Package ‘ribiosUtils’

February 3, 2026

Type Package

Title Utilities from and Interface to the 'Bioinfo-C' ('BIOS') Library

Version 1.7.7

Date 2026-01-22

Description Provides interface to the 'Bioinfo-C' (internal name: 'BIOS') library and utilities. 'ribiosUtils' is a Swiss-knife for computational biology in drug discovery, providing functions and utilities with minimal external dependency and maximal efficiency.

Depends R (>= 3.4.0)

Imports methods, stats, utils

Suggests devtools, testthat

License GPL-3

URL `https://github.com/bedapub/ribiosUtils`,
`https://bedapub.github.io/ribiosUtils/`

BugReports `https://github.com/bedapub/ribiosUtils/issues`

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Jitao David Zhang [aut, cre, ctb] (ORCID:
<`https://orcid.org/0000-0002-3085-0909`>),
Clemens Broger [aut, ctb],
F.Hoffmann-La Roche AG [cph],
Junio C Hamano [cph],
Jean Thierry-Mieg [cph],
Konrad Rudolph [cph],
Richard Durbin [cph]

Maintainer Jitao David Zhang <`jitao_david.zhang@roche.com`>

Repository CRAN

Date/Publication 2026-02-03 10:20:02 UTC

Contents

`allIdentical` *Testing whether several objects are all identical with each other*

Description

Given several objects, the function tests whether all of them are identical.

Usage

```
allIdentical(...)
```

Arguments

`...`	Objects to be tested. Can be given as a list, or simplying appending names separated by commas, see example.

Value

Logical, whether all objects are the same

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`identical`

Examples

```
test1 <- test2 <- test3 <- LETTERS[1:3]
allIdentical(test1, test2, test3)
allIdentical(list(test1, test2, test3))

num1 <- num2 <- num3 <- num4 <- sqrt(3)
allIdentical(num1, num2, num3, num4)
```

`applyTopOrIncAndNotExclFilter` *Apply isTopOrIncAndNotExcl filter to a matrix*

Description

Apply isTopOrIncAndNotExcl filter to a matrix

Usage

```
applyTopOrIncAndNotExclFilter(matrix, MARGIN, top = 1, falseValue = 0, ...)
```

Arguments

`matrix`	A matrix.
`MARGIN`	Integer, 1 stands for row and 2 stands for column, passed to `apply`.
`top`	Integer, how many top elements should be kept, passed to `isTopOrIncAndNotExcl`.
`falseValue`	The same type as data in the matrix, used to replace values that is `FALSE` when judged by `isTopOrIncAndNotExcl`.
`...`	Further parameters passed to `isTopOrIncAndNotExcl`, including `incFunc`, `excFunc`, and `decreasing`. The function applies the filter function `isTopOrIncAndNotExcl` to each row or each column to a matrix, keeps the values that are `TRUE` based on the logical vector returned by function, and replaces the values that are `FALSE` with the value defined by `falseValue`.

Value

A matrix with the same dimnames but with elements not satisfying `isTopOrIncAndNotExcl` replaced by `falseValue`.

Examples

```
myMat <- matrix(c(1,2,3,4,8,7,6,5,12,9,11,10), nrow=3, byrow=TRUE,
   dimnames=list(c("A", "B", "C"), c("Alpha", "Beta", "Gamma", "Delta")))
print(myMat)
applyTopOrIncAndNotExclFilter(myMat, 1, top=2, falseValue=-1)
applyTopOrIncAndNotExclFilter(myMat, 2, top=2, falseValue=-1)
applyTopOrIncAndNotExclFilter(myMat, 2, top=2, falseValue=-1, decreasing=FALSE)
applyTopOrIncAndNotExclFilter(myMat, 1, top=2, falseValue=-1, incFunc=function(x) x%%2==0)
applyTopOrIncAndNotExclFilter(myMat, 1, top=2, falseValue=-1,
   incFunc=function(x) x%%2==0, excFunc=function(x) x<5)
```

`asNumMatrix` *Convert string-valued data frame or matrix into a numeric matrix*

Description

Convert string-valued data frame or matrix into a numeric matrix

Usage

```
asNumMatrix(x)
```

Arguments

`x`	A data.frame or matrix, most likely with string values

Value

A numeric matrix with the same dimension

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
testDf <- data.frame(a=c("2.34", "4.55"), b=c("7.33", "9.10"))
asNumMatrix(testDf)

testMatrix <- matrix(c("2.34", "4.55", "9E-3","-2.44", "7.33", "9.10"), nrow=2)
asNumMatrix(testMatrix)
```

`assertColumnName` *Assert whether the required column names exist*

Description

The function calls `matchColumnName` internally to match the column names.

Usage

```
assertColumnName(data.frame.cols, reqCols, ignore.case = FALSE)
```

Arguments

`data.frame.cols`	column names of a data.frame. One can also provide a data.frame, which may however cause worse performance since the data.frame is copied
`reqCols`	required columns
`ignore.case`	logical, whether the case is considered

Value

If all required column names are present, their indices are returned *invisibly*. Otherwise an error message is printed.

Examples

```
myTestDf <- data.frame(HBV=1:3, VFB=0:2, BVB=4:6, FCB=2:4)
myFavTeams <- c("HBV", "BVB")
assertColumnName(myTestDf, myFavTeams)
myFavTeamsCase <- c("hbv", "bVb")
assertColumnName(myTestDf, myFavTeamsCase, ignore.case=TRUE)
```

`assertContrast` *Check dimensionality of contrast matrix*

Description

Check dimensionality of contrast matrix

Usage

```
assertContrast(design, contrast)
```

Arguments

`design`	Design matrix
`contrast`	Contrast matrix

Value

Side effect is used: the function stops if the ncol(design) does not equal nrow(contrast)

Examples

```
design <- matrix(1:20, ncol=5)
contrast <- matrix(c(-1,1,0,0,0, 0,1,0,-1,0), nrow=5)
assertContrast(design, contrast)
```

`assertDesign` *Check dimensionality of design matrix*

Description

Check dimensionality of design matrix

Usage

```
assertDesign(nsample, design)
```

Arguments

`nsample`	Integer, number of samples
`design`	Design matrix

Value

Side effect is used: the function stops if sample size does not equal ncol(matrix)

Examples

```
nsample <- 4
design <- matrix(1:20, ncol=5)
assertDesign(nsample, design)
```

`assertDesignContrast` *Check dimensionality of both design and contrast matrix*

Description

Check dimensionality of both design and contrast matrix

Usage

```
assertDesignContrast(nsample, design, contrast)
```

Arguments

`nsample`	Integer, number of samples
`design`	Design matrix
`contrast`	Contrast matrix

Value

Side effect is used: the function stops if there are errors in the dimensionalities

See Also

`assertDesign`, `assertContrast`

Examples

```
nsample <- 4
design <- matrix(1:20, ncol=5)
contrast <- matrix(c(-1,1,0,0,0, 0,1,0,-1,0), nrow=5)
assertDesignContrast(nsample, design, contrast)
```

`bedaInfo` *Print BEDA project information*

Description

Print BEDA project information

Usage

```
bedaInfo()
```

Value

A list, including pstore path, URL, git address, and user id The function is used at the end of the Rmarkdown report to print relevant information to help other colleagues finding relevant resources

Examples

```
bedaInfo()
```

`biomicsPstorePath2URL` *Translate BiOmics-Pathology pstore path to URL*

Description

Translate BiOmics-Pathology pstore path to URL

Usage

```
biomicsPstorePath2URL(path)
```

Arguments

`path`	Unix path

Value

Character string of biomics pstore path The URL is only visible inside Roche

Examples

```
biomicsPstorePath2URL("/pstore/data/biomics/")
```

`bound` *Set boundaries for numeric values, or perform a 0-1 normalization.*

Description

The basic concepts of these functions are borrowed from the bound function in the `Qt` framework.

Usage

```
bound(x, low, high)

boundNorm(x, low = min(x, na.rm = TRUE), high = max(x, na.rm = TRUE))
```

Arguments

`x`	A numeric vector or matrix
`low`	New lower boundary
`high`	New higher boundary

Details

`bound` sets the values smaller than `low`, or larger than `high`, to the value of `low` and `high` respectively.If no such values exist, the vector or matrix is returned unchanged.

`boundNorm` performs a 0-1 normalization. Input vector or matrix is transformed linearly onto the region defined between `low` and `high`, which has the unit length (1).

Value

A numeric vector or matrix, the same type as input.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
myVec <- c(2,4,3,-1,9,5,3,4)
bound(myVec, 0, 8)
boundNorm(myVec)
## boundNorm returns negative values if input values lie out of the
## given region between low and high
boundNorm(myVec, 0, 8)

myMat <- matrix(myVec, nrow=2)
myMat
bound(myMat, 0, 8)
boundNorm(myMat)
boundNorm(myMat, 0, 8)
```

`cbindByRownames` *Column bind by rownames*

Description

Column bind by rownames

Usage

```
cbindByRownames(..., type = c("intersect", "union"))

rbindByColnames(..., type = c("intersect", "union"))
```

Arguments

`...`	Two or more matrices, or a list of matrices.
`type`	Character string, how are row names that are not shared by all items handled, either `intersect` (intersect are kept) or `union` (union is used, `NA` is inserted whenever necessary)

Value

A `matrix`

Examples

```
mat1 <- matrix(1:9, nrow=3, byrow=FALSE,
               dimnames=list(LETTERS[1:3], LETTERS[1:3]))
mat2 <- matrix(1:9, nrow=3, byrow=FALSE,
               dimnames=list(LETTERS[2:4], LETTERS[4:6]))
mat3 <- matrix(1:9, nrow=3, byrow=FALSE,
               dimnames=list(LETTERS[c(2,4,5)], LETTERS[7:9]))
cbindByRownames(mat1, mat2, mat3, type="intersect")
cbindByRownames(mat1, mat2, mat3, type="union")
## it is also possible to pass a list
cbindByRownames(list(mat1, mat2, mat3), type="union")
mat4 <- matrix(1:9, nrow=3, byrow=FALSE,
               dimnames=list(LETTERS[1:3], LETTERS[1:3]))
mat5 <- matrix(1:9, nrow=3, byrow=FALSE,
               dimnames=list(LETTERS[4:6], LETTERS[2:4]))
mat6 <- matrix(1:9, nrow=3, byrow=TRUE,
               dimnames=list(LETTERS[7:9], LETTERS[c(2,4,6)]))
rbindByColnames(mat4, mat5, mat6, type="intersect")
rbindByColnames(mat4, mat5, mat6,  type="union")
## it is also possible to pass a list
rbindByColnames(list(mat4, mat5, mat6), type="union")
```

`checkFile` *Check whether file(s) exist*

Description

`checkFile` checks whether file exists, `assertFile` stops the program if files do not exist

Usage

```
checkFile(...)

assertFile(...)
```

Arguments

`...`	Files to be checked

Details

`assertFile` is often used in scripts where missing a file would cause the script fail.

Value

`checkFile` returns logical vector. `assertFile` returns an invisible `TRUE` if files exist, otherwise halts and prints error messages.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`isDir` and `assertDir`

Examples

```
myDesc <- system.file("DESCRIPTION", package="ribiosUtils")
myNEWS <- system.file("NEWS", package="ribiosUtils")
checkFile(myDesc, myNEWS)
assertFile(myDesc, myNEWS)
```

`chosenFew` *Print the chosen few items of a long vector*

Description

Print the chosen few (the first and the last) items of a long vector

Usage

```
chosenFew(vec, start = 3, end = 1, collapse = ",")
```

Arguments

`vec`	A vector of characters or other types that can be cast into characters
`start`	Integer, how many elements at the start shall be printed
`end`	Integer, how many elements at the end shall be printed
`collapse`	Character used to separate elements

Value

A character string ready to be printed

Note

In case the vector is shorter than the sum of `start` and `end`, the whole vector is printed.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
lvec1 <- 1:100
chosenFew(lvec1)
chosenFew(lvec1, start=5, end=3)

svec <- 1:8
chosenFew(svec)
chosenFew(svec, start=5, end=4)
```

`closeLoggerConnections` *Close connections to all loggers This function closes all open connections set up by loggers It is automatically run at the end of the R session (setup by `registerLog`)*

Description

Close connections to all loggers This function closes all open connections set up by loggers It is automatically run at the end of the R session (setup by `registerLog`)

Usage

```
closeLoggerConnections()
```

Value

Invisible `NULL`. Only side effect is used.

See Also

`registerLog`

`columnOverlapCoefficient` *Pairwise jaccard/overlap coefficient can be calculated efficiently using matrix Pairwise overlap coefficient of binary matrix by column*

Description

Pairwise jaccard/overlap coefficient can be calculated efficiently using matrix Pairwise overlap coefficient of binary matrix by column

Usage

```
columnOverlapCoefficient(x, y = NULL)
```

Arguments

`x`	An integer matrix, other objects will be coereced into a matrix
`y`	An integer matrix, other objects will be coereced into a matrix. In case of `NULL`, pairwise overlap coefficients by column of `x` is returned.

Value

A matrix of column-wise pairwise overlap coefficients of the binary matrix. `NaN` is reported when neither of the columns have any non-zero element.

Examples

```
set.seed(1887)
testMatrix1 <- matrix(rbinom(120, 1, 0.2), nrow=15)
columnOverlapCoefficient(testMatrix1)

testMatrix2 <- matrix(rbinom(150, 1, 0.2), nrow=15)
testMatrix12Poe <- columnOverlapCoefficient(testMatrix1,
  testMatrix2)
```

`compTwoVecs` *Compare two vectors by set operations*

Description

Basic set operations are used to compare two vectors

Usage

```
compTwoVecs(vec1, vec2)
```

Arguments

`vec1`	A vector of atomic types, e.g. integers, characters, etc.
`vec2`	A vector of the same type as `vec1`

Value

A vector of six integer elements

`vec1.setdiff`	Number of unique items only in `vec1` but not in `vec2`
`intersect`	Number of items in both `vec1` and `vec2`
`vec2.setdiff`	Number of unique items only in `vec2` but not in `vec1`
`vec1.ulen`	Number of unique items in `vec1`
`vec2.ulen`	Number of unique items in `vec2`
`union`	Number of unique items in `vec1` and `vec2`

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
year1 <- c("HSV", "FCB", "BVB", "S04", "FCN")
year2 <- c("HSV", "FCK", "S04")
compTwoVecs(year1, year2)
```

`corByRownames` *Calculate correlation coefficients using common rows of the two matrices*

Description

Calculate correlation coefficients using common rows of the two matrices

Usage

```
corByRownames(mat1, mat2, ...)
```

Arguments

`mat1`	A numeric matrix
`mat2`	Another numeric matrix
`...`	Passed `cor` At least two rows of both matrices must share the same names, otherwise the function will report an error.

