

Package ‘circlizePlus’

July 22, 2025

Title Using 'ggplot2' Feature to Write Readable R Code for Circular Visualization

Version 0.9.1

Description A wrapper for 'circlize'. All components are based on classes and objects. Users can use the addition symbol (+) to combine components for a circular visualization with 'ggplot2' style. The package is described in Zhang Z, Cao T, Huang Y and Xia Y (2025) <[doi:10.3389/fgene.2025.1535368](https://doi.org/10.3389/fgene.2025.1535368)>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Collate 'track.R' 'heatmap.R' 'cell-geom.R' 'track-geom.R' 'link.R' 'param.R' 'utils.R' 'initialize.R' 'add.R' 'circlizePlus-package.R' 'data.R'

Depends circlize (>= 0.4.16), R (>= 4.4.0)

Imports methods, stats

Suggests png, ape, dendextend, ComplexHeatmap, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

URL <https://github.com/TianzeLab/circlizePlus>,
<https://tianzelab.github.io/circlizePlus/>,
<https://doi.org/10.3389/fgene.2025.1535368>

BugReports <https://github.com/TianzeLab/circlizePlus/issues>

VignetteBuilder knitr

NeedsCompilation no

Author Tianze Cao [aut, cre] (ORCID: <<https://orcid.org/0000-0003-4308-1878>>),
Zheyu Zhang [aut]

Maintainer Tianze Cao <hnrcao@qq.com>

Repository CRAN

Date/Publication 2025-05-23 09:22:02 UTC

Contents

addition-rules	3
ccArrow	4
ccBarplot	5
ccBoxplot	6
ccCell	8
ccCell-class	8
ccCellGeom-class	9
ccCells	9
ccCells-class	10
ccDendrogram	10
ccGenomicAxis	11
ccGenomicCellGeom-class	12
ccGenomicDensity	13
ccGenomicHeatmap	14
ccGenomicIdeogram	15
ccGenomicLabels	16
ccGenomicLines	18
ccGenomicLink	19
ccGenomicLink-class	20
ccGenomicPoints	21
ccGenomicRainfall	22
ccGenomicRect	23
ccGenomicText	24
ccGenomicTrack	26
ccGenomicTrack-class	27
ccHeatmap	28
ccHeatmap-class	30
ccHeatmapLink	31
ccHeatmapLink-class	31
ccLines	32
ccLink	33
ccLink-class	35
ccPar	36
ccPar-class	36
ccPlot	37
ccPlot-class	39
ccPoints	39
ccPolygon	40
ccRaster	41
ccRect	42
ccSegments	43
ccText	44
ccTrack	45
ccTrack-class	47
ccTrackGeom-class	48
ccTrackHist	48

ccTrackLines	50
ccTrackPoints	51
ccTrackText	52
ccViolin	53
ccXaxis	54
ccYaxis	56
data-set	57
show,ccHeatmap-method	58
show,ccPlot-method	59

Index	61
--------------	-----------

addition-rules	<i>Addition rules in circlizePlus</i>
----------------	---------------------------------------

Description

ccPlot(contain n ccTracks)+ccTrack=ccPlot(contain n+1 ccTracks),n>=0 ccPlot(contain n ccLinks)+ccLink=ccPlot(contain n+1 ccLinks),n>=0 ccTrak(contain n ccTrakGeoms)+ccTrackGeom=ccTrack(contain n+1 ccTrack-Geoms),n>=0 ccTrack(contain n ccCells)+ccCell=ccTrack(contain n+1 ccCells),n>=0 ccCell(contain n ccCellGeoms)+ccCellGeom=ccCell(contain n+1 ccCellGeoms),n>=0

Usage

```
## S4 method for signature 'ccPlot,ccPar'
e1 + e2

## S4 method for signature 'ccPlot,ccTrack'
e1 + e2

## S4 method for signature 'ccPlot,ccLink'
e1 + e2

## S4 method for signature 'ccTrack,ccTrackGeom'
e1 + e2

## S4 method for signature 'ccTrack,ccCells'
e1 + e2

## S4 method for signature 'ccTrack,ccCell'
e1 + e2

## S4 method for signature 'ccCell,ccCellGeom'
e1 + e2

## S4 method for signature 'ccCells,ccCellGeom'
e1 + e2
```

```
## S4 method for signature 'ccHeatmap,ccPar'
e1 + e2

## S4 method for signature 'ccHeatmap,ccTrack'
e1 + e2

## S4 method for signature 'ccHeatmap,ccLink'
e1 + e2
```

Arguments

e1 A object defined in circlizePlus
e2 A object defined in circlizePlus

Value

A object defined in circlizePlus

Examples

```
NULL
```

ccArrow

Draw an arrow

Description

Object `ccCellGeom` will call the function `circlize::circos.arrow` while drawing.

Usage

```
ccArrow(
  x1,
  x2,
  y,
  width,
  arrow.head.length = NULL,
  arrow.head.width = width * 2,
  arrow.position = c("end", "start"),
  tail = c("normal", "point"),
  border = "black",
  col = "#FFCCCC",
  lty = par("lty"),
  ...
)
```

Arguments

<code>x1</code>	Start position of the arrow on the x-axis.
<code>x2</code>	End position of the arrow on the x-axis. Note <code>x2</code> should be larger than <code>x1</code> . The direction of arrows can be controlled by <code>arrow.position</code> argument.
<code>y</code>	Position of the arrow on the y-axis. Note this is the center of the arrow on y-axis.
<code>width</code>	Width of the arrow body.
<code>arrow.head.length</code>	Length of the arrow head. Note the value should be smaller than the length of the arrow itself (which is <code>x2 - x1</code>).
<code>arrow.head.width</code>	Width of the arrow head.
<code>arrow.position</code>	Where is the arrow head on the arrow. If you want to the arrow in the reversed direction, set this value to "start".
<code>tail</code>	The shape of the arrow tail (the opposite side of arrow head).
<code>border</code>	Border color of the arrow.
<code>col</code>	Filled color of the arrow.
<code>lty</code>	Line style of the arrow.
<code>...</code>	Pass to polygon .

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:4], xlim = c(0, 10))
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "a") + ccArrow(x1 = 1, x2 = 9, y=0.5, width=0.5)
track <- track + cell
cc + track
```

ccBarplot

Draw barplots

Description

Object [ccCellGeom](#) will call the function [circlize::circos.barplot](#) while drawing.

Usage

```
ccBarplot(
  value,
  pos,
  bar_width = 0.6,
  col = NA,
  border = "black",
  lwd = par("lwd"),
  lty = par("lty")
)
```

Arguments

value	A numeric vector or a matrix. If it is a matrix, columns correspond to the height of bars.
pos	Positions of the bars.
bar_width	Width of bars. It assumes the bars locating at $x = 1, 2, \dots$
col	Filled color of bars.
border	Color for the border.
lwd	Line width.
lty	Line style.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:4], xlim = c(0, 10))
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "a") + ccBarplot(value = runif(10), pos = 1:10 - 0.5, col = 1:10)
track <- track + cell
cc + track
```

ccBoxplot

Draw boxplots

Description

Object [ccCellGeom](#) will call the function [circlize::circos.boxplot](#) while drawing.

Usage

```
ccBoxplot(
  value,
  pos,
  outline = TRUE,
  box_width = 0.6,
  col = NA,
  border = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  cex = par("cex"),
  pch = 1,
  pt.col = par("col")
)
```

Arguments

value	A numeric vector, a matrix or a list. If it is a matrix, boxplots are made by columns (each column is a box).
pos	Positions of the boxes.
outline	Whether to draw outliers.
box_width	Width of boxes.
col	Filled color of boxes.
border	Color for the border as well as the quantile lines.
lwd	Line width.
lty	Line style
cex	Point size.
pch	Point type.
pt.col	Point color.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:4], xlim = c(0, 10))
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "a") + ccBoxplot(value = replicate(runif(10),
n = 10, simplify = FALSE), pos = 1:10 - 0.5, col = 1:10)
track <- track + cell
cc + track
```

ccCell	<i>Generate a cell container that belongs to a particular sector</i>
--------	--

Description

Generate a cell container that belongs to a particular sector

Usage

```
ccCell(sector.index = NULL)
```

Arguments

`sector.index` character. It is the index that corresponds to the sector.

Value

Object [ccCell](#)

Examples

```
library(circlizePlus)
sectors <- c("a", "a", "a", "a", "b", "b", "b", "b", "c", "c", "c", "c", "d", "d", "d", "d")
x1 <- c(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
y1 <- c(1, 2, 3, 4, 4, 3, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2)
cc <- ccPlot(initMode = "initialize", sectors = sectors, x = x1)
track1 <- ccTrack(sectors = sectors, x = x1, y = y1)
cell_single <- ccCell(sector.index = letters[3]) + ccPoints(y = \(x, y){
  y
})
track1 <- track1 + cell_single
cc + track1
```

ccCell-class	<i>S4 class ccCell</i>
--------------	------------------------

Description

A cell container that belongs to a particular sector.

Slots

`sector.index` character. It is the index that corresponds to the sector.

`geoms` list. The elements in the list should all be of type [ccCellGeom](#) or [ccGenomicCellGeom](#).

Examples

```
NULL
```

ccCellGeom-class	<i>S4 class ccCellGeom</i>
------------------	----------------------------

Description

Objectified representation of the R package `circlize`'s plotting functions and corresponding parameters at the cell level.

Slots

`func` character. The name of the plot function in the R package `circlize`.

`params` list. When the function corresponding to the parameter `param` is called, it represents the argument of this function.

Examples

NULL

ccCells	<i>Generate a list of multiple object ccCell-class</i>
---------	--

Description

Generate a list of multiple object `ccCell`-class

Usage

```
ccCells(sector.indexes = list())
```

Arguments

`sector.indexes` list. A list of indexes that corresponds to the sectors.

Value

Object `ccCells`

Examples

```
library(circlizePlus)
sectors <- c("a", "a", "a", "a", "b", "b", "b", "b", "c", "c", "c", "c", "d", "d", "d", "d")
x1 <- c(1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)
y1 <- c(1, 2, 3, 4, 4, 3, 2, 1, 1, 1, 1, 1, 2, 1, 2)
cc <- ccPlot(initMode = "initialize", sectors = sectors, x = x1)
cells <- ccCells(sector.indexes = letters[1:4])
cc_point <- ccPoints()
cells <- cells + cc_point + ccLines()
```

```
track1 <- ccTrack(sectors = sectors, x = x1, y = y1)
track1 <- track1 + cells
cc + track1
```

ccCells-class

S4 class ccCells

Description

A list of multiple ccCell. Any ccCellGeom and ccCells are added together as if they were added to each ccCell contained in the ccCells.

Examples

```
NULL
```

ccDendrogram

Draw dendrogram plots in a track

Description

Object [ccCellGeom](#) will call the function [circlize::circo.dendrogram](#) while drawing.

Usage

