Package ‘kwb.utils’

June 6, 2016

Version 0.2.1

Title some basic functions used by other kwb packages

Maintainer Hauke Sonnenberg <hauke.sonnenberg@kompetenz-wasser.de>

Author Hauke Sonnenberg

Imports PKI

Description some basic functions used by other kwb packages.

License GPL

R topics documented:

`kwb.utils-package` *some basic functions used by other kwb packages*

Description

some basic functions used by other kwb packages.

Details

Package:	kwb.utils
Version:	0.2.1
Title:	some basic functions used by other kwb packages
Maintainer:	Hauke Sonnenberg <hauke.sonnenberg@kompetenz-wasser.de>
Author:	Hauke Sonnenberg
Imports:	PKI
License:	GPL

Author(s)

Hauke Sonnenberg

`addRowWithName` *addRowWithName*

Description

add row to data frame and give a row name at the same time

Usage

```
addRowWithName(x, y, row.name)
```

Arguments

`x`	data frame to which row is to be appended
`y`	data frame containing the row to be appended (exacly one row expected)
`row.name`	name of row to be given in result data frame

Value

x with row of *y* (named *row.name*) appended to it

Author(s)

Hauke Sonnenberg

`allAreEqual` *allAreEqual*

Description

allAreEqual

Usage

```
allAreEqual(elements)
```

Arguments

`elements`

Author(s)

Hauke Sonnenberg

`allTheSame` *allTheSame*

Description

are all elements in x the same?

Usage

```
allTheSame(x)
```

Arguments

`x`	vector of elements to be compared

Author(s)

Hauke Sonnenberg

appendSuffix — *append suffix to (selected) character values*

Description

append suffix to (selected) character values

Usage

```
appendSuffix(values, suffix, valuesToOmit = NULL)
```

Arguments

`values`	vector of character values to which *suffix* is to be appended
`suffix`	(character) suffix to be pasted to *values* that are not in *valuesToOmit*
`valuesToOmit`	vector of values in *values* to which no suffix is to be appended

Value

values with *suffix* appended to those values that are not in *valuesToOmit*

Author(s)

Hauke Sonnenberg

Examples

```
values <- c("a", "b", "c")

# Append ".1" to all values
appendSuffix(values, ".1")

# Append ".1" to all values but "c"
appendSuffix(values, ".1", valuesToOmit = "c")
```

arglist — *merge argument lists or arguments*

Description

creates a list of arguments from given argument lists and arguments. This function allows to create argument lists for function calls. You may start with some basic argument list and then merge other argument lists or single argument assignments into this list. Merging means that elements of the same name are overriden and elements with new names are appended.

Usage

```
arglist(...)
```

Arguments

`...`	list of arguments to this function. All unnamed arguments are assumed to be argument lists which are merged using `merge.lists` first. All named arguments are then merged into this list.

Value

merged list of arguments

Author(s)

Hauke Sonnenberg

See Also

`callWith`

Examples

```
# define some default arguments
args.default <- list(xlim = c(0, 10), ylim = c(0, 10), col = "red", lwd = 2)

# call plot with the default arguments
do.call(plot, arglist(args.default, x = 1:10))

# call plot with the default arguments but override the colour
do.call(plot, arglist(args.default, x = 1:10, col = "blue"))
```

`assignGlobally` *assignGlobally*

Description

assign variable in .GlobalEnv

Usage

```
assignGlobally(x, value)
```

Arguments

`x`	name of variable
`value`	value of variable

Author(s)

Hauke Sonnenberg

`atLeastOneRowIn` *at least one row in data frame*

Description

returns TRUE if data frame has at least one row, else FALSE

Usage

```
atLeastOneRowIn(dframe)
```

Arguments

`dframe`	data frame

Author(s)

Hauke Sonnenberg

`breakInSequence` *breakInSequence*

Description

breakInSequence

Usage

```
breakInSequence(x, expectedDiff = 1)
```

Arguments

`x`	vector of numeric
`expectedDiff`	expected difference between elements in x. A bigger difference is recognised as a break. Default: 1

Value

index of elements after which a break occurs or integer(0) if no break occurs at all

Author(s)

Hauke Sonnenberg

`callWith` *call a function with given arguments*

Description

call a function with the given arguments. Unnamed arguments are expected to be lists containing further argument assignments. Multiple argument lists are merged using `arglist` in the order of their appearence.

Usage

```
callWith(FUN, ...)
```

Arguments

`FUN`

`...`

Value

the return value is the return value of the function `FUN`.

Author(s)

Hauke Sonnenberg

See Also

`arglist`

Examples

```
# define some default arguments
args.default <- list(xlim = c(0, 10), ylim = c(0, 10), col = "red", lwd = 2)

# call plot with the default arguments
callWith(plot, x = 1:10, args.default)

# call plot with the default arguments but override the colour
callWith(plot, x = 1:10, args.default, col = "blue")
```

`catIf` *call cat if condition is met*

Description

call cat if condition is met

Usage

```
catIf(condition, ...)
```

Arguments

`condition`	if TRUE, cat is called, else not
`...`	arguments passed to cat

Author(s)

Hauke Sonnenberg

`checkForMissingColumns` *Check for column existence*

Description

Stops if data frame *frm* does not contain all columns of which the names are given in *reqCols*.

Usage

```
checkForMissingColumns(frm, reqCols, do.stop = TRUE)
```

Arguments

`frm`	data frame
`reqCols`	vector of names of which existence in *frm* shall be checked
`do.stop`	if TRUE, stop() is called else warning() if a column is missing

Value

TRUE if all required columns are available, else FALSE

Author(s)

Hauke Sonnenberg

`clearConsole` *Clear the R Console*

Description

Clear the R Console

Usage

```
clearConsole()
```

Author(s)

Hauke Sonnenberg

`cmdLinePath` *cmdLinePath*

Description

cmdLinePath

Usage

```
cmdLinePath(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`colMaxima` *colMaxima*

Description

maximum per column

Usage

```
colMaxima(dataFrame, na.rm = FALSE)
```

Arguments

`dataFrame`

`na.rm`

Author(s)

Hauke Sonnenberg

`colMinima` *colMinima*

Description

minimum per column

Usage

```
colMinima(dataFrame, na.rm = FALSE)
```

Arguments

`dataFrame`

`na.rm`

Author(s)

Hauke Sonnenberg

`colNaNumbers` *colNaNumbers*

Description

number of NA values per column

Usage

```
colNaNumbers(dataFrame)
```

Arguments

`dataFrame`

Author(s)

Hauke Sonnenberg

`colStatisticOneFunction` *colStatisticOneFunction*

Description

applies a statistical function to all columns of a data frame

Usage

```
colStatisticOneFunction(dataFrame, FUN, na.rm = FALSE)
```

Arguments

`dataFrame`	
`FUN`	statistical function to be applied on each column of dataFrame possible values: "sum", "mean", "min", "max", "number.na" (number of NA values), "length" (number of values)
`na.rm`	if TRUE, NA values are removed before applying the statistical function

Author(s)

Hauke Sonnenberg

`colStatistics` *colStatistics*

Description

applies statistical functions to all columns of a data frame

Usage