Value

A matrix of the dimension $m \times n$, where `m` and `n` are number of columns in `mat1` and `mat2`, respectively. The matrix has an attribute, `commonRownames`, giving the common rownames shared by the two matrices.

Examples

```
myMat1 <- matrix(rnorm(24), nrow=6, byrow=TRUE,
  dimnames=list(sprintf("R%d", 1:6), sprintf("C%d", 1:4)))
myMat2 <- matrix(rnorm(35), nrow=7, byrow=TRUE,
  dimnames=list(sprintf("R%d", 7:1), sprintf("C%d", 1:5)))
corByRownames(myMat1, myMat2)
```

`countTokens` *Count tokens by splitting strings*

Description

Count tokens by splitting strings

Usage

```
countTokens(str, split = "\t", ...)
```

Arguments

`str`	A character string vector
`split`	Character used to split the strings
`...`	Other parameters passed to the `strsplit` function

Value

Integer vector: count of tokens in the strings

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`strsplit` to split strings, or a convenient wrapper `strtoken` in this package.

Examples

```
myStrings <- c("HSV\t1887\tFavorite", "FCB\t1900", "FCK\t1948")
countTokens(myStrings)

## the function deals with factors as well
countTokens(factor(myStrings))
```

`createDir` *Create a directory if it does not exist, and then make sure the creation was successful.*

Description

The function is particularly useful for scripting.

Usage

```
createDir(dir, showWarnings = FALSE, recursive = TRUE, mode = "0777")
```

Arguments

`dir`	Directory name
`showWarnings`	Passed to dir.create
`recursive`	Passed to dir.create
`mode`	Passed to dir.create

Value

Directory name (invisible)

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
tempdir <- tempdir()
createDir(tempdir)
```

`cumJaccardIndex` *Cumulative Jaccard Index*

Description

Cumulative Jaccard Index

Usage

```
cumJaccardIndex(list)

cumJaccardDistance(list)
```

Arguments

`list`	A list of characters or integers

Value

The cumulative Jaccard Index, a vector of values between 0 and 1, of the same length as the input list

The cumulative Jaccard Index is calculated by calculating the Jaccard Index of element `i` and the union of elements between `1` and `i-1`. The cumulative Jaccard Index of the first element is set as 0.0.

The cumulative Jaccard distance is defined in almost the same way, with the only difference the distance is returned. The value of the first element is 1.0.

Note

An advantage of using cumulative overlap coefficient over cumulative Jaccard Index is that it is monotonic: the value is garanteed to decrease from 1 to 0, whereas the cumulative Jaccard Index may not be monotic.

See Also

`cumOverlapCoefficient`

Examples

```
myList <- list(first=LETTERS[1:5], second=LETTERS[6:10], third=LETTERS[8:12], fourth=LETTERS[1:12])
cumJaccardIndex(myList)
cumJaccardDistance(myList)
```

`cumOverlapCoefficient` *Cumulative overlap coefficient*

Description

Cumulative overlap coefficient

Usage

```
cumOverlapCoefficient(list)

cumOverlapDistance(list)
```

Arguments

`list`	A list of characters or integers

Value

The cumulative overlap coefficients, a vector of values between 0 and 1, of the same length as the input list

The cumulative overlap coefficient is calculated by calculating the overlap coefficient of element `i` and the union of elements between `1` and `i-1`. The cumulative overlap coefficient of the first element is set as 0.0.

The cumulative overlap distance is defined in almost the same way, with the only difference the distance is returned. The value of the first element is 1.0. Pratically it is calculated by `1-cumOverlapCoefficient`.

Since the denominator of the overlap coefficient is the size of the smaller set of the two, which is bound to be the size of element `i`, the cumulative overlap distance can be interpreted as the proportion of new items in each new element that are unseen in previous elements. Similarly, the cumulative overlap coefficient can be interpreted as the proportion of items in each new element that have been seen in previous elements. See examples below.

Note

An advantage of using cumulative overlap coefficient over cumulative Jaccard Index is that it is monotonic: the value is garanteed to decrease from 1 to 0, whereas the cumulative Jaccard Index may not be monotic.

Examples

```
myList <- list(first=LETTERS[1:5], second=LETTERS[6:10], third=LETTERS[8:12], fourth=LETTERS[1:12])
cumOverlapCoefficient(myList)
cumOverlapDistance(myList)
```

`cumsumprop` *Proportion of cumulative sum over sum*

Description

Proportion of cumulative sum over sum

Usage

```
cumsumprop(x)
```

Arguments

`x`	Numeric vector

Value

the proportion cumulative sum over sum

Examples

```
x <- 1:4
cumsumprop(x) ## 0.1, 0.3, 0.6, 1
```

`cutInterval` *Cut a vector of numbers into interval factors.*

Description

Three types of labels (levels) are supported: “cut.default” (Interval labels returned by `cut` as default), “left” (Left boundary of intervals), and “right” (Right boundary of intervals).

Usage

```
cutInterval(
  x,
  step = 1,
  labelOption = c("cut.default", "left", "right"),
  include.lowest = FALSE,
  right = TRUE,
  dig.lab = 3,
  ordered_result = FALSE,
  ...
)
```

Arguments

`x`	A vector of numbers
`step`	Step size.
`labelOption`	How is the label displayed.See `details` section.
`include.lowest`	Logical, passed to `cut`
`right`	Logial, passed to `cut`
`dig.lab`	See `cut`
`ordered_result`	See `cut`
`...`	Other parameters that are passed to `cut`

Value

A vector of factors

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`cut`

Examples

```
testNum <- rnorm(100)
(testFac <- cutInterval(testNum, step=1, labelOption="cut.default"))

## compare the result to
(testFacCut <- cut(testNum, 10))
```

`cutreeIntoOrderedGroups`

Cut a tree into groups of ordered sizes

Description

Cut a tree into groups of ordered sizes

Usage

```
cutreeIntoOrderedGroups(tree, k = NULL, h = NULL, decreasing = TRUE)
```

Arguments

`tree`	a tree as produced by `hclust`, will be passed to `cutree`
`k`	an integer scalar or vector with the desired number of groups
`h`	numeric scalar or vector with heights where the tree should be cut.
`decreasing`	logical, should be the first group the largest? Cut a tree, e.g. as resulting from `hclust`, into groups, with the groups being ordered by their size.

Value

A named integer vector of cluster assignments, ordered by cluster size (largest first by default). If multiple values of `k` or `h` are provided, a matrix with one column per value.

See Also

`cutree`

Examples

```
hc <- hclust(dist(USArrests))
hck5 <- cutreeIntoOrderedGroups(hc, k = 5)
table(hck5)
## compare with cutree, which does not order the groups
table(cutree(hc, k=5))

hck25 <- cutreeIntoOrderedGroups(hc, k = 2:5)
apply(hck25, 2, table)
```

`dfFactor` *Get a factor vector for a data.frame*

Description

The function try to assign a factor vector for a `data.frame` object. See details below.

Usage

```
dfFactor(df, sample.group)
```

Arguments

`df`	A `data.frame`
`sample.group`	A character, number or a vector of factors, from which the factor vector should be deciphered. See details below.

Details

The function tries to get a factor vector of the same length as the number of rows in the `data.frame`. The determination is done in the following order: Step 1: It tries to find a column in the `data.frame` with the name as given by `sample.group`. If found, this column is transformed into a factor if not and returned. Step 2: It tries to interpret the `sample.group` as an integer, as the index of the column in the `data.frame` giving the factor. Step 3: When `sample.group` itself is a vector of the same length as the data.frame, it is cast to factor when it is still not and returned.

Otherwise the program stops with error.

Value

A factor vector with the same length as the `data.frame`

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
df <- data.frame(gender=c("M", "M", "F", "F", "M"),
age=c(12,12,14,12,14), score=c("A", "B-", "C", "B-", "A"))
dfFactor(df, "gender")
dfFactor(df, "score")
dfFactor(df, 1L)
dfFactor(df, 2L)
dfFactor(df, df$score)
```

`dfFactor2Str` *Convert factor columns in a data.frame into character strings*

Description

Convert factor columns in a data.frame into character strings

Usage

```
dfFactor2Str(df)
```

Arguments

`df`	A data.frame

Value

A data.frame with factor columns coereced into character strings

Examples

```
exampleDf <- data.frame(Teams=c("HSV", "FCB", "FCB", "HSV"),
             Player=c("Mueller", "Mueller", "Robben", "Holtby"),
             scores=c(3.5, 1.5, 1.5, 1.0), stringsAsFactors=TRUE)
strDf <- dfFactor2Str(exampleDf)
stopifnot(identical(strDf[,1], c("HSV", "FCB", "FCB", "HSV")))
stopifnot(identical(exampleDf[,1], factor(c("HSV", "FCB", "FCB", "HSV"))))
```

`equateWellLabelWidth` *Format labels for wells in microwell plates to equal widths*

Description

Format labels for wells in microwell plates to equal widths

Usage

```
equateWellLabelWidth(wells)
```

Arguments

`wells`	A vector of character strings indicating well positions, they may be of different widths, for instance A1, A10, A12

Value

A vector of the same length, with all labels adjusted to the equal width, with left-padding zeros added whenever it makes sense. If the input labels are already of the same length, no change is applied.

Examples

```
equateWellLabelWidth(c("A1", "C10", "D12"))
```

`extname` *Get the base and extension(s) of file name(s)*

Description

Many files have base and extensions in their names, for instance for the file `mybook.pdf`, the base is `mybook` and the extension is `pdf`. `basefilename` `extname` functions extract these information from one or more file names.

Usage

```
extname(x, ifnotfound = NA, lower.case = FALSE)
```

Arguments

`x`	Character vector of file names; other classes will be coereced to characters
`ifnotfound`	If no extension name was found, the value to be returned. Default is `NA`
`lower.case`	Logical, should the names returned in lower case?

Value

The base file name or the extension as characters, of the same length as the input file name character. In case that a file name does not contain a extension, `NA` will be returned.

Note

In case there are multiple dots in the input file name, the last field will be taken as the extension, and the rest as the base name. For instance for file `test.out.txt`, returned base name is `test.out` and extension is `txt`.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
extname("mybook.pdf")
extname("sequence.in.fasta")
extname(c("/path/mybook.pdf", "test.doc"))
extname("README")
extname("README", ifnotfound="")
extname("/path/my\ home/Holiday Plan.txt")

basefilename("mybook.pdf")
basefilename("sequence.in.fasta")
basefilename(c("/path/mybook.pdf", "test.doc"))
basefilename("README")
basefilename("/path/my\ home/Holiday Plan.txt")

basefilename("myBook.pdf", lower.case=TRUE)
extname("myBook.PDF", lower.case=TRUE)
```

`firstUp` *Make the first alphabet of strings uppercase*

Description

Make the first alphabet of strings uppercase

Usage

```
firstUp(str)
```

Arguments

`str`	A vector of character strings

Value

A vector of the same length, with the first alphabet in uppercase

See Also

`toTitleCase`

Examples

```
firstUp('test string')
firstUp(strsplit('many many years ago', ' ')[[1]])
```

`fixWidthStr` *Shorten strings to strings with a fix width of characters*

Description

Shorten strings to strings with a fix width of characters

Usage

```
fixWidthStr(str, nchar = 8, align = c("left", "right"))
```

Arguments

`str`	A vector of strings
`nchar`	The fixed with
`align`	Character, how to align Strings with more or fewer characters than `nchar` are either shortened or filled (with spaces)

Value

A vector of strings with fixed widths

Note

`NA` will be converted to characters and the same fixed width will be applied. The behavior is different from `shortenStr`, where `NA` is kept as it is.

See Also

`shortenStr`

Examples

```
inputStrs <- c("abc", "abcd", "abcde", "abcdefg", "NA", NA)
outputStrs <- fixWidthStr(inputStrs, nchar=4)
stopifnot(all(nchar(outputStrs)==4))
```

`haltifnot` *Ensure the Truth of R Expressions and Print Defined Error Message if NOT*

Description

If any of the expressions in '...' are not *all TRUE*, *stop* is called, producing an error message defined by the *msg* parameter.

Usage

```
haltifnot(..., msg = "Error undefined. Please contact the developer")
```

Arguments

`...`	any number of 'logical' R expressions, which should evaluate to `TRUE`
`msg`	Error message.

Details

The function is adapted from the `stopifnot` function, with the difference that the error message can be defined the programmer. With `haltifnot` error message can be more informative, which is desired for diagnostic and user-interation purposes.

Value

`NULL` if all statements in `...` are `TRUE`

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`stop`, `warning` and `stopifnot`

Examples