```
ccDendrogram(
  dend,
  facing = c("outside", "inside"),
  max_height = NULL,
  use_x_attr = FALSE
)
```

Arguments

dend	A dendrogram object.
facing	Is the dendromgrams facing inside to the circle or outside?
max_height	Maximum height of the dendrogram. This is important if more than one dendrograms are drawn in one track and making them comparable. The height of a dendrogram can be obtained by <code>attr(dend, "height")</code> .
use_x_attr	Whether use the x attribute to determine node positions in the dendrogram, used internally.

Value

Object [ccCellGeom](#)

Examples

```
library(ape)
suppressPackageStartupMessages(library(dendextend))
library(circlizePlus)
data(bird.orders)
hc <- as.hclust(bird.orders)
labels <- hc$labels
ct <- cutree(hc, 6)
n <- length(labels)
dend <- as.dendrogram(hc)
par1 <- ccPar(cell.padding = c(0, 0, 0, 0))
cc <- ccPlot(sectors = "a", xlim = c(0, n)) # only one sector
dend <- color_branches(dend, k = 6, col = 1:6)
dend_height <- attr(dend, "height")
t1 <- ccTrack(ylim = c(0, dend_height), bg.border = NA, track.height = 0.4)
cell1 <- ccCell(sector.index = "a") + ccDendrogram(dend = dend)
cc + par1 + (t1 + cell1)
```

ccGenomicAxis

Add genomic axes

Description

Object `ccGenomicCellGeom` will call the function `circlize::circos.genomicAxis` while drawing.

Usage

```
ccGenomicAxis(
  h = "top",
  major.at = NULL,
  labels = NULL,
  major.by = NULL,
  tickLabelsStartFromZero = TRUE,
  labels.cex = 0.4 * par("cex"),
  ...
)
```

Arguments

<code>h</code>	Position of the axes. "top" or "bottom".
<code>major.at</code>	Major breaks. If <code>major.at</code> is set, <code>major.by</code> is ignored.
<code>labels</code>	labels corresponding to <code>major.at</code> . If <code>labels</code> is set, <code>major.at</code> must be set.
<code>major.by</code>	Increment of major ticks. It is calculated automatically if the value is not set (about every 10 degrees there is a major tick).
<code>tickLabelsStartFromZero</code>	Whether axis tick labels start from 0? This will only affect the axis labels while not affect x-values in cells.

labels.cex The font size for the axis tick labels.
 ... Other arguments pass to `circos.axis`.

Value

Object `ccGenomicCellGeom`

Examples

```
library(circlizePlus)
cc <- ccPlot(initMode = "initializeWithIdeogram",
  chromosome.index = paste0("chr", 1:4), plotType = NULL)
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "chr1") + ccGenomicAxis()
e <- track + cell
cc + e
```

ccGenomicCellGeom-class

S4 class ccGenomicCellGeom

Description

It is a subclass of `ccCellGeom`. It only works if the plotted data is genomic data. Objectified representation of the R package `circlize`'s plotting functions and corresponding parameters at the cell level.

Slots

func character. The name of the plot function in the R package `circlize`.

params list. When the function corresponding to the parameter `param` is called, it represents the argument of this function.

Examples

NULL

ccGenomicDensity	Create a track of density plot
------------------	--------------------------------

Description

Object `ccGenomicTrack` will call the function `circlize::circos.genomicDensity` while drawing.

Usage

```
ccGenomicDensity(
  data,
  ylim.force = FALSE,
  window.size = NULL,
  overlap = TRUE,
  count_by = c("percent", "number"),
  col = ifelse(area, "grey", "black"),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  area = TRUE,
  area.baseline = NULL,
  baseline = 0,
  border = NA,
  ...
)
```

Arguments

<code>data</code>	A bed-file-like data frame or a list of data frames. If the input is a list of data frames, there will be multiple density plot in one same track.
<code>ylim.force</code>	Whether to force upper bound of <code>ylim</code> to be 1. Ignored if <code>count_by</code> is set to <code>number</code> .
<code>window.size</code>	Pass to <code>genomicDensity</code> .
<code>overlap</code>	Pass to <code>genomicDensity</code> .
<code>count_by</code>	Pass to <code>genomicDensity</code> .
<code>col</code>	Colors. It should be length of one. If data is a list of data frames, the length of <code>col</code> can also be the length of the list. If multiple sets of genomic regions are visualized in one single track, you should set the colors with transparency to distinguish them.
<code>lwd</code>	Width of lines, the same setting as <code>col</code> argument.
<code>lty</code>	Style of lines, the same setting as <code>col</code> argument.
<code>type</code>	Type of lines, see <code>circos.lines</code> .
<code>area</code>	See <code>circos.lines</code> .
<code>area.baseline</code>	Deprecated, use <code>baseline</code> instead.

baseline See [circos.lines](#).
border See [circos.lines](#).
... Pass to [circos.trackPlotRegion](#).

Value

Object [ccGenomicTrack](#)

Examples

```
library(circlizePlus)
load(system.file(package = "circlize", "extdata", "DMR.RData"))
cc = ccPlot(initMode="initializeWithIdeogram", chromosome.index = paste0("chr", 1:22))
t2 = ccGenomicDensity(DMR_hyper, col = c("#FF000080"), track.height = 0.1)
t3 = ccGenomicDensity(DMR_hypo, col = c("#0000FF80"), track.height = 0.1)
cc + t2 + t3
circos.clear()
```

ccGenomicHeatmap

Define a heatmap track for genomic graph

Description

Object [ccGenomicTrack](#) will call the function [circlize::circos.genomicHeatmap](#) while drawing.

Usage

```
ccGenomicHeatmap(
  bed,
  col,
  na_col = "grey",
  numeric.column = NULL,
  border = NA,
  border_lwd = par("lwd"),
  border_lty = par("lty"),
  connection_height = mm_h(5),
  line_col = par("col"),
  line_lwd = par("lwd"),
  line_lty = par("lty"),
  heatmap_height = 0.15,
  side = c("inside", "outside"),
  track.margin = circos.par("track.margin")
)
```

Arguments

bed	A data frame in bed format, the matrix should be stored from the fourth column.
col	Colors for the heatmaps. The value can be a matrix or a color mapping function generated by colorRamp2 .
na_col	Color for NA values.
numeric.column	Column index for the numeric columns. The values can be integer index or character index. By default it takes all numeric columns from the fourth column.
border	Border of the heatmap grids.
border_lwd	Line width for borders of heatmap grids.
border_lty	Line style for borders of heatmap grids.
connection_height	Height of the connection lines. If it is set to NULL, no connection will be drawn. Use mm_h/cm_h/inches_h to set a height in absolute unit.
line_col	Color of the connection lines. The value can be a vector.
line_lwd	Line width of the connection lines.
line_lty	Line style of the connection lines.
heatmap_height	Height of the heatmap track
side	Side of the heatmaps. Is the heatmap facing inside or outside?
track.margin	Bottom and top margins.

Value

Object [ccGenomicTrack](#)

Examples

```
library(circlizePlus)
cc = ccPlot(initMode = "initializeWithIdeogram")
bed = generateRandomBed(nr = 100, nc = 4)
col_fun = colorRamp2(c(-1, 0, 1), c("green", "black", "red"))
t1 = ccGenomicHeatmap(bed, col = col_fun, side = "inside", border = "white")
cc + t1
circos.clear()
```

ccGenomicIdeogram	<i>Define an ideograms track for genomic graph</i>
-------------------	--

Description

Object [ccGenomicTrack](#) will call the function [circlize::circos.genomicIdeogram](#) while drawing.

Usage

```
ccGenomicIdeogram(
  cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"),
  species = NULL,
  track.height = mm_h(2),
  track.margin = circos.par("track.margin")
)
```

Arguments

cytoband	A data frame or a file path, pass to read.cytoband .
species	Abbreviations of the genome, pass to read.cytoband .
track.height	Height of the ideogram track.
track.margin	Margins for the track.

Value

Object [ccGenomicTrack](#)

Examples

```
library(circlizePlus)
cc = ccPlot(initMode = "initializeWithIdeogram", chromosome.index = "chr1", plotType = NULL)
human_cytoband = read.cytoband(species = "hg19")$df
t2=ccGenomicIdeogram(human_cytoband)
cc+t2
circos.clear()
```

ccGenomicLabels

Add labels to specific genomic track

Description

Object [ccGenomicTrack](#) will call the function [circlize::circos.genomicLabels](#) while drawing.

Usage

```
ccGenomicLabels(
  bed,
  labels = NULL,
  labels.column = NULL,
  facing = "clockwise",
  niceFacing = TRUE,
  col = par("col"),
  cex = 0.8,
  font = par("font"),
```

```
padding = 0.4,
connection_height = mm_h(5),
line_col = par("col"),
line_lwd = par("lwd"),
line_lty = par("lty"),
labels_height = NULL,
side = c("inside", "outside"),
labels.side = side,
track.margin = circos.par("track.margin")
)
```

Arguments

bed	A data frame in bed format.
labels	A vector of labels corresponding to rows in bed.
labels.column	If the label column is already in bed, the index for this column in bed.
facing	ffacing of the labels. The value can only be "clockwise" or "reverse.clockwise".
niceFacing	Whether automatically adjust the facing of the labels.
col	Color for the labels.
cex	Size of the labels.
font	Font of the labels.
padding	Padding of the labels, the value is the ratio to the height of the label.
connection_height	Height of the connection track.
line_col	Color for the connection lines.
line_lwd	Line width for the connection lines.
line_lty	Line type for the connection lines.
labels_height	Height of the labels track.
side	Side of the labels track, is it in the inside of the track where the regions are marked?
labels.side	Same as side. It will replace side in the future versions.
track.margin	Bottom and top margins.

Value

Object [ccGenomicTrack](#)

Examples

```
library(circlizePlus)
bed = generateRandomBed(nr = 50, fun = function(k) sample(letters, k, replace = TRUE))
bed[1, 4] = "aaaaa"
cc = ccPlot(initMode = "initializeWithIdeogram", plotType = NULL)
t1 = ccGenomicLabels(bed, labels.column = 4, side = "outside",
                     col = as.numeric(factor(bed[[1]])), line_col = as.numeric(factor(bed[[1]])))
cc + t1
circos.clear()
```

ccGenomicLines

*Add lines for genomic data visualization***Description**

Object `ccGenomicCellGeom` will call the function `circlize::circos.genomicLines` while drawing.

Usage