```
colStatistics(dataFrame, functions = c("sum", "mean", "min",
    "max", "number.na", "length"), na.rm = FALSE, functionColumn = FALSE)
```

Arguments

`dataFrame`	data frame with numeric columns only
`functions`	vector of statistical functions to be applied on each column of dataFrame possible values: "sum", "mean", "min", "max", "number.na" (number of NA values), "length" (number of values)
`na.rm`	if TRUE, NA values are removed before applying the statistical function(s)
`functionColumn`	if TRUE, a column containing the function name is contained in the result data frame, otherwise the function names become the row names of the result data frame

Author(s)

Hauke Sonnenberg

`columnDescriptor` *columnDescriptor*

Description

columnDescriptor

Usage

```
columnDescriptor(match = ".*", fixed = FALSE)
```

Arguments

`match`	pattern or fixed text to match in header line
`fixed`	if TRUE, *match* is taken as a fixed string to be looked for in the header line, otherwise it is interpreded as a regular expression

Author(s)

Hauke Sonnenberg

`commaCollapsed` *commaCollapsed*

Description

commaCollapsed

Usage

```
commaCollapsed(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

compareDataFrames *compare two data frames by columns*

Description

compare two data frames by columns

Usage

```
compareDataFrames(x, y)
```

Arguments

x	first data frame
y	second data frame

Value

list of logical

Author(s)

Hauke Sonnenberg

Examples

```
x <- data.frame(a = 1:2, b = 2:3)
y <- x

test1 <- all(unlist(compareDataFrames(x, y)))

z <- compareDataFrames(x, y[, c("b", "a")])
expectedFalse <- c("identical", "identicalExceptAttributes", "sameColumnNames")
test2 <- all(names(which(!unlist(z))) == expectedFalse)

test1 && test2
```

containsNulString *containsNulString*

Description

check for nul string in file

Usage

```
containsNulString(filepath)
```

Arguments

`filepath`

Value

TRUE if first two bytes of file are FF FE, else FALSE

Author(s)

Hauke Sonnenberg

---

createDirAndReturnPath *create directory if it does not exist*

Description

create directory if it does not exist

Usage

```
createDirAndReturnPath(dir.to.create, dbg = TRUE, confirm = FALSE)
```

Arguments

`dir.to.create`

`dbg`

`confirm`

Author(s)

Hauke Sonnenberg

---

createMatrix *matrix with row and column names*

Description

Create a matrix by giving row and column names and with all elements being set to a default value

Usage

```
createMatrix(rowNames, colNames = rowNames, value = 0)
```

Arguments

`rowNames`	character vector of row names to be given to the matrix
`colNames`	character vector of column names to be given to the matrix
`value`	value to be given to each matrix element

Value

matrix with `rowNames` as row names and `colNames` as column names, filled with *value* at each position

Author(s)

Hauke Sonnenberg

Examples

```
## Initialise a matrix with rows A to E and columns x to z of value -1
createMatrix(c("A", "B", "C", "D", "E"), c("x", "y", "z"), -1)

## By default the column names are assumed to be equal to the row names
createMatrix(c("A", "B", "C"))

## Initialise a square matrix with NA
createMatrix(c("A", "B", "C"), value = NA)
```

`createPasswordFile` *create encrypted password file for account*

Description

create encrypted password file for account

Usage

```
createPasswordFile(account, keyFile, passwordFile)
```

Arguments

`account`	name of account the user is asked to enter the password for
`keyFile`	file containing the encryption/decryption key
`passwordFile`	

Author(s)

Hauke Sonnenberg

`csvTextToDataFrame` *csvTextToDataFrame*

Description

csvTextToDataFrame

Usage

```
csvTextToDataFrame(text, ...)
```

Arguments

`text`	character vector representing lines of comma separated values
`...`	arguments passed to read.table

Author(s)

Hauke Sonnenberg

`defaultWindowsProgramFolders` *default windows program folders*

Description

default windows program folders

Usage

```
defaultWindowsProgramFolders()
```

Author(s)

Hauke Sonnenberg

`DIN.A4` *width and height of a DIN A4 paper*

Description

width and height of a DIN A4 paper

Usage

```
DIN.A4()
```

Author(s)

Hauke Sonnenberg

`extendLimits` *extendLimits*

Description

extendLimits

Usage

```
extendLimits(limits, left = 0.05, right = left, absolute = FALSE)
```

Arguments

`limits`	vector of two elements as e.g. used for xlim or ylim
`left`	percentage of limit range (*absolute == FALSE*) or absolute value (*absolute == TRUE*) by which the left limit is extended to the left.
`right`	percentage of limit range (*absolute == FALSE*) or absolute value (*absolute == TRUE*) by which the right limit is extended to the right.
`absolute`	Default: FALSE

Author(s)

Hauke Sonnenberg

`extractRowRanges` *extract row ranges by pattern*

Description

extract row ranges by pattern

Usage

```
extractRowRanges(dataFrame, columnName, pattern, startOffset = 1,
    stopOffset = 1, nameByMatch = FALSE, nameColumnsByMatch = TRUE,
    renumber = TRUE)
```

Arguments

`dataFrame`	data frame
`columnName`	name of column in which to search for *pattern*
`pattern`	pattern to be searched for in `dataFrame[[columnNa,e]]`
`startOffset`	row offset to be added to row number in which the pattern matches
`stopOffset`	row offset to be subtracted from row number in which the pattern matches
`nameByMatch`	logical. if TRUE, the elements in the result list are named by the matching values in `dataFrame[[columnName]]`. Defaults to FALSE.
`nameColumnsByMatch`	
`renumber`	

Value

list of data frames containing the rows of *dataFrame* between rows matching *pattern* in `dataFrame[[columnName]]`.

Author(s)

Hauke Sonnenberg

Examples

```
dataFrame <- as.data.frame(
  matrix(
    c("Date", "Value",
      "1.1.", "1",
      "2.1.", "2",
      "", "",
      "Date", "Value",
      "3.1.", "3",
      "4.1.", "4"),
    ncol = 2,
    byrow = TRUE
  ),
  stringsAsFactors = FALSE
)

y <- extractRowRanges(
  dataFrame, columnName = "V1", pattern = "Date", stopOffset = 2
)

expected <- list(
  data.frame(
    Date = c("1.1.", "2.1."),
    Value = c("1", "2"),
    stringsAsFactors = FALSE
  ),
  data.frame(
    Date = c("3.1.", "4.1."),
    Value = c("3", "4"),
    stringsAsFactors = FALSE
  )
)

identical(y, expected)
```

`finishAndShowPdf` *finishAndShowPdf*

Description

finish and display pdf file

Usage

```
finishAndShowPdf(PDF, ...)
```

Arguments

`PDF`	
`...`	

Author(s)

Hauke Sonnenberg

`finishAndShowPdfIf` *finishAndShowPdfIf*

Description

finish and display pdf file if condition is met

Usage

```
finishAndShowPdfIf(to.pdf, PDF, ...)
```

Arguments

`to.pdf`	
`PDF`	
`...`	

Author(s)

Hauke Sonnenberg

`firstElement` *first element*

Description

Returns the first element using the function head

Usage

```
firstElement(x)
```

Arguments

`x`	object

Value

first element: x[1]

Author(s)