```
haltifnot(1==1, all.equal(pi, 3.14159265), 1<2) ## all TRUE
m <- matrix(c(1,3,3,1), 2,2)
haltifnot(m == t(m), diag(m) == rep(1,2)) ## all TRUE

op <- options(error = expression(NULL))
# "disable stop(.)"  << Use with CARE! >>

haltifnot(all.equal(pi, 3.141593),  2 < 2, all(1:10 < 12), "a" < "b",
          msg="not all conditions are TRUE. Please contact the devleoper")
options(op)# revert to previous error handler
```

`headhead` *head/tail function for matrix or data.frame*

Description

These two functions reassembles `head` and `tail`, showing the first rows and columns of 2D data structures, e.g. matrix or data.frame.

Usage

```
headhead(x, m = 6L, n = 6L)
```

Arguments

`x`	A `data.frame` or `matrix`
`m`	Integer, number of rows to show
`n`	Integer, number of columns to show

Details

While `head` and `tail` can be applied to `data.frame` or `matrix` as well, they show all columns of the first (last) rows even if the matrix has a large number of columns. These two function, `headhead` and `tailtail`, circumvent this problem by showing only the first rows AND the first columns.

Value

The first rows/columns of the input object

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`head`, `tail`

Examples

```
myMat <- matrix(rnorm(10000), nrow=10L)
head(myMat)
headhead(myMat)
tailtail(myMat)
```

`headtail` *Print head and tail elements of a vector*

Description

This function prints head and tail elements of a vector for visualization purposes. See examples for its usage.

Usage

```
headtail(vec, head = 2, tail = 1, collapse = ", ")
```

Arguments

`vec`	A vector of native types (e.g. character strings)
`head`	Integer, number of head elements to be printed
`tail`	Integer, number of tail elements to be printed
`collapse`	Character string, used to collapse elements

Details

Head and tail elements are concatenated with ellipsis, if there are any elements that are not shown in the vector.

Value

A character string representing the vector

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`head`, `tail`

Examples

```
testVec1 <- LETTERS[1:10]
headtail(testVec1)
headtail(testVec1, head=3, tail=3)
headtail(testVec1, head=3, tail=3, collapse="|")

testVec2 <- letters[1:3]
headtail(testVec2, head=1, tail=1)
headtail(testVec2, head=2, tail=1)
```

`identicalMatrix` *Test whether two matrices are identical by values and by dim names*

Description

Test whether two matrices are identical by values and by dim names

Usage

```
identicalMatrix(x, y, epsilon = 1e-12)
```

Arguments

`x`	a matrix
`y`	another matrix
`epsilon`	accuracy threshold: absolute differences below this threshold is ignored

Value

Logical

Examples

```
set.seed(1887); x <- matrix(rnorm(1000), nrow=10, dimnames=list(LETTERS[1:10],NULL))
set.seed(1887); y <- matrix(rnorm(1000), nrow=10, dimnames=list(LETTERS[1:10],NULL))
set.seed(1887); z <- matrix(rnorm(1000), nrow=10, dimnames=list(letters[1:10],NULL))
stopifnot(identicalMatrix(x,y))
stopifnot(!identicalMatrix(x,z))
```

`identicalMatrixValue` *Test whether two matrices have the same numerica values given certain accuracy*

Description

Test whether two matrices have the same numerica values given certain accuracy

Usage

```
identicalMatrixValue(x, y, epsilon = 1e-12)
```

Arguments

`x`	a matrix
`y`	another matrix
`epsilon`	accuracy threshold: absolute differences below this threshold is ignored

Value

Logical

Examples

```
set.seed(1887); x <- matrix(rnorm(1000), nrow=10)
set.seed(1887); y <- matrix(rnorm(1000), nrow=10)
set.seed(1882); z <- matrix(rnorm(1000), nrow=10)
stopifnot(identicalMatrixValue(x,y))
stopifnot(!identicalMatrixValue(x,y+1E-5))
stopifnot(!identicalMatrixValue(x,y-1E-5))
stopifnot(!identicalMatrixValue(x,z))
```

`imatch` *Case-insensitive match and pmatch*

Description

Case-insensitive `match` and `pmatch` functions, especially useful in parsing user inputs, e.g. from command line.

Usage

```
imatch(x, table, ...)
```

Arguments

`x`	String vector
`table`	A vector to be matched
`...`	Other parameters passed to `match` or `pmatch`

Details

`imatch` and `ipmatch` works similar as `match` and `pmatch`, except that they are case-insensitive.

`matchv`, `imatchv` and `ipmatchv` are shot-cuts to get the matched value (therefore the ‘v’) if the match succeeded, or `NA` if not. `match(x, table)` is equivalent to `table[match(x, table)]`. See examples.

Value

`imatch` and `ipmatch` returns matching indices, or `NA` (by default) if the match failed.

`matchv`, `imatchv` and `ipmatchv` returns the matching element in `table`, or `NA` if the match failed. Note that when cases are different in `x` and `table`, the one in `table` will be returned. This is especially useful for cases where user’s input has different cases as the internal options.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`match` and `pmatch`

Examples

```
user.input <- c("hsv", "BvB")
user.input2 <- c("HS", "BV")
internal.options <- c("HSV", "FCB", "BVB", "FCN")

match(user.input, internal.options)
imatch(user.input, internal.options)
ipmatch(user.input, internal.options)
ipmatch(user.input2, internal.options)

matchv(user.input, internal.options)
matchv(tolower(user.input), tolower(internal.options))
imatchv(user.input, internal.options)
ipmatchv(user.input, internal.options)
ipmatchv(user.input2, internal.options)
```

`invertList` *Invert the names and elements of a list*

Description

Invert the names and elements of a list

Usage

```
invertList(inputList, simplify = FALSE)
```

Arguments

`inputList`	a list, other classed (e.g. named vectors) will be converted to lists
`simplify`	Logical, if yes and if no duplicated names, return a vector

Value

A list with values from the input becoming names and vice versa. When `simplify=TRUE` and there are no duplicated names, a named character vector is returned instead.

Examples

```
myList <- list("A"=c("a", "alpha"), "B"=c("b", "Beta"), "C"="c")
invertList(myList)
invertList(myList, simplify=TRUE)
```

`isDir` *Checks existing directory*

Description

Checks whether given character strings point to valid directories

Usage

```
isDir(...)

checkDir(...)

assertDir(...)
```

Arguments

`...`	One or more character strings giving directory names to be tested

Details

`isDir` tests whether the given string represent a valid, existing directory. `assertDir` performs a logical test, and stops the program if the given string does not point to a given directory.

`checkDir` is synonymous to `isDir`

Value

`isDir` returns logical vector.

`assertDir` returns an invisible `TRUE` if directories exist, otherwise halts and prints error messages.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`file.info`, `checkFile` and `assertFile`

Examples

```
dir1 <- tempdir()
dir2 <- tempdir()

isDir(dir1, dir2)
assertDir(dir1, dir2)
```

`isError` *Tell whether an object is an error*

Description

Determines whether an object is of class `try-error`

Usage

```
isError(x)
```

Arguments

`x`	Any object, potentially produced within a `try-error` structure.

Value

Logical value, `TRUE` if `x` inherits the `try-error` class.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
if(exists("nonExistObj")) rm(nonExistsObj)
myObj <- try(nonExistObj/5, silent=TRUE)
isError(myObj)
```

`isOdd` *Whether an integer is odd (or even)*

Description

Whether an integer is odd (or even)

Usage

```
isOdd(x)

isEven(x)
```

Arguments

`x`	An integer.

Value

Logical, whether the input number is odd or even.

`isOdd` and `isEven` returns whether an integer is odd or even, respectively.

Examples

```
isOdd(3)
isEven(4)
```

`isRocheCompoundID` *Tell whether a character string is a Roche compound ID*

Description

Tell whether a character string is a Roche compound ID

Usage

```
isRocheCompoundID(str)
```

Arguments

`str`	Character string(s)

Value

A logical vector of the same length as `str`, indicating whether each element is a Roche compound ID or not.

Note

Short versions (RO[1-9]{2,7}) are supported.

Examples

```
isRocheCompoundID(c("RO1234567", "RO-1234567",
                    "RO1234567-000", "RO1234567-000-000",
                    "ROnoise-000-000"))
```

`isTopOrIncAndNotExcl` *Logical vector of being top or included and not excluded*

Description

Logical vector of being top or included and not excluded

Usage

```
isTopOrIncAndNotExcl(x, top = 1, incFunc, excFunc, decreasing = TRUE)
```

Arguments

`x`	An atomic vector that can be sorted by `sort`, for instance integers and character strings.
`top`	Integer, number of top elements that we want to consider.
`incFunc`	Function, applied to `x` to return a logical vector of the same length, indicating whether the values should be included even if it does not belong to the top elements.
`excFunc`	Function, applied to `x` to return a logical vector of the same length, indicating whether the values should be excluded even if it does belong to the top elements.
`decreasing`	Logical, passed to `sort`. The default value is set to `TRUE`, which means that the highest values are considered the top elements. If set to `FALSE`, the lowest values are considered the top elements.

Value

A logical vector of the same length as the input `x`, indicating whether each element is being either top or included, and not excluded. The function can be used to keep top elements of a vector while considering both inclusion and exclusion criteria.

Examples

```
myVal <- c(2, 4, 8, 7, 1)
isTopOrIncAndNotExcl(myVal, top=1)
isTopOrIncAndNotExcl(myVal, top=3)
isTopOrIncAndNotExcl(myVal, top=3, incFunc=function(x) x>=2)
isTopOrIncAndNotExcl(myVal, top=3, excFunc=function(x) x%%2==1)
isTopOrIncAndNotExcl(myVal, top=3, incFunc=function(x) x>=2, excFunc=function(x) x%%2==1)
myVal2 <- c("a", "A", "a", "A", "A")
isTopOrIncAndNotExcl(myVal2, 2)
isTopOrIncAndNotExcl(myVal2, 2, incFunc=function(x) x=="A")
isTopOrIncAndNotExcl(myVal2, 4)
isTopOrIncAndNotExcl(myVal2, 4, excFunc=function(x) x=="a")

## the function returns all TRUEs if top is larger than the length of the vector
isTopOrIncAndNotExcl(myVal, top=9)
```

`jaccardIndex` *Calculate the Jaccard Index between two vectors*

Description

Calculate the Jaccard Index between two vectors

Usage

```
jaccardIndex(x, y)

jaccardDistance(x, y)
```

Arguments

`x`	A vector
`y`	A vector

Value

The Jaccard Index, a number between 0 and 1

`JaccardDistance` is defined as `1-JaccardIndex`.

Examples

```
myX <- 1:6
myY <- 4:9
jaccardIndex(myX, myY)
jaccardDistance(myX, myY)

myX <- LETTERS[1:5]
```

```
myY <- LETTERS[6:10]
jaccardIndex(myX, myY)
jaccardDistance(myX, myY)
```

`keepMaxStatRow` *KEEP ROWS WITH THE MAXIMUM STATISTIC*

Description

A common task in expression analysis is to collapse multiple features that are mapped to the same gene by some statistic. This function does this job by keeping the matrix row (normally features) with the higheest statistic specified by the user.

Usage

```
keepMaxStatRow(
  matrix,
  keys,
  keepNArows = TRUE,
  stat = function(x) mean(x, na.rm = TRUE),
  levels = c("rownames", "attribute", "discard"),
  ...
)
```

Arguments

`matrix`	A numeric matrix
`keys`	A vector of character giving the keys the rows are mapped to. A common scenario is that each row represents one probeset, while the vector keys give the genes that the probesets are mapped to. Thus keys can be redundant, namely multiple probesets can map to the same gene.
`keepNArows`	Logical, whether rows with `NA` as their keys should be kept (`TRUE`) or should be discarded (`FALSE`)
`stat`	The function to calculate the univariate statistic. By default the `NA`-robust mean is used.
`levels`	How should the information of the levels of keys, e.g. unique keys, be kept. `dicard` will discard this information, `rownames` will make the unique keys (potentially with `NA`s) as row names of the output matrix, and `attribute` will append an attribute named `levels` to the output matrix.
`...`	Other parameters passed to the `stat` function

Details

`isMaxStatRow` returns a logical vector, with rows with maximal statistics each key as `TRUE` and otherwise as `FALSE`. `keepMaxStatRowInd` returns the integer indices of such rows. Finally `keepMaxStatRow` returns the resulting matrices.

