which creates a layout suitable for labels allowing to adjust the placement of the label with respect to the original points and to add extra margins.

Usage

```
add.layout(x, y, w, h, rs = 0.01, as = 0.22, ri = 0, ai = 0, l = NULL)
add.labels(x, y, txt, w, h, adj = 0.5, mar = 0.1, ...)
```

Arguments

x	x coordinates of the points to add. For rectangles it is the left edge.
y	y coordinates of the points to add. For rectangles it is the bottom edge.
w	width of the rectangles (optional for labels where it defaults to the label widths)
h	height of the rectangles (optional for labels where it defaults to the label height)
rs	radius step for the placement algorithm
as	angle step for the placement algorithm
ri	initial radius for the placement algorithm
ai	initial angle for the placement algorithm
l	layout to add to or NULL if a new layout is to be created
txt	text labels to add
adj	adjustment of the text position with respect to the supplied points (see adj in text)
mar	additional margins around the text (relative to the width or height). If present, it will be recycled to length two for horizontal and vertical margins respectively.
...	additional parameters passed through to add.layout

Details

The layout attempts to sequentially place rectangles of the given width and height at the specified points. If a placement of a new rectangle would overlap with any existing rectangle, it is moved until it no longer overlaps. The current algorithm defines the movement as a clockwise spiral.

Value

add.layout returns an obejct of the class box.layout which consists of a list with elements x, y, w and h. If the input was non-overlapping it would be equivalent to the (recycled) arguments. If there are overlaps the x and y coordinates will differ accordingly.

If l is specified on input, it is expected to be box.layout as well and the layout is extended by adding the rectangles defined in the arguments.

add.labels returns an object of the class label.layout which is a subclass of box.layout. It adds the components lx and ly which are the label coordinates (as opposed to the rectangle coordinates). If adj is zero the label and box coordinates are equal.

Examples

```
x = rnorm(100)
y = rnorm(100)
txt = sprintf("%.2f", rnorm(100))
plot(x, y, pch=3, col=2)
l = add.labels(x, y, txt, mar=0.2)
rect(l$x, l$y, l$x + l$w, l$y + l$h, col="#00000020", border=NA)
text(l$lx, l$ly, txt, col=4)
segments(x, y, l$lx, l$ly, col=3)
```

`cloud` *Word cloud (aka tag cloud) plot*

Description

`cloud` creates a siple word cloud plot

Usage

```
cloud(w, col, yspace=1.3, xspace=0.01, minh=0, ...)
```

Arguments

`w`	named weights - the acutal data to display. Names are used as tags, actual weight is used for the size. Note that plotting an unnamed vector results in an empty plot.
`col`	color for the tags.
`yspace`	space between lines as a multiple of the text height.
`xspace`	space (padding) between tags (in native coordinates).
`minh`	minimal height of a line.
`...`	optional arguments, currently ignored.

Details

The coordinates of the plot are [0, 1] in both x and y. The plot is created by starting in the upper left, proceedeing from left to right and top to bottom. The algorithm used is simply filling up

Note that resizing a world could usually destroys its properties. You'll have to re-run `cloud` after resizing, because it relies on exact extents of the text.

Value

Invisible TRUE.

Examples

```
# a really silly one
words <- c(apple=10, pie=14, orange=5, fruit=4)
cloud(words)

# or with more words - take the license for example
r <- paste(readLines(file.path(R.home(),"COPYING")), collapse=' ')
r <- gsub("[\f\t.,;:`'\\\"\\(\\)<>]+", " ", r)
w <- tolower(strsplit(r, " +")[[1]])
cloud(sqrt(table(w)))

# remove infrequent words
wt <- table(w)
wt <- log(wt[wt > 8])
cloud(wt, col = col.br(wt, fit=TRUE))
```

col *Flexible quantitative color schemes*

Description

`col.base` is the base for all other functions and allows full flexibility in defining arbitrary quantitative color schemes.

`col.bbr` provides blue-black-red diverging color scheme (suitable for text and borderless glyphs on white background).