```
ccGenomicLines(
  region = NULL,
  value = NULL,
  numeric.column = NULL,
  posTransform = NULL,
  col = ifelse(area, "grey", "black"),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch"),
  ...
)
```

Arguments

<code>region</code>	A data frame contains 2 column which correspond to start positions and end positions.
<code>value</code>	A data frame contains values and other information.
<code>numeric.column</code>	Which column in value data frame should be taken as y-value. If it is not defined, the whole numeric columns in value will be taken.
<code>posTransform</code>	Self-defined function to transform genomic positions, see posTransform.default for explanation.
<code>col</code>	col of lines/areas. If there are more than one numeric column, the length of col can be either one or number of numeric columns. If there is only one numeric column and type is either segment or h, the length of col can be either one or number of rows of region. pass to circos.lines
<code>lwd</code>	Settings are similar as col. Pass to circos.lines .
<code>lty</code>	Settings are similar as col. Pass to circos.lines .
<code>type</code>	There is an additional option segment which plot segment lines from start position to end position. Settings are similar as col. Pass to circos.lines .

area	Settings are similar as col. Pass to circos.lines .
area.baseline	Deprecated, use baseline instead.
border	Settings are similar as col. Pass to circos.lines .
baseline	Settings are similar as col. Pass to circos.lines .
pt.col	Settings are similar as col. Pass to circos.lines .
cex	Settings are similar as col. Pass to circos.lines .
pch	Settings are similar as col. Pass to circos.lines .
...	Mysterious parameters.

Value

Object [ccGenomicCellGeom](#)

Examples

```
library(circlizePlus)
data <- generateRandomBed(nr = 30, nc = 2)
all_chr <- c("chr1", "chr2", "chr3", "chr4", "chr5", "chr6", "chr7", "chr8",
"chr9", "chr10", "chr11", "chr12", "chr13", "chr14", "chr15", "chr16",
"chr17", "chr18", "chr19", "chr20", "chr21", "chr22", "chrX", "chrY")
cc <- ccPlot(initMode = "initializeWithIdeogram", plotType = NULL)
t1 <- ccGenomicTrack(data = data, numeric.column = 4)
cells1 <- ccCells(sector.indexes = all_chr) +
ccGenomicLines(numeric.column = 2)
t1 <- t1 + cells1
show(cc + t1)
```

ccGenomicLink

Add links between two sets of genomic positions

Description

Object [ccGenomicLink](#) will call the function [circlize::circos.genomicLink](#) while drawing.

Usage

```
ccGenomicLink(
  region1,
  region2,
  rou = NULL,
  rou1 = rou,
  rou2 = rou,
  col = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  border = col,
  ...
)
```

Arguments

region1	A data frame in bed format.
region2	A data frame in bed format.
rou	Pass to circos.link .
rou1	Pass to circos.link .
rou2	Pass to circos.link .
col	Pass to circos.link , length can be either one or nrow of region1.
lwd	Pass to circos.link , length can be either one or nrow of region1.
lty	Pass to circos.link , length can be either one or nrow of region1.
border	Pass to circos.link , length can be either one or nrow of region1.
...	Pass to circos.link .

Value

Object [ccGenomicLink](#)

Examples

```
library(circlizePlus)
set.seed(123)

bed1 = generateRandomBed(nr = 100)
bed1 = bed1[sample(nrow(bed1), 20), ]
bed2 = generateRandomBed(nr = 100)
bed2 = bed2[sample(nrow(bed2), 20), ]
par1 = ccPar("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
cc = ccPlot(initMode="initializeWithIdeogram")

link1 = ccGenomicLink(bed1, bed2, col = sample(1:5, 20, replace = TRUE), border = NA)
cc + par1 + link1
```

ccGenomicLink-class *S4 class ccGenomicLink*

Description

S4 class ccGenomicLink

Slots

func character. Normally it is "circos.genomicLink".
 params list. A **named** list that stores the parameters of the function [circlize::circos.genomicLink](#) called by the backend.

Examples

NULL

ccGenomicPoints	<i>Add points for genomic data visualization</i>
-----------------	--

Description

Object `ccGenomicCellGeom` will call the function `circlize::circo.genomicPoints` while drawing.

Usage

```
ccGenomicPoints(
  region = NULL,
  value = NULL,
  numeric.column = NULL,
  posTransform = NULL,
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg"),
  ...
)
```

Arguments

<code>region</code>	A data frame contains 2 columns which correspond to start positions and end positions.
<code>value</code>	A data frame contains values and other information.
<code>numeric.column</code>	Which column in value data frame should be taken as y-value. If it is not defined, the whole numeric columns in value will be taken.
<code>posTransform</code>	Self-defined function to transform genomic positions, see <code>posTransform.default</code> for explanation
<code>pch</code>	Type of points. Settings are similar as <code>col</code> . Pass to <code>circo.points</code> .
<code>col</code>	Color of points. If there is only one numeric column, the length of <code>col</code> can be either one or number of rows of <code>region</code> . If there are more than one numeric column, the length of <code>col</code> can be either one or number of numeric columns. Pass to <code>circo.points</code> .
<code>cex</code>	Size of points. Settings are similar as <code>col</code> . Pass to <code>circo.points</code> .
<code>bg</code>	Background colors for points.
<code>...</code>	Mysterious parameters.

Value

Object `ccGenomicCellGeom`

Examples

```
library(circlizePlus)
data <- generateRandomBed(nr = 30, nc = 2)
all_chr <- c("chr1", "chr2", "chr3", "chr4", "chr5", "chr6", "chr7", "chr8",
"chr9", "chr10", "chr11", "chr12", "chr13", "chr14", "chr15", "chr16",
"chr17", "chr18", "chr19", "chr20", "chr21", "chr22", "chrX", "chrY")
cc <- ccPlot(initMode = "initializeWithIdeogram", plotType = NULL)
t1 <- ccGenomicTrack(data = data, numeric.column = 4)
cells1 <- ccCells(sector.indexes = all_chr) +
ccGenomicPoints(region = \(region, value){
  region
}, value = \(region, value){
  value
}, numeric.column = 2)
t1 <- t1 + cells1
show(cc + t1)
```

ccGenomicRainfall	Create a rainfall plot
-------------------	------------------------

Description

Object `ccGenomicTrack` will call the function `circlize::circos.genomicRainfall` while drawing.

Usage

```
ccGenomicRainfall(
  data,
  mode = "min",
  ylim = NULL,
  col = "black",
  pch = par("pch"),
  cex = par("cex"),
  normalize_to_width = FALSE,
  ...
)
```

Arguments

<code>data</code>	A bed-file-like data frame or a list of data frames.
<code>mode</code>	How to calculate the distance of two neighbouring regions, pass to rainfallTransform .
<code>ylim</code>	<code>ylim</code> for rainfall plot track. If <code>normalize_to_width</code> is FALSE, the value should correspond to $\log_{10}(\text{dist}+1)$, and if <code>normalize_to_width</code> is TRUE, the value should correspond to $\log_2(\text{rel_dist})$.
<code>col</code>	Color of points. It should be length of one. If <code>data</code> is a list, the length of <code>col</code> can also be the length of the list.
<code>pch</code>	Style of points.

cex Size of points.
 normalize_to_width If it is TRUE, the value is the relative distance divided by the width of the region.
 ... Pass to `circos.trackPlotRegion`.

Value

Object `ccGenomicTrack`

Examples

```

library(circlizePlus)
load(system.file(package = "circlize", "extdata", "DMR.RData"))
cc = ccPlot(initMode="initializeWithIdeogram", chromosome.index = paste0("chr", 1:22))
bed_list = list(DMR_hyper, DMR_hypo)
t1 = ccGenomicRainfall(bed_list, pch = 16, cex = 0.4, col = c("#FF000080", "#0000FF80"))
cc + t1
circos.clear()

```

ccGenomicRect

Draw rectangle for genomic data visualization

Description

Object `ccGenomicCellGeom` will call the function `circlize::circos.genomicRect` while drawing.

Usage

```

ccGenomicRect(
  region = NULL,
  value = NULL,
  ytop = NULL,
  ybottom = NULL,
  ytop.column = NULL,
  ybottom.column = NULL,
  posTransform = NULL,
  col = NA,
  border = "black",
  lty = par("lty"),
  ...
)

```

Arguments

region	A data frame contains 2 column which correspond to start positions and end positions.
value	A data frame contains values and other information.
ytot	A vector or a single value indicating top position of rectangles.
ybottom	A vector or a single value indicating bottom position of rectangles.
ytot.column	If ytot is in value, the index of the column.
ybottom.column	If ybottom is in value, the index of the column.
posTransform	Self-defined function to transform genomic positions, see posTransform.default for explanation.
col	The length of col can be either one or number of rows of region. Pass to circos.rect .
border	Settings are similar as col. Pass to circos.rect .
lty	Settings are similar as col. Pass to circos.rect .
...	Mysterious parameters.

Value

Object [ccGenomicCellGeom](#)

Examples

```
library(circlizePlus)
par1 <- ccPar("track.height" = 0.1, cell.padding = c(0, 0, 0, 0))
cc <- ccPlot(initMode = "initializeWithIdeogram", plotType = NULL)
bed1 <- generateRandomBed(nr = 100)
bed2 <- generateRandomBed(nr = 100)
bed_list <- list(bed1, bed2)
f <- colorRamp2(breaks = c(-1, 0, 1), colors = c("green", "black", "red"))
track1 <- ccGenomicTrack(data = bed_list, stack = TRUE)
all_chr <- c("chr1", "chr2", "chr3", "chr4", "chr5", "chr6", "chr7", "chr8",
"chr9", "chr10", "chr11", "chr12", "chr13", "chr14", "chr15", "chr16",
"chr17", "chr18", "chr19", "chr20", "chr21", "chr22", "chrX", "chrY")
rect1 <- ccGenomicRect(col = 1, border = NA)
cells1 <- ccCells(sector.indexes = all_chr) + rect1
cc + par1 + (track1 + cells1)
```

ccGenomicText

Add text for genomic data visualization

Description

Object [ccGenomicCellGeom](#) will call the function [circlize::circos.genomicText](#) while drawing.

Usage

```
ccGenomicText(
  region = NULL,
  value = NULL,
  y = NULL,
  labels = NULL,
  labels.column = NULL,
  numeric.column = NULL,
  posTransform = NULL,
  direction = NULL,
  facing = "inside",
  niceFacing = FALSE,
  adj = par("adj"),
  cex = 1,
  col = "black",
  font = par("font"),
  padding = 0,
  extend = 0,
  align_to = "region",
  ...
)
```

Arguments

region	A data frame contains 2 column which correspond to start positions and end positions.
value	A data frame contains values and other information.
y	A vector or a single value indicating position of text.
labels	Labels of text corresponding to each genomic positions.
labels.column	If labels are in value, index of column in value.
numeric.column	Which column in value data frame should be taken as y-value. If it is not defined, only the first numeric columns in value will be taken.
posTransform	Self-defined function to transform genomic positions, see posTransform.default for explanation.
direction	Deprecated, use facing instead.
facing	Passing to circos.text . Settings are similar as col.
niceFacing	Should the facing of text be adjusted to fit human eyes?
adj	Pass to circos.text . Settings are similar as col.
cex	Pass to circos.text . Settings are similar as col.
col	Pass to circos.text . The length of col can be either one or number of rows of region.
font	Pass to circos.text . Settings are similar as col.
padding	pass to posTransform if it is set as posTransform.text .
extend	pass to posTransform if it is set as posTransform.text .

`align_to` pass to `posTransform` if it is set as `posTransform.text`.
`...` Mysterious parameters.

Value

Object `ccGenomicCellGeom`

Examples

