Package: kwb.utils (via r-universe)

August 25, 2024

Title General Utility Functions Developed at KWB

Version 0.15.0

Description This package contains some small helper functions that aim at improving the quality of code developed at Kompetenzzentrum Wasser gGmbH (KWB).

License MIT + file LICENSE

URL https://github.com/kwb-r/kwb.utils

BugReports https://github.com/kwb-r/kwb.utils/issues

Imports methods

Suggests PKI, testthat, covr, knitr, magrittr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

Repository https://kwb-r.r-universe.dev

RemoteUrl https://github.com/KWB-R/kwb.utils

RemoteRef HEAD

RemoteSha 4415aa211e0d689a99c1a21f2ae99c36f206ff70

Contents

`.log` *Write Log Message to Console*

Description

Write Log Message to Console

Usage

```
.log(...)
```

Arguments

`...`	parts of the message that will be passed to `cat`

`.logline` *Write Log Message and New Line Character to Console*

Description

Write Log Message and New Line Character to Console

Usage

```
.logline(...)
```

Arguments

`...`	parts of the message that will be passed to `cat`

.logok *Write "ok" and New Line Character to Console*

Description

Write "ok" and New Line Character to Console

Usage

```
.logok(dbg = TRUE)
```

Arguments

dbg	logical to switch logging on (dbg = TRUE) or off (dbg = FALSE)

.logstart *Write Log Message to Console if in Debugging Mode*

Description

Write Log Message to Console if in Debugging Mode

Usage

```
.logstart(dbg = TRUE, ...)
```

Arguments

dbg	logical to switch logging on (dbg = TRUE) or off (dbg = FALSE)
...	parts of the message that will be passed to catIf

.OStype *see tools:::.OStype*

Description

This is a copy of tools:::.OStype. It is redefined here so that it can be used within this package. R CMD build would complain if I used tools:::.OStype!

Usage

```
.OStype()
```

addClass *Add a Class to an Object*

Description

Extend the class attribute by a given class name

Usage

```
addClass(x, className, first = TRUE, dbg = FALSE)
```

Arguments

x	R object
className	name of the class to be added to x
first	if TRUE (default) the className is prepended to the vector of existing class names, otherwise appended to the end of the vector
dbg	if TRUE (default), debug messages are shown.

Value

x with the class attributed extended by className (only if x did not yet inherit from className)

addRowWithName *Add a Row with a Name*

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

addSuffixToColumns *Add Suffix to Column Names*

Description

Add Suffix to Column Names

Usage

```
addSuffixToColumns(data, suffix, except = NULL)
```

Arguments

data	data frame
suffix	suffix to be added to each column name except the columns given in except
except	names of columns to which no suffix is to be given

Value

data with renamed columns

Examples

```
d1 <- data.frame(id = 1, a = 2, b = 3)
d2 <- data.frame(id = 1, c = 2, d = 3)

# Using merge the origin of the column gets lost
merge(d1, d2)

# Add a suffix before merging
merge(
  addSuffixToColumns(d1, ".first", except = "id"),
  addSuffixToColumns(d2, ".second", except = "id"),
  by = "id"
)
```

allAreEqual *are all elements in x the same?*

Description

are all elements in x the same?

Usage

```
allAreEqual(x, method = 1)
```

Arguments

x	vector of elements to be compared
method	Select one of two methods. 1: check if the length of unique elements is equal to one, 2: check if all elements are equal to the first element.

Value

TRUE if all elements in x are equal to each other, otherwise FALSE

---

allAreIdentical *Are all list elements identical to each other?*

Description

Are all list elements identical to each other?

Usage

```
allAreIdentical(x)
```

Arguments

x	a list

Value

TRUE if all elements in x are identical, otherwise FALSE

---

almostEqual *Compare Numerical Vectors with Tolerance*

Description

Compare numerical vectors accepting a tolerance

Usage

```
almostEqual(x, y, tolerance = 1e-12)
```

Arguments

x	vector of numerical
y	vector of numerical
tolerance	tolerance, i.e. accepted difference between values in x and y. Default: 1e-12

Value

vector of logical

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*

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(..., warn.on.NULL = FALSE)
```

Arguments

...	list of arguments to this function. All unnamed arguments are assumed to be argument lists which are merged using mergeLists first. All named arguments are then merged into this list.
warn.on.NULL	if TRUE (default is FALSE) a warning is given if any of the arguments given to this function is NULL

Value

merged list of arguments

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"))
```

---

argsCsv *Language Abbreviation to "sep" and "dec" argument*

Description

Language Abbreviation to "sep" and "dec" argument

Usage

```
argsCsv(lng = "de")
```

Arguments

lng	"de" for German, "en" for Englisch

Value

list with elements sep (column separator), dec (character used for decimal points)

arrayToDataFrame *Convert Array to Data Frame*

Description

Convert Array to Data Frame

Usage

```
arrayToDataFrame(x, name = deparse(substitute(x)))
```

Arguments

x	an array
name	name to be given to the value column

Value

data frame with as many rows as there are elements in the input array x. The data frame has one column per dimension of x containing the dimension names and one additional column called according to name containing the array values.

Examples

```
x <- array(1:24, dim = c(2, 3, 4), dimnames = list(
  c("a", "b"),
  c("x", "y", "z"),
  c("r", "s", "t", "u")
))

arrayToDataFrame(x)
```

asColumnList *Matrix to List of Matrix Columns*

Description

Matrix to List of Matrix Columns

Usage

```
asColumnList(x)
```

Arguments

x	matrix

Value

list with as many elements as there are columns in x and each element representing one column

Examples

```
x <- matrix(1:12, nrow = 3)

column_list <- asColumnList(x)

for (i in 1:ncol(x)) print(identical(column_list[[i]], x[, i]))
```

---

asNoFactorDataFrame *Convert to Data Frame without Factors*

Description

Use as.data.frame with stringsAsFactors = FALSE

Usage

```
asNoFactorDataFrame(...)
```

Arguments

...	passed to as.data.frame

Examples

```
data_matrix <- matrix(LETTERS[1:6], nrow = 2)

# as.data.frame() by default converts character to factor
str(as.data.frame(data_matrix))

# asNoFactorDataFrame keeps character as character
str(asNoFactorDataFrame(data_matrix))
```

asRowList *Matrix to List of Matrix Rows*

Description

Matrix to List of Matrix Rows

Usage

```
asRowList(x)
```

Arguments

x	matrix

Value

list with as many elements as there are rows in x and each element representing one row

Examples

```
x <- matrix(1:12, nrow = 3)

row_list <- asRowList(x)

for (i in 1:nrow(x)) print(identical(row_list[[i]], x[i, ]))
```

assertFinalSlash *Make Sure that Strings End With Slash*

Description

Make Sure that Strings End With Slash

Usage

```
assertFinalSlash(x, method = 1L)
```

Arguments

x	vector of character
method	integer value specifying the implementation method. 1 (default): Find strings without ending slash and append slash to these strings. 2: Remove one or more slashes at the end and append slash to all strings. 3: Append slash to all strings and replace multiple occurrences of slash at the end with one slash. Method 1 is the fastest but does not replace multiple trailing slashes with only one trailing slash (see examples).

Examples

```
assertFinalSlash(c("a", "b", "c"))
assertFinalSlash(c("a/", "b/", "c/"))
assertFinalSlash(c("a//", "b", "c/"))

# Use method 2 or 3 to replace multiple slashes with one slash
assertFinalSlash(c("a//", "b", "c/"), method = 2)
assertFinalSlash(c("a//", "b", "c/"), method = 3)
```

---

assertRowsAndColumns *Assert Row and Column Names of a Matrix*

Description

Make sure that a matrix contains rows and columns of the given names in the given order.

Usage

```
assertRowsAndColumns(x, row_names = NULL, col_names = NULL, fill_value = 0)
```

Arguments

x	A matrix
row_names	character vector of row names
col_names	character vector of column names
fill_value	value to fill a row or column with if a row or column does not exist in x

Examples

```
m <- matrix(1:12, nrow = 3, ncol = 4, dimnames = list(
  rows = paste0("row", 1:3), cols = paste0("col", 1:4)
))

# Add two rows, reverse order of rows, add one column, remove one column
assertRowsAndColumns(
  m,
  row_names = paste0("row", 4:0),
  col_names = paste0("col", 0:2)
)
```

assignAll *Call assign for each List Element*

Description

Provide all assignments of a list as objects in the environment

Usage

```
assignAll(x, pos = 1, ...)
```

Arguments

x	list of key = value assignments
pos	passed to assign,
...	further arguments passed to assign

assignArgumentDefaults *Assign Argument Default Values in the Global Environment*

Description

Assign Argument Default Values in the Global Environment

Usage

```
assignArgumentDefaults(FUN)
```

Arguments

FUN	a function (a closure or a primitive). If FUN is a character string then the function with that name is found and used.

Examples

```
# Assign the default values of breakInSequence() in the global environment
assignArgumentDefaults(kwb.utils::breakInSequence)

# The argument "expectedDiff" is now in the global environment
ls()

# Its value is 1 which is the default value defined in the function definition
expectedDiff
```

assignGlobally *assignGlobally*

Description

assign variable in .GlobalEnv

Usage

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

Arguments

x	name of variable
value	value of variable

assignObjects *Assign Package Functions to the Global Environment*

Description

This function provides all (also non-exported) function definitions of this package in the Global environment. This is useful for debugging the code of a function that calls non-exported functions.

Usage

```
assignObjects()
```

Examples

```
## Not run:
# Provide all functions of kwb.utils in the global environment
assignObjects()

## End(Not run)
```

assignPackageObjects *Assign all Package Objects to the Global Environment*

Description

Assign all Package Objects to the Global Environment

Usage

```
assignPackageObjects(package)
```

Arguments

package	package name

Examples

```
assignPackageObjects("kwb.utils")
```

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

backspace *String of n Backspaces*

Description

String of n Backspaces

Usage

```
backspace(n)
```

Arguments

n	number of backspace characters

Value

vector of character of length one

Examples

```
update <- function(x) cat(backspace(3), x)
x <- "value: 123"
cat(x)
cat(paste0(x, backspace(3), "987"))
```

breakInSequence *Find "Breaks" in a Sequence of Numbers*

Description

Find "Breaks" in a Sequence of Numbers

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

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	function to be called
...	(unnamed) lists containing argument assignments passed to FUN or (named) arguments passed to FUN

Value

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

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")
```

callWithData *Call a Function with Argument Combinations from a Data Frame*

Description

Call a Function with Argument Combinations from a Data Frame

Usage

```
callWithData(
  FUN,
  data,
  ...,
  threshold = 0.5,
  SIMPLIFY = TRUE,
  USE.NAMES = TRUE
)
```

Arguments

FUN	function to be called
data	data frame with one column per argument of FUN
...	further (constant) arguments to FUN that are passed to mapply via MoreArgs
threshold	if the ratio of unique value combinations in the relevant columns in data to all value combinations in these columns is below this threshold value then FUN will be called only with the unique value combinations. This should increase performance.
SIMPLIFY	passed tomapply, default: TRUE
USE.NAMES	passed tomapply, default: TRUE

Value

vector of length nrow(data) with the result values returned by FUN

Examples

```
combis <- expand.grid(x = 1:2, y = c(10, 20, 30))
combis

callWithData(`+`, combis)
```

`callWithStringsAsFactors` *Call a Function with Option "stringsAsFactors" set temporarily*

Description

Set the option "stringsAsFactors", run a function and reset the option.

Usage

```
callWithStringsAsFactors(stringsAsFactors, FUN, ...)
```

Arguments

`stringsAsFactors`	TRUE or FALSE. Before calling `FUN` the option "stringsAsFactors" is set to the value given here. After the function call the option is reset to what it was before.
`FUN`	function to be called
`...`	arguments passed to `FUN`

Value

This function returns what `FUN` returns when called with the arguments given in `...`

Examples

```
option.bak <- getOption("stringsAsFactors")

d1 <- callWithStringsAsFactors(
  TRUE,
  rbind,
  data.frame(id = 1, name = "Peter"),
  data.frame(id = 2, name = "Paul"),
  data.frame(id = 3, name = "Mary")
)

d2 <- callWithStringsAsFactors(
  FALSE,
  rbind,
  data.frame(id = 1, name = "Peter"),
  data.frame(id = 2, name = "Paul"),
  data.frame(id = 3, name = "Mary")
)

str(d1)
str(d2)

# The option "stringsAsFactors" has not changed!
stopifnot(option.bak == getOption("stringsAsFactors"))
```

catAndRun *Print Debug Messages Before and After Running Code*

Description

Print Debug Messages Before and After Running Code

Usage

```
catAndRun(
  messageText = "Running code",
  expr,
  newLine = 2L,
  dbg = TRUE,
  log_time = TRUE
)
```

Arguments

messageText	text to be printed before running the code
expr	expressions to be run. Enclose more than one expression in curly braces
newLine	integer controlling new lines. 0: no extra new line, 1: new line after messageText, 2: new line after "ok.", 3: new line after both, messageText and "ok."
dbg	logical. If FALSE, output is suppressed.
log_time	logical. If TRUE, the time elapsed during the evaluation of expr is printed.

Value

This function returns the evaluation of expr. The result is returned invisibly so that the result of an assignment as the last expression in exprt does not show up on the console.

Examples

```
for (newLine in 0:3) {

  catAndRun("work hard", newLine = newLine, {
    cat("hard\nworking\n")
  })

  cat("here.\n\n")
}
```

catChanges *Print Differences Between Two Vectors*

Description

Print Differences Between Two Vectors

Usage

```
catChanges(x, y)
```

Arguments

x	vector
y	vector

Examples

```
catChanges(c(1, 2, 3), c(1, 22, 3))
```

catChangesIf *Print Differences Between Two Vectors if Condition is Met*

Description

Print Differences Between Two Vectors if Condition is Met

Usage

```
catChangesIf(dbg, x, y)
```

Arguments

dbg	if TRUE catChanges is called on x and y
x	vector
y	vector

catchWarning *Return Warning Message in Attribute*

Description

This function evaluates an expression and returns, if warnings occurred, the warning message in the attribute "warningMessage" of the returned object.

Usage

```
catchWarning(expr, dbg = TRUE)
```

Arguments

expr	expression to be evaluated
dbg	if TRUE (the default) the warning text is also printed on the console, otherwise the warning message is suppressed.

Examples

```
catchWarning(as.numeric("1.23"))
result <- catchWarning(as.numeric("x"))
result <- catchWarning(as.numeric("x"), dbg = FALSE)
str(result)
```

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

catLines *Print Character Vector to the Console*

Description

Call cat on character vector, pasted with collapse = <new line>

Usage

```
catLines(x)
```

Arguments

x	vector of character representing text lines to be printed

catNewLineIf *Print New Line Character to the Console if Condition is Met*

Description

Print New Line Character to the Console if Condition is Met

Usage

```
catNewLineIf(condition)
```

Arguments

condition	if TRUE the new line is printed else not

Value

Returns the condition, invisibly so that it can be reused

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,
  dataFrameName = deparse(substitute(frm))
)
```

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
dataFrameName	the name of the data frame to be shown in the error message if a column was missing

Value

TRUE if all required columns are available, else FALSE

clearConsole *Clear the R Console*

Description

Clear the R Console

Usage

```
clearConsole()
```

clipMatrix *Remove Leading/Trailing Empty Rows/Columns from Matrix*

Description

Remove Leading/Trailing Empty Rows/Columns from Matrix

Usage

```
clipMatrix(m)
```

Arguments

m	a matrix

Value

m with leading trailing empty rows (full of NA) removed

cmdLinePath *Path in Quotes for Usage in Command Line*

Description

Set the given path in quotes so that in can be used in a command line

Usage

```
cmdLinePath(x)
```

Arguments

x	path to be quoted

collapsed *Shortcut to paste(x, collapse = collapse)*

Description

This is just a shortcut to paste(x, collapse = collapse)