`col.bwr` provides blue-white-red diverging color scheme (suitable for maps and gryphs with borders).

`col.br` provides blue-red color scheme.

`col.bw` provides black-white color scheme.

`col.q` returns transformed quantity between 0 and 1 for each datapoint (suitable for pass-though to other color schemes). `gray(col.q(...))` has the same effect (except for lack of alpha support) as `col.bw`.

Usage

```
col.base(x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA, FN)
col.bbr (x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA)
col.bwr (x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA)
col.br  (x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA)
col.bw  (x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA)
col.q   (x, alpha=1, lim=c(0, 1), center = mean(lim), fit=FALSE,
         trans="id", na.col=NA)
```

Arguments

`x`	data values (treated as numeric vector).
`alpha`	alpha value. It will be passed directly to the call to `rgb` except for `col.q` where is used to multiply the resulting value.
`lim`	data cut-off limits (range). Values outside this range will be clipped to the nearest value in the range.
`center`	center of the scale (mostly useful to calibrate center color of diverging scales).
`fit`	if set to `TRUE` then the data is shifted and scaled to fit the entire `lim` range.
`trans`	transformation of the resulting values. It can be either a function or one of the character strings `"id"`, `"sin"` or `"asin"`. The transformation function may not return values larger than 1 or smaller than 0.
`na.col`	color (or value when used in `col.q`) to use for missing values, will be used as-is (e.g., alpha is not applied).

FN — function accepting three arguments (a, b, alpha). a is mapped by a linear function descending from 1 to 0 between lim[1] and center. b is a mapped by the correspondingly increasing function between center and lim[2]. alpha is passed from the original function call.

Value

col.base returns the result of the FN function.

col.q returns a vector of numeric values, all other functions return a vector of colors as created by the rgb function.

Examples

```
plot(0:10, rep(0, 11), ylim=c(0,5), cex=3, pch=19, col=col.bbr(0:10, fit=TRUE))
points(0:10, rep(1,11), cex=3, pch=21, bg=col.br(0:10, fit=TRUE), col=1)
points(0:10, rep(2,11), cex=3, pch=21, bg=col.bw(0:10, fit=TRUE), col=1)
points(0:10, rep(3,11), cex=3, pch=21, bg=col.bwr(0:10, fit=TRUE, trans=sqrt),
 col=1)
points(0:10, rep(4,11), cex=3, pch=21, bg=col.bwr(0:10, fit=TRUE), col=1)
points(0:10, rep(5,11), cex=3, pch=21, bg=col.bwr(0:10, fit=TRUE,
 trans=function(x) x^2), col=1)
```

---

fd — *Fluctuation diagram*

Description

Draws a fluctuation diagram.

Usage

```
fd(x, ...)
## S3 method for class 'matrix'
fd(x, add = FALSE, vals = FALSE, at.x, at.y, axes = TRUE,
frame.plot = FALSE, main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
zmax = max(x, na.rm = TRUE), xlim, ylim, asp = 1, panel.first = NULL,
panel.last = NULL, ann = par("ann"), col = "grey", border = "black",
...)
## S3 method for class 'table'
fd(x, add = FALSE, vals = FALSE, at.x, at.y, axes = TRUE,
frame.plot = FALSE, main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
zmax = max(x, na.rm = TRUE), xlim, ylim, asp = 1, panel.first = NULL,
panel.last = NULL, ann = par("ann"), col = "grey", border = "black",
...)
```

Arguments

x — object to draw fluctuation diagram of (most commonly a table)

add — a logical value indicating whether to add to an existing plot (TRUE) or to create a new plot (FALSE).

vals — a logical value indicating whether to draw values into the rectangles (discouraged and unimplemented).