For use see examples

Value

A numeric matrix with rows mapped to unique keys, selected by the maximum statistics. See examples below

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
myFun1 <- function(x) mean(x, na.rm=TRUE)
myFun2 <- function(x) sd(x, na.rm=TRUE)
mat1 <- matrix(c(1,3,4,-5,
                 0,1,2,3,
                 7,9,5,3,
                 0,1,4,3), ncol=4, byrow=TRUE)
keys1 <- c("A", "B", "A", "B")

isMaxStatRow(mat1, keys1, stat=myFun1)
isMaxStatRow(mat1, keys1, stat=myFun2)

keepMaxStatRowInd(mat1, keys1, stat=myFun1)
keepMaxStatRowInd(mat1, keys1, stat=myFun2)

keepMaxStatRow(mat1, keys1, stat=myFun1)
keepMaxStatRow(mat1, keys1, stat="myFun2")
keepMaxStatRow(mat1, keys1, stat="myFun2", levels="discard")
keepMaxStatRow(mat1, keys1, stat="myFun2", levels="attribute")

mat2 <- matrix(c(1,3,4,5,
                 0,1,2,3,
                 7,9,5,3,
                 0,1,4,3,
                 4,0,-1,3.1,
                 9,4,-3,2,
                 8,9,1,2,
                 0.1,0.2,0.5,NA,
                 NA, 4, 3,NA), ncol=4, byrow=TRUE,
               dimnames=list(LETTERS[1:9], NULL))
keys2 <- c("A", "B", "A", "B", NA, NA, "C", "A", "D")

isMaxStatRow(mat2, keys2, keepNArows=FALSE, stat=myFun1)
keepMaxStatRowInd(mat2, keys2, keepNArows=FALSE, stat=myFun1)

keepMaxStatRow(mat2, keys2, keepNArows=FALSE, stat=myFun1)
keepMaxStatRow(mat2, keys2, keepNArows=TRUE, stat=myFun1)
keepMaxStatRow(mat2, keys2, keepNArows=TRUE, stat=myFun1, levels="discard")
keepMaxStatRow(mat2, keys2, keepNArows=TRUE, stat=myFun1, levels="attribute")
```

`lastChar` *Return last characters from strings*

Description

Return last characters from strings

Usage

```
lastChar(str)
```

Arguments

`str`	A vector of character strings

Value

A vector of the same length, containing last characters

Examples

```
lastChar("Go tell it on the mountain")
lastChar(c("HSV", "FCB", "BVB"))
```

`libordie` *Load a library mutedly and quit (die) in case of failing*

Description

The specified library is loaded mutedly by suppressing all messages. If the library is not found, or its version under the specification of `minVer`, the R session dies with a message.

Usage

```
libordie(package, minVer, missing.quit.status = 1, ver.quit.status = 1)
```

Arguments

`package`	One package name (can be character or non-quoted symbol (see examples)
`minVer`	Optional, character string, the minimum working version
`missing.quit.status`	Integer, the status of quitting when the package was not found
`ver.quit.status`	Integer, the status of quitting when the package was found, but older than the minimum working version

Details

Only one package should be tested once.

Value

`NULL` if success, otherwise the session will be killed.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

The function calls `qqmsg` internally to kill the session

Examples

```
libordie(stats)
libordie("methods")
libordie(base, minVer="2.15-1")
```

`list2df` *Transform a list of character strings into a data.frame*

Description

Transform a list of character strings into a data.frame

Usage

```
list2df(list, names = NULL, col.names = c("Name", "Item"))
```

Arguments

`list`	A list of character strings
`names`	Values in the 'Name' column of the result, used if the input list has no names
`col.names`	Column names of the `data.frame`

Value

A `data.frame`

Examples

```
myList <- list(HSV=c("Mueller", "Papadopoulos", "Wood"), FCB=c("Lewandowski", "Robben", "Hummels"),
               BVB=c("Reus", "Goetze", "Kagawa"))
list2df(myList, col.names=c("Club", "Player"))
```

`listOverlapCoefficient` *Pairwise overlap coefficient of lists*

Description

Pairwise overlap coefficient of lists

Usage

```
listOverlapCoefficient(x, y = NULL, checkUniqueNonNA = TRUE)
```

Arguments

`x`	A list of vectors that are interpreted as sets of elements
`y`	A list of vectors that are interpreted as sets of elements. In case of `NULL`, pairwise overlap coefficient of lists in `x` is returned.
`checkUniqueNonNA`	Logical, should vectors in the list be first cleaned up so that NA values are removed and the elements are made unique? Default is set as `TRUE`; if the user is confident that the vectors are indeed valid sets, this option can be set as `FALSE` to speed up the code

Value

A matrix of column-wise pairwise overlap coefficients.

Examples

```
set.seed(1887)
testSets1 <- sapply(rbinom(10, size=26, prob=0.3),
  function(x) sample(LETTERS, x, replace=FALSE))
names(testSets1) <- sprintf("List%d", seq(along=testSets1))
testSets1Poe <- listOverlapCoefficient(testSets1)
testSets1PoeNoCheck <- listOverlapCoefficient(testSets1, checkUniqueNonNA=FALSE)
stopifnot(identical(testSets1Poe, testSets1PoeNoCheck))

testSets2 <- sapply(rbinom(15, size=26, prob=0.3),
  function(x) sample(LETTERS, x, replace=FALSE))
names(testSets2) <- sprintf("AnotherList%d", seq(along=testSets2))
testSets12Poe <- listOverlapCoefficient(testSets1, testSets2)
```

`longdf2matrix` *Convert a long-format data frame into matrix*

Description

Input data.frame must contain at least three columns: one contains row names (specified by `row.col`), one contains column names (`column.col`), and one contains values in matrix cells (`value.col`). The output is a 2D matrix.

Usage

```
longdf2matrix(
  df,
  row.col = 1L,
  column.col = 2L,
  value.col = 3L,
  missingValue = NULL
)
```

Arguments

`df`	Long-format data frame
`row.col`	Character or integer, which column of the input data.frame contains row names?
`column.col`	Character or integer, which column contains column names?
`value.col`	Character or integer, which column contains matrix values?
`missingValue`	Values assigned in case of missing data

Value

A 2D matrix equivalent to the long-format data frame

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`matrix2longdf`

Examples

```
test.df <- data.frame(H=c("HSV", "BVB", "HSV", "BVB"),
A=c("FCB", "S04", "S04", "FCB"),
score=c(3, 1, 1, 0))
longdf2matrix(test.df, row.col=1L, column.col=2L, value.col=3L)

data(Indometh)
```

```
longdf2matrix(Indometh, row.col="time", column.col="Subject",value.col="conc")
longdf2matrix(Indometh, row.col="Subject", column.col="time", value.col="conc")
```

`matchColumn` *Match a column in data.frame to a master vector*

Description

Given a vector known as master vcector, a data.frame and one column of the data.frame, the function `matchColumnIndex` matches the values in the column to the master vector, and returns the indices of each value in the column with respect to the vector. The function `matchColumn` returns whole or subset of the data.frame, with the matching column in the exact order of the vector.

Usage

```
matchColumn(vector, data.frame, column, multi = FALSE)
```

Arguments

`vector`	A vector, probably of character strings.
`data.frame`	A `data.frame` object
`column`	The column name (character) or index (integer between 1 and the column number), indicating the column to be matched. Exceptionally `0` is as well accepted, which will match the row names of the `data.frame` to the given vector.
`multi`	Logical, deciding what to do if a value in the vector is matched to several values in the data.frame column. If set to `TRUE`, all rows containing the matched value in the specified column are returned; otherwise, when the value is set to `FALSE`, one arbitrary row is returned. See details and examples below.

Details

See more details below.

The function is used to address the following question: how can one order a `data.frame` by values of one of its columns, the order for which is given in a vector (known as “master vector”). `matchColumnIndex` and `matchColumn` provide thoroughly-tested implementation to address this question.

For `one-to-one` cases, where both the column and the vector have no duplicates and can be matched one-to-one, the question is straightforward to solve with the `match` function in R. In `one-to-many` or `many-to-many` matching cases, the parameter `multi` determines whether multiple rows matching the same value should be shown. If `mutli=FALSE`, then the sorted data.frame that are returned has exactly the same row number as the input vector; otherwise, the returned data.frame has more rows. See the examples below.

In either case, in the returned `data.frame` object by `matchColumn`, values in the column used for matching are overwritten by the master vector.If `multi=TRUE`, the order of values in the column is also obeying the order of the master vector, with exceptions of repeating values casued by mutliple matching.

The `column` parameter can be either character string or non-negative integers. In the exceptional case, where `column=0L` (“L” indicates integer), the row names of the `data.frame` is used for matching instead of any of the columns.

Both functions are NA-friendly, since NAs in neither vector nor column should break the code.

Value

For `matchColumnIndex`, if `multi` is set to `FALSE`, an integer vector of the same length as the master vector, indicating the order of the `data.frame` rows by which the column can be re-organized into the master vector. When `multi` is `TRUE`, the returning object is a list of the same length as the master vector, each item containing the index (indices) of data.frame rows which match to the master vector.

For `matchColumn`, a data.frame is always returned. In case `multi=FALSE`, the returning data frame has the same number of rows as the length of the input master vector, and the column which was specified to match contains the master vector in its order. If `multi=TRUE`, returned data frame can contain equal or more numbers of rows than the master vector, and multiple-matched items are repeated.

Note

When `multi=TRUE`, the indices within each list element (for `matchColumnIndex`) are returned in ascending order.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

See `match` for basic matching operations.

Examples

```
df <- data.frame(Team=c("HSV", "BVB", "HSC", "FCB", "HSV"),
                 Pkt=c(25,23,12,18,21),
                 row.names=c("C", "B", "A", "F", "E"))
teams <- c("HSV", "BVB", "BRE", NA)
ind <- c("C", "A", "G", "F", "C", "B", "B", NA)

matchColumnIndex(teams, df, 1L, multi=FALSE)
matchColumnIndex(teams, df, 1L, multi=TRUE)
matchColumnIndex(teams, df, "Team", multi=FALSE)
matchColumnIndex(teams, df, "Team", multi=TRUE)
matchColumnIndex(teams, df, 0, multi=FALSE)
matchColumnIndex(ind, df, 0, multi=FALSE)
matchColumnIndex(ind, df, 0, multi=TRUE)

matchColumn(teams, df, 1L, multi=FALSE)
```

```
matchColumn(teams, df, 1L, multi=TRUE)
matchColumn(teams, df, "Team", multi=FALSE)
matchColumn(teams, df, "Team", multi=TRUE)
matchColumn(ind, df, 0, multi=FALSE)
matchColumn(ind, df, 0, multi=TRUE)
```

`matchColumnName` *Match a given vector to column names of a data.frame or matrix*

Description

Match a given vector to column names of a data.frame or matrix

Usage

```
matchColumnName(data.frame.cols, reqCols, ignore.case = FALSE)
```

Arguments

`data.frame.cols`	column names of a data.frame. One can also provide a data.frame, which may however cause worse performance since the data.frame is copied
`reqCols`	required columns
`ignore.case`	logical, whether the case is considered

Value

A vector of integers as indices

Examples

```
myTestDf <- data.frame(HBV=1:3, VFB=0:2, BVB=4:6, FCB=2:4)
myFavTeams <- c("HBV", "BVB")
matchColumnName(myTestDf, myFavTeams)
myFavTeamsCase <- c("hbv", "bVb")
matchColumnName(myTestDf, myFavTeamsCase, ignore.case=TRUE)
## NA will be returned in this case if ignore.case is set to FALSE
matchColumnName(myTestDf, myFavTeamsCase, ignore.case=FALSE)
```

`matrix2longdf` *Transform a matrix into a long-format data.frame*

Description

The function converts a matrix into a long-format, three-column data.frame, containing row, columna nd value. Such 'long' data.frames can be useful in data visualization and modelling.

Usage

```
matrix2longdf(
  mat,
  row.names,
  col.names,
  longdf.colnames = c("row", "column", "value")
)
```

Arguments

`mat`	A matrix
`row.names`	Character, row names to appear in the `data.frame`. If missing, the `rownames` of the matrix will be used. If set to `NULL`, or if the matrix `rownames` are `NULL`, a integer index vector starting from 1 will be used.
`col.names`	Charater, column names to appear in the `data.frame`. The rule of handling missing or NULL parameters is the same as `row.names` described above.
`longdf.colnames`	Character, column names of the output long data frame

Details

The function converts a matrix into a three-column, 'long' format data.frame containing row names, column names, and values of the matrix.

Value

A `data.frame` object with three columns: `row`, `column` and `value`. If the input matrix is of dimesion `MxN`, the returning `data.frame` is of the dimension `MNx3`.

Note

The length of `row.names` and `col.names` should be as the same as the matrix dimension. Otherwise the function raises warnings.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
test.mat <- matrix(1:12, ncol=4, nrow=3, dimnames=list(LETTERS[1:3],
LETTERS[1:4]))
print(test.mat)
print(matrix2longdf(test.mat))
print(matrix2longdf(test.mat, longdf.colnames=c("From", "To", "Time")))
```

`mergeInfreqLevelsByCumsumprop`

Merge infrequent levels by setting the threshold of the proportion of cumulative sum over sum a.k.a. cumsumprop

---

Description

Merge infrequent levels by setting the threshold of the proportion of cumulative sum over sum a.k.a. cumsumprop

Usage

```
mergeInfreqLevelsByCumsumprop(
  classes,
  thr = 0.9,
  mergedLevel = "others",
  returnFactor = TRUE
)
```

Arguments

`classes`	Character strings or factor.
`thr`	Numeric, between 0 and 1, how to define frequent levels. Default: 0.9, namely levels which make up over 90% of all instances.
`mergedLevel`	Character, how the merged level should be named.
`returnFactor`	Logical, whether the value returned should be coereced into a factor.

Value

A character string vector or a factor, of the same length as the input `classes`, but with potentially fewer levels.

Note

In case only one class is deemed as infrequent, its label is unchanged.