Usage

```
collapsed(x, collapse = " ")
```

Arguments

x	vector of character
collapse	character string to separate the elements in x (passed to paste)

colMaxima *Columnwise Maxima*

Description

Calculate the maxima within each column

Usage

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

Arguments

dataFrame	data frame of which to calculate columnwise maxima
na.rm	passed to the max function

colMinima *Columnwise Minima*

Description

Calculate the minima within each column

Usage

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

Arguments

dataFrame	data frame of which to calculate columnwise minima
na.rm	passed to the min function

colNaNumbers *Columnwise Number of NA*

Description

Calculate the number of NA values within each column

Usage

```
colNaNumbers(dataFrame)
```

Arguments

dataFrame	data frame of which to calculate columnwise NA values

colStatisticOneFunction *Apply Function to All Columns*

Description

Applies a statistical function to all columns of a data frame

Usage

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

Arguments

dataFrame	a data frame of which statistics are to be calculated
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

colStatistics *Column Statistics*

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

columnDescriptor *Column Descriptor*

Description

Column Descriptor

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

columnToDate *Convert Column in Data Frame To Date*

Description

Convert Column in Data Frame To Date

Usage

```
columnToDate(df, column, dbg = TRUE)
```

Arguments

df	data frame
column	name of column in x
dbg	if TRUE a debug message is shown

Value

df with column converted to class Date with as.Date

Examples

```
df <- data.frame(id = 1:2, date = c("2018-10-23", "2018-10-23"))
str(df)

df <- columnToDate(df, "date")
str(df)
```

columnwisePercentage *Columnwise Percentage*

Description

Calculate the percentage (value divided by sum of values in the column) for each column

Usage

```
columnwisePercentage(x, default = 0, digits = 1)
```

Arguments

x	two dimensional numeric data structure
default	default value to be used if the calculated percentage is NA.
digits	number of digits (default: 1) to which the resulting percentages are to be rounded. Set to NA to suppress rounding

Examples

```
# Create a random matrix of integer values
M1 <- matrix(sample(100, 12), nrow = 4, dimnames = list(LETTERS[1:4], 1:3))

# Introduce some NA
values <- as.numeric(M1)
values[sample(length(values), 3)] <- NA
M2 <- matrix(values, nrow = nrow(M1), dimnames = dimnames(M1))

M1
columnwisePercentage(M1)

M2
columnwisePercentage(M2)
columnwisePercentage(M2, default = 0)
```

`combineAlternatingly` *Combine Two Vectors Alternatingly*

Description

Combine Two Vectors Alternatingly

Usage

```
combineAlternatingly(a, b)
```

Arguments

a	first vector
b	second vector

Value

vector x with all `x[c(1, 3, 5, ...)]` == a and all `x[c(2, 4, 6, ...)]` == b

Examples

```
a <- paste0("a", 1:5)
b <- paste0("b", 1:5)

combineAlternatingly(a, b)
combineAlternatingly(b, a)

combineAlternatingly(list(a = 1, b = 2), list(c = 3, d = 4))
```

commaCollapsed *Paste With Collapse = ","*

Description

Paste With Collapse = ","

Usage

```
commaCollapsed(x)
```

Arguments

x	vector of character

commonNames *Common Names in two Objects with Names Attribute*

Description

Common Names in two Objects with Names Attribute

Usage

```
commonNames(x, y)
```

Arguments

x	object with names attribute, e.g. data.frame, named vector
y	object with names attribute, e.g. data.frame, named vector

Value

vector of names occurring in both x and y

Examples

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

compareDataFrames *Compare two data frames by columns*

Description

Compare two data frames by columns

Usage

```
compareDataFrames(
  x,
  y,
  dbg = FALSE,
  xname = deparse(substitute(x)),
  yname = deparse(substitute(y))
)
```

Arguments

x	first data frame
y	second data frame
dbg	if TRUE (default) the result of comparing the dimensions and the column names is printed on the screen
xname	name of first data frame to appear in the output if dbg = TRUE
yname	name of second data frame to appear in the output if dbg = TRUE

Value

list of logical

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
```

compareSets *Compare the values in two vectors*

Description

Prints the result of comparing the values of two vectors with each other (which values are in the first vector but not in the second one and which values are in the second vector but not in the first one) on the screen.

Usage

```
compareSets(
  x,
  y,
  subject = "Values",
  xname = deparse(substitute(x)),
  yname = deparse(substitute(y))
)
```

Arguments

x	first vector
y	second vector
subject	name of objects to be compared that will appear in the message. Default: "Values".
xname	optional name of the first vector that will be used in the message
yname	optional name of the second vector that will be used in the message

Examples

```
compareSets(1:10, 3:13)
compareSets(1:10, 3:13, "numbers", "set 1", "set 2")
```

containsNulString *Check for nul String in File*

Description

Check for nul String in File

Usage

```
containsNulString(filepath)
```

Arguments

filepath	full path to file to be checked

Value

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

---

convertCsvFile *Modify the format of a CSV file*

Description

This function allows you to change the format (such as column delimiter, decimal character) of a CSV file. It uses read.table to read a CSV file and write.table to rewrite the file with modified format to a new file. All arguments of read.table and write.table are supported. Arguments that are provided by both functions appear as two arguments <argument_name>_in and <argument_name>_out in this function.

Usage

```
convertCsvFile(
  file_in,
  sep_in = formals(utils::read.table)$sep,
  sep_out = sep_in,
  dec_in = formals(utils::read.table)$dec,
  dec_out = dec_in,
  file_out = NULL,
  header = TRUE,
  quote_in = formals(utils::read.table)$quote,
  quote_out = formals(utils::write.table)$quote,
  row.names_in = formals(utils::read.table)$row.names,
  col.names_in = formals(utils::read.table)$col.names,
  row.names_out = FALSE,
  col.names_out = TRUE,
  fileEncoding_in = formals(utils::read.table)$fileEncoding,
  fileEncoding_out = fileEncoding_in,
  dbg = TRUE,
  ...
)
```

Arguments

file_in	path to input file
sep_in	column separator in input file
sep_out	column separator in output file
dec_in	decimal character in input file

dec_out	decimal character inoutput file
file_out	path to output file
header	passed to read.table
quote_in	passed as quote to read.table
quote_out	passed as quote to write.table
row.names_in	passed as row.names to read.table
col.names_in	passed as col.names to read.table
row.names_out	passed as row.names to write.table
col.names_out	passed as col.names to write.table
fileEncoding_in	passed as fileEncoding to read.table
fileEncoding_out	passed as fileEncoding to write.table
dbg	if TRUE (default) debug messages are shown
...	further arguments passed to either read.table or write.table

Value

path to the created CSV file

Examples

```
# Write the iris dataset to a temporary file with "," as column separator
csv_in <- tempfile(fileext = ".csv")
write.table(iris, csv_in, row.names = FALSE)

# Review the first lines of the file
catLines(readLines(csv_in, 6))

# Convert the column separator (from " " which was the default) to ";"
csv_out <- convertCsvFile(csv_in, sep_out = ";")

# Review the result
catLines(readLines(csv_out, 6))

# Delete the file so that it can be recreated
unlink(csv_out)

# Convert the column separator and the decimal character
csv_out <- convertCsvFile(csv_in, sep_out = ";", dec_out = ",")

# Review the result
catLines(readLines(csv_out, 6))
```

copyAttributes *Copy Attributes Between Two Objects*

Description

Copy Attributes Between Two Objects

Usage

```
copyAttributes(x, y, attrNames)
```

Arguments

x	object to which to copy attributes of y
y	object from which to copy attributes to x
attrNames	vector of character containing the names of attributes in y to be copied to x

Examples

```
x <- structure(1, a = 2, b = 3)
y <- structure(2, c = 4)
copyAttributes(x, y, "c")
```

copyDirectoryStructure *Copy Directory Structure*

Description

Copy the full directory structure from a source directory to a target directory

Usage

```
copyDirectoryStructure(
  sourcedir,
  targetdir,
  excludePattern = "^$",
  recursive = TRUE,
  dbg = TRUE
)
```

Arguments

sourcedir	path to the source directory
targetdir	path to the target directory
excludePattern	pattern matching directory names to be excluded from the copying process
recursive	if TRUE (default) the full tree of directories and subdirectories is copied, otherwise only the top-level directories
dbg	if TRUE (default) debug messages are shown

Value

This function invisibly returns a vector of character containing the full paths of the directories that were created.

---

copyListElements *Copy List Elements into a List of Lists*

Description

Copy List Elements into a List of Lists

Usage

```
copyListElements(x, y, name = deparse(substitute(y)))
```

Arguments

x	list of lists
y	list of elements
name	name of target list element

Value

x with each sublist being extended by list element name having been taken from y

Examples

```
x <- list(list(a = 1), list(b = 2), list(c = 3))
y <- list("b1", "b2", "b3")
str(copyListElements(x, y, "b"))
```

countNaInColumn *Count NA in one Column of a Data Frame*

Description

Count NA in one Column of a Data Frame

Usage

```
countNaInColumn(data, column)
```

Arguments

data	data frame
column	column name

Value

number of NA in column of data

countOrSum *Count or Sum Up Values Within Groups of rows*

Description

Count or Sum Up Values Within Groups of rows

Usage

```
countOrSum(x, by = NULL, sum.up = NULL)
```

Arguments

x	data frame
by	vector of names of columns in x to be grouped by
sum.up	name of column in x containing numeric values to be summed up. If NULL (default) rows within groups are counted instead of summing up values within each group

Value

object of class xtabs with as many dimensions as there are values in by

Examples

```
# Create a data frame with example data
x <- data.frame(
  Group = rep(c("A", "B", "C"), 4),
  Even = rep(c(FALSE, TRUE), 6),
  Value = seq_len(12)
)

# Count the number of rows for each group
countOrSum(x, "Group")
countOrSum(x, c("Group", "Even"))

# Sum up the values in column "Value" for each group
countOrSum(x, "Group", sum.up = "Value")
countOrSum(x, c("Group", "Even"), sum.up = "Value")
```

`createAccessor` *Create Function to Safely Access Data Frame Columns or List Elements*

Description

Create Function to Safely Access Data Frame Columns or List Elements

Usage

```
createAccessor(x)
```

Arguments

`x`	a data frame or a list

Examples

```
getcol <- createAccessor(data.frame(a = 1:2, b = 2:3))
getcol("a")
getcol("b")
getcol(c("b", "a"))
# getcol("c") # error with info about existing columns
```

createDirAndReturnPath *Create a Directory including required "upward" Folders*

Description

Create a Directory including required "upward" Folders

Usage

```
createDirAndReturnPath(path, dbg = TRUE, confirm = FALSE)
```

Arguments

path	character string representing the path to the directory to be created
dbg	if TRUE messages about created or found directories are shown
confirm	if TRUE (the default is FALSE!) the user is asked to confirm the creation of a directory

Value

created path or NULL if the path could not be created

createDirectories *Create Directories*

Description

Create Directories

Usage

```
createDirectories(paths, ...)
```

Arguments

paths	paths to directories to be created
...	further arguments passed to createDirectory

Value

paths to created directories

createDirectory *Create Directory if it does not exist*

Description

Create Directory if it does not exist

Usage

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

Arguments

dir.to.create	character string representing the path to the directory to be created
dbg	if TRUE messages about created or found directories are shown
confirm	if TRUE (the default is FALSE!) the user is asked to confirm the creation of a directory

Value

created path or NULL if the path could not be created

createFunctionAssignObjects *Create Function to Provide Package Objects in Global Environment*

Description

Create Function to Provide Package Objects in Global Environment

Usage

```
createFunctionAssignObjects(package)
```

Arguments

package	name of package for which to create a shortcut to assignPackageObjects(package = package)

Value

This function returns a function that internally calls assignPackageObjects(package = package)

Examples

```
provideFunctions <- createFunctionAssignObjects("kwb.utils")
## Not run:
provideFunctions()

## End(Not run)
```

`createFunctionExtdataFile` *Create Function to Compose Path in Package*

Description

Create Function to Compose Path in Package

Usage

```
createFunctionExtdataFile(package)
```

Arguments

package	name of package for which to create a shortcut to `system.file("extdata", ..., package = package)`

Value

This function returns a function that internally calls `system.file("extdata", ..., package = package)`

Examples

```
extdataFile <- createFunctionExtdataFile("kwb.utils")
dir(extdataFile())
```

`createIdAlong` *Create Unique Identifiers along Input Vector*

Description

This function helps you create strings that can be used as identifiers.

Usage

```
createIdAlong(x, base_name = NULL)
```

Arguments

x	vector of objects for which to create identifiers
base_name	base string to which the hexadecimal suffixes are to be appended. By default the name of the object x, with "s" at the end removed, is used. For example, passing a vector called "files" to this function results in ids "file_01", "file_02", ...

Value

vector of character as long as x

Examples

```
# Create ids for 32 numbers
createIdAlong(1:32, "number")
# Take base name from x, with the plural's "s" removed
numbers <- 1:32
createIdAlong(numbers)
```

---

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,
  name.row = NULL,
  name.col = NULL
)
```

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
name.row	optional. Name to be given to the row dimension
name.col	optional. Name to be given to the column dimension

Value

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

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)

## Give a name to the row dimension
createMatrix(c("A", "B", "C"), name.row = "Letters")
```

---

createPasswordFile *Create a File Storing a Password in Encrypted Form*

Description

Create a File Storing a Password in Encrypted Form

Usage

```
createPasswordFile(account = NULL, keyFile, passwordFile, password = NULL)
```

Arguments

account	optional. Name of account the user is asked to enter the password for. Only used for the input prompt.
keyFile	path to the file containing the encryption/decryption key, as generated by generateKeyFile
passwordFile	path to the password file to be created. An existing file will be overwritten!
password	password for account. If NULL (default) the user will be asked to enter the password on the console

See Also

generateKeyFile, getPassword

createStorage *Create Storage Object to Save/Load .rds or .RData files*

Description

This function returns an object that implements the methods `list()`, `save()`, `load()`, `remove()` that list, store, load or remove, respectively .rds or .RData files. The object is associated to a folder that is given to this function.

Usage

```
createStorage(path, type = "rds")
```

Arguments

`path`	path to the folder in which to store the RDS files. If the folder does not exist it is attempted to be created.
`type`	one of `"rds"`, `"RData"`, specifying the format used to store R objects.

Value

list with "member functions" `list()`, `save()`, `load()`, `remove()`, and variables `path`, `type`

Examples

```
# Create an empty test folder
path <- file.path(tempdir(), "test")
dir.create(path)

# Create a storage object pointing to a temporary test folder
storage <- kwb.utils::createStorage(path)

# At the beginning, the storage is empty
storage$list()

# Store objects
apple <- 5
storage$save(apple, banana = list("Hello", "World"))

## Not run:
# List the objects
storage$list()

# Retrieve an object
storage$load("banana")

# Remove objects
storage$remove("apple")
storage$remove("banana")
```

```
# The storage is empty again
storage$list()

## End(Not run)
```

---

csvTextToDataFrame *CSV Text to Data Frame*

Description

CSV Text to Data Frame

Usage

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

Arguments

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

---

decode *Decode a Vector of Character*

Description

For an example see encode.

Usage

```
decode(x)
```

Arguments

x	vector of encoded character strings as returned by encode

Value

vector of decoded character strings

defaultIf *Return a Default Value for Values Meeting a Condition*

Description

Return a Default Value for Values Meeting a Condition

Usage

```
defaultIf(FUN.test, x, default, count = FALSE)
```

Arguments

FUN.test	function returning vector of logical, to be applied to x
x	vector in which to set default values
default	value to be set in x at those positions where FUN.text returns TRUE
count	if TRUE (the default is FALSE) the number of replaced values is returned in the attributes count

defaultIfNA *Default Value if Object is NA*

Description

Return the given object or a default value if the object is NA

Usage

```
defaultIfNA(x, default, count = FALSE)
```

Arguments

x	vector possibly containing NA values
default	default value that is returned if x is NA
count	if TRUE (the default is FALSE) the number of replaced values is returned in the attributes count

Value

x if x is not NA and default otherwise

See Also

defaultIfNULL, defaultIfZero

Examples

```
defaultIfNA(NA, "default") # returns the default value
defaultIfNA("actual", "default") # returns the "actual" value
```

`defaultIfNULL` *Default Value if Object is NULL*

Description

Return the given object or a default value if the object is NULL

Usage

```
defaultIfNULL(x, default, count = FALSE)
```

Arguments

`x`	R object to be checked for NULL
`default`	default value that is returned if `x` is NULL
`count`	if `TRUE` (the default is `FALSE`) the number of replaced values is returned in the attributes `count`

Value

`x` if `x` is not NULL and `default` otherwise

See Also

`defaultIfNA`, `defaultIfZero`

Examples