```
library(circlizePlus)
cc <- ccPlot(initMode = "initializeWithIdeogram", plotType = NULL)
bed <- generateRandomBed(nr = 20)
track1 <- ccGenomicTrack(data = bed, ylim = c(0, 1))
all_chr <- c("chr1", "chr2", "chr3", "chr4", "chr5", "chr6", "chr7", "chr8",
"chr9", "chr10", "chr11", "chr12", "chr13", "chr14", "chr15", "chr16",
"chr17", "chr18", "chr19", "chr20", "chr21", "chr22", "chrX", "chrY")
text1 <- ccGenomicText(y = 0.5, labels = "text")
cells1 <- ccCells(sector.indexes = all_chr) + text1
cc + (track1 + cells1)
```

<code>ccGenomicTrack</code>	<i>Define a track for genomic data visualization</i>
-----------------------------	--

Description

Object `ccGenomicTrack` will call the function `circlize::circos.genomicTrackPlotRegion` while drawing.

Usage

```
ccGenomicTrack(
  data = NULL,
  ylim = NULL,
  stack = FALSE,
  numeric.column = NULL,
  jitter = 0,
  panel.fun = function(region, value, ...) {
    NULL
  },
  ...
)
```

Arguments

<code>data</code>	A bed-file-like data frame or a list of data frames
<code>ylim</code>	If it is <code>NULL</code> , the value will be calculated from <code>data</code> . If <code>stack</code> is set to <code>TRUE</code> , this value is ignored.
<code>stack</code>	whether to plot in a "stack" mode.

<code>numeric.column</code>	Columns of numeric values in data that will be used for plotting. If data is a data frame list, <code>numeric.column</code> should be either length of one or length of data. If value of <code>numeric.column</code> is not set, its value will depend on the structure of data. If data is a data frame, the default value for <code>numeric.column</code> is all the numeric column starting from the fourth column. If data is a list of data frame, the default value for <code>numeric.column</code> is a vector which have the same length as data and the value in default <code>numeric.column</code> is the index of the first numeric column in corresponding data frame.
<code>jitter</code>	Numeric. Only works for adding points in <code>circos.genomicTrackPlotRegion</code> under stack mode
<code>panel.fun</code>	Self-defined function which will be applied on each sector. Please not it is different from that in circos.trackPlotRegion . In this function, there are two arguments (region and value) plus ... In them, region is a two-column data frame with start positions and end positions in current genomic category (e.g. chromosome). value is a data frame which is derived from data but excluding the first three columns. Rows in value correspond to rows in region. ... is mandatory and is used to pass internal parameters to other functions. The definition of value will be different according to different input data (data frame or list of data frame) and different settings (stacked or not), please refer to 'details' section and vignettes to detailed explanation.
...	Pass to circos.trackPlotRegion .

Value

Object [ccGenomicTrack](#)

Examples

```
library(circlizePlus)
cc = ccPlot(initMode = "initializeWithIdeogram", chromosome.index = "chr1", plotType = NULL)
bed = generateRandomBed(nr = 300)
t1 = ccGenomicTrack(bed, panel.fun = function(region, value, ...) {
  circos.genomicPoints(region, value, pch = 16, cex = 0.5, ...)
})
cc+t1
circos.clear()
```

ccGenomicTrack-class *S4 class ccGenomicTrack*

Description

S4 class `ccGenomicTrack`

Slots

func character. Normally it is "circos.genomicTrack" or "circos.genomicIdeogram" or "circos.genomicHeatmap" or "circos.genomicLabels" or "circos.genomicRainfall" or "circos.genomicDensity".

params list. A **named** list that stores the parameters of the function called by the backend.

trackGeoms list. A list where [ccTrackGeom](#) are stored.

cells list. A list where [ccCell](#) are stored.

Examples

```
library(circlizePlus)
cc = ccPlot(initMode = "initializeWithIdeogram", chromosome.index = "chr1", plotType = NULL)
bed = generateRandomBed(nr = 300)
t1 = ccGenomicTrack(bed, panel.fun = function(region, value, ...) {
  circos.genomicPoints(region, value, pch = 16, cex = 0.5, ...)
})
cc+t1
circos.clear()
```

ccHeatmap

Object generator for S4 class ccHeatmap

Description

Object [ccHeatmap](#) will call the function [circlize::circos.heatmap](#) while drawing.

Usage

```
ccHeatmap(
  mat,
  split = NULL,
  col,
  na.col = "grey",
  cell.border = NA,
  cell.lty = 1,
  cell.lwd = 1,
  bg.border = NA,
  bg.lty = par("lty"),
  bg.lwd = par("lwd"),
  ignore.white = is.na(cell.border),
  cluster = TRUE,
  clustering.method = "complete",
  distance.method = "euclidean",
  dend.callback = function(dend, m, si) reorder(dend, rowMeans(m)),
  dend.side = c("none", "outside", "inside"),
  dend.track.height = 0.1,
  rownames.side = c("none", "outside", "inside"),
```

```

    rownames.cex = 0.5,
    rownames.font = par("font"),
    rownames.col = "black",
    show.sector.labels = FALSE,
    cell_width = rep(1, nrow(mat)),
    clear = TRUE,
    ...
)

```

Arguments

<code>mat</code>	A matrix or a vector. The vector is transformed as a one-column matrix.
<code>split</code>	A categorical variable. It splits the matrix into a list of matrices.
<code>col</code>	If the values in the matrices are continuous, the color should be a color mapping generated by colorRamp2 . If the values are characters, the color should be a named color vector.
<code>na.col</code>	Color for NA values.
<code>cell.border</code>	Border color of cells. A single scalar.
<code>cell.lty</code>	Line type of cell borders. A single scalar.
<code>cell.lwd</code>	Line width of cell borders. A single scalar.
<code>bg.border</code>	Color for background border.
<code>bg.lty</code>	Line type of the background border.
<code>bg.lwd</code>	Line width of the background border.
<code>ignore.white</code>	Whether to draw the white color?
<code>cluster</code>	whether to apply clustering on rows. The value can also be a dendrogram/hclust object or other objects that can be converted to with as.dendrogram .
<code>clustering.method</code>	Clustering method, pass to hclust .
<code>distance.method</code>	Distance method, pass to dist .
<code>dend.callback</code>	A callback function that is applied to the dendrogram in every sector.
<code>dend.side</code>	Side of the dendrograms relative to the heatmap track.
<code>dend.track.height</code>	Track height of the dendrograms.
<code>rownames.side</code>	Side of the row names relative to the heatmap track.
<code>rownames.cex</code>	Cex of row names.
<code>rownames.font</code>	Font of row names.
<code>rownames.col</code>	Color of row names.
<code>show.sector.labels</code>	Whether to show sector labels.
<code>cell_width</code>	Relative widths of heatmap cells.
<code>clear</code>	Whether to call circlize::circos.clear before drawing.
<code>...</code>	Pass to circos.track which draws the heatmap track.

Value

Object [ccHeatmap](#)

Examples

```
library(circlizePlus)
set.seed(123)
mat1 <- rbind(
  cbind(
    matrix(rnorm(50 * 5, mean = 1), nr = 50),
    matrix(rnorm(50 * 5, mean = -1), nr = 50)
  ),
  cbind(
    matrix(rnorm(50 * 5, mean = -1), nr = 50),
    matrix(rnorm(50 * 5, mean = 1), nr = 50)
  )
)
rownames(mat1) <- paste0("R", 1:100)
colnames(mat1) <- paste0("C", 1:10)
mat1 <- mat1[sample(100, 100), ] # randomly permute rows
split <- sample(letters[1:5], 100, replace = TRUE)
split <- factor(split, levels = letters[1:5])
col_fun1 <- colorRamp2(c(-2, 0, 2), c("blue", "white", "red"))
ccHeatmap(mat = mat1, split = split, col = col_fun1)
```

ccHeatmap-class

S4 class ccHeatmap

Description

ccHeatmap is a special class. It can be used not only as a single track but also as the result of adding a heatmap track to a ccPlot

Slots

func character. Normally it is "circos.heatmap".

params list. A **named** list that stores the parameters of the function [circlize::circos.heatmap](#) called by the backend.

trackGeoms list. A list where [ccTrackGeom](#) are stored.

cells list. A list where [ccCell](#) are stored.

tracks list. A list where [ccTrack](#) or [ccGenomicTrack](#) or [ccHeatmap](#) are stored.

links list. A list where [ccLink](#) or [ccGenomicLink](#) or [ccHeatmapLink](#) are stored.

pars list. A list where [ccPar](#) are stored.

clear logical. Whether to call [circlize::circos.clear](#) before drawing.

ccHeatmapLink	<i>Draw a link between two matrix rows in the circular heatmap</i>
---------------	--

Description

Object `ccHeatmapLink` will call the function `circlize::circos.heatmap.link` while drawing.

Usage

```
ccHeatmapLink(row_from, row_to, ...)
```

Arguments

<code>row_from</code>	The row index where the link starts. The value should be length 1. If you want to draw multiple links, put the function in a for loop.
<code>row_to</code>	The row index where the link ends.
<code>...</code>	Pass to <code>circos.link</code> .

Value

Object `ccHeatmapLink`

Examples

```
library(circlizePlus)
set.seed(123)
mat = matrix(rnorm(100*10), nrow = 100)
rownames(mat) = paste0("R", 1:100)
col_fun = colorRamp2(c(-2, 0, 2), c("blue", "white", "red"))
cc = ccHeatmap(mat, col = col_fun, rownames.side = "outside")
link1 = ccHeatmapLink(10, 60)
cc + link1
```

ccHeatmapLink-class	<i>S4 class ccHeatmapLink</i>
---------------------	-------------------------------

Description

S4 class `ccHeatmapLink`

Slots

`func` character. Normally it is "circos.heatmap.link".

`params` list. A **named** list that stores the parameters of the function `circlize::circos.heatmap.link` called by the backend.

Examples

NULL

ccLines

Draw lines in a region

Description

Object `ccCellGeom` will call the function `circlize::circos.lines` while drawing.

Usage

```
ccLines(
  x = NULL,
  y = NULL,
  col = ifelse(area, "grey", par("col")),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  straight = FALSE,
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch")
)
```

Arguments

<code>x</code>	Data points on x-axis, measured in "current" data coordinate.
<code>y</code>	Data points on y-axis, measured in "current" data coordinate.
<code>col</code>	Line color.
<code>lwd</code>	Line width.
<code>lty</code>	Line style.
<code>type</code>	Line type, similar as <code>type</code> argument in <code>lines</code> , but only in <code>c("l", "o", "h", "s")</code>
<code>straight</code>	Whether draw straight lines between points.
<code>area</code>	Whether to fill the area below the lines. If it is set to TRUE, <code>col</code> controls the filled color in the area and <code>border</code> controls color of the line.
<code>area.baseline</code>	deprecated, use <code>baseline</code> instead.
<code>border</code>	color for border of the area.

baseline	The base line to draw areas. By default it is the minimal of y-range (bottom). It can be a string or a number. If a string, it should be one of bottom and top. This argument also works if type is set to h.
pt.col	If type is "o", point color.
cex	If type is "o", point size.
pch	If type is "o", point type.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
sectors <- letters[1:9]
par <- ccPar(points.overflow.warning = FALSE)
cc <- ccPlot(sectors = sectors, xlim = c(0, 10))
cc <- cc + par
track <- ccTrack(sectors = sectors, ylim = c(0, 10), track.height = 0.5)
cells <- ccCell(sector.index = "a") + ccLines(sort(x = runif(10) * 10), y = runif(10) * 10)
track <- track + cells
cc + track
```

ccLink

Add a link

Description

Object [ccLink](#) will call the function [circlize::circos.link](#) while drawing.