at.x	locations of the colums (by default 1:ncol)
at.y	locations of the rows (by default 1:nrow)
axes	a logical value indicating whether both axes should be drawn on the plot. Use graphical parameter xaxt or yaxt to suppress just one of the axes.
frame.plot	a logical indicating whether a box should be drawn around the plot.
main	a main title for the plot, see also title.
sub	a subtitle for the plot.
xlab	a label for the x axis.
ylab	a label for the y axis.
zmax	value representing the total size of an allocated box.
xlim	the x limits (x1, x2) of the plot. The default is the range of at.x with an additional 0.5 margin of the ends.
ylim	the y limits of the plot.
asp	the y/x aspect ratio, see plot.window.
panel.first	an expression to be evaluated after the plot axes are set up but before any plotting takes place. This can be useful for drawing background grids
panel.last	an expression to be evaluated after plotting has taken place.
ann	see "ann" graphical parameter
col	color of the boxes to be filled with, will be recycled to match the shape of x.
border	color of the box borders - only scalar value is supported at the moment.
...	additional graphical parameters

Value

Returns (invisibly) a data frame describing the sparse representation of the boxes as location and radius.

Examples

```
## this is best viewed on a wide-screen device...
par(mfrow=c(1,2))
for (sex in dimnames(HairEyeColor)$Sex)
  fd(HairEyeColor[,,sex], main=sex)
```

---

gpx *Create a GPX (GPS Exchange Format) output.*

Description

gpx creates XML output in the GPX format (GPS Exchange Format) from latitude, longitude and time.

Usage

```
gpx(lat, lon, time, file = NULL)
```

Arguments

lat	latitude of the points
lon	longitude of the points
time	time of the points (optional)
file	destination - can be a character string naming the output file or a connection to use for output or NULL in which case the function return a character vector with teh output.

Details

The resulting output is in GPX format contining exactly one track with the specified values (NAs are currently not supported!). If the time value is present then the track entries will include a time nodes as well. No checking is done on time entries so the user must ensure that they are exactly of the form YYYY-MM-DD HH:MM:SS, assumed to be in UTC.

(Note that OSM requires time stamps to be present in uploaded tracks.)

Value

If file is NULL then the value is a character vector containing the lines of the GPX output.

Examples

```
lat <- c(40.779, 40.777)
lon <- c(-74.428,-74.418)
cat(gpx(lat, lon), sep='\n')
cat(gpx(lat, lon, Sys.time()), sep='\n')
```

---

mfrow *Recursive grid layout functions for graphics*

Description

mfrow and mfcol setup a multi-figure layout scheme on a grid in the current graphics device. Unlike other layout methods the schemes can be used recursively to create complex layouts with very simple commands.

Usage

```
mfrow(rows, cols, n, asp = 1, add = FALSE, times = NA,
      fig = if (add) par("fig") else c(0, 1, 0, 1), ...)
mfcol(rows, cols, n, asp = 1, add = FALSE, times = NA,
      fig = if (add) par("fig") else c(0, 1, 0, 1), ...)
```

Arguments

rows	either a numeric vector of length one defining the number of equally spaced rows or a vector specifying the relative height of each row.
cols	either a numeric vector of length one defining the number of equally spaced columns or a vector specifying the relative width of each row.

n	if rows and cols are both not specified, then n specifies the minimal number of cells that the layout should contain. In that case rows and cols are computed such that their product is at least n and the resulting aspect ratio of the figures is as close to asp as possible.
asp	numeric, desired aspect ratio of the figures when n is used to specify the grid
add	logical, if TRUE then the layout scheme is added to the layout stack, allowing recursive layouts. Otherwise the currently layout is replaced with the new grid layout.
times	number of times this layout should be applied or NA for unlimited. Any number larger than 1e6 is interpreted as NA.
fig	boundaries of the figure that will be split using this layout scheme.
...	additional arguments that will be passed to par when a new figure is setup.

Details

mfrow and mfcol have a similar effect as the corresponding graphics parameters, but they are more flexible (allowing individual withs and heights) and construct a scheme that can be used recursively with add = TRUE.

Note that the scheme layout method is incompatible with all other layout methods that are not based on the scheme stack, such as par(mfrow) or layout. It can be to a degree combined with split.screen with add = TRUE if used as a sub-layout thereof since screen uses a cooperative manipulation of figures. However, in that case you should make sure that the stack layout does not overflow (i.e., you plot only at most as many plots as there are figures) or it will clear the device.