```
defaultIfNULL(NULL, "default") # returns the default value
defaultIfNULL("actual", "default") # returns the "actual" value
```

defaultIfZero *Default Value if Object is 0 (zero)*

Description

Return the given object or a default value if the object is 0 (zero)

Usage

```
defaultIfZero(x, default, count = FALSE)
```

Arguments

x	vector possibly containing zeroes
default	value to be used instead of zero
count	if TRUE (the default is FALSE) the number of replaced values is returned in the attributes count

Value

x if x is not 0 and default otherwise. The returned object has an attribute count containing the number of values that have been set to the default value.

See Also

defaultIfNA, defaultIfNULL

Examples

```
defaultIfZero(c(1, 2, 0, 4, 5, 0, 6), NA) # returns the default value (NA)
(out <- defaultIfZero(1:6, NA, count = TRUE)) # returns the "actual" values

# The number of values that were zero is returned in the attribute "count"
attr(out, "count")
```

defaultLevels *Default Factor Levels*

Description

If x is numeric and all values are whole numbers use the sequence between the smallest and the highest value

Usage

```
defaultLevels(x, step = 1)
```

Arguments

x	numeric or character vector
step	step to be used in sequence generation

Value

if x is numeric the sequence between the lowest and highest value in x with the given step is returned. If x is a vector of character strings, the sorted unique values are returned.

Examples

```
defaultLevels(c(1, 3, 4, 5, 4))
defaultLevels(c(1920, 1950, 1970), step = 10)
```

---

defaultWindowsProgramFolders *Default Windows Program Folders*

Description

Default Windows Program Folders

Usage

```
defaultWindowsProgramFolders()
```

---

desktop *Path to Your Desktop*

Description

Get the path to your desktop

Usage

```
desktop(osType = .OStype())
```

Arguments

osType	Optional. Type of operating system, one of "unix", "windows"

Value

character string representing the path to your desktop

diffrows *Differences between Matrix Rows*

Description

Differences between Matrix Rows

Usage

```
diffrows(x)
```

Arguments

x	matrix

Value

matrix with one row less than in input matrix `x` and each row `i` representing the difference `x[i+1, ]-x[i, ]` between rows `i+1` and `i` in `x`

Examples

```
x <- matrix(1:12, nrow = 3)

d <- diffrows(x)

x[2, ] - x[1, ] == d[1, ]
x[3, ] - x[2, ] == d[2, ]
```

DIN.A4 *Width and Height of a DIN A4 Paper*

Description

Width and Height of a DIN A4 Paper

Usage

```
DIN.A4()
```

directoryName *Safe Version of Base Function dirname()*

Description

The base function dirname may fail if the path passed is too long. This version checks if the call of dirname() failed and gives a clear error message.

Usage

```
directoryName(x)
```

Arguments

x	a file path of which to get the path to the directory only

Value

path to directory of file path given in x

Examples

```
## Not run:
directoryName(repeated("verylongpath/", 50))

## End(Not run)
```

dropDim *Drop Array Dimension(s) of Length One*

Description

Drop Array Dimension(s) of Length One

Usage

```
dropDim(x, dimension = which(dim(x) == 1))
```

Arguments

x	an array
dimension	number(s) of dimension(s) of length one to be removed

Value

array with dimensions of which the numbers are given in dimension removed

Examples

```
# Define an array of two matrices
A <- array(
1:8, dim = c(2, 2, 2), dimnames = list(
    paste0("x", 1:2), paste0("y", 1:2), paste0("z", 1:2))
)

# The aim is to select the first column of the first matrix with
# the matrix structure being kept. This cannot be done with the
# standard "[" operator. It has indeed a "drop" argument but this
# acts on all dimensions:

# By default, drop is TRUE. The result is a named vector
A[, 1, 1]

# With drop = FALSE we get a 3D-array again and not a matrix
A[, 1, 1, drop = FALSE]

# Use dropDim to remove the third dimension of an array that
# has already one dimension of length one
dropDim(A[, 1, 1, drop = FALSE], dimension = 3)
```

---

dropUnusedFactorLevels

Drop Unused Factor Levels in all Factor Columns

Description

Drop Unused Factor Levels in all Factor Columns

Usage

```
dropUnusedFactorLevels(data, dbg = TRUE)
```

Arguments

data	data frame in which to remove unused levels in all columns that are factors
dbg	if TRUE, debug messages are shown

Value

data with unused factors removed in all columns being factors

Examples

```
# Create an example data frame with two factor columns
data <- data.frame(
  id = 1:3,
  factor_1 = factor(c("a", "b", "a"), levels = c("a", "b", "c")),
  factor_2 = factor(c("x", "x", "y"), levels = c("x", "y", "z")),
  no_factor = c("A", "B", "C"),
  stringsAsFactors = FALSE
)

# Review the structure of the data frame
str(data)

# Review the structure of the data frame with unused factors removed
str(dropUnusedFactorLevels(data))
```

---

encode *Encode a Vector of Character*

Description

Encode a Vector of Character

Usage

```
encode(x, level = 1, chars = printable_chars(level))
```

Arguments

x	vector of character
level	one of 1, 2, or 3 defining which character set to use for encoding, see printable_chars.
chars	vector of characters to be used for encoding

Value

vector of character as long as x with each element containing the encoded version of the corresponding element in x. The returned vector has an attribute codes being a named vector. This vector contains the unique values in x as elements. Each element is named by the code that was used to encode the corresponding element.

Examples

```
x <- c("very-long-word", "very-long-word", "very-very-long-word")
encoded <- encode(x)
encoded
decode(encoded)
```

enlargeVector *Enlarge a Vector to Given Length*

Description

Enlarge a vector to the given length, filling with given element

Usage

```
enlargeVector(x, length.out, fill.with = "")
```

Arguments

x	vector
length.out	desired length of output vector
fill.with	element to fill the vector up with (default: "")

Value

x, filled up with fill.with to a final length of length.out

Examples

```
kwb.utils::enlargeVector(1:5, 10, fill.with = 0)
kwb.utils::enlargeVector(1:5, 10, fill.with = NA)
kwb.utils::enlargeVector(c("a", "b", "c"), 10)
kwb.utils::enlargeVector(c("a", "b", "c"), 10, fill.with = "?")
```

excludeNULL *Exclude all NULL Entries from a List*

Description

Exclude all NULL Entries from a List

Usage

```
excludeNULL(x, dbg = TRUE)
```

Arguments

x	a list
dbg	if TRUE (default) a message is shown if elements are removed

Value

list x with all NULL entries excluded

Examples

```
L <- list(a = 1, b = NULL, c = "three")
L

excludeNULL(L)
```

---

expandGrid *Wrapper around expand.grid*

Description

Same as expand.grid but with stringsAsFactors = FALSE by default and with the values of the first argument being changed last, not first.

Usage

```
expandGrid(..., stringsAsFactors = FALSE)
```

Arguments

...	arguments passed to expand.grid, but in reversed order
stringsAsFactors	passed to expand.grid

Examples

```
persons <- c("Peter", "Paul", "Mary")
fruits <- c("apple", "pear")

# With expand.grid() the values of the first argument change first...
(grid_1 <- expand.grid(person = persons, fruit = fruits))

#... with expandGrid() they change last.
(grid_2 <- expandGrid(person = persons, fruit = fruits))

# With expand.grid() character strings are converted to factors by default...
str(grid_1)

# ... with expandGrid() character strings are not converted by default.
# Also, there is no attribute "out.attrs" as it is set by expand.grid().
str(grid_2)
```

extdataFile *Path to File in Installed Package*

Description

Path to File in Installed Package

Usage

```
extdataFile(..., must_exist = TRUE, dbg = FALSE)
```

Arguments

...	parts of the file path to be passed to system.file
must_exist	if TRUE (the default) and the specified file does not exist, the program stops with an error message
dbg	if TRUE (the default is FALSE) debug messages are shown

Value

path to file in the package installation folder in the R library or "" if the path does not exist

path to the specified file

Examples

```
# List the files provided in the "extdata" folder of kwb.utils
dir(extdataFile())
```

extendLimits *Extend the Limits of a Range Vector*

Description

Extend the Limits of a Range Vector

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

`extractRowRanges` *Extract Row Ranges by Pattern*

Description

Extract Row Ranges by Pattern

Usage

```
extractRowRanges(
  x,
  pattern,
  column = NULL,
  starts = NULL,
  startOffset = 1L,
  stopOffset = 1L,
  nameByMatch = FALSE,
  nameColumnsByMatch = TRUE,
  renumber = TRUE
)
```

Arguments

`x`	data frame or matrix or vector of character (representing e.g. the rows read from a text file)
`pattern`	pattern to be searched for, either in `[, column]` (if `x` is a data frame or a matrix) or in `x` (if `x` is a vector of character)
`column`	name of column in which to search for *pattern* if `x` is a data frame or a matrix
`starts`	optional. Vector of indices representing the starts of the row ranges to be extracted. This argument overrides `pattern`. Instead of using the pattern to find the start indices, the indices given here are used.
`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. Defaults to `FALSE`.
`nameColumnsByMatch`	if `TRUE` (default) the columns of the result data frames or matrices are named (if `x` is a data frame or a matrix)
`renumber`	if `TRUE` (default) and `x` is a data frame or a matrix the row names of the result data frames or matrices are reset to `NULL`, i.e. their rows are renumbered

Value

list of data frames or list of matrices or list of vectors of character. Each list element represents one section of the input, found between two consecutive matches of `pattern`.

Examples

```
textLines <- c(
  "Date,Value",
  "1.1.,1",
  "2.1.,2",
  ",",
  "Date,Value",
  "3.1.,3",
  "4.1.,4",
  ",",
  "Date,Value",
  "5.1.,5",
  "6.1.,6"
)

# Convert textLines to data frame. The function should be able to handle both.
(dataFrame <- read.table(text = textLines, sep = ",", stringsAsFactors = FALSE))

# stopOffset = 2L: skip empty line at the bottom of each sub-table
extractRowRanges(
  textLines,
  pattern = "Date",
  stopOffset = 2L,
)

extractRowRanges(
  dataFrame,
  pattern = "Date",
  column = "V1",
  stopOffset = 2L
)

# Extract sections after a header line
# startOffset = 2L: skip empty line after header "topic: ..."
textLines <- c(
  "topic: A",
  "",
  " a.1",
  " a.2",
  " a.3",
  "topic: B",
  "",
  " b.1",
  "topic: C",
  "",
  " c.1",
  " c.2"
)

extractRowRanges(
  textLines,
  pattern = "topic",
```

```
  startOffset = 2L,
  nameByMatch = TRUE
)
```

`extractSubstring` *Extract Substrings Defined by Regular Expressions*

Description

Extract substrings defined by regular expressions from a vector of strings

Usage

```
extractSubstring(pattern, x, index, stringsAsFactors = FALSE)
```

Arguments

`pattern`	regular expression containing parts in pairs of opening and closing parentheses defining the part(s) to be extracted
`x`	vector of character strings
`index`	index(es) of parenthesized subexpression(s) to be extracted. If the length of `x` is greater than one a data frame is returned with each column containing the substrings matching the subexpression at the corresponding index. If `index` is named, the names will be used as column names.
`stringsAsFactors`	if `TRUE` (default is FALSE) and a data frame is returned then the columns in the returned data frame are of factors, otherwise vectors of character.

Examples

```
# Define pattern matching a date
pattern <- "([^ ]+), ([0-9]+) of ([^ ]+)"

# Extract single sub expressions from one string
datestring <- "Thursday, 8 of December"
extractSubstring(pattern, datestring, 1) # ""Thursday""
extractSubstring(pattern, datestring, 2) # "8"
extractSubstring(pattern, datestring, 3) # "December"

# Extract single sub expressions from a vector of strings
datestrings <- c("Thursday, 8 of December", "Tuesday, 14 of January")
extractSubstring(pattern, datestrings, 1) # "Thursday" "Tuesday"
extractSubstring(pattern, datestrings, 2) # "8"  "14"
extractSubstring(pattern, datestrings, 3) # "December" "January"

# Extract more than one subexpression at once -> data.frame
extractSubstring(pattern, datestrings, 1:3)
```

```
#   subexp.1 subexp.2 subexp.3
#   1 Thursday        8 December
#   2  Tuesday       14  January

# Name the sub expressions by naming their number in index (3rd argument)
extractSubstring(pattern, datestrings, index = c(weekday = 1, 2, month = 3))
#    weekday subexp.2    month
# 1 Thursday        8 December
# 2  Tuesday       14  January
```

`fileExtension` *Get Extension of Full File Paths*

Description

Get Extension of Full File Paths

Usage

```
fileExtension(x)
```

Arguments

`x`	vector of file paths

Examples

```
# Define example paths
paths <- c("C:/example/file.csv", "file2.txt", "D:/e/f/ghi.jkl.zip")

# Get the file name extensions
fileExtension(paths)

# Empty string is returned for paths without file name extensions
fileExtension("C:/NEWS")
```

`findChanges` *Find Value Changes in a Vector*

Description

Finds the positions in a vector where the value changes an returns the "index regions", i.e. the regions between indices `starts_at` and `ends_at` whithin each of which the value does not change, i.e. `length(unique(x[starts_at:ends_at])) == 1L` is `TRUE`.

Usage

```
findChanges(x)
```

Arguments

x	vector of atomic mode (e.g. logical, numeric, character)

Details

The input vector x must not contain any NA because it is not clear how to handle this. The function stops with an error if there are any NA in x.

Value

data frame with one row more than there are value changes in x. If the value in x does not change from one index to the next, it is assumed to belong to the same index region. If the value changes, a new index region begins. In the result data frame each index region is given by starts_at and ends_at which are the indices of the first and last element, respectively, of each index region. The function returns NULL for input vectors x of length 0.

Examples

```
findChanges(c(1,2,2,3,3,3))
findChanges(c("a", "a", "b", "c", "c", "d"))
findChanges(c(TRUE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, FALSE))
```

---

findPartialDuplicates *Find Paritally Duplicated Rows in a Data Frame*

Description

Find Rows in a data frame that are identical in the key columns but not identical in all columns

Usage

```
findPartialDuplicates(data, key_columns, skip_columns = NULL)
```

Arguments

data	data frame
key_columns	names of columns in data in which to look for duplicated (combined) values
skip_columns	names of columns to be skipped when looking for duplicated rows

Value

NULL if there are no rows in data that have identical values in the key_columns or if all groups of rows that have identical values in the key_columns are also identical in all the other columns (except for those named in skip_columns). Otherwise a list is returned with the one element per duplicate in the key columns. The list elements are subsets of data representing the rows of data that are identical in the key columns but different in at least one of the other columns.

Examples

```
findPartialDuplicates(key_columns = "id", data = rbind(
  data.frame(id = 1, value = 1),
  data.frame(id = 2, value = 2),
  data.frame(id = 2, value = 3),
  data.frame(id = 3, value = 3),
  data.frame(id = 3, value = 3),
  data.frame(id = 3, value = 3.1)
))
```

---

finishAndShowPdf *Close Device and Open PDF File in Viewer*

Description

Close the PDF device (default: dev.cur()) and open the PDF file in a PDF viewer

Usage

```
finishAndShowPdf(PDF, which = grDevices::dev.cur(), ...)
```

Arguments

PDF	full path to the PDF file to be opened in the PDF Viewer
which	passed to grDevices::dev.off
...	further arguments passed to hsShowPdf

finishAndShowPdfIf *Finish and Display PDF File if Condition is Met*

Description

Finish and Display PDF File if Condition is Met

Usage

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

Arguments

to.pdf	if TRUE the pdf device is closed and the pdf file is opened in a viewer
PDF	path to the opened pdf file
...	arguments passed to finishAndShowPdf

firstElement *First Element*

Description

Returns the first element using the function head

Usage

```
firstElement(x)
```

Arguments

x	object

Value

first element: x[1]

firstPosixColumn *data/time column of data frame*

Description

data/time column of data frame

Usage

```
firstPosixColumn(x)
```

Arguments

x	data frame in which to find a column of class "POSIXt"

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)

frequencyTable *Number of value occurrences in columns*

Description

Counts the number of occurrences of the different values in each column of a data frame

Usage