Usage

```
ccLink(
  sector.index1,
  point1,
  sector.index2,
  point2,
  rou = NULL,
  rou1 = rou,
  rou2 = rou,
  h = NULL,
  h.ratio = 0.5,
  w = 1,
  h2 = h,
  w2 = w,
  inverse = FALSE,
  col = "black",
```

```

lwd = par("lwd"),
lty = par("lty"),
border = col,
directional = 0,
arr.length = ifelse(arr.type == "big.arrow", 0.02, 0.4),
arr.width = arr.length/2,
arr.type = "triangle",
arr.lty = lty,
arr.lwd = lwd,
arr.col = col,
reduce_to_mid_line = FALSE
)

```

Arguments

sector.index1	Index for the first sector where one link end locates
point1	A single value or a numeric vector of length 2. If it is a 2-elements vector, then the link would be a belt/ribbon.
sector.index2	Index for the other sector where the other link end locates
point2	A single value or a numeric vector of length 2. If it is a 2-elements vector, then the link would be a belt/ribbon.
rou	The position of the the link ends (if rou1 and rou2 are not set). It is the percentage of the radius of the unit circle. By default its value is the position of bottom margin of the most inner track.
rou1	The position of end 1 of the link.
rou2	The position of end 2 of the link.
h	Height of the link, measured as percent to the radius to the unit circle. By default it is automatically infered.
h.ratio	systematically change the link height. The value is between 0 and 1.
w	Since the link is a Bezier curve, it controls the shape of Bezier curve.
h2	Height of the bottom edge of the link if it is a ribbon.
w2	Shape of the bottom edge of the link if it is a ribbon.
inverse	Whether the link is inversed.
col	Color of the link. If the link is a ribbon, then it is the filled color for the ribbon.
lwd	Line (or border) width
lty	Line (or border) style
border	If the link is a ribbon, then it is the color for the ribbon border.
directional	0 for no direction, 1 for direction from point1 to point2, -1 for direction from point2 to point1. 2 for two directional. The direction is important when arrow heads are added.
arr.length	Length of the arrows, measured in 'cm', pass to Arrowhead . If arr.type is set to big.arrow, the value is percent to the radius of the unit circle.
arr.width	Width of the arrows, pass to Arrowhead .

arr.type	Type of the arrows, pass to Arrowhead . Default value is triangle. There is an additional option big.arrow.
arr.lty	Line type of arrows, pass to Arrowhead .
arr.lwd	Line width of arrows, pass to Arrowhead .
arr.col	Color of the arrows, pass to Arrowhead .
reduce_to_mid_line	Only use the middle points of point1 and point2 to draw the link.

Value

Object [ccLink](#)

Examples

```
library(circlizePlus)
set.seed(999)
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE), x = rnorm(n), y = runif(n))
cc = ccPlot(initMode = "initialize", sectors = df$sectors, x = df$x)
track1 = ccTrack(df$sectors, y = df$y)
col = rep(c("#FF0000", "#00FF00"), 4)
tPoint1 = ccTrackPoints(df$sectors, df$x, df$y, col = col, pch = 16, cex = 0.5)
link1 = ccLink("a", 0, "b", 0, h = 0.4)
link2 = ccLink("c", c(-0.5, 0.5), "d", c(-0.5, 0.5), col = "red", border = "blue", h = 0.2)
link3 = ccLink("e", 0, "g", c(-1, 1), col = "green", border = "black", lwd = 2, lty = 2)
cc + (track1 + tPoint1) + link1 + link2 + link3
```

ccLink-class

S4 class ccLink

Description

S4 class ccLink

Slots

func character. Normally it is "circos.link".

params list. A **named** list that stores the parameters of the function [circlize::circos.link](#) called by the backend.

Examples

NULL

ccPar	<i>Parameters for the circular layout</i>
-------	---

Description

Object `ccPar` will call the function `circlize::circos.par` while drawing.

Usage

```
ccPar(...)
```

Arguments

```
...           Arguments passed on to circlize::circos.par
RESET reset to default values
READ.ONLY please ignore
LOCAL please ignore
ADD please ignore
```

Value

Object `ccPar`

Examples

```
library(circlizePlus)
set.seed(999)
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE), x = rnorm(n), y = runif(n))
par1 = ccPar("track.height" = 0.1)
cc = ccPlot(initMode = "initialize", sectors = df$sectors, x = df$x)
track1 = ccTrack(df$sectors, y = df$y)
col = rep(c("#FF0000", "#00FF00"), 4)
tPoint1 = ccTrackPoints(df$sectors, df$x, df$y, col = col, pch = 16, cex = 0.5)
cc + par1 + (track1 + tPoint1)
```

ccPar-class	<i>Object generator for S4 class ccPar</i>
-------------	--

Description

Object generator for S4 class `ccPar`

Slots

params params list. A **named** list that stores the parameters of the function `circlize::circos.par` called by the backend.

Examples

NULL

ccPlot	<i>Object generator for S4 class ccPlot</i>
--------	---

Description

Object **ccPlot** calls one of the following functions based on the value of `initMode`: [circlize::circos.initialize](#), [circlize::circos.genomicInitialize](#), [circlize::circos.initializeWithIdeogram](#), [circlize::circos.heatmap.initialize](#). The correct way to call it is as follows: `ccPlot(initMode = 'initialize', clear = TRUE, sectors = NULL, x = NULL, xlim = NULL, sector.width = NULL, factors = sectors, ring = FALSE) ccPlot(initMode = 'genomicInitialize', clear = TRUE, data=NULL, sector.names = NULL, major.by = NULL, plotType = c("axis", "labels"), tickLabelsStartFromZero = TRUE, axis.labels.cex = 0.4*par("cex"), labels.cex = 0.8*par("cex"), track.height = NULL, ...)` `ccPlot(initMode = 'initializeWithIdeogram', clear = TRUE, cytoband = system.file(package = "circlize", "extdata", "cytoBand.txt"), species = NULL, sort.chr = TRUE, chromosome.index = usable_chromosomes(species), major.by = NULL, plotType = c("ideogram", "axis", "labels"), track.height = NULL, ideogram.height = convert_height(2, "mm"), ...)` `ccPlot(initMode = 'heatmap.initialize', clear = TRUE, mat=NULL, split = NULL, cluster = TRUE, clustering.method = "complete", distance.method = "euclidean", dend.callback = function(dend, m, si) reorder(dend, rowMeans(m)), cell_width = rep(1, nrow(mat)))`

Usage

```
ccPlot(initMode = "initialize", clear = TRUE, ...)
```

Arguments

<code>initMode</code>	It can only be the following values: "initialize", "genomicInitialize", "initializeWithIdeogram", "heatmap.initialize".
<code>clear</code>	Whether to call circlize::circos.clear before drawing.
<code>...</code>	Arguments passed on to circlize::circos.initialize , circlize::circos.genomicInitialize , circlize::circos.initializeWithIdeogram , circlize::circos.heatmap.initialize
<code>sectors</code>	A factor variable or a character vector which represent data categories
<code>factors</code>	The same as <code>sectors</code> . It will be removed in future versions.
<code>x</code>	Data on x-axes, a vector
<code>xlim</code>	Ranges for values on x-axes, see "details" section for explanation of the format
<code>sector.width</code>	Width for each sector. The length of the vector should be either 1 which means all sectors have same width or as same as the number of sectors. Values for the vector are relative, and they will be scaled by dividing their summation. By default, it is NULL which means the width of sectors correspond to the data range in sectors.

ring Whether the sector represented as a ring. If yes, there should only be one sector in the circle.

data A data frame in bed format.

sector.names Labels for each sectors which will be drawn along each sector. It will not modify values of sector index.

major.by Increment of major ticks. It is calculated automatically if the value is not set (about every 10 degrees there is a major tick).

plotType If it is not NULL, there will create a new track containing axis and names for sectors. This argument controls which part should be drawn, axis for genomic axis and labels for chromosome names

tickLabelsStartFromZero Whether axis tick labels start from 0? This will only affect the axis labels while not affect x-values in cells.

axis.labels.cex The font size for the axis tick labels.

labels.cex The font size for the labels.

track.height If PlotType is not NULL, height of the annotation track.

cytoband A path of the cytoband file or a data frame that already contains cytoband data. By default it is cytoband for hg19. Pass to [read.cytoband](#).

species Abbreviations of species. e.g. hg19 for human, mm10 for mouse. If this value is specified, the function will download cytoBand.txt.gz from UCSC website automatically. If there is no cytoband for user's species, it will keep on trying to download chromInfo file. Pass to [read.cytoband](#) or [read.chromInfo](#).

chromosome.index subset of chromosomes, also used to reorder chromosomes.

sort.chr Whether chromosome names should be sorted (first sort by numbers then by letters). If chromosome.index is set, this argument is enforced to FALSE

ideogram.height Height of the ideogram track

mat A matrix or a vector. The vector is transformed as a one-column matrix.

split A categorical variable. It splits the matrix into a list of matrices.

cluster whether to apply clustering on rows. The value can also be a dendrogram/hclust object or other objects that can be converted to with [as.dendrogram](#).

clustering.method Clustering method, pass to [hclust](#).

distance.method Distance method, pass to [dist](#).

dend.callback A callback function that is applied to the dendrogram in every sector.

cell_width Relative widths of heatmap cells.