Value

object of the class scheme.

Note

All layout functions require R 2.13.0 or higher! If plots seemingly don't move from the first figure then you either have old version or other code has removed the neessary "before.plot.new" hook.

Author(s)

Simon Urbanek

Examples

```
## one advantage is that you can simply specify the number of plots
## the layout will then depend on the aspect ratio of the device
mfrow(n=8)
par(mar=c(2,2,0,0))
for (i in 1:8) plot(rnorm(100), rnorm(100), col=i, pch=19)

## or you can create recursive layouts
mfrow(2, 4)                   ## 4 x 2 base grid
mfrow(c(1, 4), 1, add=TRUE)   ## each cell split asymetrically 1:4

par(mar=rep(0,4))
for (i in 1:8) {
   plot(0:1, 0:1, type='n', axes=FALSE); rect(0,0,1,1); text(0.5, 0.5, i)
```

```
    plot(rnorm(100), rnorm(100), col=i, pch=19, axes=FALSE)
}
```

`osm-tools` *Tools converting from lat/lon coordinates to OSM tiles and back.*

Description

`osm.ll2xy` converts lat/lon coordinates to OpenStreetMap tile numbers. `osm.xy2ll` performs the inverse conversion.

Usage

```
osm.xy2ll(x, y, zoom = 16)
osm.ll2xy(lon, lat, zoom = 16)
```

Arguments

`x`	number of the tile in the x direction
`y`	number of the tile in the y direction
`lon`	longitude
`lat`	latitude
`zoom`	zoom factor of the tiles

Details

`osm.ll2xy` computes tile numbers for given latitude and longitude, `osm.xy2ll` does the inverse.

Value

`osm.ll2xy` returns a list with the components `x` and `y`

`osm.xy2ll` returns a list with the components `lon` and `lat`

See Also

`osmap`

`osmap` *Draw tiles from OpenStreetMap or any other compatible tile source.*

Description

`osmap` fetches and draws OpenStreetMap tiles (or any compatible tiles) in the current graphics device assuming latitude and longitude coordinates.

Usage

```
osmap(alpha = 1, zoom, area = par()$usr, tiles.url, cache.dir,
      tile.coord = FALSE)
```

Arguments

`alpha`	in theory this is the desired opacity of the map but in practice the final plot is faded to white by this alpha value (1 = no fading, 0 = entriely white)
`zoom`	zoom level of the map to draw (optional). If missing the zoom level is determined automatically to cover the area with about five tiles in the horizontal direction
`area`	minimal area to cover - the required tiles are computed to cover at least this area
`tiles.url`	URL to the tiles server (excluding the `zoom/x/y.png` part). If missing, `osm.tiles.url` option is consulted and if also missing then an OSM tile server is used.
`cache.dir`	if set, the tiles are first looked up in this directory and the directory is used for caching downloaded tiles
`tile.coord`	if `FALSE` then the plot coordinates are assumed to be latitude and longitude - this is the default. Otherwise this must be an integer and it is assumed that the coordinates of the plot are tile coordinates at the specified `tile.coord` zoom level (this is useful for plotting data projected in the same Mercator projection as the tiles). Note taht `zoom` and `tile.coord` can be different zoom levels.

Examples

```
par(mar=rep(0,4))
# plot any lat/lon data - here just an area around the AT&T Labs
plot(c(-74.44, -74.39), c(40.76, 40.79), type='n')
# draw the map (needs internet connection to get the tiles)
osmap()
# to draw the world, use zoom level=0 as the base so that
# the area is simply [0,1]
plot(c(0,1), c(0,1), type='n')
osmap(tile.coord=0L)
```

rfBin *Read binary file*

Description

`rfBin` is a front-end to `readBin` that reads the entire content of a binary file.