```
frequencyTable(
  data,
  columns = names(data),
  orderByLeastLevels = TRUE,
  as.data.frame = TRUE,
  useNA = c("no", "ifany", "always")[2]
)
```

Arguments

data	data frame
columns	columns of data to be included in the frequency analysis. Default: all columns of data
orderByLeastLevels	if TRUE (default) the list elements in the output list each of which represents one column of data or the sections of rows in the output data frame are orderd by length(unique(data[[column]]))
as.data.frame	if TRUE (default) the result is a data frame, otherwise a list (see below)
useNA	passed to table see there. Default: "ifany"

Value

for as.data.frame = FALSE a list of data frames each of which represents the frequency statistics for one column of data. Each data frame has columns *column* (name of the column of data), *value* (value occurring in *column* of data), *count* (number of occurrences). For as.data.frame = TRUE one data frame being the result of rbind-ing together these data frames.

Examples

```
# Some example data
(data <- data.frame(
  A = c("a1", "a2", "a1", "a1", "a2", "", "a2", NA, "a1"),
  B = c("b1", "b1", NA, "b2", "b2", "b1", " ", "b3", "b2")
))

frequencyTable(data) # results in a data frame

frequencyTable(data, as.data.frame = FALSE) # results in a list
```

---

fullWinPath *Windows-Compatible Expanded File Path*

Description

Expanded file path with backslashes (each duplicated if doubleBackslashes is TRUE). Duplicated backslashes can be useful when being used in RMarkdown documents

Usage

```
fullWinPath(x, doubleBackslashes = FALSE)
```

Arguments

x	file path (with slashes or backslashes as separators)
doubleBackslashes	if TRUE the backslashes in the Windows compatible path are duplicated each

Value

Windows compatible file path with backslashes (duplicated if `doubeBackslashes` is `TRUE`)

Examples

```
cat(fullWinPath("~/a/b"))
cat(fullWinPath("~/a/b", doubleBackslashes = TRUE))
```

`fullySorted` *Sort a Data Frame by all of its Columns*

Description

Sort a Data Frame by all of its Columns

Usage

```
fullySorted(x, decreasing = FALSE, ..., renumber.rows = TRUE)
```

Arguments

`x`	data frame
`decreasing`	passed to `order`
`...`	further arguments passed to `order`
`renumber.rows`	if `TRUE` (default) the rows in the returned data frame are renumbered from 1 to the number of rows in `x`

Examples

```
fullySorted(head(iris))
fullySorted(head(iris), decreasing = TRUE)
fullySorted(head(iris[, 5:1]))
fullySorted(head(iris[, 5:1]), decreasing = TRUE)
```

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

See Also

createPasswordFile, getPassword

getAttribute *Safely get the Attribute of an Object*

Description

Safely get the attribute of an object. An error is given if the attribute does not exist (and do.stop = TRUE)

Usage

```
getAttribute(x, attributeName, do.stop = TRUE)
```

Arguments

x	object from which to take the attribute
attributeName	name of the attribute to be returned
do.stop	if TRUE (default) an error is raised if the attribute does not exist.

Examples

```
x <- structure(1, a = 2)
# getAttribute(x, "b") # gives a clear error message
identical(getAttribute(x, "a"), attr(x, "a")) # is TRUE

# Get an attribute's attribute by means of a "path" notation
y <- structure(1, a = structure(2, b = 3))
z <- structure(4, y = y)

str(y)
str(z)

kwb.utils::getAttribute(y, "a/b")
kwb.utils::getAttribute(z, "y/a/b")
```

---

getByPositiveOrNegativeIndex

Get Vector Elements by Positive or Negative Index

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*

getElementLengths *Get the Lenghts of List Elements*

Description

Get the Lenghts of List Elements

Usage

```
getElementLengths(x)
```

Arguments

x	a list

Value

vector of integer

Examples

```
x <- list(a = 1:3, b = list(x = 1, y = 2), c = 1:100)
getElementLengths(x)
```

getEvenNumbers *Get even Numbers out of a Vector of Integers*

Description

Get even Numbers out of a Vector of Integers

Usage

```
getEvenNumbers(x)
```

Arguments

x	vector of integer

getFunctionValueOrDefault *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

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

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.

getListNode *Get List Element in Nested List Structure by Path*

Description

This function allows to access a list element of a nested list structure with a "path". The path "a/b/c", for example, applied to a list L refers to list element L[["a"]][["b"]][["c"]] (same as L[[c("a", "b", "c")]]).

Usage

```
getListNode(x, path)
```

Arguments

x	a list
path	"path" to list element in the form <key1>/<key2>/<key3>...

Examples

```
L <- list(
  a1 = list(
    b1 = list(c = 1, d = 2, e = 3),
    b2 = list(c = list(c1 = 1, c2 = 2, c3 = 3))
  ),
  a2 = list(b3 = 22, b4 = 44)
)

getListNode(L, "a1/b2/c/c2")
getListNode(L, "a1/b2/c")
getListNode(L, "a2/b3")
```

`getNamesOfObjectsInRDataFiles`
Deprecated. Use `listObjects` *instead.*

Description

Deprecated. Use `listObjects` instead.

Usage

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

Arguments

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

`getObjectFromRDataFile`
Deprecated. Please use `loadObject` *instead.*

Description

Deprecated. Please use `loadObject` instead.

Usage

```
getObjectFromRDataFile(file, objectname = NULL, dbg = TRUE)
```

Arguments

`file`	path to .RData file
`objectname`	name of object to be loaded
`dbg`	if `TRUE` a message about which object is loaded from which file is shown

Value

R object as specified in *objectname*. If an object of that name does not exist in the .RData file an error is thrown

---

getOddNumbers *Get odd Numbers out of a Vector of Integers*

Description

Get odd Numbers out of a Vector of Integers

Usage

```
getOddNumbers(x)
```

Arguments

x	vector of integer

---

getPassword *Get Encrypted Password from File Using Key*

Description

Given the path to a file containing an encrypted password and given the path to the file containing the key that was used to do the encryption this function returns the original password, invisibly.

Usage

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

Arguments

passwordFile	path to the password file, as generated by createPasswordFile.
keyFile	path to the key file, as generated by generateKeyFile.

Value

The password (character) or NA if no password is stored, both invisibly.

See Also

generateKeyFile, createPasswordFile

getPathsAndValuesFromRecursiveList *Get Paths and String Values from Recursive List*

Description

Get Paths and String Values from Recursive List

Usage

```
getPathsAndValuesFromRecursiveList(x, path = "")
```

Arguments

x	a list
path	start path

Value

data frame with columns `path` and `value`. The data frame contains all non-list elements that are contained in `x`, coerced to character, in column `value`, together with the sequence of element names "leading" to the value when starting at `x`. For example, the path to element `x$a$a1` is `/a/a1` (see example).

Examples

```
# Define a recursive list
x <- list(
  a = list(a1 = "A1", a2 = "A2"),
  b = list(b1 = "B1", b2 = "B2", b3 = "B3"),
  c = list(c1 = list(c11 = "C11"), c2 = list(c21 = "C21", c22 = "C22"))
)

# Get all non-list-elements and their "path" as a data frame
getPathsAndValuesFromRecursiveList(x)
```

getTagNames *Find <tag>-tags in string*

Description

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

Usage

```
getTagNames(x, bt = c("<>", "[]")[1], dbg = FALSE, expected.length = length(x))
```

Arguments

x	vector of character
bt	bracket type, must be one of c("<>", "[]"). Default: "<>"
dbg	if TRUE (default is FALSE) debug messages are shown
expected.length	if given and different from the length of x an error is thrown

---

get_homedir *Get File Path of User's Home Directory*

Description

Get File Path of User's Home Directory

Usage

```
get_homedir(osType = .OStype())
```

Arguments

osType	Optional. Type of operating system, one of "unix", "windows"

Value

path of user's home directory on windows (full path home directory path but without '/Documents') or linux

---

guessSeparator *Guess Column Separator Used in File*

Description

Guess Column Separator Used in 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	vector of possible column separator characters the file is to be checked for

guessSeparator.1 *Guess Column Separator (Version 1)*

Description

Guess Column Separator (Version 1)

Usage

```
guessSeparator.1(textlines, separators, comment.char = "#")
```

Arguments

textlines	vector of character representing the first lines of a file
separators	vector of possible column separator characters the file is to be checked for
comment.char	comment character

guessSeparator.2 *Guess Column Separator (Version 2)*

Description

Guess Column Separator (Version 2)

Usage

```
guessSeparator.2(textlines, separators)
```

Arguments

textlines	vector of character representing the first lines of a file
separators	vector of possible column separator characters the file is to be checked for

hasFinalSlash *Does a String End with Slash?*

Description

Does a String End with Slash?

Usage

```
hasFinalSlash(x)
```

Arguments

x	vector of character

Value

vector of boolean being TRUE at indices where the elements in x end with a slash ("/")

Examples

```
(is_directory <- hasFinalSlash(c("a", "b/", "c", "d/")))
```

hasZeroLength *Does an Object Have a Length of Zero?*

Description

This is just a "shortcut" to "length(x) == 0L"

Usage

```
hasZeroLength(x)
```

Arguments

x	R object that has a length attribute, e.g. vector or list

Value

TRUE if length(x) == 0L, otherwise FALSE

Examples

```
hasZeroLength(character()) # TRUE

hasZeroLength(list()) # TRUE

# Do not confuse, has nothing to do with the length of a string
# here: vector of length one
hasZeroLength("") # FALSE

# Remember that the length of a data frame is the number of its columns
hasZeroLength(data.frame(a = character())) # FALSE
```

---

headtail *Print First and Last Rows of a Data Frame*

Description

Print the first and last rows of a data frame using head and tail, respectively. Print the number of omitted rows

Usage

```
headtail(x, n = 6, pattern = "[%d rows omitted]")
```

Arguments

x	data frame
n	total number of rows to be printed.
pattern	pattern given to sprintf containing a %d placeholder to print the number of omitted rows

Value

number of omitted rows, invisibly

Examples

```
x <- data.frame(number = 1:26, letter = LETTERS)
headtail(x)
headtail(x, 10)
headtail(x, 16)
headtail(x[10:20, ], 10)
```

hsAddMissingCols *Add missing Columns to a 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*

hsChrToNum *Character to Numeric*

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.

Value

vector of numeric(s). In case of conversion the function stops (if stopOnError = TRUE) or returns NA for those indices for which the conversion failed. In the latter case an attribute "errorIndices" containing the corresponding indices is assigned to the result vector.

hsCountInStr *Count Pattern in String*

Description

Count occurrences of `chr` in `str`

Usage

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

Arguments

`chr`	character string (or pattern) to be looked for and counted in `str`
`str`	character string in which to count for `chr`

Value

number of occurrences of `char` in `str`

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, FUN = function(x) all(is.na(x)), drop = FALSE)
```

Arguments

`dataFrame`	data frame of which empty columns (NA in all rows) are to be removed
`FUN`	function to be applied to each column to decide whether the column is empty or not. Default: `function(x) all(is.na(x))`
`drop`	if `TRUE` (the default is `FALSE`) one dimension is dropped (a vector is returned instead of a data frame) in case that all but one columns are removed.

Value

copy of input data frame but with all empty columns removed

See Also

`removeEmptyColumns`

`hsMatrixToListForm` *Convert "Matrix Form" (wide format) to "List Form" (long format)*

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" (long format)

See Also

`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.

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 = !isNetworkPath(startdir)
)
```

Arguments

startdir	directory to be opened in Windows Explorer
use.shell.exec	if TRUE shell.exec is used instead of running the system command cmd /C explorer

hsPrepPdf *Prepare Writing of PDF File*

Description

Deprecated. Please use preparePdf instead.

Usage

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

Arguments

strPdf	see argument pdfFile in preparePdf
boolLandscape	see argument landscape in preparePdf
bordW	see argument borderWidth.cm in preparePdf
bordH	see argument borderHeight.cm in preparePdf
makeCur	see argument makeCurrent in preparePdf
...	see ... in preparePdf

hsQuoteChr *Quote Character Strings*

Description

quotes objects of type character with quoting character

Usage

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

Arguments

x	vector or list of character strings
qchar	quoting character to be used. Default: single quote "'"
escapeMethod	one of "double", "backslash", "none" deciding how to treat the quote character if it occurs within the string to be quoted

hsRenameColumns *Rename Columns in a Data Frame (deprecated)*

Description

Rename Columns in a Data Frame (deprecated, use renameColumns instead)

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>

hsResolve *Resolve Placeholders in Dictionary*

Description

Resolve strings according to substitutions defined in dictionary

Usage

```
hsResolve(x, dict = NULL, ..., 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.
...	additional named arguments that are added to dict before resolving
dbg	if TRUE (the default is FALSE) debug messages are shown

Value

The (resolved) value is returned

hsRestoreAttributes *Restore Object Attributes*

Description

Restore 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

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.

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	character string passed to shell
...	additional arguments passed to shell

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	if TRUE (default) the command used to open the PDF file is shown

See Also

preparePdf

Examples

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

preparePdf(tmpPdf)

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

# Finish PDF file.
grDevices::dev.off()

## Not run:
# Open PDF file in viewer.
hsShowPdf(tmpPdf)

## End(Not run)
```

hsStringToDate *Convert String to Date*

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)

hsStringToDouble *Convert String to Double*

Description

Convert string to double considering given decimal sign in input string

Usage

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

Arguments

strDbl	character string representing a double value
dec	decimal character (default: ".")

Value

double representation of input string strDbl

hsSubstSpecChars *Deprecated. Use substSpecialChars instead.*

Description

Deprecated. Use substSpecialChars instead.

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.

hsSystem *Wrapper around "system"*

Description

Wrapper around "system"

Usage

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

Arguments

commandLine	character string passed to system
...	additional arguments passed to system

hsTrim *Remove Leading and Trailing Spaces*

Description

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

Usage

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

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
dbg	if TRUE (the default is FALSE) debut messages are shown

Value

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

hsValidValue *Value in Correct English/German Notation?*

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	vector of character strings
lng	language code: "en" for English or "de" for German
dbg	if TRUE (the default is FALSE) debug messages are shown
accept.na	if TRUE (default) TRUE is returned for NA values within x

indent *Indentat a String (Add Spaces to the Left)*

Description

Indentat a String (Add Spaces to the Left)

Usage

```
indent(x, depth = 0L, tabLength = 2L)
```

Arguments

x	vector of character
depth	indentation level. Default: 0
tabLength	number of spaces per indentation level. Default: 2

inRange *check for values within minimum and maximum value*

Description

check for values within minimum and maximum value

Usage

```
inRange(values, min.value, max.value, rng = NULL)
```

Arguments

values	vector of values
min.value	minimum value (inclusive)
max.value	maximum value (inclusive)
rng	optional. Vector of two elements representing min.value and max.value, respectively

Value

vector of boolean

`insertColumns` *Insert new Column(s) into a Data Frame*

Description

Insert one or more new columns into a data frame before or after the given column

Usage

```
insertColumns(
  Data,
  ...,
  before = "",
  after = "",
  stringsAsFactors = defaultIfNULL(options()$stringsAsFactors, FALSE)
)
```

Arguments

`Data`	data frame
`...`	named objects each of which will be a new column in the data frame. Each object must have as many elements as Data has rows.
`before`	name of column before which to insert the new column(s)
`after`	name of column after which to insert the new column(s)
`stringsAsFactors`	passed on to data.frame() and cbind()

Value

data frame `Data` with new columns inserted before the column named as given in `before` or after the column named as given in `after`

Examples

```
Data <- data.frame(A = 1:5, B = 2:6)

# Insert new columns X and Y before column "B"
insertColumns(Data, before = "B", X = paste0("x", 1:5), Y = paste0("y", 1:5))

# This is the same as inserting new columns X and Y after column "A":
insertColumns(Data, after = "A", X = paste0("x", 1:5), Y = paste0("y", 1:5))

# You may also insert before the first...
insertColumns(Data, before = "A", X = paste0("x", 1:5), Y = paste0("y", 1:5))

# ... or after the last column
insertColumns(Data, after = "B", X = paste0("x", 1:5), Y = paste0("y", 1:5))
```

intToNumeralSystem *Convert Integers to Numeral System*

Description

Convert Integers to Numeral System

Usage

```
intToNumeralSystem(x, base)
```

Arguments

x	vector of integers
base	base of the numeral system

Value

matrix with as many rows as there are elements in x and as many columns as digits that are required to represent the integers in x in the numeral system in base base. The elements of x appear as row names whereas the powers of base appear as column names.

Examples

```
intToNumeralSystem(1:16, base = 2) # binary system
intToNumeralSystem(1:16, base = 10) # decimal system
intToNumeralSystem(1:16, base = 8) # octal system
```

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

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
```

---

isASCII *Do Strings Consist only of ASCII Characters?*

Description

The code has been "stolen" from showNonASCII

Usage

```
isASCII(x)
```

Arguments

x	vector of character

Value

vector of logical with TRUE at positions i where x[i] contains only ASCII characters

Examples

```
months <- c("Januar", "Februar", "M\ue4rz")

cat(months, "\n")

isASCII(months)
```

---

isDotOrDoubleDot *Does a string end with one or two dots (".", "..")?*

Description

Does a string end with one or two dots (".", "..")?