Examples

```
set.seed(1887)
myVals <- sample(c(rep("A", 4), rep("B", 3), rep("C", 2), "D"))
## in the example below, since A, B, C make up of 90% of the total,
## D is infrequent. Since it is alone, it is not merged
mergeInfreqLevelsByCumsumprop(myVals, 0.9)
mergeInfreqLevelsByCumsumprop(myVals, 0.9, returnFactor=FALSE) ## return characters
## in the example below, since A and B make up 70% of the total,
## and A, B, C 90%, they are all frequent and D is infrequent.
## Following the logic above, no merging happens
mergeInfreqLevelsByCumsumprop(myVals, 0.8)
mergeInfreqLevelsByCumsumprop(myVals, 0.7) ## A and B are left, C and D are merged
mergeInfreqLevelsByCumsumprop(myVals, 0.5) ## A and B are left, C and D are merged
mergeInfreqLevelsByCumsumprop(myVals, 0.4) ## A is left
mergeInfreqLevelsByCumsumprop(myVals, 0.3) ## A is left
```

`midentical` *Multiple identical*

Description

Testing whether multiple objects are identical

Usage

```
midentical(
  ...,
  num.eq = TRUE,
  single.NA = TRUE,
  attrib.as.set = TRUE,
  ignore.bytecode = TRUE,
  ignore.environment = FALSE,
  ignore.srcref = TRUE,
  extptr.as.ref = FALSE
)
```

Arguments

`...`	Objects to be tested, or a list of them
`num.eq`, `single.NA`, `attrib.as.set`, `ignore.bytecode`	See `identical`
`ignore.environment`, `ignore.srcref`	See `identical`
`extptr.as.ref`	See `identical`, new parameter since R-4.2

Details

`midentical` extends `identical` to test multiple objects instead of only two.

Value

A logical value, `TRUE` if all objects are identical

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`identical`

Examples

```
set1 <- "HSV"
set2 <- set3 <- set4 <- c("HSV", "FCB")

midentical(set1, set2)
midentical(list(set1, set2))

midentical(set2, set3, set4)
midentical(list(set2, set3, set4))

## other options passed to identical
midentical(0, -0, +0, num.eq=FALSE)
midentical(0, -0, +0, num.eq=TRUE)
```

`mmatch` *Multiple matching*

Description

Multiple matching between two vectors. Different from R-native `match` function, where only one match is returned even if there are multiple matches, `mmatch` returns all of them.

Usage

```
mmatch(x, table, nomatch = NA_integer_)
```

Arguments

`x`	vector or `NULL`: the values to be matched.
`table`	vector or `NULL`: the values to be matched against.
`nomatch`	the value to be returned in case when no match is found.

Details

Multiple matches can be useful in many cases, and there is no native R function for this purpose. User can write their own functions combining `lapplying` with `match` or `%in%`, our experience however shows that such non-vectorized function can be extremely slow, especially when the `x` or `table` vector gets longer.

`mmatch` delegates the multiple-matching task to a C-level function, which is optimized for speed. Internal benchmarking shows improvement of hundred fold, namely using `mmatching` costs about 1/100 of the time used by R-implementation.

Value

A list of the same length as the input `x` vector. Each list item contains the matching indices in ascending order (similar to `match`).

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>, C-code was adapted from the program written by Roland Schmucki.

See Also

`match`

Examples

```
vec1 <- c("HSV", "BVB", "FCB", "HSV", "BRE", "HSV", NA, "BVB")
vec2 <- c("FCB", "FCN", "FCB", "HSV", "BVB", "HSV", "FCK", NA, "BRE", "BRE")

mmatch(vec1, vec2)

## compare to match
match(vec1, vec2)
```

`munion` *Operations for multiple sets*

Description

Set operation functions in the base package, `union`, `intersect` and `setdiff`, can only be applied to binary manipulations involving two sets. Following functions, `munion`, `mintersect` and `msetdiff`, extend their basic versions to deal with multiple sets.

Usage

```
munion(...)
```

Arguments

`...`	Vectors of items, or a list of them. See examples below.

Details

These functions apply set manipulations (union, intersect, or difference) in a sequential manner: the first two sets are considered first, then the third, the fourth and so on, till all sets have been visited.

Value

A vector of set operation results. Can be an empty vector if no results were returned.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`union`, `intersect` and `setdiff`.

Examples

```
set1 <- c("HSV", "FCB", "BVB", "FCN", "HAN")
set2 <- c("HSV", "FCB", "BVB", "HAN")
set3 <- c("HSV", "BVB", "FSV")

munion(set1, set2, set3)
mintersect(set1, set2, set3)
msetdiff(set1, set2, set3)

## sets can be given in a list as well
munion(list(set1, set2, set3))
mintersect(list(set1, set2, set3))
msetdiff(list(set1, set2, set3))
```

`na.false` *Replace NA with FALSE*

Description

Replace `NA` in a vector with `FALSE`

Usage

```
na.false(x)
```

Arguments

`x`	A logical vector or matrix

Value

Logical vector or matrix with NAs replaced by FALSE

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

myX <- c("HSV", "FCK", "FCN", NA, "BVB") res <- myX == "HSV" na.false(res)

`naivePairwiseDist` *Calculate pairwise distances between each pair of items in a list*

Description

Calculate pairwise distances between each pair of items in a list

Usage

```
naivePairwiseDist(list, fun = jaccardIndex)
```

Arguments

`list`	A list
`fun`	A function that receives two vectors (such as jaccardIndex) and returns a number (scale)

Value

A symmetric matrix of dimension `mxm`, where `m` is the length of the list

This function is inefficient compared with matrix-based methods. It is exported just for education and for verifying results of matrix-based methods.

Examples

```
myList <- list(first=LETTERS[3:5], second=LETTERS[1:3], third=LETTERS[1:5], fourth=LETTERS[6:10])
naivePairwiseDist(myList, fun=jaccardIndex)
## despite of the name, any function that returns a number can work
naivePairwiseDist(myList, fun=jaccardDistance)
```

`ofactor` *Ordered factor*

Description

Build a factor using the order of input character strings

Usage

```
ofactor(x, ...)
```

Arguments

`x`	A vector of character strings
`...`	Other parameters passed to `factor`

Value

Factor with levels in the same order of the input strings.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`factor`

Examples

```
testStrings <- c("A", "C", "B", "B", "C")
(testFac <- factor(testStrings))
(testOfac <- ofactor(testStrings))

stopifnot(identical(levels(testOfac), c("A", "C", "B")))
```

`orderCutgroup` *Reorder the groups by their group size*

Description

Reorder the groups by their group size

Usage

```
orderCutgroup(groups, decreasing = TRUE)
```

Arguments

`groups`	Named vectors of integers as group indices
`decreasing`	Logical, should the first group be the largest? The function permutes a vector of names integers so that the names matching the same integer match to the same or another integer, while assuring that the permuted group matching the first integer (or the last integer if `decreasing` is set to `FALSE`) is the largest group by count.

`overlapCoefficient` *Overlap coefficient, also known as Szymkiewicz-Simpson coefficient*

Description

Overlap coefficient, also known as Szymkiewicz-Simpson coefficient

Usage

```
overlapCoefficient(x, y, checkUniqueNonNA = FALSE)

overlapDistance(x, y, checkUniqueNonNA = FALSE)
```

Arguments

`x`	A vector
`y`	A vector
`checkUniqueNonNA`	Logical, if `TRUE`, `x` and `y` are made unique and non-NA

Value

The overlap coefficient

See Also

`jaccardIndex`

`overlapCofficient` calculates the overlap coefficient, and `overlapDistance` is defined by 1-`overlapCoefficient`.

Examples

```
myX <- 1:6
myY <- 4:9
overlapCoefficient(myX, myY)

myY2 <- 4:10
overlapCoefficient(myX, myY2)
## compare the result with Jaccard Index
jaccardIndex(myX, myY2)

## overlapDistance
overlapDistance(myX, myY2)
```

`overwriteDir` *Overwrite a directory*

Description

Overwrite a directory

Usage

```
overwriteDir(dir, action = c("ask", "overwrite", "append", "no"))
```

Arguments

`dir`	Chacater, path to a directory.
`action`	Ask the user to input the option (`ask`), or one of the following options: `overwrite`, `append`, and `no`. See below for other options.

Value

If `action` is set to `overwrite`, the directory will be deleted recursively if it exists, a new directory with the same name will be created, and the function returns `TRUE`. If `append` is set, the function creates the directory if necessary and returns `TRUE`. If `no` is set, the function does nothing and returns.

If `action` is set to `ask`, user will be prompted for actions.

If `overwrite` is set, the directory will be removed and written anew.

If `append` is set, in contrast to `overwrite`, the directory and the files in it are not removed if they exists. In this case, files with the same name will be overwritten. Otherwise, new directories or files will be simply created. On the other hand, if the directory does not exist, it will be created.

If `no` is set, no action will be taken. The funciton returns `FALSE`.

Examples

```
createTempDir <- function() {
  tmpdir <- tempdir()
  tmpfile1 <- tempfile(tmpdir=tmpdir)
  tmpfile2 <- tempfile(tmpdir=tmpdir)

  writeLines("First file", tmpfile1)
  writeLines("Second file", tmpfile2)
  return(tmpdir)
}
newTempFile <- function(tmpdir) {
  writeLines("Third file", tempfile(tmpdir=tmpdir))
}

  tmpdir <- createTempDir()
  overwriteDir(tmpdir, action="ask")

  ## overwrite: delete the directory and create it a new
  tmpdir <- createTempDir()
  fileCount <- length(dir(tmpdir))
  dir(tmpdir) ## two files should be there
  overwriteDir(tmpdir, action="overwrite")
  newTempFile(tmpdir)
  dir(tmpdir) ## now there should be only one file
  stopifnot(length(dir(tmpdir))==1)

  ## append: append files, and overwrite if a file of the same name is there
  overwriteDir(tmpdir, action="append")
  newTempFile(tmpdir)
  dir(tmpdir) ## a new file is written
  stopifnot(length(dir(tmpdir))==2)

  ## no: no action, and returns FALSE
  noRes <- overwriteDir(tmpdir, action="no")
  stopifnot(!noRes)
```

`pAbsLog10Score` *Transform p-values to continuous scores with the absolute-log10 transformation*

Description

The function maps p values between 0 and 1 to continuous scores ranging on `R` by the following equation: $abs(log10(p)) * sign$

Usage

```
pAbsLog10Score(p, sign = 1, replaceZero = TRUE)
```

Arguments

`p`	*p*-value(s) between (0,1]
`sign`	Sign of the score, either positive (in case of positive numbers), negative (in case of negative numbers), or zero. In case a logical vector, `TRUE` is interpreted as positive and `FALSE` is interpreted as negative.
`replaceZero`	Logical, whether to replace zero p-values with the minimal double value specified by the machine. Default is `TRUE`. If set to `FALSE`, results will contain infinite values.

Value

A numeric vector of transformed p-values using signed -log10 transformation.

See Also

`pQnormScore`, `pScore`, `replaceZeroPvalue`

Examples

```
testPvals <- c(0.001, 0.01, 0.05, 0.1, 0.5, 1)
pAbsLog10Score(testPvals)
testPvalSign <- rep(c(-1,1), 3)
pAbsLog10Score(testPvals, sign=testPvalSign)
testLog <- rep(c(TRUE, FALSE),3)
pAbsLog10Score(testPvals, testLog)
```

`pairwiseJaccardIndex` *Calculate pairwise Jaccard Indices between each pair of items in a list*

Description

Calculate pairwise Jaccard Indices between each pair of items in a list

Usage

```
pairwiseJaccardIndex(list)

pairwiseJaccardDistance(list)
```

Arguments

`list`	A list

Value

A symmetric matrix of dimension `mxm`, where `m` is the length of the list

`pairwiseJaccardDistance` is defined as `1-pairwiseJaccardIndex`.

Examples

```
myList <- list(first=LETTERS[3:5], second=LETTERS[1:3], third=LETTERS[1:5], fourth=LETTERS[6:10])
pairwiseJaccardIndex(myList)

poormanPJI <- function(list) {
  sapply(list, function(x) sapply(list, function(y) jaccardIndex(x,y)))
}
stopifnot(identical(pairwiseJaccardIndex(myList), poormanPJI(myList)))
```

`pairwiseOverlapDistance` *Calculate pairwise overlap coefficients between each pair of items in a list*

Description

Calculate pairwise overlap coefficients between each pair of items in a list

Usage

```
pairwiseOverlapDistance(list)

pairwiseOverlapCoefficient(list)
```

Arguments

`list`	A list

Value

A symmetric matrix of dimension `mxm`, where `m` is the length of the list

`pairwiseOverlapDistance` is defined the pairwise overlap distance.

Examples

```
myList <- list(first=LETTERS[3:5], second=LETTERS[1:3], third=LETTERS[1:5], fourth=LETTERS[6:10])
pairwiseOverlapCoefficient(myList)
pairwiseOverlapDistance(myList)

poormanPOC <- function(list) {
  sapply(list, function(x) sapply(list, function(y) overlapCoefficient(x,y)))
}
stopifnot(identical(pairwiseOverlapCoefficient(myList), poormanPOC(myList)))
```

`percentage` *Print a decimal number in procent format*

Description

Print a decimal number in procent format

Usage

```
percentage(x, fmt = "1.1")
```

Arguments

`x`	a decimal number, usually between -1 and 1
`fmt`	format string, '1.1' means a digit before and after the decimal point will be printed

Value

Character string

Examples

```
percentage(c(0,0.1,0.25,1))
percentage(c(0,0.1,0.25,1), fmt="1.4")
percentage(c(0,-0.1,0.25,-1), fmt="+1.1")
```

`pQnormScore` *Transform p-values to continuous scores with the quantile function of the normal distribution*

Description

Quantile function, also known as the inverse of cumulative distribution function of the normal distribution, is used to map p-values to continuous scores raging on R. The signs of the resulting scores are positive by default and are determined by the parameter `sign`.