Hauke Sonnenberg

`firstPosixColumn` *data/time column of data frame*

Description

data/time column of data frame

Usage

```
firstPosixColumn(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`frenchToAscii` *French unicode letter to ASCII letter(s)*

Description

French unicode letter to ASCII letter(s)

Usage

```
frenchToAscii()
```

Value

list of ASCII characters (list elements) replacing unicode characters (element names)

Author(s)

Hauke Sonnenberg

`generateKeyFile` *generate a decryption key file*

Description

generate a decryption key file

Usage

```
generateKeyFile(target)
```

Arguments

`target`	full path to the file to which the key shall be written

Author(s)

Hauke Sonnenberg

`getByPositiveOrNegativeIndex` *getByPositiveOrNegativeIndex*

Description

get element from vector, counting from head or tail

Usage

```
getByPositiveOrNegativeIndex(elements, index)
```

Arguments

`elements`	vector of elements
`index`	positive or negative index(es) with absolute value between 1 and length(*elements*)

Value

element(s) out of *elements* corresponding to the index(es) given in *index*

Author(s)

Hauke Sonnenberg

`getEvenNumbers` *getEvenNumbers*

Description

getEvenNumbers

Usage

```
getEvenNumbers(x)
```

Arguments

`x`	

Author(s)

Hauke Sonnenberg

`getFunctionName` *get the name of a function*

Description

get the name of a function

Usage

```
getFunctionName(FUN)
```

Arguments

`FUN`	R object representing a function

Author(s)

Hauke Sonnenberg

`getFunctionValueOrDefault` *take function value or default if NA*

Description

take the function value or a default value if the function value is NA

Usage

```
getFunctionValueOrDefault(values, FUN, default, warningMessage = NA)
```

Arguments

`values`	vector of values given to *FUN*
`FUN`	function to which values are passed and which offers the argument "na.rm"
`default`	default value to be returned if all values are NA
`warningMessage`	Warning message given if the default was taken

Author(s)

Hauke Sonnenberg

`getGlobally` *getGlobally*

Description

gat variable from .GlobalEnv

Usage

```
getGlobally(x, default = NULL, create.if.not.existing = TRUE)
```

Arguments

`x`	name of variable
`default`	default value to which the variable is assigned (if create.if.not.existing = TRUE) in case that it does not yet exist. Default: NULL
`create.if.not.existing`	if TRUE and if the variable does not yet exist, it is created and initialised with the value given in *default*. Default: TRUE

Author(s)

Hauke Sonnenberg

`getKeywordPositions` *localise keywords in data frame*

Description

localise keywords in data frame

Usage

```
getKeywordPositions(dataFrame, keywords, asDataFrame = TRUE)
```

Arguments

`dataFrame`	data frame or matrix in which to search for given keywords
`keywords`	(list of) keywords to be looked for in *data frame*
`asDataFrame`	if TRUE (default), a data frame is returned, otherwise a matrix

Value

data frame (if *asDataFrame* = TRUE) or matrix with one column per keyword that was given in *keywords*. The first row contains the row numbers and the second row contains the column numbers, respectively, of the fields in *dataFrame* in which the corresponding keywords were found.

Author(s)

Hauke Sonnenberg

`getNamesOfObjectsInRDataFiles` *get names of objects in .RData files*

Description

get names of objects in .RData files

Usage

```
getNamesOfObjectsInRDataFiles(files.rdata)
```

Arguments

`files.rdata`	vector of full paths to .RData files

Author(s)

Hauke Sonnenberg

Examples

```
## Not run

## Search for available .RData files below "searchdir"
#searchdir <- "//poseidon/projekte$/SUW_Department/Projects/SEMA/WP/20_Braunschweig"
#files.rdata <- dir(searchdir, pattern = "\\.RData$", recursive = TRUE, full.names = TRUE)

## Get the names of the objects in the .RData files
#objectsInFiles <- getNamesOfObjectsInRDataFiles(files.rdata = files.rdata)

## Which file contains the object "DataQ"?
#dataQ.found <- sapply(objectsInFiles, function(x) {"DataQ" %in% x})

#cat("DataQ was found in the following files:",
#    paste(files.rdata[dataQ.found], collapse = "\n  "))
```

`getOddNumbers` *getOddNumbers*

Description

getOddNumbers

Usage

```
getOddNumbers(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`getPassword` *get encrypted password from file using key*

Description

get encrypted password from file using key

Usage

```
getPassword(passwordFile, keyFile)
```

Arguments

`passwordFile`

`keyFile`

Value

NA if no password is stored

Author(s)

Hauke Sonnenberg

`guessSeparator` *guess column separator from file*

Description

guess column separator from file

Usage

```
guessSeparator(csvFile, n = 10, separators = c(";", ",", "\t"))
```

Arguments

`csvFile`	full path to text file containing 'comma separated values'
`n`	number of first lines in the file to be looked at
`separators`	

Author(s)

Hauke Sonnenberg

`hsAddMissingCols` *Add missing columns to data frame*

Description

Adds missing columns to the given data frame so that the resulting data frame contains all columns given in the vector *colNames*. Added columns are filled with NA values.

Usage

```
hsAddMissingCols(dataFrame, colNames, fill.value = NA)
```

Arguments

`dataFrame`	data frame to which column names are to be appended
`colNames`	vector containing names of columns that shall be contained in *dataFrame*
`fill.value`	value to be inserted into newly created columns. Default: NA

Value

data frame with columns as listed in *colNames*

Author(s)

Hauke Sonnenberg

`hsAddToDict` *hsAddToDict*

Description

add assignements given in ... list to *dictionary*

Usage

```
hsAddToDict(dictionary = NULL, ...)
```

Arguments

`dictionary`	
`...`	

Author(s)

Hauke Sonnenberg

`hsChrToNum` *hsChrToNum*

Description

conversion of text representing a number to a valid number

Usage

```
hsChrToNum(x, country, stopOnError = TRUE)
```

Arguments

`x`	(vector of) text value(s) to be converted to numeric
`country`	"en" if value(s) in *x* is (are) given in English format (decimal point ".", thousands separator ",") or "de" if value is given in German format (decimal point ",", thousands separator ".").
`stopOnError`	if TRUE (default) the program stops if any of the given values could not be converted.

Author(s)

Hauke Sonnenberg

`hsCountInStr` *hsCountInStr*

Description

Count occurrences of *chr* in *str*

Usage

```
hsCountInStr(chr, str)
```

Arguments

`chr`	
`str`	

Value

number of orrurrences of *char* in *str*

Author(s)

Hauke Sonnenberg

`hsDelEmptyCols` *Delete empty columns of data frame*

Description

Returns data frame in which all empty columns (NA in all rows) are removed

Usage

```
hsDelEmptyCols(dataFrame)
```

Arguments

`dataFrame`	data frame of which empty columns (NA in all rows) are to be removed

Value

copy of input data frame but with all empty columns removed

Author(s)

Hauke Sonnenberg

`hsMatrixToListForm` *convert "matrix form" to "list form"*

Description

converts a data frame in "matrix form" to a data frame in "list form"

Usage

```
hsMatrixToListForm(df, keyFields, parFields = setdiff(names(df),
    keyFields), colNamePar = "parName", colNameVal = "parVal",
    stringsAsFactors = FALSE)
```

Arguments

`df`	data frame
`keyFields`	names of key fields (e.g. date/time)
`parFields`	names of fields representing differen parameters. Default: column names that are not in *keyFields*
`colNamePar`	name of column in result data frame that will contain the parameter names
`colNameVal`	name of column in result data frame that will contain the parameter values
`stringsAsFactors`	if TRUE, columns of type character in the result data frame are converted to factors. Parameter is passed to cbind, rbind.