Usage

```
isDotOrDoubleDot(x)
```

Arguments

x	vector of character

Value

vector of logical

Examples

```
isDotOrDoubleDot(c("a", "b.", "c..", "d", "efg..h"))
```

---

isEvenNumber *Check for Even Numbers*

Description

Check for Even Numbers

Usage

```
isEvenNumber(x)
```

Arguments

x	vector of numeric

---

isLoaded *Has a Script Been Loaded (with source)?*

Description

Check whether a script has been loaded (with source) or not. The information is read from a list that stores a logical value (TRUE or FALSE) for each script name for which setLoaded has been called.

Usage

```
isLoaded(scriptName)
```

Arguments

scriptName	name of the script for which to get the "loaded" state

See Also

setLoaded (see example there)

Examples

```
# For an example see kwb.utils::setLoaded()...

# In fact, the information about loaded scripts is stored in the
# R option "kwb.utils.loaded":
setLoaded("myScript")

options("kwb.utils.loaded")[[1]]
```

`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).

`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).

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)

isNetworkPath *Does the Path Represent a Network Path?*

Description

Does the Path Represent a Network Path?

Usage

```
isNetworkPath(x)
```

Arguments

x	vector of character representing paths

Examples

```
isNetworkPath("//server/folder/file.txt")
```

`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.

`isOddNumber` *Check for Odd Numbers*

Description

Check for Odd Numbers

Usage

```
isOddNumber(x)
```

Arguments

`x`	vector of numeric

isTryError *Does an Object Inherit from "try-error"?*

Description

Does an Object Inherit from "try-error"?

Usage

```
isTryError(x)
```

Arguments

x	R object

Value

logical of length one, TRUE if x inherits from "try-error", otherwise FALSE

Examples

```
result <- try(stop("Stop!"), silent = TRUE)
isTryError(result) # TRUE
```

lastElement *Last Element*

Description

Returns the last element using the function tail

Usage

```
lastElement(x)
```

Arguments

x	object

Value

The last element of `object` is returned: `x[[length(x)]]` if `x` is a list (and not a data frame), otherwise `tail(x, 1)`.

left *Left Part of a String*

Description

Left Part of a String

Usage

```
left(x, n)
```

Arguments

x	vector of character
n	number of characters to be kept from the beginning of each character string within x

Value

vector of character

Examples

```
left("Good Morning", 4)
```

leftSubstringEquals *Is Left Substring of X Equal To Y?*

Description

TODO: Simply use startsWith?

Usage

```
leftSubstringEquals(x, y)
```

Arguments

x	String of which the left part is compared with y
y	String to be compared with the left part of x

Examples

```
leftSubstringEquals("Great job", "Great")
leftSubstringEquals("Great car", "Great")
leftSubstringEquals("Boring job", "Great")
```

limitToRange *Limit Values to Interval*

Description

limit the values in x so that each value lies within the closed interval [left, right]

Usage

```
limitToRange(x, left = .Machine$double.eps, right = 1)
```

Arguments

x	vector of numeric values
left	lower boundary
right	upper boundary

Examples

```
limitToRange(1:20, left = 5, right = 15)
```

linearCombination *Linear Combination of a Matrix*

Description

Calculate the linear combination of a matrix

Usage

```
linearCombination(x, coeffs, version = 1)
```

Arguments

x	numeric matrix
coeffs	numeric vector of coefficients
version	1 or 2 (default: 1). Allows for two different versions of calculation both of which should return the same!

Examples

```
(x <- randomMatrix(c(4, 2)))
(coeffs <- rnorm(ncol(x)))

# Calculate the linear combination manually
LC1 <- x[, 1] * coeffs[1] + x[, 2] * coeffs[2]

# Caluclate with linearCombination()
LC2 <- linearCombination(x, coeffs)

# The result shoulc be the same!
all.equal(LC1, LC2) # TRUE
```

---

listObjects *Get Names of Objects in .RData files*

Description

Get Names of Objects in .RData files

Usage

```
listObjects(files)
```

Arguments

files	vector of full paths to .RData files

Examples

```
## Not run

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

## Get the names of the objects in the .RData files
#objectsInFiles <- listObjects(files)

## 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[dataQ.found], collapse = "\n  "))
```

`listToDepth` *List Elements Recursively up to Depth*

Description

List Elements Recursively up to Depth

Usage

```
listToDepth(
  path,
  max_depth = 0L,
  full_info = FALSE,
  FUN = listFiles,
  ...,
  depth = 0,
  prob_mutate = 0,
  template = NULL
)
```

Arguments

`path`	path to the element at which to start listing
`max_depth`	maximal depth level of which to list elements. A value of `0` means non-recursive listing, a value of `NA` represents fully recursive listing.
`full_info`	return only `path` and `isdir` information or the full information provided by `FUN(full_info = TRUE)`?
`FUN`	function called to get the listing of the element given in `path`. The function must accept a path as its first argument and it must define the argument `full_info` second. It may accept further arguments. It must always return a data frame. For `full_info = FALSE` the data frame must have columns `file` and `isdir` (is the "file" a directory?). For `full_info = TRUE` the function may return further columns. The function must provide an empty data frame with the expected columns when being called with `character()` as the first argument. The function is expected to set the attribute "failed" to the given path in case that the path could not be accessed (e.g. because of a broken internet connection if the listing is done remotely). See `kwb.utils:::listFiles` for an example implementation that somehow simulates the behaviour of the `dir` function. See `kwb.dwd::list_url()` for a more advanced usage of this function in order to recursively list the files on an FTP server (FTP = file transfer protocol).
`...`	further arguments passed to `FUN`
`depth`	start depth of recursion if `max_depth > 0`. This argument is for internal use and not intended to be set by the user!
`prob_mutate`	probability to alter the path so that it becomes useless. This is zero by default. Set the value only if you want to test how the function behaves if the listing of a path fails.

template	empty data frame (zero rows) and with columns that are identical to the columns in the data frame returned by FUN. If not provided the function will call FUN once with a zero-length path argument expecting to retrieve the template that is expected here.

Value

data frame containing at least the columns file and isdir. If full_info = TRUE the result data frame may contain further columns, as provided by the function given in FUN for full_info = TRUE.

Examples

```
# Example list function provided in this package (file listing)
FUN <- kwb.utils:::listFiles

# The list function must provide empty records when no path is given. The
# returned data frame must provide the columns "file" and "isdir" ...
FUN()
FUN(full_info = TRUE)

# ... even when being called with an empty character vector
FUN(character())
FUN(character(), full_info = TRUE)

# Call the function recursively up to the given depth level
kwb.utils:::listToDepth(".", max_depth = 1, FUN = FUN)
```

---

loadFunctions *Load Functions from Scripts into Attached Namespace*

Description

This function parses R scripts at the given paths and provides all objects in an environment that is attached to the search path.

Usage

```
loadFunctions(paths, attach = TRUE, name = "kwb", dbg = TRUE)
```

Arguments

paths	vector of character with paths to files (R scripts) or directories (in which to look for R scripts).
attach	if TRUE the environment with all parsed objects is attached to the search path, otherwise the environment is returned
name	name that is used when attaching the environment with all parsed objects to the search path. Default: kwb
dbg	logical indicating whether or not to show debug messages

Value

if `attach` is `FALSE` the environment with all parsed objects is returned, otherwise `NULL`, invisibly.

---

loadObject *Load R object from .RData file*

Description

Load an R object of given name from a .RData file

Usage

```
loadObject(file, objectname = NULL, dbg = TRUE)
```

Arguments

`file`	path to .RData file
`objectname`	name of object to be loaded
`dbg`	if `TRUE` a message about which object is loaded from which file is shown

Value

R object as specified in *objectname*. If an object of that name does not exist in the .RData file an error is thrown

---

mainClass *Main Class of an Object*

Description

Returns the first element of what class(x) returns

Usage

```
mainClass(x)
```

Arguments

`x`	any R object

Value

name of main class of `x` (vector of character of length one)

Examples

```
class(as.POSIXct("2022-06-02"))
mainClass(as.POSIXct("2022-06-02"))
```

makeUnique *Make Duplicated Character Strings Unique adds ".1", ".2", etc. to duplicate values*

Description

Make Duplicated Character Strings Unique

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

Usage

```
makeUnique(x, warn = TRUE, sep = ".", simple = FALSE)
```

Arguments

x	vector of character strings
warn	if TRUE (default) a warning showing the duplicated values is given
sep	separator between name and suffix number. Default: "."
simple	if TRUE all elements with identical name (e.g. "a") are numbered (e.g. "a.1", "a.2", "a.3"), otherwise the first element is kept unchanged and all but the first one are numbered (e.g. "a", "a.1", "a.2"). The default is FALSE.

Value

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

matchesCriteria *Do Data Frame Row Match Given Criteria?*

Description

are data frame rows matching given criteria?

Usage

```
matchesCriteria(
  Data,
  criteria = NULL,
  na.to.false = FALSE,
  add.details = FALSE,
  dbg = TRUE
)
```

Arguments

Data	data frame
criteria	vector of character containing conditions, in which the column names of Data, e.g. A can appear unquoted, e.g. "A == 'x'"
na.to.false	if TRUE (the default is FALSE) NA in the resulting vector will be replaced with FALSE
add.details	if TRUE (the default is FALSE) a matrix containing the evaluation of each criterion is returned in attribute details
dbg	if TRUE (default) for each criterion in criteria it is shown for how many rows in Data the criterion is TRUE and for how many rows it is FALSE

Value

vector of logical containing TRUE at positions representing rows in Data fulfilling the conditions and FALSE elsewhere

Examples

```
# Define an example data frame
Data <- data.frame(A = c("x", "y", "z", NA),
                   B = c( NA,   2,   3, 4))

# Define one or more criteria
criteria <- c("A %in% c('y', 'z')", "B %in% 1:3")

# For which rows the criteria are met (vector of logical)?
matchesCriteria(Data, criteria, dbg = FALSE)

# You may use the function in the context of indexing:
Data[matchesCriteria(Data, criteria), ]

# Filtering for non-NA values
D1 <- Data[matchesCriteria(Data, "! is.na(A) & ! is.na(B)"), ]

# the same result is returned by:
D2 <- Data[matchesCriteria(Data, c("! is.na(A)", "! is.na(B)")), ]

identical(D1, D2)
```

---

matrixToDataFrame *Convert a Matrix to a Data Frame (in "Long" Format)*

Description

Convert a Matrix to a Data Frame (in "Long" Format)

Usage

```
matrixToDataFrame(
  x,
  name_row = NULL,
  name_column = NULL,
  name_value = "value",
  row_first = TRUE
)
```

Arguments

x	matrix
name_row	name to be given to the data frame column containing the row "coordinates". Default: `names(dimnames(x))[1]` unless `NULL`, `"row"` otherwise.
name_column	name to be given to the data frame column containing the column "coordinates". Default: `names(dimnames(x))[2]` unless `NULL`, `"column"` otherwise.
name_value	name to be given to the data frame column containing the matrix values. Default: `value`
row_first	if `TRUE` (the default), the "row column" will come first, else the "column column".

Value

data frame with three columns: 1. row "coordinate", 2. column "coordinate", 3. value

Examples

```
m1 <- matrix(1:12, nrow = 3, dimnames = list(NULL, letters[1:4]))
m2 <- matrix(1:12, nrow = 3, dimnames = list(index = NULL, letters[1:4]))
m3 <- matrix(1:12, nrow = 3, dimnames = list(NULL, letter = letters[1:4]))

matrixToDataFrame(x = m1)
matrixToDataFrame(x = m1, row_first = FALSE)
matrixToDataFrame(x = m2)
matrixToDataFrame(x = m3)
matrixToDataFrame(x = m3, "myrow", "mycol", "myval")
```

---

maxStringLength *Maximum String Length in Vector of Character*

Description

Maximum String Length in Vector of Character

Usage

```
maxStringLength(x)
```

Arguments

x	vector of character

Value

integer representing the length of the longest string in x

Examples

```
maxStringLength(c("a", "ab", "abc", "x", "xy-z"))
```

---

mergeAll *Merge Multiple Data Frames*

Description

Merge multiple data frames, given in a list

Usage

```
mergeAll(dataFrames, by, ..., dbg = TRUE)
```

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
dbg	if TRUE (default) debug messages showing the process of merging are shown

Value

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

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)
```

---

mergeLists *Merge Lists Overriding Elements of the Same Name*

Description

Merge Lists Overriding Elements of the Same Name

Usage

```
mergeLists(..., warn.on.NULL = TRUE)
```

Arguments

...	lists
warn.on.NULL	if TRUE (default) a warning is given if any of the arguments given to this function is NULL

Value

list containing the elements given in ...

See Also

arglist

Examples

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

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

mergeNamedArrays *Merge List of Named Arrays*

Description

Merge List of Named Arrays

Usage

```
mergeNamedArrays(x, check_dim = TRUE)
```

Arguments

x	list of arrays of the same dimension
check_dim	logical. If TRUE, it is checked whether the source dimension names are available in the target dimension names

Examples

```
a1 <- array(
  1:12,
  dim = c(2, 4, 2),
  dimnames = list(paste0("x", 1:2), paste0("y", 1:4), paste0("z", 1:2))
)

a2 <- array(
  11:16,
  dim = c(1, 3, 2),
  dimnames = list("x3", paste0("y", 2:4), paste0("z", 1:2))
)

mergeNamedArrays(list(a1, a2))
```

moveColumnsToFront *Move Columns to the Left*

Description

Move columns of a data frame or matrix to the left

Usage

```
moveColumnsToFront(x, columns = NULL)
```

Arguments

x	data frame
columns	vector of column names

Value

data frame or matrix with columns being the leftmost columns

Examples

```
x <- data.frame(a = 1:5, b = 2:6, c = 3:7)

moveColumnsToFront(x, "b")
moveColumnsToFront(x, c("b", "a"))
```

---

moveToFront *Move elements to the start of a vector*

Description

Move elements to the start of a vector

Usage

```
moveToFront(x, elements)
```

Arguments

x	vector
elements	elements out of x to be moved to the front

Value

vector with elements coming first

Examples

```
moveToFront(1:10, 5)
moveToFront(c("a", "b", "c", "x", "y", "d"), c("x", "y"))
```

movingSum *movingSum*

Description

Calculate moving sum of n values "around" values

Usage

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

Arguments

x	vector of values of which moving sum 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 sum is calculated by: x[i-(n-1)/2] + ... + x[i-1] + x[i] + x[i+1] + ... + x[i+(n-1)/2]
na.rm	logical. Should missing values (including NaN) be omitted from the calculations?

Value

Vector of moving sums 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 at the end of the vector, respectively, to calculate the sum.

Examples

```
x <- rnorm(30)

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

times <- 2:4

for (i in times) {

  lines(movingSum(x, n = 2 * i - 1), col = i, type = "b", lwd =  2)
}

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

msgAvailableFields *Message Listing Available Fields*

Description

Message to be shown if fields/columns are missing

Usage

```
msgAvailableFields(x)
```

Arguments

x	vector of character

multiColumnLookup *Lookup by Matching Values in Multiple Columns*

Description

Lookup by Matching Values in Multiple Columns

Usage

```
multiColumnLookup(data, lookup, value = NULL, drop = TRUE, includeKeys = FALSE)
```

Arguments

data	data frame for which to lookup values in the lookup table
lookup	lookup table defining the key values to be matched against the values in the corresponding columns in data and the corresponding lookup values that are to be returned
value	name of column(s) in lookup containing the value(s) to be looked up. Default: name of the last column in lookup
drop	logical indicating whether or not to return a vector instead of a one-column data frame in case that there is only one value column. The default is TRUE
includeKeys	logical indicating whether or not to include the key column(s) in the returned data frame (for more than one value column or drop = FALSE). The default is FALSE

Value

If value is of length one and drop = TRUE) a vector with as many elements as there are rows in data is returned. Otherwise a data frame with as many rows as there are rows in data and with the columns named in value is returned.