Usage

```
rfBin(filename, what = 1L, ...)
```

Arguments

`filename`	name of the file to read
`what`	either an integer or real vector defining the payload type
`...`	additional parameters passed to `readBin`

Value

Same as `readBin`

Author(s)

Simon Urbanek

scheme *Scheme hanlding functions*

Description

The following functions manage the stack of layout "schemes" created by corresponding scheme constructors such as `mfrow` or `mfcol`. Most of them are called by the implementation and are not expected to be used by the user directly.

`advance.scheme` is called when a new plot is about to be drawn and advances to the next scheme layout according to the stack hierarchy.

`pop.scheme` removes the topmost scheme from the stack.

Usage

```
advance.scheme()
pop.scheme()
## S3 method for class 'scheme'
print(x, ...)
```

Arguments

`x`	scheme to be printed
`...`	additional arguments passed through

Note

"scheme" is not a typo but rather a play on the meaning of the word 'scheme' in the context of something that may be called 'schema' (a grand plan of figure layout if you will) – the former is an evolution of the latter word from its original form, anyway.

Author(s)

Simon Urbanek

See Also

`mfrow`, `mfcol`

`screen2data` *Functions converting between pixel and data coordinates.*

Description

`screen2data` converts between screen (pixel) coordinates of device output and the data coordinates.

`data2screen` performs the inverse conversion.

Both functions can be parametrized manually but default to obtaining all necessary parameters from the active graphics device. It is most useful when used with bitmap dvices such as the Cairo device (see `Cairo` package) for the purpose of adding interactivity (e.g., via JavaScript).

Usage

```
screen2data(x, y, width = par("fin")[1] * dpi, height = par("fin")[2] * dpi,
            dpi = 72, plt = par("plt"), usr = par("usr"), flip = FALSE)
data2screen(x, y, width = par("fin")[1] * dpi, height = par("fin")[2] * dpi,
            dpi = 72, plt = par("plt"), usr = par("usr"), flip = FALSE)
```

Arguments

`x`	x coordinates of locations to convert or a two-column matrix (if `y` is missing)
`y`	y coordinates of locations to convert (if missing `x` must be a matrix and the second column of `x` is interpreted as `y`)
`width`	width of the figure region (usually the size of the resulting file in pixels)
`height`	height of the figure region (usually the size of the resulting file in pixels)
`dpi`	resolution (only used to compute the width and height from figure size if they are not specified
`plt`	the 'plt' parameter
`usr`	the 'usr' parameter
`flip`	if set to `TRUE` then `y` axis in pixels is assumed to be flipped (0 on top)

Value

The result is a two-column matrix with the columns `x` and `y`. The special case of one row input is returned as a named vector of length two.

Note

If x and y are vectors they are recycled to match.

Examples

```
plot(0:1,0:1)
## where on the bitmap is the data point 0,0 ?
data2screen(0, 0)
## if I click on 200, 100 with flipped coordinates, what coordinates do
## I hit?
screen2data(200, 100, flip=TRUE)
## go there and back
screen2data(data2screen(c(0, 0.5), c(1, 0.5)))
```

---

setup.figure *Method defining a layout scheme*

Description

Layout schemes are defined by the scheme object and the implmentation of the setup.figure method for that object.

The purpose of setup.figure is to set the "fig" graphical parameter according to the state represented by the scheme object. This allows implementation of arbitrary layout schemes.

Usage

```
setup.figure(scheme)
## S3 method for class 'gridColScheme'
setup.figure(scheme)
## S3 method for class 'gridRowScheme'
setup.figure(scheme)
```

Arguments

scheme	scheme defining the current state which is to be reflected in the graphics parameters

Details

The scheme will contain the enclosing figure region in scheme$fig and it is up to the setup.figure method implementation to use the advancement index scheme$index to determine the appropriate region to set the "fig" graphics parameter. Clearly, the scheme object can contain any additonal necessary needed for the method to perform its function.

For example, the grid layout schemes keep the matrix of the grid in the scheme object and use simple modulo operation to determine the approriate column and row to set "fig" accordingly.

Value

scheme

Author(s)

Simon Urbanek

Index