Value

data frame in "list form"

Author(s)

Hauke Sonnenberg

See Also

stats::reshape

`hsMovingMean` *moving mean*

Description

Calculate moving mean of *n* values "around" values

Usage

```
hsMovingMean(x, n, na.rm = FALSE)
```

Arguments

`x`	vector of values of which moving mean is to be calculated
`n`	number of values "around" the values in *x*, including the values in *x*, of which the mean is calculated. Only odd numbers 1, 3, 5, ... allowed. For each x[i] in x the moving mean is calculated by: (x[i-(n-1)/2] + ... + x[i-1] + x[i] + x[i+1] + ... + x[i+(n-1)/2]) / n
`na.rm`	logical. Should missing values (including NaN) be omitted from the calculations?

Value

Vector of moving means with the same number of values as there are in *x*. If na.rm is FALSE, the first *(n-1)/2* values and the last *(n-1)/2* values are NA since there are not enough values at the start and the end of the vector to calculate the mean.

Author(s)

Hauke Sonnenberg

Examples

```
x <- rnorm(30)

plot(x, type = "b", main = "Moving mean over 3, 5, 7 points")

times <- 2:4

for (i in times) {
  lines(hsMovingMean(x, n = 2*i - 1), col = i, type = "b", lwd =  2)
}

legend("topright", fill = times, legend = sprintf("n = %d", 2*times - 1))
```

`hsOpenWindowsExplorer` *open Windows Explorer*

Description

open Windows Explorer

Usage

```
hsOpenWindowsExplorer(startdir = tempdir(), use.shell.exec = TRUE)
```

Arguments

`startdir`	directory to be opened in Windows Explorer
`use.shell.exec`	

Author(s)

Hauke Sonnenberg

hsPrepPdf — *Prepare writing of PDF file*

Description

Opens a PDF device in A4 paper format. After calling this function all plots go into the specified PDF file in *strPdf*. Important: The PDF file needs to be closed explicitely with dev.off() after all desired plots have been made.

Usage

```
hsPrepPdf(strPdf = tempfile(fileext = ".pdf"), boolLandscape = TRUE,
    bordW = 2, bordH = 2, makeCur = TRUE, ...)
```

Arguments

`strPdf`	Full path to PDF file to be created
`boolLandscape`	If TRUE, orientation in PDF file will be landscape, else portrait
`bordW`	(Total) border width in "width" direction in cm
`bordH`	(Total) border width in "height" direction in cm
`makeCur`	if TRUE, the new pdf device will become the current device, otherwise the current device will be restored
`...`	further arguments passed to pdf

Author(s)

Hauke Sonnenberg

See Also

hsShowPdf

Examples

```
# Set path to PDF file and open PDF device
pdfFile <- file.path(tempdir(), "ex_hsPrepPdf.pdf")
hsPrepPdf(pdfFile)

## Plot something
plot(x <- seq(-pi,pi,pi/100), sin(x), type = "l")

## Close PDF device
dev.off()

## Open PDF file in viewer
hsShowPdf(pdfFile)
```

hsQuoteChr *hsQuoteChr*

Description

quotes objects of type character with quoting character

Usage

```
hsQuoteChr(x, qchar = "'", escapeMethod = c("double", "backslash",
    "none"))
```

Arguments

x	vector or list of elements
qchar	quoting character to be used. Default: single quote "’"
escapeMethod	

Author(s)

Hauke Sonnenberg

hsRenameColumns *rename columns in a data frame*

Description

rename columns in a data frame giving tupels of original name and substitute name as named elements in list "renames"

Usage

```
hsRenameColumns(dframe, renames)
```

Arguments

dframe	data.frame,
renames	list with named elements each of which defines a column rename in the form <old-name> = <new-name>

Value

dframe with columns renamed as specified in *renames*

Author(s)

Hauke Sonnenberg

`hsResolve` *hsResolve*

Description

Resolve strings according to substitutions defined in dictionary

Usage

```
hsResolve(x, dict, dbg = FALSE)
```

Arguments

`x`	(vector of) string expression(s) to be resolved using the dictionary *dict*.
`dict`	dictionary: list with named elements where the element name represents the key and the element value represents the value assigned to the key.
`dbg`	

Author(s)

Hauke Sonnenberg

`hsRestoreAttributes` *Restores object attributes*

Description

Restores given attributes that are not object attributes any more

Usage

```
hsRestoreAttributes(x, attribs)
```

Arguments

`x`	object
`attribs`	former attributes of x (as retrieved by attributes(x)) to be restored

Author(s)

Hauke Sonnenberg

hsSafeName *Non-existing desired name*

Description

Returns a name that is not yet contained in a vector *myNames* of existing names.

Usage

```
hsSafeName(myName, myNames)
```

Arguments

myName	desired name.
myNames	vector of existing names.

Value

If *myName* is not contained in *myNames* it is returned. Otherwise *myName* is modified to *myName*_01, *myName*_02, ... until a non-existing name is found that is then returned.

Author(s)

Hauke Sonnenberg

Examples

```
existing <- c("a", "b")
myName <- hsSafeName("c", existing)
myName                              # "c"
myName <- hsSafeName("a", existing)
myName                              # "a_1"
hsSafeName("a", c(existing, myName)) # "a_2"
```

hsShell *wrapper around "shell"*

Description

wrapper around "shell"

Usage

```
hsShell(commandLine, ...)
```

Arguments

commandLine	
...	

Author(s)

Hauke Sonnenberg

---

hsShowPdf *Open PDF file in PDF viewer*

Description

Opens the PDF file of which the full path is given in *Pdf* in a PDF viewer.

Usage

```
hsShowPdf(Pdf, dbg = TRUE)
```

Arguments

`Pdf`	full path to PDF file
`dbg`	

Author(s)

Hauke Sonnenberg

See Also

`hsPrepPdf`

Examples

```
# Set path to PDF file and open PDF device
tmpPdf <- tempfile("ex_hsFinishPdf", fileext = ".pdf")
hsPrepPdf(tmpPdf)

## Plot something
plot(x <- seq(-pi,pi,pi/100), sin(x), type = "l")

## Finish PDF file.
dev.off()

## Open PDF file in viewer.
hsShowPdf(tmpPdf)
```

`hsStringToDate` *hsStringToDate*

Description

Convert date string to string and stop on failure

Usage

```
hsStringToDate(strDate, dateFormat)
```

Arguments

`strDate`	(vector of) string(s) representing date(s)
`dateFormat`	date format specifier describing the format in which dates are represented in the csv file. Use placeholders , `"%d"` (day), `"%m"` (month), `"%y"` (2-digit year), `"%Y"` (4-digit year) to describe the date format. `"%d.%m.%Y"`, `"%d/%m/%y"`, `"%Y-%m-%d"` are examples for valid format specifiers.