Examples

```
(persons <- rbind(
  noFactorDataFrame(name = "Peter", city = "Berlin"),
  noFactorDataFrame(name = "Paul", city = "Paris"),
  noFactorDataFrame(name = "Mary", city = "Berlin"),
  noFactorDataFrame(name = "Paul", city = "Berlin"),
  noFactorDataFrame(name = "Peter", city = "Paris")
))

# Who is cool, which city is cool and which combination is coolest?
(is_cool <- kwb.utils::safeRowBindAll(list(
  noFactorDataFrame(name = "Paul", city = "Berlin", value = "astro-cool"),
  noFactorDataFrame(city = "Berlin", value = "cool"),
  noFactorDataFrame(name = "Paul", value = "mega-cool"),
  noFactorDataFrame(city = "Paris", value = "ca va")
)))

# Lookup the coolness based on name and city
coolness <- multiColumnLookup(persons, is_cool, value = "value")

# Add the coolness column
cbind(persons, coolness)
```

`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, ..., dbg = FALSE)
```

Arguments

`strings`	vector of character
`replacements`	list of pattern = replacement pairs.
`...`	additional arguments passed to gsub
`dbg`	if `TRUE` (the default is `FALSE`) it is shown which strings were replaced

mySystemTime *Elapsed Time of a Function Call*

Description

Call a function and show the elapsed time using `system.time`)

Usage

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

Arguments

`FUN`	function to be called
`args`	list of arguments passed to `FUN`

nameByElement *Name List Elements by Sublist Element*

Description

name the elements of a list of lists by the value of the element `elementName` of each sublist

Usage

```
nameByElement(x, elementName = "name")
```

Arguments

`x`	list of lists
`elementName`	name of element to be looked up in each sublist of `x`

Examples

```
L <- list(
  list(group = "A", value = 1),
  list(group = "B", value = 2)
)

nameByElement(L, "group")
```

namedVectorFromColumns

Create Named Vector from two Columns of a Data Frame

Description

Create Named Vector from two Columns of a Data Frame

Usage

```
namedVectorFromColumns(data, valueColumn, nameColumn)
```

Arguments

data	a data frame
valueColumn	name of column in data containing the names
nameColumn	name of column in data containing the values

Value

vector with values in data[[valueColumn]], named by data[[nameColumn]]

Examples

```
data <- data.frame(a = 1:3, name = LETTERS[1:3])
namedVectorFromColumns(data, valueColumn = "a", nameColumn = "name")
```

namedVectorToDataFrame

Convert a Named Vector to a Data Frame

Description

Convert a Named Vector to a Data Frame

Usage

```
namedVectorToDataFrame(x)
```

Arguments

x	named vector

Value

data frame with columns name, containing the names of x and value, containing the values of x

Examples

```
namedVectorToDataFrame(c(a = 1, b = 2, c = 3))
```

---

naToLastNonNa *Replace NA With "Last" non-NA*

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	vector in which NA are to be replaced with the last non-NA value (at greatest of smaller indices) in the vector
method	integer (1 or 2) distinguishing two different methods

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 method = 1 and the first element is NA!

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

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

---

nNA *Number of NA values*

Description

Number of NA values

Usage

```
nNA(x)
```

Arguments

x	vector

Value

number of NA values in x

Examples

```
nNA(1:3)
nNA(c(1, NA, 3))
```

---

noFactorDataFrame *Create Data Frame without Factors*

Description

Use data.frame with stringsAsFactors = FALSE

Usage

```
noFactorDataFrame(...)
```

Arguments

...	passed to data.frame

Examples

```
# data.frame() by default converts character to factor
str(data.frame(id = 1:3, letter = LETTERS[1:3]))

# noFactorDataFrame keeps character as character
str(noFactorDataFrame(id = 1:3, letter = LETTERS[1:3]))
```

noSuchElements *Message on Missing Elements*

Description

Message on Missing Elements

Usage

```
noSuchElements(x, available, type = "element", sorted = TRUE, suffix = "")
```

Arguments

x	name of element that was not found
available	names of elements that are available
type	type of element to appear in the message. Default: "element"
sorted	logical. Whether or not to print the available elements in lexical order. Default: TRUE
suffix	suffix to be appended to type. Can be used to distinguish between singular and plural form. Default: ""

Examples

```
cat(kwb.utils:::noSuchElements("x", LETTERS[1:3]))
cat(kwb.utils:::noSuchElements(c("x", "y"), LETTERS[1:3], suffix = "s"))
```

nUnique *Number of Unique Values*

Description

Number of Unique Values

Usage

```
nUnique(x)
```

Arguments

x	R object that function unique can be applied on

Value

Number of unique values in x (integer)

Examples

```
nUnique(1:3)
nUnique(c(1, 1, 2))
```

`objectSize` *Object Size and Sizes of Sub Structures in Mb*

Description

Object Size and Sizes of Sub Structures in Mb

Usage

```
objectSize(x, max_depth = 2, units = "auto", depth = 0)
```

Arguments

`x`	object
`max_depth`	number of sub levels of list structures to be shown
`units`	passed to `object.size`
`depth`	depth of recursive call (for internal use only)

Value

if `x` is a list, a list of the same structure is returned with each list elemenet replaced by its size, otherwise the object size of `x`. Each list or sub list with more than one element is assigned an attribute "total" to containing the total size of the list.

`objectToText` *Convert R Object to Text Representation*

Description

Convert R Object to Text Representation

Usage

```
objectToText(x)
```

Arguments

`x`	R object

Value

vector of character representing the R code that reproduces the object `x`

Examples

```
objectToText(1:10)
objectToText((1:10)[-5])
cat(objectToText(head(iris)))
```

---

orderBy *Order a Data Frame by One or more Columns*

Description

Order a Data Frame by One or more Columns

Usage

```
orderBy(df, by = NULL, ...)
```

Arguments

df	data frame
by	vector of column names specifying the columns by which to order
...	further arguments passed to order, such as decreasing

Value

df being sorted and with newly renumbered rows

Examples

```
orderBy(iris, by = "Sepal.Length")
orderBy(iris, by = "Species", decreasing = TRUE)
orderBy(
  iris,
  by = c("Species", "Petal.Width", "Petal.Length"),
  decreasing = TRUE
)
```

orderDecreasinglyBy *Order Data Frame Decreasingly by one Column*

Description

Order Data Frame Decreasingly by one Column

Usage

```
orderDecreasinglyBy(df, column)
```

Arguments

df	data frame
column	name of column by which to order decreasingly.

Examples

```
(df <- data.frame(a = 1:3, b = 11:13))
orderDecreasinglyBy(df, "a")
```

pairwise *Reorder Strings So That Matching Strings are Neighbours*

Description

Reorder strings so that strings that start with the same characters appear next to each other

Usage

```
pairwise(x, starts = defaultStarts(x, split), split = "_")
```

Arguments

x	vector of character
starts	vector of character defining the start strings that are looked for in x to find strings that belong together. The default is to take the unique strings appearing before a split character (if any)
split	split character used to create default start strings

Examples

```
x <- c("a.1", "b_hi", "c", "a.2", "d", "b_bye")

# You have the most control when setting the starts argument
pairwise(x, starts = c("a.", "b_"))

# Use default starts resulting from splitting at a split character
pairwise(x, split = "_")

# This is actually the default
pairwise(x)

# Note that the split parameter is interpreted as a pattern where the
# dot has a special meaning unless it is escaped or enclosed in []
pairwise(x, split = "[.]")

# Same result as in the first example
pairwise(x, split = "[._]")
```

---

parallelNonNA *Merge two Vectors selecting non-NA Values*

Description

two vectors *a* and *b* of same length are merged in such a way that at index i the result vector contains (1) a[i] if a[i] is not NA and b[i] is NA or b[i] == a[i], (2) b[i] if b[i] is not NA and a[i] is NA or a[i] == b[i], (3) "" if a[i] is not NA and b[i] is not NA and a[i] != b[i] or if both a[i] and b[i] are NA

Usage

```
parallelNonNA(a, b)
```

Arguments

a	vector 1
b	vector 2

Value

character vector of same length as *a* and *b*

Examples

```
parallelNonNA(c(1, NA, 3), c(NA, 2, NA))  # "1" "2" "3"

parallelNonNA(c(1, NA, NA), c(NA, 2, NA)) # "1" "2" ""

## A warning is given if (non-NA) values at the same index differ
```

```
y <- parallelNonNA(c(1, 2, 3), c(1, 2, 4))

y # "1" "2" ""

## attribute "invalid" contains the index and values of the differing values
attr(y, "invalid")
```

`pasteColumns` *Paste Columns of Data Frame With Separator*

Description

Paste Columns of Data Frame With Separator

Usage

```
pasteColumns(x, columns = names(x), sep = " ", ...)
```

Arguments

`x`	data frame
`columns`	names of columns to be pasted. Default: all columns
`sep`	separator character. Default: space (" ")
`...`	args passed to `selectColumns`, e.g. `do.stop` to control whether the function shall stop if not all columns exist

Value

vector of character with each element representing the values of the selected columns of one row, being pasted with the separator character

Examples

```
x <- data.frame(A = 1:3, B = 2:4)
pasteColumns(x, sep = ";")
```

pasteColumns0 *Paste Columns of Data Frame Without Separator*

Description

Paste Columns of Data Frame Without Separator

Usage

```
pasteColumns0(x, columns = names(x), ...)
```

Arguments

x	data frame
columns	names of columns to be pasted. Default: all columns
...	args passed to pasteColumns

Value

vector of character with each element representing the values of the selected columns of one row, being pasted without a separator

Examples

```
x <- data.frame(A = 1:3, B = 2:4)
pasteColumns0(x)
```

percentage *Percentage*

Description

x / basis, in percent

Usage

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

Arguments

x	numeric
basis	numeric

Value

100 * x / basis

percentageOfMaximum *Percentage of Maximum*

Description

Percentage of Maximum

Usage

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

Arguments

x	vector of numeric values
na.rm	passed to max

Value

100 * x / max(x)

percentageOfSum *Percentage of the Sum of Values*

Description

Percentage of the Sum of Values

Usage

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

Arguments

x	vector of numeric values
na.rm	passed to max

Value

100 * x / sum(x)

Examples

```
p <- percentageOfSum(1:10)
stopifnot(sum(p) == 100)
```

posixColumnAtPosition *Indices of POSIX columns in a Data Frame*

Description

Indices of POSIX columns in a Data Frame

Usage

```
posixColumnAtPosition(x)
```

Arguments

x	data frame containing a date/time column

preparePdf *Open PDF Device with DIN A4 Dimensions by Default*

Description

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

Usage

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

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, otherwise the current device will be restored
paper	passed to pdf. By default "A4" (if landscape = FALSE) or "A4r" (if landscape = TRUE). Use paper = "special" to use the dimensions of the plot.
...	further arguments passed to pdf

Value

full path to pdf file

See Also

finishAndShowPdf

Examples

```
## Not run:
# Open PDF file by giving a path to preparePdf(). The path is returned.
pdf_file <- preparePdf(file.path(tempdir(), "example_preparePdf.pdf"))

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

# Open PDF file in viewer
finishAndShowPdf(pdf_file)

## End(Not run)
```

---

preparePdfIf *Prepare PDF File if Condition is Met*

Description

Prepare PDF File if Condition is Met

Usage

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

Arguments

to.pdf	condition determining whether preparePdf is called or not
PDF	full path to PDF file. If this is an empty string (default) a temporary file is created and its path returned
...	arguments passed to preparePdf

Value

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

---

print.repro_sample *Print Method for Object of Class "repro_sample"*

Description

Print Method for Object of Class "repro_sample"

Usage

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

Arguments

x	object to be printed
...	further arguments, not used.

---

printable_chars *Different Sets of Printable ASCII Characters*

Description

Different Sets of Printable ASCII Characters

Usage

```
printable_chars(level = 1)
```

Arguments

level	one of 1, 2, or 3. Level 1 characters comprise the ten digits 0 to 9 and 26 uppercase letters. Level 2 characters comprise the characters of level 1 as well as 26 lowercase letters. Level 3 characters comprise altogether 88 printable characters.

Value

vector of character

Examples

```
printable_chars(1)
printable_chars(2)
printable_chars(3)
```

printIf *Call Print If Condition Is Met*

Description

Call Print If Condition Is Met

Usage

```
printIf(condition, x, caption = deparse(substitute(x)))
```

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*

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

randomMatrix *Create a Matrix with Random Integer Values*

Description

Create a matrix of given dimension and fill it with random integer values

Usage

```
randomMatrix(dim = c(sample(10, 1), sample(10, 1)), values = seq_len(100))
```

Arguments

dim	integer vector of length two containing the number of rows and columns, respectively, that the output matrix shall contain
values	set of values to be used within the matrix

Examples

```
# By default, the matrix has a random number of rows between 1 and 10 and
# a random number of columns between 1 and 10 and random values of 1:100
randomMatrix()

# You may specify the dimensions (here: 5 rows, 3 columns)...
randomMatrix(dim = c(5, 3))

# ... and the set of values to be used within the matrix
randomMatrix(dim = c(5, 3), values = c(0, 0.5, 1, NA))
```

randomValuesWithSum *Vector of random Integer Values of given Sum*

Description

Vector of random Integer Values of given Sum

Usage

```
randomValuesWithSum(n, sumOfValues, names = seq_len(n))
```

Arguments

n	number of values
sumOfValues	sum of values in the result vector
names	names of elements in the result vector. Default: seq_len(n)

Value

named vector of integer values with `sum(values) == sumOfValues`

---

rangeToSequence *Create Sequence from Range*

Description

Create Sequence from Range

Usage

```
rangeToSequence(x)
```

Arguments

x	numeric vector with exactly two elements

Value

integer sequence between `x[1]` and `x[2]`

Examples

```
rangeToSequence(c(1, 10))
```

---

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

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)
```

---

readArglists *Read Argument Lists from CSV File*

Description

Read argument lists from CSV (or MS Excel) file

Usage

```
readArglists(
```

```
  file = NULL,
  configTable = readArglistsTable.csv(safePath(file), dbg = dbg),
  dbg = FALSE
)
```

Arguments

file	full path to comma separated file containing the argument specification
configTable	Instead of the file you may provide a data frame containing the configuration
dbg	if TRUE, debug messages are shown

---

readCsvInputFile *Read CSV File Giving Column Descriptions*

Description

Read CSV File Giving Column Descriptions

Usage

```
readCsvInputFile(
  csv,
  sep,
  dec,
  headerRow = 1,
  headerPattern = "",
  columnDescription = NULL,
  maxRowToLookForHeader = 10,
  stopOnMissingColumns = TRUE,
  fileEncoding = "UTF-8",
  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
fileEncoding	encoding of the input file
encoding	passed to readLines, "Latin-1" or "UTF-8"
...	further arguments passed to read.table

`readDictionaries` *Read Dictionary Files into a List of Dictionaries*

Description

Read files from a folder, specified by a file name pattern, into a list of dictionaries

Usage

```
readDictionaries(folder, pattern = "^dictionary_(.*)[.]txt$")
```

Arguments

folder	path to the folder containing the files to be read
pattern	regular expression to match the names of the files to be read. The pattern is expected to contain a pair of parentheses around the part of the file that shall be used as element name in the returned list

See Also

readDictionary

`readDictionary` *Read Dictionary from Text File*

Description

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

Usage

```
readDictionary(file, sorted = TRUE, fileEncoding = "", ...)
```

Arguments

file	full path to dictionary file or connection as e.g. opened with file
sorted	if TRUE (default) the entries in the dictionary will be sorted by their keys
fileEncoding	passed to readLinesWithEncoding
...	further arguments passed to readLinesWithEncoding

See Also

readDictionaries

Examples