Usage

```
pQnormScore(p, sign = 1, replaceZero = TRUE)
```

Arguments

`p`	*p*-value(s) between $(0, 1]$
`sign`	Signs of the scores, either positive (in case of positive numbers), negative (in case of negative numbers), or zero. In case of a logical vector, `TRUE` is interpreted as positive and `FALSE` is interpreted as negative.
`replaceZero`	Logical, whether to replace zero p-values with the minimal double value specified by the machine. Default is `TRUE`. If set to `FALSE`, results will contain infinite values.

Value

A numeric vector of transformed p-values using signed quantile normal transformation.

See Also

`pAbsLog10Score`, `pScore`, `double`

Examples

```
testPvals <- c(0.001, 0.01, 0.05, 0.1, 0.5, 1)
pQnormScore(testPvals)
testPvalSign <- rep(c(-1,1), 3)
pQnormScore(testPvals, sign=testPvalSign)
testLog <- rep(c(TRUE, FALSE),3)
pQnormScore(testPvals, testLog)
```

`print.BEDAinfo` *Print BEDAinfo object*

Description

Print BEDAinfo object

Usage

```
## S3 method for class 'BEDAinfo'
print(x, ...)
```

Arguments

`x`	A BEDA info object, returned by `bedaInfo`
`...`	Ignored

Value

Invisible `NULL`, only side effect is used

Examples

```
print(bedaInfo())
```

`pScore` *Transform p-values to continuous scores*

Description

The function wraps other functions to map p values ranging on $(0, 1]$ to continuous scores ranging on R in a number of ways.

Usage

```
pScore(p, sign = 1, method = c("qnorm", "absLog10"), replaceZero = TRUE)
```

Arguments

`p`	p-value between (0,1]
`sign`	Sign of the score, either positive (in case of positive numbers), negative (in case of negative numbers), or zero. In case a logical vector, `TRUE` is interpreted as positive and `FALSE` is interpreted as negative.
`method`	Currently available methods include `qnorm` and `absLog10`.
`replaceZero`	Logical, whether to replace zero p-values with the minimal double value specified by the machine. Default is `TRUE`. If set to `FALSE`, results will contain infinite values.

Value

A numeric vector of transformed p-values using the specified method.

See Also

`pAbsLog10Score`, `pQnormScore`

Examples

```
testPvals <- c(0.001, 0.01, 0.05, 0.1, 0.5, 1)
pScore(testPvals, method="absLog10")
pScore(testPvals, method="qnorm")
testPvalSign <- rep(c(-1,1), 3)
pScore(testPvals, sign=testPvalSign, method="absLog10")
pScore(testPvals, sign=testPvalSign, method="qnorm")
testLog <- rep(c(TRUE, FALSE),3)
pScore(testPvals, testLog, method="absLog10")
pScore(testPvals, testLog, method="qnorm")

testPvals <- 10^seq(-5, 0, 0.05)
```

```
plot(pScore(testPvals, method="qnorm"),
     pScore(testPvals, method="absLog10"),
     xlab="pQnormScore", ylab="pAbsLog10Score"); abline(0,1, col="red", lty=2)
```

`putColsFirst` *Rearrange columns to put some columns to far left*

Description

This function is helpful to export tables where certain columns are desired to be placed to the most left of the data.frame

Usage

```
putColsFirst(data.frame, columns)
```

Arguments

`data.frame`	Data.frame
`columns`	Character vector, names of columns which are to be put to the left

Value

data.frame with re-arranged columns

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
clubs <- data.frame(Points=c(21,23,28,24), Name=c("BVB", "FCB", "HSV",
"FCK"), games=c(12,11,11,12))
putColsFirst(clubs, c("Name"))
putColsFirst(clubs, c("Name", "games"))
```

`pwdecode` *Decode password with function implemented with pwencode*

Description

Decode password encypted with pwencode.

Usage

```
pwdecode(password)
```

Arguments

`password`	Character string to be decoded. If starting with a empty character, the string is sent for decoding; otherwise, it is deemed as clear text password and returned.

Details

See pwdecode function documentation in BIOS for implemetnation details.

Note that since R does not support strings embedding null values (`\000`), the password to be decoded has to be given with two slashes, e.g. ‘ \001\000\129\235’.

Value

Decoded character string, or empty string if decoding fails

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>. The C library code was written by Detlef Wolf.

Examples

```
mycode <- " \\001\\000\\141\\314\\033\\033\\033\\033\\033\\142\\303\\056\\166\\311\\037\\042"
pwdecode(mycode)
```

`pwencode` *Encode a password*

Description

Encode a password

Usage

```
pwencode(label = "VAR", key)
```

Arguments

`label`	label used to encode the password
`key`	password key

Value

Character string, encoded password

`qqmsg` *Quitely Quit with Messages*

Description

Quitely quit R with messages in non-interactive sessions

Usage

```
qqmsg(..., status = 0, save = FALSE, runLast = TRUE)
```

Arguments

`...`	Messages to be passed to `message`
`status`	Quit stats
`save`	Logical, should current working environment be saved?
`runLast`	Logical, should `.Last()` be executed?

Details

The function prints messages in any case, and quits R if the current session is non-interactive, e.g. in the command-line running Rscript mode

Value

Invisible `NULL`, only side effect is used.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`quit`

Examples

```
## the example should not run because it will lead the R session to quit
## Not run:
  qqmsg()
  qqmsg("die", status=0)
  qqmsg("Avada kedavra", status=-1)
  qqmsg("Crucio!", "\n", "Avada kedavra", status=-100)

## End(Not run)
```

`qsystem` *Quietly runs a system command*

Description

Quietly runs a system command: the output is internalized and returned as an invisible variable, and the standard error output is ignored.

Usage

```
qsystem(command)
```

Arguments

`command`	A system command

Details

The function runs the system command in a quiet mode. The function can be useful in CGI scripts, for instance

Value

(Invisibly) the internalized output of the command

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
dateIntern <- system("date")
```

`refactorNum` *Sort numeric factor levels by values*

Description

Factor variables with numbers as levels are alphabetically ordered by default, which requires rearrangements for various purposes, e.g. modelling or visualizations. This function re-orders levels of numeric factor variables numerically.

Usage

```
refactorNum(x, decreasing = FALSE)
```

Arguments

`x`	A factor variable with numeric values as levels
`decreasing`	Logical, should the levels sorted descendingly?

Value

A factor variable, with sorted numeric values as levels

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
(nums <- factor(c("2","4","24","1","2","125","1","2","125")))
(nums.new <- refactorNum(nums))
```

`registerLog` *The functions* `registerLog` *and* `doLog` *provide a simple mechanism to handle loggings (printing text messages to files or other types of connections) in R.*

Description

Users can register arbitrary numbers of loggers with `registerLog`, and the functions take care of low-level details such as openning and closing the connections.

Usage

```
registerLog(..., append = FALSE)
```

Arguments

`...`	Arbitrary numbers of file names (character strings) or connection objects (see example).
`append`	Logical, log will be appended to the existing file but not overwriting. Only valid for files but not for connections such as standard output.

Details

Input parameters can be either character strings or connections (such as the objects returned by `stdout()` or `pipe()`.

If a character string is registered as a logger, it is assumed as a file name (user must make sure that it is writable/appendable). In case the file exists, new logging messages will be *appended*; otherwise if the file does not exists, it will be created and the logging messages will be `written` to the file.

A special case is the parameter value `"-"`: it will be interpreted as standard output.

if a connection is registered as a logger, it must be writable in order to write the logging messages.

Each parameter will be converted to a `connection` object, which will be `closed` (when applicable) automatically before R quits.

If the parameter is missing (or set to `NA` or `NULL`), no logging will take place.

Value

No value returned: its side effect is used.

Note

Currently, the loggers are stored in a variable in the namespace of ribiosUtils named `RIBIOS_LOGGERS`. This is only for internal use of the package and may change any time, therefore users are not advised to manipulate this variable directly.

To clear the registered loggers, use `clearLog`.To flush the registered loggers, use `flushLog`. Usually it is not necessary to use `flushLog` in R scripts, since by program exit the active R session will automatically flush and close the connections (in addition, frequent flushing may decrease the program's efficiency). However, if used in interactive sessions, sometimes `flushLog` is needed to force R write all log files to all connections that are registered.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`doLog` writes messages iteratively to each connection registered by `registerLog`.

Examples

```
logfile1 <- tempfile()
logfile2 <- tempfile()
logcon3 <- stdout()
if(.Platform$OS.type == "unix") {
  registerLog("/dev/null")
} else {
  registerLog(tempfile())
}
registerLog(logfile1)
registerLog(logfile2)
registerLog(logcon3)

doLog("Start logging")
doLog("Do something...")
doLog("End logging")

flushLog() ## usually not needed, see notes

txt1 <- readLines(logfile1)
txt2 <- readLines(logfile2)

cat(txt1)
cat(txt2)

clearLog()

registerLog(logfile1, logfile2, logcon3)

doLog("Start logging - round 2")
doLog("Do something again ...")
doLog("End logging - for good")

flushLog() ## usually not needed, see notes

txt1 <- readLines(logfile1)
txt2 <- readLines(logfile2)

cat(txt1)
```

```
cat(txt2)

## clean up files and objects to close unused connections
closeLoggerConnections()
```

`relevels` *Relevel a factor by a named or unnamed vector.*

Description

This function wraps `relevelsByNamedVec` for named vector and `relevelsByNotNamedVec` for not named vectors

Usage

```
relevels(
  x,
  refs,
  missingLevels = c("pass", "warning", "error"),
  unrecognisedLevels = c("warning", "pass", "error")
)
```

Arguments

`x`	A factor or a character string vector that will be cast into factor
`refs`	A named vector or unnamed vector.
`missingLevels`	Actions taken in case existing levels are missing: 'pass', 'warning', or 'error'.
`unrecognisedLevels`	Actions taken in case unrecognised levels are found: 'pass', 'warning', or 'error'.

Value

A vector of factor

See Also

`relevelsByNamedVec` and `relevelsByNotNamedVec`

Examples

```
oldFactor <- factor(c("A", "B", "A", "C", "B"), levels=LETTERS[1:3])
refLevels <- c("B", "C", "A")
refDict <- c("A"="a", "B"="b", "C"="c")
newFactor <- relevels(oldFactor, refLevels)
stopifnot(identical(newFactor, factor(c("A", "B", "A", "C", "B"), levels=c("B", "C", "A"))))
newFactor2 <-  relevels(oldFactor, refDict)
stopifnot(identical(newFactor2, factor(c("a", "b", "a", "c", "b"), levels=c("a", "b", "c"))))
```

`relevelsByNamedVec` *Relevel a factor by a named vector.*

Description

If names contain character strings other than the levels in the old factor and warning is set to `TRUE`, a warning will be raised.

Usage

```
relevelsByNamedVec(
  x,
  refs,
  missingLevels = c("pass", "warning", "error"),
  unrecognisedLevels = c("warning", "pass", "error")
)
```

Arguments

`x`	A factor
`refs`	A named vector. The names of the vector are all or a subset of levels in the old factor. And the values are new levels
`missingLevels`	Actions taken in case existing levels are missing: 'pass', 'warning', or 'error'.
`unrecognisedLevels`	Actions taken in case unrecognised levels are found: 'pass', 'warning', or 'error'.

Details

The levels of the factor are the names of the `ref` vector, and the order of the `ref` vector matters: it is the levels of the new factor.

Value

A vector of factor

Examples

```
oldFactor <- factor(c("A", "B", "A", "C", "B"), levels=LETTERS[1:3])
factorDict <- c("A"="a", "B"="b", "C"="c")
newFactor <- relevelsByNamedVec(oldFactor, factorDict)
stopifnot(identical(newFactor, factor(c("a", "b", "a", "c", "b"), levels=c("a", "b", "c"))))
## TODO: test warning and error
```

`relevelsByNotNamedVec` *Relevel a factor by a unnamed vector.*

Description

If names contain character strings other than the levels in the old factor and warning is set to `TRUE`, a warning will be raised

Usage

```
relevelsByNotNamedVec(
  x,
  refs,
  missingLevels = c("pass", "warning", "error"),
  unrecognisedLevels = c("warning", "pass", "error")
)
```

Arguments

`x`	A factor
`refs`	A unnamed vector. The values of the vector are levels of `x`.
`missingLevels`	Actions taken in case existing levels are missing: 'pass', 'warning', or 'error'.
`unrecognisedLevels`	Actions taken in case unrecognised levels are found: 'pass', 'warning', or 'error'.

Value

A vector of factor

Examples

```
oldFactor <- factor(c("A", "B", "A", "C", "B"), levels=LETTERS[1:3])
refLevels <- c("B", "C", "A")
newFactor <- relevelsByNotNamedVec(oldFactor, refLevels)
stopifnot(identical(newFactor, factor(c("A", "B", "A", "C", "B"), levels=c("B", "C", "A"))))
## TODO: test warning and error
```

`reload` *Reload a package*

Description

Reload a package by first detaching and loading the library.

Usage

```
reload(pkg)
```

Arguments

`pkg`	Character string, name of the package

Value

Side effect is used.

Note

So far only character is accepted

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`detach` and `library`

Examples

```
## the example should not run because it will reload the package
## Not run:
  reload(ribiosUtils)

## End(Not run)
```

`removeColumns` *Remove columns*

Description

Remove columns from a data.frame object

Usage

```
removeColumns(data.frame, columns, drop = FALSE)
```

Arguments

`data.frame`	data.frame
`columns`	names of columns to be removed
`drop`	Logical, whether the matrix should be dropped to vector if only one column is left

Details

The function is equivalent to the subsetting operation with brackets. It provides a tidy programming interface to manupulate data.frames.

Value

data.frame with specified columns removed

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
clubs <- data.frame(Points=c(21,23,28,24), Name=c("BVB", "FCB", "HSV",
"FCK"), games=c(12,11,11,12))
removeColumns(clubs,c("Name"))
```

`removeColumnsByFunc` *Remove rows or column by function*

Description

Remove rows or column by function

Usage

```
removeColumnsByFunc(matrix, removeFunc)

removeRowsByFunc(matrix, removeFunc)
```

Arguments

`matrix`	A matrix
`removeFunc`	A function which should return boolean results

Value

A matrix with rows or columns whose return value of `removeFunc` is `TRUE`

Examples

```
myMat <- matrix(c(1, 3 ,5, 4, 5, 6, 7, 9, 11), byrow=FALSE, nrow=3)
removeColumnsByFunc(myMat, removeFunc=function(x) any(x %% 2 == 0))
removeRowsByFunc(myMat, removeFunc=function(x) any(x %% 2 == 0))
```

`removeColumnsWithNA` *Remove columns in a matrix that contains one or more NAs*

Description

Remove columns in a matrix that contains one or more NAs

Usage

```
removeColumnsWithNA(mat)
```

Arguments

`mat`	A matrix

Value

A matrix, with columns containing one or more NAs removed

Examples

```
myMat <- matrix(c(1:9, NA, 10:17), nrow=6, byrow=TRUE,
  dimnames=list(sprintf("R%d", 1:6), sprintf("C%d", 1:3)))
removeColumnsWithNA(myMat)
```

`removeInvarCol` *Remove invariable columns from a data frame or matrix*

Description

Columns with one unique value are invariable. The functions help to remove such columns from a data frame (or matrix) in order to highlight the variables.