Value

(vector of) Date object(s)

Author(s)

Hauke Sonnenberg

`hsStringToDouble` *convert string to double*

Description

convert string to double considering given decimal sign in input string

Usage

```
hsStringToDouble(strDbl, dec = ".")
```

Arguments

`strDbl`

`dec`

Value

double representation of input string *strDbl*

Author(s)

Hauke Sonnenberg

`hsSubstSpecChars` *hsSubstSpecChars*

Description

Substitution of special characters

Usage

```
hsSubstSpecChars(x)
```

Arguments

`x`	string containing special characters to be substituted

Value

input string *x* with special characters being substituted by a meaningful represenation or underscore, multiple underscores replaced by a single underscore and multiple underscores at the end removed.

Author(s)

Hauke Sonnenberg

`hsSystem` *wrapper around "system"*

Description

wrapper around "system"

Usage

```
hsSystem(commandLine, ...)
```

Arguments

`commandLine`

`...`

Author(s)

Hauke Sonnenberg

hsTags *Find <tag>-tags in string*

Description

Return tags of the form <tag> that are contained in the string *x*.

Usage

```
hsTags(x, bt, dbg = FALSE)
```

Arguments

x	
bt	bracket type, must be one of c("<>", "[]")
dbg	

Author(s)

Hauke Sonnenberg

hsTags2 *Find <tag>-tags in string*

Description

Return tags of the form <tag> that are contained in the string *x*.

Usage

```
hsTags2(x, bt, dbg = FALSE)
```

Arguments

x	
bt	bracket type, must be one of c("<>", "[]")
dbg	

Author(s)

Hauke Sonnenberg

`hsTrim` *Remove leading and trailing spaces*

Description

Remove leading, trailing (and, if requested, duplicate) spaces

Usage

```
hsTrim(str, trim.multiple.spaces = TRUE)
```

Arguments

`str`	vector of character containing the strings to be trimmed
`trim.multiple.spaces`	if TRUE (default), multiple consecutive spaces are replaced by one space

Value

input string *str* without leading or trailing spaces and with multiple consecutive spaces being replaced by a single space

Author(s)

Hauke Sonnenberg

`hsValidValue` *hsValidValue*

Description

returns TRUE if text representation of number is in correct format in terms of decimal character and (optionally) thousand's separator character.

Usage

```
hsValidValue(x, lng, dbg = FALSE, accept.na = TRUE)
```

Arguments

`x`

`lng`

`dbg`

`accept.na`

Author(s)

Hauke Sonnenberg

`inRange` *inRange*

Description

check for values within minimum and maximum value

Usage

```
inRange(values, min.value, max.value)
```

Arguments

`values`	vector of values
`min.value`	minimum value (inclusive)
`max.value`	maximum value (inclusive)

Value

vector of boolean

Author(s)

Hauke Sonnenberg

`is.unnamed` *are list elements unnamed?*

Description

returns a vector of logical as long as *x* holding TRUE at indices where the list element at the same indices are named and FALSE at positions where the list element at the same indices are not named.

Usage

```
is.unnamed(x)
```

Arguments

`x`	list

Value

vector of logical

Author(s)

Hauke Sonnenberg

Examples

```
is.unnamed(list(1, b = 2)) # TRUE FALSE
is.unnamed(list(a = 1, 2)) # FALSE TRUE
is.unnamed(list()) # logical(0)
is.unnamed(list(a = 1, 2, c = 3)) # FALSE  TRUE FALSE
```

`isEvenNumber` *check for even numbers*

Description

check for even numbers

Usage

```
isEvenNumber(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`isNaInAllColumns` *isNaInAllColumns*

Description

isNaInAllColumns

Usage

```
isNaInAllColumns(dataFrame)
```

Arguments

`dataFrame`	data frame or matrix

Value

logical vector with as many elements as there are rows in *dataFrame* (TRUE for rows in which all elements are NA, FALSE for rows in which there is at least one non-NA element).

Author(s)

Hauke Sonnenberg

`isNaInAllRows` *isNaInAllRows*

Description

isNaInAllRows

Usage

```
isNaInAllRows(dataFrame)
```

Arguments

`dataFrame`	data frame or matrix

Value

logical vector with as many elements as there are columns in *dataFrame* (TRUE for columns in which all elements are NA, FALSE for columns in which there is at least one non-NA element).

Author(s)

Hauke Sonnenberg

`isNaOrEmpty` *NA or the empty string ""?*

Description

is an object NA or equal to the empty string "" (after trimming)?

Usage

```
isNaOrEmpty(x)
```

Arguments

`x`	object to be tested for NA or being empty (equal to "", after trimming)

Value

(vector of) logical, being TRUE for each element in *x* that is NA or the empty string "" (after trimming)

Author(s)

Hauke Sonnenberg

`isNullOrEmpty` *isNullOrEmpty*

Description

isNullOrEmpty

Usage

```
isNullOrEmpty(x)
```

Arguments

`x`	object to be tested for NULL or being empty (vector or list of length 0 or data frame with no rows)

Value

TRUE if x is NULL or x is a vector of length 0 or x is a data frame with no rows.

Author(s)

Hauke Sonnenberg

`isOddNumber` *check for odd numbers*

Description

check for odd numbers

Usage

```
isOddNumber(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`lastElement` *last element*

Description

Returns the last element using the function tail

Usage

```
lastElement(x)
```

Arguments

`x`	object

Value

last element: x[length(x)]

Author(s)

Hauke Sonnenberg

`makeUnique` *adds ".1", ".2", etc. to duplicate values*

Description

# adds ".1", ".2", etc. to duplicate values

Usage

```
makeUnique(x, warn = TRUE)
```

Arguments

`x`	vector of values
`warn`	

Value

x with duplicate elements being modified to "element.1", "element.2", etc.

Author(s)

Hauke Sonnenberg

`merge.lists` *merge lists overriding elements of the same name*

Description

merge lists overriding elements of the same name

Usage

```
## S3 method for class 'lists'
merge(...)
```

Arguments

`...`	lists

Value

list containing the elements given in `...`

Author(s)

Hauke Sonnenberg

See Also

`arglist`

Examples

```
# merge two lists with different elements
merge.lists(list(a = 1), list(b = 2))

# merge two lists with one element of the same name: override element "b"
merge.lists(list(a = 1, b = 2), list(b = 3, c = 4))
```

`mergeAll` *merge multiple data frames*

Description

merge multiple data frames, given in a list

Usage

```
mergeAll(dataFrames, by, ...)
```

Arguments

`dataFrames`	list of data frames. If the list elements are named, the element names are used as suffixes in the column names, otherwise suffixes ".1", ".2", etc are used
`by`	vector of column names to be merged by, passed on to `merge`
`...`	additional arguments passed to `merge`

Value

data frame being the result of merging all the data frames given in *dataFrames* by consecutively calling `merge`

Author(s)

Hauke Sonnenberg

Examples

```
peter <- data.frame(fruit = c("apple", "pear", "banana"), kg = 1:3)
paul <- data.frame(fruit = c("banana", "apple", "lemon"), kg = c(10, 20, 30))
mary <- data.frame(fruit = c("lemon", "organger", "apple"), kg = c(22, 33, 44))

# By default only categories that are in all data frames are returned
mergeAll(list(peter = peter, paul = paul, mary = mary), by = "fruit")

# Use the arguments supported by merge to change that behaviour
mergeAll(list(peter = peter, paul = paul, mary = mary), by = "fruit", all = TRUE)
```

`multiSubstitute` *multiple substitutions*

Description

apply multiple substitutions on a vector of character. For each element in *replacements* gsub is called with the element name being the pattern and the element value being the replacement.