```
file <- system.file("extdata", "dictionary.txt", package = "kwb.utils")

dictionary <- readDictionary(file)

resolve("file.out", dictionary, extension = "csv")
resolve("file.out", dictionary, extension = "pdf")
```

---

readLinesWithEncoding *Read Lines by Giving the File Encoding*

Description

Read Lines by Giving the File Encoding

Usage

```
readLinesWithEncoding(file, ..., fileEncoding = "", encoding = "unknown")
```

Arguments

file	a connection object or character string
...	arguments passed to readLines
fileEncoding	The name of the encoding to be assumed. Passed as encoding to file, see there.
encoding	passed to readLines.

readPackageFile *Read File from Package's extdata Folder*

Description

Read File from Package's extdata Folder

Usage

```
readPackageFile(file, package, stringsAsFactors = FALSE, ...)
```

Arguments

file	name of file (without path)
package	name of the package from which to read the file
stringsAsFactors	passed to utils::read.csv (default: FALSE)
...	further arguments passed to utils::read.csv

Value

result of reading file with utils::read.csv

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: ""

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

relativeCumulatedSum *Relative Cumulated Sum*

Description

relative cumulated sum of a vector of values

Usage

```
relativeCumulatedSum(values)
```

Arguments

values	vector of numeric values

removeAttributes *Remove all or selected Attributes of an Object*

Description

Remove all or selected Attributes of an Object

Usage

```
removeAttributes(x, names = NULL)
```

Arguments

x	object
names	names of attributes to be removed. If NULL (default), all attributes are removed.

Value

x, but with its attributes removed

---

removeColumns *Remove Columns from a Data Frame*

Description

Remove Columns from a Data Frame

Usage

```
removeColumns(
  dframe,
  columns = NULL,
  columnsToRemove = NULL,
  pattern = NULL,
  drop = FALSE,
  dbg = FALSE
)
```

Arguments

dframe	data frame,
columns	vector of column names giving the columns to remove
columnsToRemove	deprecated. Use argument columns instead.
pattern	regular expression matching the names of the columns to be removed. Will only be evaluated if no explicit column names are given in columns.
drop	if FALSE, a data frame is returned in any case, otherwise the result may be a vector if only one column remains
dbg	if TRUE (the default is FALSE), the deletion of columns is reported on the screen

Value

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

removeDuplicates *Remove Duplicated Values from a Vector*

Description

Remove Duplicated Values from a Vector

Usage

```
removeDuplicates(x, dbg = TRUE)
```

Arguments

x	vector from which to remove duplicates
dbg	if TRUE a debug message is shown

Value

x with duplicated values being removed

Examples

```
removeDuplicates(c(1, 1, 2, 3, 4, 4))
```

removeElements *Remove Elements from a List*

Description

Remove Elements from a List

Usage

```
removeElements(x, elements)
```

Arguments

x	a list
elements	names of elements to remove

Value

x with elements with names given in elements being removed. User attributes of *x* are restored.

removeEmpty *Remove Empty Elements from Vector*

Description

Remove Empty Elements from Vector

Usage

```
removeEmpty(x)
```

Arguments

x	vector

Value

x with elements for which isNaOrEmpty is TRUE removed

Examples

```
removeEmpty(c(1, NA, 3))
removeEmpty(c("a", "", "b", NA, "c", " ", "d"))
```

removeEmpty2 *Remove Empty Elements from Vector of Character*

Description

Remove Empty Elements from Vector of Character

Usage

```
removeEmpty2(x, dbg = FALSE)
```

Arguments

x	vector of character
dbg	if TRUE (the default is FALSE) a message about the number of removed elements is printed

Examples

```
removeEmpty2(c("a", "", "c"))
removeEmpty2(c("a", "", "c", "", "e"), dbg = TRUE)
```

removeEmptyColumns *Remove empty Columns from a Data Frame*

Description

Remove empty Columns from a Data Frame

Usage

```
removeEmptyColumns(
  x,
  drop = FALSE,
  FUN = function(x) all(is.na(x)),
  dbg = TRUE
)
```

Arguments

x	data frame
drop	if TRUE and only one column remains the column is returned as a vector
FUN	function called on each column to determine if all values in the column are empty. Default: `function(x) all(is.na(x))`
dbg	if TRUE debug messages are shown

Value

data frame x with empty columns (columns with NA in all rows) being removed

See Also

hsDelEmptyCols

removeExtension *Remove File Name Extension*

Description

Remove File Name Extension

Usage

```
removeExtension(x)
```

Arguments

x	vector of character

Examples

```
removeExtension("example.R")
removeExtension("any/path/example.txt")
```

`removeLeadingSlashes` *Remove Leading Slashes of Strings*

Description

Remove Leading Slashes of Strings

Usage

```
removeLeadingSlashes(x)
```

Arguments

x	vector of character

Value

modified version of `x` with all leading slashes in all elements of `x` removed

Examples

```
removeLeadingSlashes(c("a", "/b", "//c"))
```

`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

renameAndSelect *Rename and Select Columns of a Data Frame*

Description

Rename and Select Columns of a Data Frame

Usage

```
renameAndSelect(data, renames, columns = unlist(renames))
```

Arguments

data	data frame
renames	list defining renames in the form of "oldName" = "newName" pairs
columns	(new) names of columns to be selected

renameColumns *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

```
renameColumns(x, renamings = NULL)
```

Arguments

x	data.frame
renamings	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*

repeated *Repeated Substring*

Description

Repeated Substring

Usage

```
repeated(x, n)
```

Arguments

x	substring to be repeated and pasted together to a new string
n	number of times to repeat the substring

Value

vector of character of length one

Examples

```
repeated("no ", 2)
repeated("yes ", 3)
repeated("yes no ", 3)
```

replaceFileExtension *Replace File Name Extension*

Description

Replace the file name extension in a file name or file path

Usage

```
replaceFileExtension(file, extension)
```

Arguments

file	file name or full path to a file
extension	new file name extension (including the dot, e.g. ".txt")

Examples

```
replaceFileExtension(file = "run.bat", ".txt")
```

`reproducibleSample` *Reproducible Call to the Sample Function*

Description

This function calls `sample` and stores the values that are required to reproduce the exact same sampling in the attribute `random_seed` of the returned vector. When this attribute is passed to another call of this function, the values returned will be the same as in the first call.

Usage

```
reproducibleSample(..., FUN = sample, random_seed = NULL)
```

Arguments

`...`	arguments passed to `FUN`.
`FUN`	the sample function to be called. Default: `sample`.
`random_seed`	vector of integer as stored in `.Random.seed`.

Value

This function returns what `sample` returns with an attribute `random_seed` attached.

Examples

```
# Take a sample
x <- reproducibleSample(1:100, 10)
x

# The full seed vector is returned in the attribute "random_seed"
random_seed <- attr(x, "random_seed")

# Take a new sample, this time passing the seed vector
y <- reproducibleSample(1:100, 10, random_seed = random_seed)
y

# The values are identical to the values of the first sampling
identical(x, y)
```

resetRowNames *Reset row names to 1:n*

Description

Reset the row names of a data frame x to 1:nrow(x) by setting the `row.names` attribute to `NULL`.

Usage

```
resetRowNames(x)
```

Arguments

`x`	data frame or matrix

Examples

```
persons <- data.frame(id = c(1, 2, 3), name = c("Peter", "Paul", "Mary"))

persons.ordered <- persons[order(persons$name), ]

# Original row names
persons.ordered

# Reset row names
resetRowNames(persons.ordered)
```

resolve *Resolve String(s) Using a Dictionary*

Description

Resolve String(s) Using a Dictionary

Usage

```
resolve(x, ...)
```

Arguments

`x`	vector of character to be resolved or a list of which all elements will be resolved using itself as a "dictionary". A dictionary is a list of `key = value` pairs defining string replacements.
`...`	Unnamed arguments are treated as (further) dictionaries. These are merged first to one dictionary before merging further (named) `key = value` pairs.

Examples

```
file <- system.file("extdata", "dictionary.txt", package = "kwb.utils")

dictionary <- readDictionary(file)

# Resolve the dictionary
resolve(dictionary)

# Resolve the dictionary by setting an undefined placeholder
resolve(dictionary, extension = "pdf")

# Resolve a string
resolve("dir.project", dictionary)

# Set a placeholder "on-the-fly"
resolve("file.out", dictionary, extension = "pdf")

# Override a placeholder "on-the-fly"
resolve("dir.project", dictionary, project = "new_project")

# Resolve a vector of strings
resolve(c("dir.root", "dir.project"), dictionary, project = "vector")
```

---

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.
...	additional assignments of the form <key> = <value> that are temporarily added to the dictionary before doing the resolving

Examples

```
# Define a dictionary in the form of a list
dictionary <- list(
  basedir = "C:/myNicefolder",
  projectdir = "<basedir>/projects/<project_name>",
  inputdir = "<projectdir>/input",
```

```
  outputdir = "<projectdir>/output"
)

# Resolve all entries in the dictionary, with different values for the
# placeholder "<project_name> which is undefined in the original dictionary
dictionary.1 <- resolveAll(dictionary, project_name = "project_1")
dictionary.2 <- resolveAll(dictionary, project_name = "project_2")

# Define entries of the dictionary to resolve
keys <- c("inputdir", "outputdir")

# Resolve the entries using the two different dictionaries
resolve(keys, dictionary.1)
resolve(keys, dictionary.2)
```

`revertListAssignments` *Revert List Assignments*

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

Examples

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

revertListAssignments(abbreviation)

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

right *Right Part of a String*

Description

Right Part of a String

Usage

```
right(x, n)
```

Arguments

x	vector of character
n	number of characters to be kept from the end of each character string within x

Value

vector of character

Examples

```
right("Good Morning", 7)
```

roundColumns *Round Columns to given Number of Digits*

Description

Round Columns to given Number of Digits

Usage

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

Arguments

dframe	data frame containing numeric columns to be rounded
columns	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: <column_name> = <n_digits>. If you give a list here, then there is no need to set the argument columnNames.
pattern	regular expression matching the names of the columns to be rounded. Will only be evaluated if no explicit column names are given in columnNames.
columnNames	deprecated. Use argument columns instead.

Value

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

---

rowOrColumnwisePercentage

Rowwise or Columnwise Percentage

Description

Calculate the percentage (value divided by sum of values in the row/column) for each row/column

Usage

```
rowOrColumnwisePercentage(x, rowwise, default = 0, digits = 1)
```

Arguments

x	two dimensional numeric data structure
rowwise	if TRUE the percentage is calculated by row, else by column
default	default value to be used if the calculated percentage is NA.
digits	number of digits (default: 1) to which the resulting percentages are to be rounded. Set to NA to suppress rounding

See Also

rowwisePercentage, columnwisePercentage

rowwisePercentage *Rowwise Percentage*

Description

Calculate the percentage (value divided by sum of values in the row) for each row

Usage

```
rowwisePercentage(x, default = 0, digits = 1)
```

Arguments

x	two dimensional numeric data structure
default	default value to be used if the calculated percentage is NA.
digits	number of digits (default: 1) to which the resulting percentages are to be rounded. Set to NA to suppress rounding

Examples

```
# Create a random matrix of integer values
M1 <- matrix(sample(100, 12), nrow = 4, dimnames = list(LETTERS[1:4], 1:3))

# Introduce some NA
values <- as.numeric(M1)
values[sample(length(values), 3)] <- NA
M2 <- matrix(values, nrow = nrow(M1), dimnames = dimnames(M1))

M1
rowwisePercentage(M1)

M2
rowwisePercentage(M2)
rowwisePercentage(M2, default = 0)
```

rStylePath *R compatible file path*

Description

R compatible file path with backslashes replaced with forward slashes

Usage

```
rStylePath(path)
```

Arguments

path	character string representing a file path

Value

path in which backslashes are replaced with forward slashes

---

runBatchfileInDirectory *Run a Batch File in a given Directory*

Description

Run a Batch File in a given Directory

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

---

runInDirectory *Change Working Directory and Run Function*

Description

Change Working Directory and Run Function

Usage

```
runInDirectory(target.dir = tempdir(), FUN, ..., .dbg = FALSE)
```

Arguments

target.dir	target directory. Default: tempdir()
FUN	function to be invoked
...	arguments to be passed to function FUN
.dbg	if TRUE, debug messages on changing the directory are shown. Default: FALSE

safeColumnBind *"Safe" version of cbind.*

Description

If x1 is NULL x2 is returned otherwise cbind(x1, x2)

Usage

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

Arguments

x1	first object to be passed to cbind
x2	second object to be passed to cbind

Value

result of cbind(x1, x2) or x2 if x1 is NULL.

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
```

safeMerge *Merge By Checking Column Existence*

Description

check existence of columns before merging

Usage

```
safeMerge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, ...)
```

Arguments

x	just as in merge
y	just as in merge
by	just as in merge
by.x	just as in merge
by.y	just as in merge
...	additional arguments passed to merge

---

safePath *Stop if Path does not Exist*

Description

Check if file or folder path exists and return the path or stop if the path does not exist

Usage

```
safePath(...)
```

Arguments

...	character vectors passed to file.path

Value

path as given in path

---

safeRowBind *"safe" rbind*

Description

rbind two data frames even if column names differ

Usage

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

Arguments

dataFrame1	first data frame
dataFrame2	second data frame

Examples

```
kwb.utils::safeRowBind(
  data.frame(A = 1:2, B = 2:3),
  data.frame(B = 3:4, C = 4:5)
)
```

---

safeRowBindAll *"safe" rbind of all data frames in a list*

Description

rbind all data frames in a list using safeRowBind

Usage

```
safeRowBindAll(x)
```

Arguments

x	list of data frames

Value

data frame resulting from "rbind"-ing all data frames in x

---

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.

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")
```

`selectColumns` *Select Columns from a Data Frame*

Description

Select columns from a data frame. Stop with message if columns do not exist

Usage

```
selectColumns(
  x,
  columns = NULL,
  pattern = NULL,
  drop = (length(columns) == 1),
  do.stop = TRUE
)
```

Arguments

`x`	data frame
`columns`	vector of column names. If `columns` is of length 0 or `NULL` (default) or `NA` `x` is returned unchanged.
`pattern`	regular expression matching the names of the columns to be selected. Will only be evaluated if no explicit column names are given in `columns`.
`drop`	if `TRUE` and if only one column is to be selected the result is a vector (one dimensional) containing the values of the selected column and not a data frame. One dimension has been *dropped* then. See the `help("[.data.frame")`. The default is `TRUE` if `length(columns) == 1`, else `FALSE`.

do.stop	this flag controls whether the function stops (do.stop = TRUE) or not (do.stop = FALSE) if there are non-existing columns to be selected. If do.stop = FALSE only those columns are selected that actually exist

Value

data frame containing the columns of x that are specified in columns or x itself if columns is NULL or a vector containing the values of column value if columns is of length 1 and drop = TRUE (which is the default in this case).

---

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 = NULL, 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.

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"))
```

setColumns *Set the column(s) of a data frame*

Description

Set the (new or existing) column(s) of a data frame.

Usage

```
setColumns(.x, ..., dbg = TRUE)
```

Arguments

.x	data frame
...	column assignment(s) in the form of <columnName> = <values>
dbg	if TRUE (default) the creation of new columns is reported on the screen

Value

data frame with columns modified or appended as specified with the assignments

Examples

```
# Create a data frame
x <- data.frame(a = 1:5)

# Option 1: use the "$" operator
x1 <- x
x1$b <- 2:6
x1$c <- 3:7

# Option 2: use setColumns
x2 <- setColumns(x, b = 2:6, c = 3:7)

# The result is the same
identical(x1, x2)

# but the creation of columns has been reported on the console (dbg = TRUE by
# default)

## Provide column 'b' to data frame 'x'... ok.
## Provide column 'c' to data frame 'x'... ok.
```

setLoaded *Set the "loaded" Status for a Script*

Description

You may use this function at the last line of your script to indicate that this script has already been loaded. This information is stored in the R option "kwb.utils.loaded" from which you may read the information back with isLoaded.

Usage

```
setLoaded(scriptName, isLoaded = TRUE)
```

Arguments

scriptName	name of the script for which we want to set the "loaded" state
isLoaded	logical. Use TRUE to indicate that the script scriptName has been loaded and use FALSE to indicate that the script is not loaded (e.g. because you cleared the workspace in the meanwhile).