Value

Object [ccPlot](#)

Examples

```
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
                x = rnorm(n), y = runif(n))
```

```
library(circlizePlus)
cc=ccPlot(initMode = 'initialize', sectors = df$sectors, x = df$x)
```

ccPlot-class	<i>S4 class ccPlot</i>
--------------	------------------------

Description

S4 class ccPlot

Slots

initMode character. It can only be the following values: "initialize", "genomicInitialize", "initializeWithIdeogram", "heatmap.initialize".

initParams list. A **named** list that stores the parameters of the function called by the backend. Based on the value of initMode, the backend function will be one of the following four: [circlize::circos.initialize](#), [circlize::circos.genomicInitialize](#), [circlize::circos.initializeWithIdeogram](#), [circlize::circos.heatmap.initialize](#).

tracks list. A list where [ccTrack](#) or [ccGenomicTrack](#) or [ccHeatmap](#) are stored.

links list. A list where [ccLink](#) or [ccGenomicLink](#) or [ccHeatmapLink](#) are stored.

pars list. A list where [ccPar](#) are stored.

clear logical. Whether to call [circlize::circos.clear](#) before drawing.

Examples

```
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
                x = rnorm(n), y = runif(n))
library(circlizePlus)
cc=ccPlot(initMode = 'initialize', sectors = df$sectors, x = df$x)
```

ccPoints	<i>Draw points in a region</i>
----------	--------------------------------

Description

Object [ccCellGeom](#) will call the function [circlize::circos.points](#) while drawing.

Usage

```
ccPoints(
  x = NULL,
  y = NULL,
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg")
)
```

Arguments

x	Data points on x-axis, measured in "current" data coordinate
y	Data points on y-axis, measured in "current" data coordinate
pch	Point type
col	Point color
cex	Point size
bg	background of points

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track1 <- ccTrack(ylim = c(0, 1), panel.fun = function(x, y) {
  circos.points(runif(10), runif(10))
})
cells <- ccCell(sector.index = "a") + ccPoints(
  x = runif(10), y = runif(10),
  pch = 16, col = "red"
)
track1 <- track1 + cells
cc + track1
```

ccPolygon

Draw polygon

Description

Object [ccCellGeom](#) will call the function [circlize::circos.polygon](#) while drawing.

Usage

```
ccPolygon(x = NULL, y = NULL, ...)
```

Arguments

x	Data points on x-axis
y	Data points on y-axis
...	pass to polygon

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track <- ccTrack(ylim = c(0, 10))
cell <- ccCell(sector.index = "a") + ccPolygon(x = c(0.5, 0.7, 1), y = c(2, 6, 8))
track <- track + cell
cc + track
```

ccRaster

Add raster image

Description

Object `ccCellGeom` will call the function `circlize::circos.raster` while drawing.

Usage

```
ccRaster(
  image,
  x,
  y,
  width,
  height,
  facing = c("inside", "outside", "reverse.clockwise", "clockwise", "downward",
    "bending.inside", "bending.outside"),
  niceFacing = FALSE,
  scaling = 1
)
```

Arguments

image	A raster object, or an object that can be converted by <code>as.raster</code> .
x	Position of the center of the raster image, measured in the data coordinate in the cell.
y	Position of the center of the raster image, measured in the data coordinate in the cell.
width	Width of the raster image. When facing is one of "inside", "outside", "clockwise" and "reverse.clockwise", the image should have absolute size where the value of width should be specified like 20mm, 1cm or 0.5inche. When facing is one of bending.inside and bending.outside, the value of width is measured in the data coordinate in the cell.
height	Height of the raster image. Same format as width. If the value of height is omit, default height is calculated by taking the aspect ratio of the original image. But when facing is one of bending.inside and bending.outside, height is mandatory to set.
facing	Facing of the raster image.

niceFacing Facing of text. Please refer to vignette for different settings.
 scaling Scaling factor to resize the raster image.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
library(png)
image <- system.file("extdata", "Rlogo.png", package = "circlize")
image <- as.raster(readPNG(image))
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:4], xlim = c(0, 10))
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "a") + ccRaster(image = image, x = 5, y = 0.5,
width = "2cm", height = "2cm", facing = "inside", niceFacing = TRUE)
track <- track + cell
cc + track
```

ccRect

Draw rectangle in a region

Description

Object [ccCellGeom](#) will call the function [circlize::circos.rect](#) while drawing.

Usage

```
ccRect(xleft = NULL, ybottom = NULL, xright = NULL, ytop = NULL, rot = 0, ...)
```

Arguments

xleft	x for the left bottom points
ybottom	y for the left bottom points
xright	x for the right top points
ytop	y for the right top points
rot	Rotation of the rectangles. The value is measured clockwise in degree. Rotation is relative to the center of the rectangles.
...	pass to polygon

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track <- ccTrack(ylim = c(0, 1), track.height = 0.3)
cell <- ccCell(sector.index = "a") + ccRect(xleft = 0.7, ybottom = 0.1, xright = 0.8, ytop = 0.9)
track <- track + cell
cc + track
```

ccSegments

*Draw segments connecting points in a region***Description**

Object `ccCellGeom` will call the function `circlize::circos.segments` while drawing.

Usage

```
ccSegments(
  x0 = NULL,
  y0 = NULL,
  x1 = NULL,
  y1 = NULL,
  straight = FALSE,
  col = par("col"),
  lwd = par("lwd"),
  lty = par("lty"),
  ...
)
```

Arguments

<code>x0</code>	x coordinates for starting points.
<code>y0</code>	y coordinates for ending points.
<code>x1</code>	x coordinates for starting points.
<code>y1</code>	y coordinates for ending points.
<code>straight</code>	Whether the segment is a straight line.
<code>col</code>	Color of the segments.
<code>lwd</code>	Line width of the segments.
<code>lty</code>	Line type of the segments.
<code>...</code>	Pass to <code>lines</code> .

Value

Object `ccCellGeom`

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track <- ccTrack(ylim = c(0, 1), track.height = 0.3)
cell <- ccCell(sector.index = "a") + ccSegments(x0 = 0.7, y0 = 0.1, x1 = 0.7, y1 = 0.9)
track <- track + cell
cc + track
```

ccText

Draw text in a cell

Description

Object `ccCellGeom` will call the function `circlize::circos.text` while drawing.

Usage

```
ccText(
  x = NULL,
  y = NULL,
  labels,
  direction = NULL,
  facing = c("inside", "outside", "reverse.clockwise", "clockwise", "downward",
    "bending", "bending.inside", "bending.outside"),
  niceFacing = FALSE,
  adj = par("adj"),
  cex = 1,
  col = par("col"),
  font = par("font"),
  ...
)
```

Arguments

x	Data points on x-axis
y	Data points on y-axis
labels	Labels for each points
direction	deprecated, use facing instead.
facing	Facing of text. Please refer to vignette for different settings
niceFacing	Should the facing of text be adjusted to fit human eyes?
adj	offset for text. By default the text position adjustment is either horizontal or vertical in the canvas coordinate system. The "circular horizontal" offset can be set as a value in degree unit and the value should be wrapped by degree .
cex	Font size
col	Font color
font	Font style
...	Pass to text

ValueObject [ccCellGeom](#)**Examples**

```

library(circlizePlus)
n <- 1000
df <- data.frame(
  sectors = sample(letters[1:8], n, replace = TRUE),
  x = rnorm(n), y = runif(n)
)
par1 <- ccPar("track.height" = 0.1)
cc <- ccPlot(sectors = df$sectors, x = df$x) + par1
track1 <- ccTrack(
  sectors = df$sectors, y = df$y,
  panel.fun = function(x, y) {
    circos.text(
      CELL_META$xcenter,
      CELL_META$cell.ylim[2] + mm_y(5),
      CELL_META$sector.index
    )
    circos.axis(labels.cex = 0.6)
  }
)
cell1 <- ccCell(sector.index = "a") + ccText(-1, 0.5, "text")
track1 <- track1 + cell1
cc <- cc + track1
cc

```

ccTrack

*Define a generic track***Description**

Object [ccTrack](#) will call the function [circlize::circos.trackPlotRegion](#) while drawing.

Usage

```

ccTrack(
  sectors = NULL,
  x = NULL,
  y = NULL,
  ylim = NULL,
  force.ylim = TRUE,
  track.index = NULL,
  track.height = circos.par("track.height"),
  track.margin = circos.par("track.margin"),
  cell.padding = circos.par("cell.padding"),
  bg.col = NA,

```

```

    bg.border = "black",
    bg.lty = par("lty"),
    bg.lwd = par("lwd"),
    panel.fun = function(x, y) {
      NULL
    },
    factors = sectors
  )

```

Arguments

sectors	A factor or a character vector which represents categories of data, if it is NULL, then it uses all sector index.
x	Data on x-axis. It is only used if <code>panel.fun</code> is set.
y	Data on y-axis
ylim	Range of data on y-axis
force.ylim	Whether to force all cells in the track to share the same <code>ylim</code> . Normally, all cells on a same track should have same <code>ylim</code> .
track.index	Index for the track which is going to be created/updated. If the specified track has already been created, this function just updated corresponding track with new plot. If the specified track is NULL or has not been created, this function just creates it. Note the value for this argument should not exceed maximum track index plus 1.
track.height	Height of the track. It is the percentage to the radius of the unit circles. The value can be set by uh to an absolute unit. If updating a track (with proper <code>track.index</code> value), this argument is ignored.
track.margin	only affect current track
cell.padding	only affect current track
bg.col	Background color for the plotting regions. It can be vector which has the same length of sectors.
bg.border	Color for the border of the plotting regions. It can be vector which has the same length of sectors.
bg.lty	Line style for the border of the plotting regions. It can be vector which has the same length of sectors.
bg.lwd	Line width for the border of the plotting regions. It can be vector which has the same length of sectors.
panel.fun	Panel function to add graphics in each cell, see "details" section and vignette for explanation.
factors	The same as sectors. It will be removed in future versions.

Value

Object [ccTrack](#)

Examples

```
library(circlizePlus)
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
                x = rnorm(n), y = runif(n))
library(circlizePlus)
par1=ccPar("track.height" = 0.1)
cc=ccPlot(sectors = df$sectors, x = df$x) + par1
track1 = ccTrack(sectors = df$sectors, y = df$y,
                 panel.fun = function(x, y) {
                   circos.text(CELL_META$xcenter,
                               CELL_META$cell.ylim[2] + mm_y(5),
                               CELL_META$sector.index)
                   circos.axis(labels.cex = 0.6)
                 })
cc=cc+track1
cc
circos.clear()
```

ccTrack-class

S4 class ccTrack

Description

S4 class ccTrack

Slots

func character. Normally it is "circos.track" or "circos.trackHist".

params list. A **named** list that stores the parameters of the function called by the backend.

trackGeoms list. A list where [ccTrackGeom](#) are stored.

cells list. A list where [ccCell](#) are stored.

Examples