Usage

```
multiSubstitute(strings, replacements, ...)
```

Arguments

`strings`	vector of character
`replacements`	list of pattern = replacement pairs.
`...`	additional arguments passed to gsub

Author(s)

Hauke Sonnenberg

`mySystemTime` *mySystemTime*

Description

mySystemTime

Usage

```
mySystemTime(FUN, args)
```

Arguments

`FUN`

`args`

Author(s)

Hauke Sonnenberg

`naToLastNonNa` *replace NA values with "last" non-NA value*

Description

replace NA values in a vector with the "last" non-NA values (at the nearest smaller indices in each case) in the vector

Usage

```
naToLastNonNa(x, method = 2)
```

Arguments

`x`

`method`

Author(s)

Hauke Sonnenberg

Examples

```
naToLastNonNa(c(1, 2, NA, NA, 3, NA, NA, 4, NA, NA, 5))
## Result: [1] 1 2 2 2 3 3 3 4 4 4 5

# You will get an error if the first element is NA!

# naToLastNonNa(c(NA, 1, NA, 2))

## Error in naToLastNonNa(c(NA, 1, NA, 2)) :
##   The first element must not be NA
```

`percentage` *percentage*

Description

x/basis, in percent

Usage

```
percentage(x, basis)
```

Arguments

`x`	
`basis`	

Value

100 * x / basis

Author(s)

Hauke Sonnenberg

`percentageOfMaximum` *percentageOfMaximum*

Description

percentageOfMaximum

Usage

```
percentageOfMaximum(x, na.rm = TRUE)
```

Arguments

`x`	vector of numeric values
`na.rm`	passed to `max`

Value

100 * x / max(x)

Author(s)

Hauke Sonnenberg

`posixColumnAtPosition` *posixColumnAtPosition*

Description

posixColumnAtPosition

Usage

```
posixColumnAtPosition(x)
```

Arguments

`x`	data frame containing a date/time column

Author(s)

Hauke Sonnenberg

`preparePdf` *open PDF device with DIN A4 dimensions*

Description

open PDF device with DIN A4 dimensions

Usage

```
preparePdf(pdfFile = tempfile(fileext = ".pdf"), landscape = TRUE,
    borderWidth.cm = 2, borderHeight.cm = 2, width.cm = .defaultWidth(landscape,
        borderWidth.cm, borderHeight.cm), height.cm = .defaultHeight(landscape,
        borderWidth.cm, borderHeight.cm), makeCurrent = TRUE,
    ...)
```

Arguments

`pdfFile`	Full path to PDF file to be created
`landscape`	If TRUE (default), orientation in PDF file will be landscape, else portrait
`borderWidth.cm`	(Total) border width in "width" direction in cm
`borderHeight.cm`	(Total) border width in "height" direction in cm
`width.cm`	page width in cm. Default according to DIN A4
`height.cm`	page height in cm. Default according to DIN A4
`makeCurrent`	if TRUE (default), the opened PDF device will become the current device
`...`	further arguments passed to `pdf`

Value

full path to pdf file

Author(s)

Hauke Sonnenberg

`preparePdfIf` *preparePdfIf*

Description

prepare pdf file if condition is met

Usage

```
preparePdfIf(to.pdf, PDF = "", ...)
```

Arguments

`to.pdf`	
`PDF`	
`...`	

Value

full path to pdf file created if condition is met or "" else

Author(s)

Hauke Sonnenberg

`printIf` *call print if condition is met*

Description

call print if condition is met

Usage

```
printIf(condition, x, caption = "")
```

Arguments

`condition`	if TRUE, print is called, else not
`x`	object to be printed
`caption`	optional. Caption line to be printed with cat before printing *x*

Author(s)

Hauke Sonnenberg

`printLines` *printLines*

Description

printLines

Usage

```
printLines(x)
```

Arguments

`x`

Author(s)

Hauke Sonnenberg

`quotient` *quotient*

Description

calculate the quotient of two numbers

Usage

```
quotient(dividend, divisor, substitute.value = Inf, warn = TRUE)
```

Arguments

`dividend`	number to be devided
`divisor`	number by which dividend is to be devided
`substitute.value`	value to be returned if divisor is 0
`warn`	if TRUE, a warning is given if the divisor is zero

Value

quotient of dividend and divisor: dividend/divisor

Author(s)

Hauke Sonnenberg

rbindAll *rbind all data frames given in a list*

Description

rbind all data frames given in a list

Usage

```
rbindAll(x, nameColumn = "", remove.row.names = TRUE, namesAsFactor = TRUE)
```

Arguments

`x`	list of data frames to be passed to `rbind`
`nameColumn`	optional. If given, an additional column of that name is added to the resulting data frame containing the name (or number if *args* is an unnamed list) of the element in *x* that the corresponding rows belong to
`remove.row.names`	if TRUE (default) row names are reset in the output data frame
`namesAsFactor`	if TRUE (default) and *nameColumn* is given the values in column *nameColumn* are converted to a factor

Author(s)

Hauke Sonnenberg

Examples

```
L <- list(
  A = data.frame(x = 1:2, y = 2:3),
  B = data.frame(x = 1:3, y = 2:4)
)

L.unnamed <- L
names(L.unnamed) <- NULL

y1 <- rbindAll(L)
y2 <- rbindAll(L, nameColumn = "group")
y3 <- rbindAll(L.unnamed, nameColumn = "group", namesAsFactor = FALSE)
y4 <- rbindAll(L.unnamed, nameColumn = "group")

expected1 <- data.frame(
  x = c(L$A$x, L$B$x),
  y = c(L$A$y, L$B$y)
)

expected2 <- cbind(
  expected1,
  group = as.factor(c(rep("A", nrow(L$A)), rep("B", nrow(L$B)))),
  stringsAsFactors = FALSE
)

expected3 <- cbind(
  expected1,
  group = c(rep(1L, nrow(L$A)), rep(2L, nrow(L$B)))
)

expected4 <- expected3
expected4$group <- as.factor(expected4$group)

identical(y1, expected1) &&
  identical(y2, expected2) &&
  identical(y3, expected3) &&
  identical(y4, expected4)
```

`readCsvInputFile` *read CSV file*

Description

read CSV file giving column descriptions

Usage

```
readCsvInputFile(csv, sep, dec, headerRow = 1, headerPattern = "",
   columnDescription = NULL, maxRowToLookForHeader = 10, stopOnMissingColumns = TRUE,
    encoding = "unknown", ...)
```

Arguments

`csv`	full path to CSV file
`sep`	column separator
`dec`	decimal character
`headerRow`	number row in which the header (containing column captions) is found
`headerPattern`	pattern matching the header row. If *headerPattern* is given *headerRow* is not considered
`columnDescription`	list of column descriptors. The list elements are named with the name of the list elements being the names that shall be used in the returned data frame. Each list element is a list with elements *match* (pattern to be looked for in the header fields), ...
`maxRowToLookForHeader`	maximum number of rows to be considered when looking for the header row
`stopOnMissingColumns`	if TRUE (default) the program stops if not all columns defined in *columnDescription* are found
`encoding`	passed to readLines, "Latin-1" or "UTF-8"
`...`	further arguments passed to read.table

Author(s)

Hauke Sonnenberg

`readDictionaryFromFile` *readDictionaryFromFile*

Description

reads a dictionary (a list of "key = value"-pairs) from a text file.