Usage

```
removeInvarCol(df)
```

Arguments

`df`	A data frame or matrix

Details

`removeInvarCol` the data frame removing invariable column(s).

`isVarCol` and `isInvarCol` are helper functions, returning a logical vector indicating the variable and invariable columns respectively.

Value

`isVarCol` and `isInvarCol` return a logical vector indicating the variable and invariable columns respectively.

`removeInvarCol` removes invariable columns.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
 testDf <- data.frame(a=1:4, b=7, c=LETTERS[1:4])
isVarCol(testDf)
isInvarCol(testDf)
removeInvarCol(testDf)
```

`removeRowsWithNA` *Remove rows in a matrix that contains one or more NAs*

Description

Remove rows in a matrix that contains one or more NAs

Usage

```
removeRowsWithNA(mat)
```

Arguments

`mat`	A matrix

Value

A matrix, with rows containing one or more NAs removed

Examples

```
myMat <- matrix(c(1:9, NA, 10:17), nrow=6, byrow=TRUE,
  dimnames=list(sprintf("R%d", 1:6), sprintf("C%d", 1:3)))
removeRowsWithNA(myMat)
```

`replaceColumnName` *Replace column names in data.frame*

Description

Replace column names in data.frame

Usage

```
replaceColumnName(data.frame, old.names, new.names)
```

Arguments

`data.frame`	A data.frame
`old.names`	Old column names to be replaced
`new.names`	New column names

Value

Data.frame with column names updated

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
clubs <- data.frame(Points=c(21,23,28,24), Name=c("BVB", "FCB", "HSV",
"FCK"), games=c(12,11,11,12))
replaceColumnName(clubs, c("Points", "games"), c("Punkte", "Spiele"))
```

`replaceZeroPvalue` *Replace p-values of zero*

Description

Replace p-values of zero

Usage

```
replaceZeroPvalue(p, factor = 1)
```

Arguments

`p`	A numeric vector, containing p-values. Zero values will be replaced by a small, non-zero value.
`factor`	A numeric vector, the minimal p-value will be multiplied by it. Useful for `pQnormScore`, because there the p-value needs to be divided by two, therefore a factor of two makes sense.

Value

A numeric vector of the same length as the input vector, with zeros replaced by the minimal absolute double value defined by the machine multiplied by the factor.

Note

Values under the minimal positive double value are considered zero and replaced.

Examples

```
ps <- seq(0,1,0.1)
replaceZeroPvalue(ps)
replaceZeroPvalue(ps, factor=2)
```

`ribiosTempdir` *A temporary directory which (1) every machine in the cluster has access to and (2) has sufficient space*

Description

A temporary directory which (1) every machine in the cluster has access to and (2) has sufficient space

Usage

```
ribiosTempdir()
```

Value

a character string of the directory name

See Also

`ribiosTempfile`

`ribiosTempfile` *A temporary file which (1) every machine in the cluster has access to and (2) there is sufficient space*

Description

A temporary file which (1) every machine in the cluster has access to and (2) there is sufficient space

Usage

```
ribiosTempfile(pattern = "file", tmpdir = ribiosTempdir(), fileext = "")
```

Arguments

`pattern`	Character string, file pattern
`tmpdir`	Character string, temp directory
`fileext`	CHaracter string, file name extension (suffix)

Value

a character string of the file name

See Also

`ribiosTempdir`

`ribiosUtils` *ribiosUtils*

Description

ribiosUtils is a swiss-knife package providing misc utilities

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>, with inputs from Clemens Broger, Martin Ebeling, Laura Badi and Roland Schmucki

---

`rmat` *Remove temporary files at a specified time interval from now*

Description

Send a `at` job to remove (probably temporary) files in the future with a specified time interval from now

Usage

```
rmat(..., days = NULL, hours = NULL, minutes = NULL, dry = TRUE)
```

Arguments

`...`	Files to be removed
`days`	Numeric, interval in days
`hours`	Numeric, interval in hours
`minutes`	Numeric, interval in minutes
`dry`	Logical, if set to `TRUE`, only the command will be returned and files are not really removed.

Details

The command will delete files, and there is usually no way to get deleted files back. *So make sure you know what you are doing!*

Days, hours, and minutes can be given in a mixed way: they will be summed up to give the interval.

Value

(Invisibly) the output of the `at` job.

Note

Since the command uses `at` internally, it is unlikely the command will work in the Windows system “out of box”.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`qsystem` for running system commands quietly.

Examples

```
tmp1 <- tempfile()
tmp2 <- tempfile()
rmat(tmp1, tmp2, minutes=1)
```

`rocheCore` *Extract core identifiers from Roche compound IDs*

Description

Extract core identifiers from Roche compound IDs

Usage

```
rocheCore(str, short = FALSE)
```

Arguments

`str`	Character strings
`short`	Logical, if `TRUE`, the short version of Roche identifiers (`RO[0-9]{4}`) is returned. Default: `FALSE`

Value

Core identifiers if the element is a Roche compound ID, the original element otherwise Non-character input will be converted to character strings first.

See Also

`isRocheCompoundID`

Examples

```
rocheCore(c("RO1234567-001", "RO1234567-001-000", "RO1234567",
    "ROnoise-001", "anyOther-not-affected"))
rocheCore(c("RO1234567-001", "RO1234567-001-000", "RO1234567",
    "ROnoise-001","anyOther-not-affected"), short=TRUE)
```

`rowscale` *S3 method for row-scaling*

Description

S3 method for row-scaling

Usage

```
rowscale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	Any object
`center`	Logical, whether centering should be done before scaling
`scale`	Logical, whether scaling should be done

Value

The input object with rows scaled

`rowscale.matrix` *Scale a matrix by row*

Description

Scaling a matrix by row can be slightly slower due to a transposing step.

Usage

```
## S3 method for class 'matrix'
rowscale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	An matrix
`center`	Logical, passed to `scale`. to `TRUE`
`scale`	Logical, passed to `scale`. `TRUE`

Value

A matrix with each row scaled.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`scale`

Examples

```
mat <- matrix(rnorm(20), nrow=4)
rs.mat <- rowscale(mat)

print(mat)
print(rs.mat)
rowMeans(rs.mat)
apply(rs.mat, 1L, sd)

rowscale(mat, center=FALSE, scale=FALSE) ## equal to mat
rowscale(mat, center=TRUE, scale=FALSE)
rowscale(mat, center=FALSE, scale=TRUE)
```

`rowscale.table` *Scale a table by row*

Description

Scaling a table by row can be slightly slower due to a transposing step.

Usage

```
## S3 method for class 'table'
rowscale(x, center = TRUE, scale = TRUE)
```

Arguments

`x`	An matrix
`center`	Logical, passed to `scale`. to `TRUE`
`scale`	Logical, passed to `scale`. `TRUE`

Value

A table with each row scaled.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`scale`

Examples

```
letterDf <- data.frame(from=c("A", "A", "B", "C"), to=c("A", "B", "C", "A"))
tbl <- table(letterDf$from, letterDf$to)
tblRowscale <- rowscale(tbl)

print(tbl)
print(tblRowscale)
rowMeans(tblRowscale)
apply(tblRowscale, 1L, sd)

rowscale(tbl, center=FALSE, scale=FALSE) ## equal to mat
rowscale(tbl, center=TRUE, scale=FALSE)
rowscale(tbl, center=FALSE, scale=TRUE)
```

`rrank` *Reverse rank order*

Description

Reverse rank order

Usage

```
rrank(x, ...)

## Default S3 method:
rrank(x, ...)
```

Arguments

`x`	A numeric, complex, character or logical vector
`...`	Passed to `rank`

Value

A vector of numbers of the same length as the input, giving reverse rank orders.

The function returns the reverse rank order, i.e. in the descending order

See Also

`rank`

Examples

```
testVec <- c(3,6,4,5)
rank(testVec)
rrank(testVec)
```

`rrank.matrix` *Get reverse rank orders in each column*

Description

Get reverse rank orders in each column

Usage

```
## S3 method for class 'matrix'
rrank(x, ...)
```

Arguments

`x`	A matrix
`...`	Passed to `rank`

Value

A matrix of the same dimension and attributes of the input matrix, with reverse rank orders of each column

Examples

```
testMatrix <- matrix(c(3,6,4,5,2,4,8,3,2,5,4,7), ncol=3, byrow=FALSE)
rrank(testMatrix)
```

`rrankInd` *Return a matrix that highlights reverse rank orders of features of interest by column*

Description

Return a matrix that highlights reverse rank orders of features of interest by column

Usage

```
rrankInd(matrix, ind, inValue = 1L, outValue = 0L, ...)
```

Arguments

`matrix`	A matrix
`ind`	An integer vector or a logical vector that gives the index
`inValue`	Value to highlight the reverse ranks indexed by `ind`, see details below
`outValue`	Values assigned to other values not indexed by `ind`
`...`	Passed to `rank`

Value

A matrix of the same dimension and attributes of the input matrix, each column contains a vector of `inValue` and `outValue`. Positions that match the reverse ranks of matrix values indexed by `ind` are assigned the `inValue`, otherwise, the `outValue`.

The function can be used to visualize the reverse ranks of features of interest (rows of the input matrix) in each sample (columns of the input matrix). This is for instance useful for rank plots of features for gene-set enrichment analysis.

Imagine that all features indexed by `ind` are the larger than all other features in each sample, then the returned matrix will have value `1` in the first rows (the number determined by the features indxed by `ind`), and `0` in the rest rows.

See Also

`rank`

Examples

```
testMatrix <- matrix(c(3,6,4,5,2,4,8,3,2,5,4,7), ncol=3, byrow=FALSE)
print(testMatrix)
testInd <- c(2,4)
## verify that the command below returns 1 in positions occupied by
## the reverse ranks of the values indexed by testInd
rrankInd(testMatrix, testInd)
testIndBool <- c(FALSE, TRUE, FALSE, TRUE)
rrankInd(testMatrix, testIndBool)
```

`rsetdiff` *Reverse setdiff*

Description

reverse setdiff, i.e. rsetdiff(x,y) equals setdiff(y,x)

Usage

```
rsetdiff(x, y)
```

Arguments

`x`	a vector
`y`	another vector

Value

Similar to setdiff, but with elements in y but not in x

Author(s)

Jitao David Zhang

Examples

```
testVec1 <- LETTERS[3:6]
testVec2 <- LETTERS[5:7]
rsetdiff(testVec1, testVec2)
```

`scriptInit` *Prepare R for an interactive script*

Description

The function prepares R for an interactive session (e.g. in a script). Currently it defines behaviour in case of errors: a file named “ribios.dump” is written.

Usage

```
scriptInit()
```

Value

Side effect is used.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`options`

Examples

```
## do not run unless the script mode is needed

  scriptInit()
```

`setDebug` *Functions for command-line Rscript debugging*

Description

These functions are used to debug command-line executable Rscripts in R sessions

Usage

```
setDebug()
```

Details

`setDebug` sets the environmental variable `RIBIOS_SCRIPT_DEBUG` as `TRUE`. `unsetDebug` unsets the variable. `isDebugging` checks whether the variable is set or not. `isIntDebugging` tests whether the scripts runs interactively or runs in the debugging mode. The last one can be useful when debugging Rscript in a R session.

A programmer wishing to debug a Rscript can explicitly set (or unset) the `RIBIOS_SCRIPT_DEBUG` variable in order to activate (inactivate) certain trunks of codes. This can be automated via `isDebugging`, or probably more conveniently, by `isIntDebugging`: if the script runs in an interactive mode, or the debugging flag is set, the function returns `TRUE`.

Value

`setDebug` and `unsetDebug` returns an invisible value indicating whether the variable setting (unsetting) was successful.