```
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
                x = rnorm(n), y = runif(n))
library(circlizePlus)
par1=ccPar("track.height" = 0.1)
cc=ccPlot(sectors = df$sectors, x = df$x) + par1
track1 = ccTrack(sectors = df$sectors, y = df$y,
                 panel.fun = function(x, y) {
                   circos.text(CELL_META$xcenter,
                               CELL_META$cell.ylim[2] + mm_y(5),
                               CELL_META$sector.index)
                   circos.axis(labels.cex = 0.6)
                 })
```

```
cc=cc+track1
cc
circos.clear()
```

ccTrackGeom-class	<i>S4 class ccTrackGeom</i>
-------------------	-----------------------------

Description

Objectified representation of the R package `circlize`'s plotting functions and corresponding parameters at the track level.

Slots

`func` character. The name of the plot function in the R package `circlize`.

`params` list. A **named** list that stores the parameters of the function called by the backend.

Examples

```
NULL
```

ccTrackHist	<i>Define a track of histograms</i>
-------------	-------------------------------------

Description

Object `ccTrack` will call the function `circlize::circos.trackHist` while drawing.

Usage

```
ccTrackHist(
  sectors,
  x,
  track.height = circos.par("track.height"),
  track.index = NULL,
  ylim = NULL,
  force.ylim = TRUE,
  col = ifelse(draw.density, "black", NA),
  border = "black",
  lty = par("lty"),
  lwd = par("lwd"),
  bg.col = NA,
  bg.border = "black",
  bg.lty = par("lty"),
  bg.lwd = par("lwd"),
  breaks = "Sturges",
```

```

    include.lowest = TRUE,
    right = TRUE,
    draw.density = FALSE,
    bin.size = NULL,
    area = FALSE,
    factors = sectors
  )

```

Arguments

sectors	A factor or a character vector which represents the categories of data
x	Data on the x-axis
track.height	Height of the track. It is the percentage to the radius of the unit circle. If to update a track, this argument is disabled.
track.index	Index for the track which is going to be updated. Setting it to NULL means creating the plotting regions in the next newest track.
ylim	Ranges on y-direction. By default, ylim is calculated automatically.
force.ylim	Whether to force all cells in the track to share the same ylim.
col	Filled color for histogram
border	Border color for histogram
lty	Line style for histogram
lwd	Line width for histogram
bg.col	Background color for the plotting regions
bg.border	Color for the border of the plotting regions
bg.lty	Line style for the border of the plotting regions
bg.lwd	Line width for the border of the plotting regions
breaks	see hist
include.lowest	see hist
right	see hist
draw.density	whether draw density lines instead of histogram bars.
bin.size	size of the bins of the histogram
area	whether to fill the area below the density lines. If it is set to TRUE, col controls the filled color in the area and border controls color of the line.
factors	The same as sectors. It will be removed in future versions.

Value

Object [ccTrack](#)

Examples

```
library(circlizePlus)
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
                x = rnorm(n), y = runif(n))
library(circlizePlus)
par1=ccPar("track.height" = 0.1)
cc=ccPlot(sectors = df$sectors, x = df$x) + par1;bgcol = rep(c("#EFEFEF", "#CCCCCC"), 4)
track2 = ccTrackHist(df$sectors, df$x, bin.size = 0.2, bg.col = bgcol, col = NA)
cc=cc+track2
cc
circos.clear()
```

ccTrackLines

Add lines on all sections of a single track.

Description

Object `ccTrackGeom` will call the function `circlize::circos.trackLines` while drawing.

Usage

```
ccTrackLines(
  sectors,
  x,
  y,
  col = par("col"),
  lwd = par("lwd"),
  lty = par("lty"),
  type = "l",
  straight = FALSE,
  area = FALSE,
  area.baseline = NULL,
  border = "black",
  baseline = "bottom",
  pt.col = par("col"),
  cex = par("cex"),
  pch = par("pch"),
  factors = sectors
)
```

Arguments

sectors	A factor or a character vector which represents the categories of data.
x	Data points on x-axis.
y	Data points on y-axis.
col	Line color.

lwd	Line width.
lty	Line style.
type	Line type, similar as type argument in lines , but only in c("l", "o", "h", "s").
straight	Whether draw straight lines between points.
area	Whether to fill the area below the lines. If it is set to TRUE, col controls the filled color in the area and border controls the color of the line.
area.baseline	Deprecated, use baseline instead.
border	Color for border of the area.
baseline	The base line to draw area, pass to circos.lines .
pt.col	If type is "o", points color.
cex	If type is "o", points size.
pch	If type is "o", points type.
factors	The same as sectors. It will be removed in future versions.

Value

Object [ccTrackGeom](#)

Examples

NULL

ccTrackPoints	<i>Add points on all sections of a single track.</i>
---------------	--

Description

Object [ccTrackGeom](#) will call the function [circlize::circos.trackPoints](#) while drawing.

Usage

```
ccTrackPoints(
  sectors,
  x,
  y,
  pch = par("pch"),
  col = par("col"),
  cex = par("cex"),
  bg = par("bg"),
  factors = sectors
)
```

Arguments

sectors	A factor or a character vector which represents the categories of data
x	Data points on x-axis
y	Data points on y-axis
pch	Point type
col	Point color
cex	Point size
bg	background color
factors	The same as sectors. It will be removed in future versions.

Value

Object [ccTrackGeom](#)

Examples

NULL

ccTrackText

Add texts on all sections of a single track.

Description

Object [ccTrackGeom](#) will call the function [circlize::circo.trackText](#) while drawing.

Usage

```
ccTrackText(
  sectors,
  x,
  y,
  labels,
  direction = NULL,
  facing = c("inside", "outside", "reverse.clockwise", "clockwise", "downward",
    "bending", "bending.inside", "bending.outside"),
  niceFacing = FALSE,
  adj = par("adj"),
  cex = 1,
  col = par("col"),
  font = par("font"),
  factors = sectors
)
```

Arguments

sectors	A factor or a character vector which represents the categories of data
x	Data points on x-axis
y	Data points on y-axis
labels	Labels
direction	deprecated, use facing instead.
facing	Facing of text
niceFacing	Should the facing of text be adjusted to fit human eyes?
adj	Adjustment for text
cex	Font size
col	Font color
font	Font style
factors	The same as sectors. It will be removed in future versions.

Value

Object [ccTrackGeom](#)

Examples

NULL

ccViolin

Draw violin plots

Description

Object [ccCellGeom](#) will call the function [circlize::circos.violin](#) while drawing.

Usage

```
ccViolin(
  value,
  pos,
  violin_width = 0.8,
  col = NA,
  border = "black",
  lwd = par("lwd"),
  lty = par("lty"),
  show_quantile = TRUE,
  pt.col = par("col"),
  cex = par("cex"),
  pch = 16,
  max_density = NULL
)
```

Arguments

value	A numeric vector, a matrix or a list. If it is a matrix, boxplots are made by columns.
pos	Positions of the boxes.
violin_width	Width of violins.
col	Filled color of boxes.
border	Color for the border as well as the quantile lines.
lwd	Line width.
lty	Line style
show_quantile	Whether to show the quantile lines.
pt.col	Point color
cex	Point size.
pch	Point type.
max_density	The maximal density value across several violins. It is used to compare between violins.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:4], xlim = c(0, 10))
track <- ccTrack(ylim = c(0, 1))
cell <- ccCell(sector.index = "a") + ccViolin(value = replicate(runif(10),
n = 10, simplify = FALSE), pos = 1:10 - 0.5, col = 1:10)
track <- track + cell
cc + track
```

ccXaxis

Draw x-axis

Description

Object [ccCellGeom](#) will call the function [circlize::circos.axis](#) while drawing.

Usage

```
ccXaxis(
  h = "top",
  major.at = NULL,
  labels = TRUE,
  major.tick = TRUE,
  labels.font = par("font"),
  labels.cex = par("cex"),
  labels.facing = "inside",
  labels.direction = NULL,
  labels.niceFacing = TRUE,
  direction = c("outside", "inside"),
  minor.ticks = 4,
  major.tick.length = NULL,
  lwd = par("lwd"),
  col = par("col"),
  labels.col = par("col"),
  labels.pos.adjust = TRUE
)
```

Arguments

<code>h</code>	Position of the x-axis, can be "top", "bottom" or a numeric value
<code>major.at</code>	If it is numeric vector, it identifies the positions of the major ticks. It can exceed <code>xlim</code> value and the exceeding part would be trimmed automatically. If it is <code>NULL</code> , about every 10 degrees there is a major tick.
<code>labels</code>	labels of the major ticks. Also, the exceeding part would be trimmed automatically. The value can also be logical (either an atomic value or a vector) which represents which labels to show.
<code>major.tick</code>	Whether to draw major tick. If it is set to <code>FALSE</code> , there will be no minor ticks neither.
<code>labels.font</code>	Font style for the axis labels.
<code>labels.cex</code>	Font size for the axis labels.
<code>labels.facing</code>	Facing of labels on axis, passing to circos.text
<code>labels.direction</code>	Deprecated, use <code>facing</code> instead.
<code>labels.niceFacing</code>	Should facing of axis labels be human-easy.
<code>direction</code>	Whether the axis ticks point to the outside or inside of the circle.
<code>minor.ticks</code>	Number of minor ticks between two close major ticks.
<code>major.tick.length</code>	Length of the major ticks, measured in "current" data coordinate. convert_y can be used to convert an absolute unit to the data coordinate.
<code>lwd</code>	Line width for ticks.
<code>col</code>	Color for the axes.

labels.col Color for the labels.
 labels.pos.adjust Whether to adjust the positions of the first label and the last label so that the first label align to its left and the last label align to its right if they exceed the range on axes. The value can be a vector of length two which correspond to the first label and the last label.