Usage

```
readDictionaryFromFile(dictionaryFile, sorted = TRUE)
```

Arguments

`dictionaryFile`	full path to dictionary file
`sorted`	if TRUE (default) the entries in the dictionary will be sorted by their keys

Author(s)

Hauke Sonnenberg

`recursiveNames` *names of all sublists of a list*

Description

returns the names of all sublists of *x* in the "$"-notation, e.g. list$sublist$subsublist$subsubsublist

Usage

```
recursiveNames(x, basename = "")
```

Arguments

`x`	R list.
`basename`	name to be used as prefix for all names found. Default: ""

Author(s)

Hauke Sonnenberg

`recycle` *"recycle" vector to given length*

Description

recycle vector to given length

Usage

```
recycle(x, n)
```

Arguments

`x`	vector to be "recycled"
`n`	target length

Author(s)

Hauke Sonnenberg

`relativeCumulatedSum` *relativeCumulatedSum*

Description

relative cumulated sum of a vector of values

Usage

```
relativeCumulatedSum(values)
```

Arguments

`values`	vector of numeric values

Author(s)

Hauke Sonnenberg

`removeAttributes` *Returns object without attributes*

Description

Returns object without attributes

Usage

```
removeAttributes(x)
```

Arguments

`x`	object

Value

x, but with its attributes removed

Author(s)

Hauke Sonnenberg

`removeColumns` *remove columns from data frame*

Description

remove columns from a data frame

Usage

```
removeColumns(dframe, columnsToRemove, drop = FALSE)
```

Arguments

`dframe`	data frame,
`columnsToRemove`	vector of column names giving the columns to remove
`drop`	if FALSE, a data frame is returned in any case, otherwise the result may be a vector if only one column remains

Value

dframe with columns given in *columnsToRemove* being removed. User attributes of *dframe* are restored.

Author(s)

Hauke Sonnenberg

`removeSpaces` *remove all spaces in string(s)*

Description

remove all spaces in string(s)

Usage

```
removeSpaces(x)
```

Arguments

`x`	(vector of) character

Value

x with all spaces removed

Author(s)

Hauke Sonnenberg

`resolveAll` *resolve all placeholders in a dictionary*

Description

resolve all placeholders in a dictionary

Usage

```
resolveAll(dictionary)
```

Arguments

`dictionary`	list with named elements where the element name represents the key and the element value represents the value assigned to the key.

Author(s)

Hauke Sonnenberg

Examples

```
dictionary <- list(
  basedir = "C:/myNicefolder",
  projectdir = "<basedir>/projects/<projectName>",
  inputdir = "<projectdir>/input",
  outputdir = "<projectdir>/output"
)

dictionary$projectName <- "project1"
dictionary1 <- resolveAll(dictionary)

dictionary$projectName <- "project2"
dictionary2 <- resolveAll(dictionary)

dictionary1$input
dictionary1$output

dictionary2$inputdir
dictionary2$output
```

`revertListAssignments` *revertListAssignments*

Description

switch list elements with their names

Usage

```
revertListAssignments(x)
```

Arguments

`x`	list of named elements

Value

list with the names of *x* as elements and the elements of *x* as names

Author(s)

Hauke Sonnenberg

Examples

```
abbreviation <- list(de = "Germany", en = "England")

revertListAssignments(abbreviation)

## reverting twice results in the original list
identical(
  abbreviation,
  revertListAssignments(revertListAssignments(abbreviation))
)
```

`roundColumns` *roundColumns*

Description

roundColumns

Usage

```
roundColumns(dframe, columnNames = NULL, digits = NULL)
```

Arguments

`dframe`	data frame containing numeric columns to be rounded
`columnNames`	names of (numeric) columns in *dframe* to be rounded.
`digits`	number of digits to be rounded to (vector of length 1 expected) or list of assignments in the form: *columnName* = *numberOfDigits*. If you give a list here, then there is no need to set the argument *columnNames*

Value

dframe with columns given in *columnNames* being rounded to *digits* digits.

Author(s)

Hauke Sonnenberg

`rStylePath` *R compatible file path*

Description

R compatible file path with backslashes replaced with forward slashes

Usage

```
rStylePath(path)
```

Arguments

`path`

Value

path in which backslashes are replaced with forward slashes

Author(s)

Hauke Sonnenberg

`runBatchfileInDirectory` *runBatchfileInDirectory*

Description

runBatchfileInDirectory

Usage

```
runBatchfileInDirectory(batchfile, directory = dirname(batchfile),
    ...)
```

Arguments

`batchfile`	full path to Windows batch file
`directory`	directory from which batchfile is to be invoked. Default: directory of batch file
`...`	arguments passed to shell.exec

Author(s)

Hauke Sonnenberg

`safeColumnBind` *cbind(x1, x2) or x2 if x1 is NULL*

Description

"Safe" version of cbind. If *x1* is NULL *x2* is returned otherwise `cbind(x1, x2)`

Usage

```
safeColumnBind(x1, x2)
```

Arguments

`x1`

`x2`

Value

result of `cbind(x1, x2)` or *x2* if *x1* is null.

Author(s)

Hauke Sonnenberg

Examples

```
x1 <- NULL

for (i in 1:3) {

  x2 <- data.frame(a = 1:3, b = rnorm(3))
  x1 <- safeColumnBind(x1, x2)

  # using cbind would result in an error:
  # x1 <- cbind(x1, x2)
}

x1
```

`safeRowBind` *"safe" rbind*

Description

rbind two data frames even if column names differ

Usage

```
safeRowBind(dataFrame1, dataFrame2)
```

Arguments

`dataFrame1`

`dataFrame2`

Author(s)

Hauke Sonnenberg

`safeRowBindOfListElements` *row-bind data frames in a list of lists*

Description

row-bind data frames in a list of lists

Usage

```
safeRowBindOfListElements(x, elementName)
```

Arguments

`x`	list of lists each of which contains a data frame in element *elementName*
`elementName`	name of list element in each sublist of *x* which contains a data frame

Value

data frame resulting from "row-binding" data frames.