`isDebugging` and `isIntDebugging` returns logical values.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
unsetDebug()
print(isDebugging())
setDebug()
print(isDebugging())
unsetDebug()
print(isDebugging())
print(isIntDebugging())
```

`shortenRocheCompoundID`

Shorten Roche compounds identifiers

Description

Shorten Roche compounds identifiers

Usage

```
shortenRocheCompoundID(str)
```

Arguments

`str`	Character strings that contains one or more Roche core identifiers (`RO` followed by seven-digit numbers)

Value

Character strings of the same length as the input, with all core identifiers shortened

In contrast to `rocheCore`, which only handles character strings that are valid Roche compound identifiers, this function takes any input string and performs a `gsub` operation to shorten Roche core numbers. Therefore, it even works when only a substring matches the pattern of a Roche compound name.

Examples

```
shortenRocheCompoundID(c("RO1234567-001", "RO1234567-001-000",
"RO1234567", "ROnoise-001", "anyOther-not-affected",
"RO1234567 and RO9876543 are two imaginary compounds."))
```

`shortenStr` *Shorten strings to a given number of characters*

Description

Shorten strings to a given number of characters

Usage

```
shortenStr(str, nchar = 8)
```

Arguments

`str`	A vector of strings
`nchar`	The maximal number of characters to keep

Value

A vector of strings of the same length as the input, with each string shortened to the desired length Strings with more characters than `nchar` will be shortened.

Note

`NA` will be kept as they are

Examples

```
inputStrs <- c("abc", "abcd", "abcde", NA)
shortenStr(inputStrs, nchar=4)
## expected outcome: abc, abcd, abcd..., NA
```

`silencio` *Keep silent by suppressing warnings and messages*

Description

The function is used to keep the command silent by suppressing warnings and messages

Usage

```
silencio(...)
```

Arguments

`...`	Any function call

Value

The same as the function call

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`suppressWarnings`, `suppressMessages`

Examples

```
wsqrt <- function(x) {warning("Beep");message("Calculating square");return(x^2)}
silencio(wsqrt(3))
```

`sortAndFilterByCumsumprop`

Sort a numeric vector and filter by a threshold of cumsumprop

Description

Sort a numeric vector and filter by a threshold of cumsumprop

Usage

```
sortAndFilterByCumsumprop(x, thr = 0.9)
```

Arguments

`x`	Numeric vector, usually named
`thr`	Threshold, default 0.9, meaning that items whose proportion of cumulative sum just above 0.9 are kept.

Value

Another numeric vector, likely shorter than x, items whose `cumsumprop` is equal or lower than `thr`. The rest items are summed into one new item, with the name `rest`

This function can be useful to extract from a long numeric vector the largest items that dominate the sum of the vector

Examples

```
x <- c("A"=1,"B"=2,"C"=3,"D"=4,"E"=400,"F"=500)
sortAndFilterByCumsumprop(x, thr=0.99) ## F and E should be returned
```

`sortByCol` *Sort data.frame rows by values in specified columns*

Description

Sort rows of an `data.frame` by values in specified columns.

Usage

```
sortByCol(
  data.frame,
  columns,
  na.last = TRUE,
  decreasing = TRUE,
  orderAsAttr = FALSE
)
```

Arguments

`data.frame`	A `data.frame` object
`columns`	Column name(s) which sould be ordered
`na.last`	Logical, whether NA should be sorted as last
`decreasing`	Logical, whether the sorting should be in the decreasing order
`orderAsAttr`	Logical, whether the order index vectors should be returned in the attribute "order" of the sorted `data.frame`

Details

Columns can be specified by integer indices, logical vectors or character names.

Value

Sorted `data.frame`

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
sample.df <- data.frame(teams=c("HSV", "BVB", "FCB", "FCN"),pts=c(18,17,17,9), number=c(7,7,6,6))
sortByCol(sample.df, 1L)
sortByCol(sample.df, 1L, decreasing=FALSE)

sortByCol(sample.df, c(3L, 1L))
sortByCol(sample.df, c(3L, 1L), decreasing=FALSE)
sortByCol(sample.df, c(3L, 2L))
```

```
sortByCol(sample.df, c(TRUE, FALSE, TRUE))

sortByCol(sample.df, c("teams", "pts"))
sortByCol(sample.df, c("pts", "number", "teams"))
sortByCol(sample.df, c("pts", "teams", "number"))
```

`sortByDimnames` *Sort matrix by dim names*

Description

Rearrange rows and columns of a matrix by dim names

Usage

```
sortByDimnames(x, row.decreasing = FALSE, col.decreasing = FALSE)
```

Arguments

`x`	A matrix or data.frame
`row.decreasing`	Logical, whether rows should be sorted decreasingly
`col.decreasing`	Logical, whether columns should be sorted decreasingly

Value

Resorted matrix or data frame

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
testMat <- matrix(1:16, nrow=4, dimnames=list(c("B", "D", "A", "C"), c("t", "f", "a", "g")))
sortByDimnames(testMat)
sortByDimnames(testMat, row.decreasing=TRUE, col.decreasing=FALSE)
```

`strtoken` *Tokenize strings by character*

Description

Tokenize strings by character in a similar way as the `strsplit` function in the `base` package. The function can return a matrix of tokenized items when `index` is missing. If `index` is given, tokenized items in the selected position(s) are returned. See examples.

Usage

```
strtoken(x, split, index, ...)
```

Arguments

`x`	A vector of character strings; non-character vectors are cast into characters.
`split`	A character to split the strings.
`index`	Numeric vector indicating which fields should be returned; if missing or set to `NULL`, a matrix containing all fields are returned.
`...`	Other parameters passed to `strsplit`

Value

A matrix if `index` is missing, `NULL`, or contains more than one integer indices; otherwise a character vector.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

References

The main body of the function is modified from the `strsplit2` function in the `limma` package.

See Also

`strsplit`

Examples

```
myStr <- c("HSV\t1887", "FCB\t1900", "FCK\t1948")
strsplit(myStr, "\t")

strtoken(myStr, "\t")
strtoken(myStr, "\t", index=1L)
strtoken(myStr, "\t", index=2L)

myFac <- factor(myStr)
```

```
strtoken(myFac, "\t")
strtoken(myFac, "\t", index=1L)
```

---

stubborngc *Repeat garbage-collecting until all resource is freed*

Description

stubborngc repeats collecting garbage untill no more resource can be freed

Usage

```
stubborngc(verbose = FALSE, reset = TRUE)
```

Arguments

`verbose`	Logical, verbose or not
`reset`	Logical, reset or not.

Value

Side effect is used.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`gc`

Examples

```
stubborngc()
```

`subsetByColumnName` *Subset a data.frame by column name, allowing differences in cases*

Description

The function calls `assertColumnName` internally to match the column names.

Usage

```
subsetByColumnName(data.frame, reqCols, ignore.case = FALSE)
```

Arguments

`data.frame`	A data.frame object
`reqCols`	required columns
`ignore.case`	logical, whether the case is considered

Value

If all required column names are present, the data.frame object will be subset to include only these columns and the result data.frame is returned. Otherwise an error message is printed.

Examples

```
myTestDf <- data.frame(HBV=1:3, VFB=0:2, BVB=4:6, FCB=2:4)
myFavTeams <- c("HBV", "BVB")
subsetByColumnName(myTestDf, myFavTeams)
myFavTeamsCase <- c("hbv", "bVb")
subsetByColumnName(myTestDf, myFavTeamsCase, ignore.case=TRUE)
```

`summarizeRows` *Summarizing rows/columns by a factor*

Description

Apply a function to summarize rows/columns that assigned to the same level by a factor vector.

Usage

```
summarizeRows(matrix, factor, fun = mean, ...)
```

Arguments

`matrix`	A numeric matrix
`factor`	A vector of factors, either of the length of `nrow(matrix)` (for `summarizeRows`), or the length of `ncol(matrix)` (for `summarizeColumns`).
`fun`	A function or a name for a function, the summarizing function applied to rows/columns sharing the same level
`...`	Further parameters passed to the function

Details

`NA` levels are neglected, and corresponding rows/columns will not contribute to the summarized matrix.

`summarizeCols` is synonymous to `summarizeColumns`

Value

A matrix, the dimension will be determined by the number of levels of the factor vector.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
my.matrix <- matrix(1:25, nrow=5)
print(my.matrix)

my.factor <- factor(c("A", "B", "A", "C", "B"))
summarizeRows(matrix=my.matrix, factor=my.factor, fun=mean)
summarizeRows(matrix=my.matrix, factor=my.factor, fun=prod)
summarizeColumns(matrix=my.matrix, factor=my.factor, fun=mean)
summarizeColumns(matrix=my.matrix, factor=my.factor, fun=prod)

## NA values in factor
my.na.factor <- factor(c("A", "B", "A", "C", NA))
summarizeRows(matrix=my.matrix, factor=my.na.factor, fun=mean)
summarizeRows(matrix=my.matrix, factor=my.na.factor, fun=prod)
summarizeColumns(matrix=my.matrix, factor=my.na.factor, fun=mean)
summarizeColumns(matrix=my.matrix, factor=my.na.factor, fun=prod)
```

`trim` *Trim leading and tailing spaces from string*

Description

The function trims leading and/or tailing spaces from string(s), using C function implemented in the BIOS library.

Usage

```
trim(x, left = " \n\r\t", right = " \n\r\t")
```

Arguments

`x`	A character string, or a vector of strings
`left`	Characters that are trimmed from the left side.
`right`	Characters that are trimmed from the right side

Details

`left` and `right` can be set to NULL. In such cases no trimming will be performed.

Value

Trimmed string(s)

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
myStrings <- c("This is a fine day\n",
               " Hallo Professor!",
               "  NUR DER HSV  ")
trim(myStrings)
```

`uniqueLength` *Length of unique elements in a vector*

Description

Length of unique elements in a vector

Usage

```
uniqueLength(x, incomparables = FALSE)
```

Arguments

`x`	A vector
`incomparables`	See `unique`

Value

An integer indicating the number of unique elements in the input vector

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`unique`

Examples

```
test.vec1 <- c("HSV", "FCB", "BVB", "HSV", "BVB")
uniqueLength(test.vec1)

test.vec2 <- c(1L, 2L, 3L, 5L, 3L, 4L, 2L, 1L, 5L)
ulen(test.vec2)
```

`uniqueNonNA` *Make a vector free of NA and unique*

Description

Make a vector free of NA and unique

Usage

```
uniqueNonNA(x)
```

Arguments

`x`	A vector

Value

A unique vector without NA

Examples

```
testVec <- c(3,4,5,NA,3,5)
uniqueNonNA(testVec)
```

`verbose` *Print messages conditional on the verbose level*

Description

The verbose level can be represented by non-negative integers. The larger the number is, the more verbose is the program: it prints then more messages for users' information.

Usage

```
verbose(..., global = 1L, this = 1L)
```

Arguments

`...`	Messages to be printed, will be passed to the `message` function
`global`	Integer, the global verbose level
`this`	Integer, the verbose level of this message

Details

This function decides whether or not to print a message, dependent on the global verbose level and the specific level of the message. If the specific level is larger than the global level, the message is suppresed; otherwise it is printed. see the details section for an example.

Suppose the global verbose level is set to `5`, and two messages have levels of `1` and `7` repsectively. Since `1` suggests a low-threshold of being verbose, the first message is printed; whereas the message of level `7` is only printed when the program should run in a more verbose way (`7,8,9,...{}`), it is suppressed in the current global verbose level.

Value

The function is used for its side effect by printing messages.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

Examples

```
Gv <- 5L
verbose("Slightly verbosing", global=Gv, this=1L)
verbose("Moderately verbosing", global=Gv, this=5L)
verbose("Heavily verbosing", global=Gv, this=9L)
```

`wellIndex2position` *Translate well index numbers to well positions*

Description

Translate well index numbers to well positions

Usage

```
wellIndex2position(ind, format = c("96", "384"))
```

Arguments

`ind`	Well index, integer numbers starting from 1, running rowwise. Non-integer parameters will be coereced to integers.
`format`	Character string, well format

Value

A `data.frame` containing three columns: input `WellIndex`, `Row` (characters) and `Column` (integers)

Examples

```
wellIndex2position(1:96, format="96")
wellIndex2position(c(3,2,5,34,85, NA), format="96")
wellIndex2position(1:384, format="384")
```

`whoami` *System user name*

Description

System user name

Usage

```
whoami()
```

Value

System user name

Examples

```
whoami()
```

`writeLog` *Write text as log to a connection*

Description

The function `writeLog` can be used to log outputs and/or running status of scripts to *one connection*. To use it one does *not* need to run `registerLog` first.

Usage

```
writeLog(fmt, ..., con = stdout(), level = 0)
```

Arguments

`fmt`	Format string to passed on to sprintf
`...`	Parameters passed on to sprintf
`con`	A connection, for instance a file (or its name) or `stdout()`
`level`	Logging level: each higher level will add one extra space before the message. See examples

Details

In contrast, `doLog` can be used to log on multiple connections that are registered by `registerLog`. Therefore, to register logger(s) with `registerLog` is a prerequisite of calling `doLog`. Internally `doLog` calls `writeLog` sequentially to make multiple-connection logging.

Value

Side effect is used.

Author(s)

Jitao David Zhang <jitao_david.zhang@roche.com>

See Also

`registerLog` to register more than one loggers so that `doLog` can write to them sequentially.

Examples

```
writeLog("This is the start of a log")
writeLog("Message 1", level=1)
writeLog("Message 1.1", level=2)
writeLog("Message 1.2", level=2)
writeLog("Message 2", level=1)
writeLog("Message 3", level=1)
writeLog("Message 3 (special)", level=4)
writeLog("End of the log");

## log with format
writeLog("This is Message %d", 1)
writeLog("Square of 2 is %2.2f", sqrt(2))

## NA is handled automatically
writeLog("This is a not available value: %s", NA, level=1)
writeLog("This is a NULL value: %s", NULL, level=1)
```

Index