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track <- ccTrack(ylim = c(0, 10))
cell <- ccCell(sector.index = "a") + ccXaxis()
track <- track + cell
cc + track
```

ccYaxis

Draw y-axis

Description

Object [ccCellGeom](#) will call the function [circlize::circos.yaxis](#) while drawing.

Usage

```
ccYaxis(
  side = c("left", "right"),
  at = NULL,
  labels = TRUE,
  tick = TRUE,
  labels.font = par("font"),
  labels.cex = par("cex"),
  labels.niceFacing = TRUE,
  lwd = par("lwd"),
  col = par("col"),
  labels.col = par("col")
)
```

Arguments

side add the y-axis on the left or right of the cell
 at If it is numeric vector, it identifies the positions of the ticks. It can exceed ylim value and the exceeding part would be trimmed automatically.

labels	labels of the ticks. The exceeding part would be trimmed automatically. The value can also be logical (either an atomic value or a vector) which represents which labels to show.
tick	Whether to draw ticks.
labels.font	font style for the axis labels
labels.cex	font size for the axis labels
labels.niceFacing	Should facing of axis labels be human-easy
lwd	line width for ticks
col	color for the axes
labels.col	color for the labels

Value

Object [ccCellGeom](#)

Examples

```
library(circlizePlus)
cc <- ccPlot(sectors = letters[1:8], xlim = c(0, 1))
track <- ccTrack(ylim = c(0, 10))
cell <- ccCell(sector.index = "a") + ccYaxis(side = "left")
track <- track + cell
cc + track
```

data-set	<i>Data set in circlizePlus</i>
----------	---------------------------------

Description

Example can be found in "4.2 Example 2: comparison of two pieces of code that use circlizePlus and circlize to implement the same requirements, respectively" allele_frequency: A dataframe with 25,000 rows and 4 columns copy_number: A dataframe with 24,788 rows and 5 columns junctions: A dataframe with 2414 rows and 28 columns r1: A dataframe with 2414 rows and 3 columns r2: A dataframe with 2414 rows and 3 columns

Usage

```
data(example2)

data(example2)

data(example2)

data(example2)

data(example2)
```

Format

An object of class data.frame with 25000 rows and 4 columns.

An object of class data.frame with 24788 rows and 5 columns.

An object of class data.frame with 2414 rows and 28 columns.

An object of class data.frame with 2414 rows and 3 columns.

An object of class data.frame with 2414 rows and 3 columns.

Source

<https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2025.1535368/full#h5>

Examples

```
data(example2)
allele_frequency
copy_number
junctions
r1
r2
```

show,ccHeatmap-method *Draw the figures described by ccHeatmap*

Description

Draw the figures described by ccHeatmap

Usage

```
## S4 method for signature 'ccHeatmap'
show(object)
```

Arguments

object Object of `ccHeatmap`

Value

No return information

Examples

```

library(circlizePlus)
set.seed(123)
mat1 <- rbind(
  cbind(
    matrix(rnorm(50 * 5, mean = 1), nr = 50),
    matrix(rnorm(50 * 5, mean = -1), nr = 50)
  ),
  cbind(
    matrix(rnorm(50 * 5, mean = -1), nr = 50),
    matrix(rnorm(50 * 5, mean = 1), nr = 50)
  )
)
rownames(mat1) <- paste0("R", 1:100)
colnames(mat1) <- paste0("C", 1:10)
mat1 <- mat1[sample(100, 100), ] # randomly permute rows
split <- sample(letters[1:5], 100, replace = TRUE)
split <- factor(split, levels = letters[1:5])
col_fun1 <- colorRamp2(c(-2, 0, 2), c("blue", "white", "red"))
show(ccHeatmap(mat = mat1, split = split, col = col_fun1))

```

show,ccPlot-method	<i>Draw the figures described by ccPlot</i>
--------------------	---

Description

Draw the figures described by ccPlot

Usage

```

## S4 method for signature 'ccPlot'
show(object)

```

Arguments

object Object of [ccPlot](#)

Value

No return information

Examples

```

library(circlizePlus)
n = 1000
df = data.frame(sectors = sample(letters[1:8], n, replace = TRUE),
               x = rnorm(n), y = runif(n))
library(circlizePlus)

```

```
par1=ccPar("track.height" = 0.1)
cc=ccPlot(sectors = df$sectors, x = df$x) + par1
track1 = ccTrack(sectors = df$sectors, y = df$y,
  panel.fun = function(x, y) {
    circos.text(CELL_META$xcenter,
      CELL_META$cell.ylim[2] + mm_y(5),
      CELL_META$sector.index)
    circos.axis(labels.cex = 0.6)
  })
cc=cc+track1
show(cc)
```

Index

- * **datasets**
 - data-set, [57](#)
- + ,ccCell, ccCellGeom-method (addition-rules), [3](#)
- + ,ccCells, ccCellGeom-method (addition-rules), [3](#)
- + ,ccHeatmap, ccLink-method (addition-rules), [3](#)
- + ,ccHeatmap, ccPar-method (addition-rules), [3](#)
- + ,ccHeatmap, ccTrack-method (addition-rules), [3](#)
- + ,ccPlot, ccLink-method (addition-rules), [3](#)
- + ,ccPlot, ccPar-method (addition-rules), [3](#)
- + ,ccPlot, ccTrack-method (addition-rules), [3](#)
- + ,ccTrack, ccCell-method (addition-rules), [3](#)
- + ,ccTrack, ccCells-method (addition-rules), [3](#)
- + ,ccTrack, ccTrackGeom-method (addition-rules), [3](#)
- addition-rules, [3](#)
- allele_frequency (data-set), [57](#)
- Arrowhead, [34](#), [35](#)
- as.dendrogram, [29](#), [38](#)
- as.raster, [41](#)
- ccArrow, [4](#)
- ccBarplot, [5](#)
- ccBoxplot, [6](#)
- ccCell, [8](#), [8](#), [28](#), [30](#), [47](#)
- ccCell-class, [8](#)
- ccCellGeom, [4–8](#), [10](#), [32](#), [33](#), [39–45](#), [53](#), [54](#), [56](#), [57](#)
- ccCellGeom-class, [9](#)
- ccCells, [9](#), [9](#)
- ccCells-class, [10](#)
- ccDendrogram, [10](#)
- ccGenomicAxis, [11](#)
- ccGenomicCellGeom, [8](#), [11](#), [12](#), [18](#), [19](#), [21](#), [23](#), [24](#), [26](#)
- ccGenomicCellGeom-class, [12](#)
- ccGenomicDensity, [13](#)
- ccGenomicHeatmap, [14](#)
- ccGenomicIdeogram, [15](#)
- ccGenomicLabels, [16](#)
- ccGenomicLines, [18](#)
- ccGenomicLink, [19](#), [19](#), [20](#), [30](#), [39](#)
- ccGenomicLink-class, [20](#)
- ccGenomicPoints, [21](#)
- ccGenomicRainfall, [22](#)
- ccGenomicRect, [23](#)
- ccGenomicText, [24](#)
- ccGenomicTrack, [13–17](#), [22](#), [23](#), [26](#), [26](#), [27](#), [30](#), [39](#)
- ccGenomicTrack-class, [27](#)
- ccHeatmap, [28](#), [28](#), [30](#), [39](#), [58](#)
- ccHeatmap-class, [30](#)
- ccHeatmapLink, [30](#), [31](#), [31](#), [39](#)
- ccHeatmapLink-class, [31](#)
- ccLines, [32](#)
- ccLink, [30](#), [33](#), [33](#), [35](#), [39](#)
- ccLink-class, [35](#)
- ccPar, [30](#), [36](#), [36](#), [39](#)
- ccPar-class, [36](#)
- ccPlot, [37](#), [37](#), [38](#), [59](#)
- ccPlot-class, [39](#)
- ccPoints, [39](#)
- ccPolygon, [40](#)
- ccRaster, [41](#)
- ccRect, [42](#)
- ccSegments, [43](#)
- ccText, [44](#)
- ccTrack, [30](#), [39](#), [45](#), [45](#), [46](#), [48](#), [49](#)
- ccTrack-class, [47](#)

- ccTrackGeom, [28, 30, 47, 50–53](#)
- ccTrackGeom-class, [48](#)
- ccTrackHist, [48](#)
- ccTrackLines, [50](#)
- ccTrackPoints, [51](#)
- ccTrackText, [52](#)
- ccViolin, [53](#)
- ccXaxis, [54](#)
- ccYaxis, [56](#)
- circlize::circos.arrow, [4](#)
- circlize::circos.axis, [54](#)
- circlize::circos.barplot, [5](#)
- circlize::circos.boxplot, [6](#)
- circlize::circos.clear, [29, 30, 37, 39](#)
- circlize::circos.dendrogram, [10](#)
- circlize::circos.genomicAxis, [11](#)
- circlize::circos.genomicDensity, [13](#)
- circlize::circos.genomicHeatmap, [14](#)
- circlize::circos.genomicIdeogram, [15](#)
- circlize::circos.genomicInitialize, [37, 39](#)
- circlize::circos.genomicLabels, [16](#)
- circlize::circos.genomicLines, [18](#)
- circlize::circos.genomicLink, [19, 20](#)
- circlize::circos.genomicPoints, [21](#)
- circlize::circos.genomicRainfall, [22](#)
- circlize::circos.genomicRect, [23](#)
- circlize::circos.genomicText, [24](#)
- circlize::circos.genomicTrackPlotRegion, [26](#)
- circlize::circos.heatmap, [28, 30](#)
- circlize::circos.heatmap.initialize, [37, 39](#)
- circlize::circos.heatmap.link, [31](#)
- circlize::circos.initialize, [37, 39](#)
- circlize::circos.initializeWithIdeogram, [37, 39](#)
- circlize::circos.lines, [32](#)
- circlize::circos.link, [33, 35](#)
- circlize::circos.par, [36](#)
- circlize::circos.points, [39](#)
- circlize::circos.polygon, [40](#)
- circlize::circos.raster, [41](#)
- circlize::circos.rect, [42](#)
- circlize::circos.segments, [43](#)
- circlize::circos.text, [44](#)
- circlize::circos.trackHist, [48](#)
- circlize::circos.trackLines, [50](#)
- circlize::circos.trackPlotRegion, [45](#)
- circlize::circos.trackPoints, [51](#)
- circlize::circos.trackText, [52](#)
- circlize::circos.violin, [53](#)
- circlize::circos.yaxis, [56](#)
- circos.axis, [12](#)
- circos.lines, [13, 14, 18, 19, 51](#)
- circos.link, [20, 31](#)
- circos.points, [21](#)
- circos.rect, [24](#)
- circos.text, [25, 55](#)
- circos.track, [29](#)
- circos.trackPlotRegion, [14, 23, 27](#)
- cm_h, [15](#)
- colorRamp2, [15, 29](#)
- convert_y, [55](#)
- copy_number (data-set), [57](#)
- data-set, [57](#)
- degree, [44](#)
- dendrogram, [10](#)
- dist, [29, 38](#)
- factor, [37, 46, 49, 50, 52, 53](#)
- genomicDensity, [13](#)
- hclust, [29, 38](#)
- hist, [49](#)
- inches_h, [15](#)
- junctions (data-set), [57](#)
- lines, [32, 43, 51](#)
- mm_h, [15](#)
- polygon, [5, 40, 42](#)
- posTransform.default, [18, 21, 24, 25](#)
- posTransform.text, [25, 26](#)
- r1 (data-set), [57](#)
- r2 (data-set), [57](#)
- rainfallTransform, [22](#)
- read.chromInfo, [38](#)
- read.cytoband, [16, 38](#)
- show, ccHeatmap-method, [58](#)
- show, ccPlot-method, [59](#)
- text, [44](#)
- uh, [46](#)