Author(s)

Hauke Sonnenberg

Examples

```
x <- list(
  list(
    number = 1,
    data = data.frame(x = 1:2, y = 2:3)
  ),
  list(
    number = 2,
    data = data.frame(x = 11:12, y = 12:13)
  )
)

safeRowBindOfListElements(x, "data")

## also working if the column names of the data frames in the "data" elements
## differ.
x[[1]]$data$z = 13:14
safeRowBindOfListElements(x, "data")
```

---

selectElements *select (and rename) elements from list*

Description

select (and rename, if required) elements from list. Stop with message if elements do not exist

Usage

```
selectElements(x, elements, do.stop = TRUE, do.warn = TRUE)
```

Arguments

x	list
elements	vector of element names. The names of named elements will be the names in the output list
do.stop	this flag controls whether the function stops (`do.stop = TRUE`) or not (`do.stop = FALSE`) if there are non-existing elements to be selected. If `do.stop = FALSE` only those elements are selected that actually exist
do.warn	if `TRUE` (default) and `do.stop = FALSE` a warning is given if elements do not exist. Set to `FALSE` to suppress warnings

Value

list containing the elements of `x` that are specified in `elements` or `x[[elements]]` if length of `elements` is 1 or `list()` if `elements` is empty. If the elements in vector `elements` are named, these names are used in the output list.

Author(s)

Hauke Sonnenberg

Examples

```
L <- list(a = 1, b = 2, c = 3, d = 4)

# Select elements
selectElements(L, c("a", "c"))

# Select and rename at the same time
selectElements(L, elements = c(a.new = "a", c.new = "c", "b"))
```

`startsToEnds` *helper function: start indices to end indices*

Description

helper function to convert start indices to end indices

Usage

```
startsToEnds(starts, lastStop, stopOffset = 1)
```

Arguments

`starts`	vector of integer
`lastStop`	number to be returned as last element of the result vector
`stopOffset`	number to be subtracted from (all but the first elements in) *starts* in order to find the ends

Value

vector of integer

Author(s)

Hauke Sonnenberg

Examples

```
starts <- c(1, 10, 20, 35)

ok <- identical(startsToEnds(starts, lastStop = 50),
                c(9, 19, 34, 50))

ok <- ok && identical(startsToEnds(starts, lastStop = 50, stopOffset = 2),
                      c(8, 18, 33, 50))
ok
```

`startsToRanges` *row numbers of start rows to from/to row ranges*

Description

a vector of row numbers is transformed to a data frame describing row ranges by numbers of first and last rows

Usage

```
startsToRanges(starts, lastStop, startOffset = 1, stopOffset = 1)
```

Arguments

`starts`

`lastStop`

`startOffset`

`stopOffset`

Author(s)

Hauke Sonnenberg

Examples

```
starts <- c(1, 10, 20, 35)

ok <- identical(startsToRanges(starts, lastStop = 50),
                data.frame(
                  from = c(2, 11, 21, 36),
                  to = c(9, 19, 34, 50)
                ))

ok <- ok && identical(
  startsToRanges(starts, lastStop = 55, startOffset = 2, stopOffset = 2),
  data.frame(
    from = c(3, 12, 22, 37),
    to = c(8, 18, 33, 55)
  ))

ok
```

`stringContains` *stringContains*

Description

stringContains

Usage

```
stringContains(x, contains)
```

Arguments

`x`	
`contains`	

Author(s)

Hauke Sonnenberg

Examples

```
stringContains(c("abc", "Kabeljau", "Arabella"), "ab")
stringContains(c("abc", "Kabeljau", "Arabella"), "abc")
```

`stringEndsWith` *stringEndsWith*

Description

stringEndsWith

Usage

```
stringEndsWith(x, endsWith)
```

Arguments

`x`	
`endsWith`	string to be searched for at the end of the string(s) in *x*

Author(s)

Hauke Sonnenberg

Examples

```
stringEndsWith(c("abc", "Kabeljau", "Arabella"), "a")
stringEndsWith(c("abc", "Kabeljau", "Arabella"), "jau")
```

`stringStartsWith` *stringStartsWith*

Description

stringStartsWith

Usage

```
stringStartsWith(x, startsWith)
```

Arguments

`x`	
`startsWith`	string to be searched for at the beginning of the string(s) in *x*

Author(s)

Hauke Sonnenberg

Examples

```
stringStartsWith(c("abc", "Kabeljau", "Arabella"), "ab")
stringStartsWith(c("abc", "Kabeljau", "Arabella"), "A")
```

`stringToExpression` *stringToExpression*

Description

stringToExpression

Usage

```
stringToExpression(expressionString)
```

Arguments

`expressionString`	

Author(s)

Hauke Sonnenberg

subExpressionMatches *subExpressionMatches*

Description

subExpressionMatches

Usage

```
subExpressionMatches(regularExpression, text, match.names = NULL,
    select = structure(seq_along(match.names), names = match.names),
    simplify = TRUE)
```

Arguments

regularExpression	regular expression containing parts in parentheses that are to be extracted from *text*
text	text to be matched against the regular expression
match.names	optional. Names that are to be given to the extracted parts in the result list,
select	named vector of numbers specifying the subexpressions in parentheses to be extracted.
simplify	if TRUE (default) and *text* has only one element, the output structure will be a list instead a list of lists

Value

If `length(text) > 1` a list is returned with as many elements as there are strings in *text* each of which is itself a list containing the strings matching the subpatterns (enclosed in parentheses in *regularExpression*) or NULL for strings that did not match. If *match.names* are given, the elements of these lists are named according to the names given in *match.names*. If *text* is of length 1 and *simplify* = TRUE (default) the top level list structure described above is omitted, i.e. the list of substrings matching the subpatterns is returned.

Author(s)

Hauke Sonnenberg

Examples

```
# split date into year, month and day
subExpressionMatches("(\\d{4})\\-(\\d{2})\\-(\\d{2})", "2014-04-23")

# split date into year, month and day (give names to the resulting elements)
x <- subExpressionMatches(
  regularExpression = "(\\d{4})\\-(\\d{2})\\-(\\d{2})", "2014-04-23",
  match.names = c("year", "month", "day")
)

cat(paste("Today is ", x$day, "/", x$month, " of ", x$year, "\n", sep=""))
```

`tempSubdirectory` *create and return path of subdirectory in temp()*

Description

create and return path of subdirectory in temp()

Usage

```
tempSubdirectory(subdir)
```

Arguments

`subdir`	name of subdirectory to be created

Value

full path to created directory

Author(s)

Hauke Sonnenberg

`test_roundColumns` *test roundColumns*

Description

test_roundColumns

Usage

```
test_roundColumns()
```

Author(s)

Hauke Sonnenberg

`toInches` *convert cm to inches*

Description

# convert cm to inches

Usage

```
toInches(cm)
```

Arguments

`cm`	vector of numeric representing length(s) in cm

Value

vector of numeric representing length(s) in inches

Author(s)

Hauke Sonnenberg

`toPositiveIndices` *toPositiveIndices*

Description

toPositiveIndices

Usage

```
toPositiveIndices(indices, n)
```

Arguments

`indices`

`n`

Author(s)

Hauke Sonnenberg

`warnIfEmpty` *warnIfEmpty*

Description

Gives a warning if the object is NULL or empty and returns the object

Usage

```
warnIfEmpty(x)
```

Arguments

`x`	object to be tested for NULL or being empty (vector of length 0 or data frame with no rows)

Author(s)

Hauke Sonnenberg

`windowsPath` *convert to MS Windows-compatible path*

Description

create MS Windows-compatible path by substituting forward slashes with backslashes

Usage

```
windowsPath(path)
```

Arguments

`path`

Author(s)

Hauke Sonnenberg

Index