See Also

isLoaded

Examples

```
# If you have a script with the main part on top and the required functions
# defined below (as recommended by Robert C. Martin, the author of "Clean
# Code") your script may look like this:

# Main part -----

# Check if the script has already been loaded (i.e. if setLoaded() has been
# called, see end of script). If yes, we can enter the main section. Otherwise
# we have to skip the main section since the function sayHello() is not yet
# defined.
if (isLoaded("welcome")) {
  sayHello(who = "Hauke")
}

# Functions -----
sayHello <- function(who) {
  clearConsole()
  cat("***\n***\n*** Hello", who, "how are you?\n***\n***\n")
}

# At the end of your script, call setLoaded() to indicate that your script is
# loaded now. If you "source" the script a second time, isLoaded("welcome")
# will return TRUE and thus the main section will be entered...
```

```
setLoaded("welcome")
```

`setMatrixColumns` *Set Matrix Columns to Values*

Description

Set matrix columns of given names to fix values

Usage

```
setMatrixColumns(m, columnValuePairs, warn = TRUE)
```

Arguments

`m`	matrix
`columnValuePairs`	list of elements each of which defines an assignment in the form `<column-name> = <value>`
`warn`	if `TRUE`, warnings are given if columns named in `columnValuePairs` do not exist in matrix `m`

`shorten` *Shorten Strings to a Maximum Length*

Description

Shorten Strings to a Maximum Length

Usage

```
shorten(x, max_chars = 10, delimiter = "...")
```

Arguments

`x`	vector of character
`max_chars`	maximum number of characters to which the strings in `x` are to be shortened
`delimiter`	string to be used as separater between the start and the end of the strings in `x` that are longer than `max_chars` characters

Value

`x` with strings longer than `max_chars` characters being shortend by replacing characters in the centre by the `delimiter` string.

showPackageObjects *Show Names of Objects/Functions in a Package*

Description

Show Names of Objects/Functions in a Package

Usage

```
showPackageObjects(package = "kwb.utils", show = TRUE)
```

Arguments

package	name of package of which to show the functions
show	if TRUE (the default) the function names are printed to the console otherwise the vector of function names is returned

sourceScripts *Load R Scripts with* source

Description

Load R Scripts with source

Usage

```
sourceScripts(scripts, dbg = TRUE)
```

Arguments

scripts	full paths to R scripts
dbg	if TRUE (default) log messages ("loading... ok") are shown

Value

the vector of script paths is returned invisibly

space *Space String Used for Indentation*

Description

Chain together `depth * tabLength` spaces

Usage

```
space(depth = 1L, tabLength = 2L)
```

Arguments

`depth`	depth of indentation
`tabLength`	number of spaces per indentation level

Value

vector of character of length one consisting of `depth * tabLength` space characters

Examples

```
cat(sprintf("%s1\n%s2\n%s3\n", space(1), space(2), space(3)))
cat(sprintf("%s1\n%s2\n%s3\n", space(1, 4), space(2, 4), space(3, 4)))
```

splitAlongDim *Split Array Along a Dimension*

Description

Split an array along its n-th dimension. The implementation was found here: https://stackoverflow.com/questions/20198751/tl
dimensional-array-to-list

Usage

```
splitAlongDim(a, n)
```

Arguments

`a`	an array
`n`	number of the dimension along which to split the array

Value

array of one dimension less than `a`

Examples

```
# Define an array
A <- array(1:8, dim = c(2, 2, 2), dimnames = list(
  paste0("x", 1:2), paste0("y", 1:2), paste0("z", 1:2)
))

splitAlongDim(A, 1)
splitAlongDim(A, 2)
splitAlongDim(A, 3)
```

---

splitIntoFixSizedBlocks *Split into Blocks of Same Size*

Description

Split a data frame or a matrix into blocks of the same size, i.e. data frames or matrices with the same number of rows (except the last one that is shorter unless the total number of rows is a multiple of the number of rows per block).

Usage

```
splitIntoFixSizedBlocks(data, blocksize)
```

Arguments

`data`	data frame or matrix
`blocksize`	number of rows in each block into which `data` is split

Value

list of data frames (if `data` is a data frame) or list of matrices (if `data` is a matrix)

---

splitIntoLines *Split Text at End of Line Characters Into Lines*

Description

Split Text at End of Line Characters Into Lines

Usage

```
splitIntoLines(x)
```

Arguments

x	vector of character of length 1.

Value

vector of character as long as there are lines in x

Examples

```
splitIntoLines("a\nb\nc")
splitIntoLines("a\r\nb\r\nc")
```

---

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

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`	integer vector of start indices
`lastStop`	integer value of the last stop index
`startOffset`	integer offset applied to the starts
`stopOffset`	integer offsets applied to the ends

Value

data frame with columns `from` and `to`

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
```

stopFormatted *Stop Function Exectution With Formatted Message*

Description

Stop Function Exectution With Formatted Message

Usage

```
stopFormatted(x, ..., call. = FALSE)
```

Arguments

x	Error message, possibly containing percentage placeholders, passed as argument fmt to sprintf
...	as many values as referenced with percentage placeholders in the error message
call.	logical, indicating if the call should become part of the error message.

Examples

```
try(stopFormatted(
  "Hi, %s, the program fails for the %d-th time.", "Dexter", 1000
))
```

stopIfNotMatrix *Stop with a Message if Input is not a Matrix*

Description

Stop with a Message if Input is not a Matrix

Usage

```
stopIfNotMatrix(x)
```

Arguments

x	object to be checked with is.matrix

stringContains *stringContains*

Description

stringContains

Usage

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

Arguments

x	vector of character
contains	vector of character

Examples

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

stringEndsWith *stringEndsWith*

Description

stringEndsWith

Usage

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

Arguments

x	vector of character to be checked if they end with endsWith
endsWith	string to be searched for at the end of the string(s) in x

Examples

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

stringList *String of Comma Separated Quoted Strings*

Description

Create a string of comma separated quoted strings

Usage

```
stringList(x, qchar = "'", collapse = ", ")
```

Arguments

x	vector of character
qchar	character to be used for quoting, default: single quote character
collapse	characters used to separate the strings. Default: ", "

stringStartsWith *stringStartsWith*

Description

stringStartsWith

Usage

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

Arguments

x	vector of character to be checked if they start with `startsWith`
startsWith	string to be searched for at the beginning of the string(s) in `x`

Examples

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

stringToExpression *Convert String to Expression*

Description

Convert String to Expression

Usage

```
stringToExpression(expressionString)
```

Arguments

expressionString	character string to be converted to an expression

subExpressionMatches *Find and Extract Regular Expressions from Strings*

Description

Find and Extract Regular Expressions from Strings

Usage

```
subExpressionMatches(
  pattern,
  text,
  match.names = NULL,
  select = stats::setNames(seq_along(match.names), match.names),
  simplify = TRUE,
  regularExpression = NULL
)
```

Arguments

pattern	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
regularExpression	deprecated. Use new argument pattern instead

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 *pattern*) 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.

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(
  pattern = "(\\\\d{4})\\\\-(\\\\d{2})\\\\-(\\\\d{2})", "2014-04-23",
  match.names = c("year", "month", "day")
)

cat(paste0("Today is ", x$day, "/", x$month, " of ", x$year, "\n"))
```

---

substSpecialChars *Substitution of Special Characters*

Description

Substitution of Special Characters

Usage

```
substSpecialChars(x, deuOnly = FALSE)
```

Arguments

`x`	string containing special characters to be substituted
`deuOnly`	logical. If `TRUE` only the German special characters Umlaute and Eszett are replaced.

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.

tableLookup *Lookup Value for Key in Table*

Description

lookup a value in the second column of a data frame x where the value in the first column matches the key.

Usage

```
tableLookup(x, key, default = NULL)
```

Arguments

x	data frame with at least two columns. Keys are expected to be in the first and values are expected to be in the second column, respectively.
key	key for which a value is to be looked up
default	default value that is returned if the key is not a key of the lookup table

Value

value looked up in the lookup table of default if key is not contained in the first column of the lookup table x

tempSubdirectory *Create and Return Path to Subdirectory in tempdir()*

Description

Create and Return Path to Subdirectory in tempdir()

Usage

```
tempSubdirectory(..., dbg = FALSE)
```

Arguments

...	parts of path to be created below tempdir()
dbg	if TRUE the directory creation is reported on. Default: FALSE

Value

full path to created directory

Examples

```
tempSubdirectory("my-folder/my-subfolder")
```

textToObject *Convert Text Representation Back to R Object*

Description

Convert Text Representation Back to R Object

Usage

```
textToObject(x)
```

Arguments

x	vector of character as returned by objectToText

Value

R object

Examples

```
textToObject("c(1:10)")
stopifnot(identical(iris, textToObject(objectToText(iris))))
```

toConditional *Convert Function to Function that is Called if Condition is Met*

Description

Convert Function to Function that is Called if Condition is Met

Usage

```
toConditional(FUN)
```

Arguments

FUN	a function

Examples

```
square <- function(x) x^2
negate <- function(x) -x
`%>%` <- magrittr::`%>%`
do_square <- TRUE
do_negate <- TRUE
10 %>%
  toConditional(square)(do_square) %>%
  toConditional(negate)(do_negate)
```

toFactor *Convert to Factor with unique Values as Levels*

Description

In contrast to the default behaviour of `base::as.factor`, this function uses the unsorted unique values of `x` as levels and not sorted unique values.

Usage

```
toFactor(x)
```

Arguments

`x`	vector to be converted to factor

Examples

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

as.factor(x) # Levels: a b c

toFactor(x) # Levels: b c a
```

toFormula *Create Formula from Left and Right Term Strings*

Description

create a formula of the form `leftterm ~ rightterms[1] + rightterms[2] + ...`

Usage

```
toFormula(leftterm, rightterms, as.formula = TRUE)
```

Arguments

`leftterm`	character. Left term of the formula
`rightterms`	vector of character. Right terms of the formula to be concatenated with "+".
`as.formula`	if TRUE (default) the formula object is returned, otherwise the formula string (character)

Value

formula object as generated by `formula` or formula string (`character`) of the form `leftterm ~ rightterms[1] + rightterms[2] + ...` if `as.formula` is FALSE

Examples

```
f1 <- toFormula("y", c("x1", "x2"))
f2 <- toFormula("y", paste0("x", 1:20))

f3 <- toFormula("BMI", c("height", "mass"), as.formula = FALSE)

# f1 and f2 are formulas ...
class(f1)
class(f2)

# ... but f3 is just "character"
class(f3)
```

---

toInches *Convert Centimeters to Inches*

Description

Convert Centimeters to Inches

Usage

```
toInches(cm)
```

Arguments

cm	vector of numeric representing length(s) in cm

Value

vector of numeric representing length(s) in inches

---

toKeysAndValues *Key Value String to List of Keys and Values*

Description

Converts a string of the form "a=1,b=2" to a list with elements `keys` (here: c("a", "b")) and `values` (here: (1,2)).

Usage

```
toKeysAndValues(x, separators = c(",", "="))
```

Arguments

x	character vector of length 1
separators	character vector of length 2 representing two types of separators. The first (default: ",") is used to split x into single key = value assignments. The second (default: "=") is used to split each assignment into key and value.

Value

list with elements keys and values

---

toLookupClass *Keys and Values to Lookup Structure*

Description

Provide the mapping between keys and values in a structure of choice

Usage

```
toLookupClass(
  keys,
  values,
  class = c("data.frame.1", "data.frame.2", "list", "vector")[1]
)
```

Arguments

keys	vector of keys
values	vector of values
class	character string determining the class of the structure returned: "data.frame.1": data frame with the keys as column names and the values in the first row; "data.frame.2": data frame with keys in the first and values in the second column; "list": list with values as elements and keys as element names; "vector": named vector with values as elements and keys as names.

Value

object according to the chosen class. See description of the class argument.

Examples

```
keys <- c("A", "B", "C")
values <- c("Apple", "Banana", "Cherry")

fruits.df1 <- toLookupClass(keys, values)
fruits.df2 <- toLookupClass(keys, values, class = "data.frame.2")
fruits.list <- toLookupClass(keys, values, class = "list")
```

```
fruits.vector <- toLookupClass(keys, values, class = "vector")

# Note how you may use the results differently

fruits.df1$A
fruits.list$A
fruits.vector["A"]

fruits.df1[c("A", "C")]
fruits.list[c("A", "C")]
fruits.vector[c("A", "C")]
```

`toLookupList` *Keys and Values to Lookup List*

Description

Keys and values to lookup list (list with elements representing the values and the names of the list elements representing the corresponding keys)

Usage

```
toLookupList(keys, values, data = NULL)
```

Arguments

`keys`	vector of character representing the keys of the dictionary
`values`	vector of character representing the values of the dictionary
`data`	Optional. Data frame with two columns of which the first is assumed to contain the keys and the second is assumed to contain the values of the dictionary

Value

dictionary: list with *values* as elements and *keys* as names

---

`toLookupTable` *Keys and Values or List to Lookup Table*

Description

Convert vectors of keys and values or a list into a lookup table (data frame)

Usage

```
toLookupTable(
  keys = NULL,
  values = NULL,
  List = NULL,
  as.twoColumnTable = FALSE,
  stringsAsFactors = FALSE
)
```

Arguments

keys	keys of the lookup table
values	values of the lookup table
List	list of named elements with the names representing the keys and the values representing the values of the lookup table
as.twoColumnTable	if TRUE (the default is FALSE) the result is a data frame with two columns *keys* and *values*, respectively.
stringsAsFactors	passed to data.frame

Value

data frame with one row containing values in columns named keys or, if as.twoColumnTable = TRUE, a data frame with the keys in column *key* and the values in column *value*

---

toNamedList *Convert to List with Names Equal to List Elements*

Description

Convert to List with Names Equal to List Elements

Usage

```
toNamedList(x)
```

Arguments

x	R object, preferably vector of character

Value

result of calling as.list on x with list element names being set to the elements of x.

Examples

```
x.vector <- c("Peter", "Paul", "Mary")
x.list <- toNamedList(x.vector)
all(names(x.list) == x.vector)

# You may use toNamedList in a call of lapply in order to get a named result
lapply(toNamedList(x.vector), function(x) sprintf("Hello, %s", x))

# Compare with
lapply(x.vector, function(x) sprintf("Hello, %s", x))
```

---

toPdf *Let Expressions Plot into a PDF File*

Description

The function opens a PDF device with pdf, executes the given expressions, closes the PDF file and displays the file in a PDF viewer.

Usage

```
toPdf(expressions, pdfFile = NULL, ...)
```

Arguments

expressions	R expressions creating plots that are to be redirected into a pdf file. You may pass multiple expressions within opening and closing curly braces
pdfFile	optional. Path to the PDF file to be created. The directory part of the path must exist. If not given or NULL, the PDF file is created in the tempdir() folder.
...	further arguments passed to pdf

Value

The function returns the path to the created PDF file.

Examples

```
## Not run:
toPdf({
  plot(1:10, 10:1)
  barplot(1:10)
  hist(rnorm(100))
})

## End(Not run)
```

toPositiveIndices *Negative Indices to Positive Indices*

Description

Convert negative indices to positive indices where negative indices are interpreted as indices counting from the back of the vector

Usage

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

Arguments

indices	vector of integer indices
n	maximum index. The index -1 is mapped to n, the index -2 to n - 1, etc.

underscoreToPercent *Replace underscore with percent sign*

Description

Replace underscore with percent sign. May be used to define time format strings as defaults in function declarations which are not supported by inlinedocs.

Usage

```
underscoreToPercent(x)
```

Arguments

x	character vector containing underscores

uniqueDirnames *Unique Directory Names*

Description

Unique Directory Names

Usage

```
uniqueDirnames(x)
```

Arguments

x	vector of file paths

Value

vector of character with the unique directories found in x

Examples

```
uniqueDirnames(c("a/b", "a/c", "b/c", "b/d"))
```

unmerge *Invert the Merging of two Data Frames*

Description

Split a data frame z into two data frames x and y so that merge(x, y) is z.

Usage

```
unmerge(z, by)
```

Arguments

z	data frame
by	vector of names of columns in z that are used to build groups of rows of z so that within each group the values in these columns do not change. For each group the columns being constant over all rows are identified. Columns that are constant in each group will appear in the data frame x whereas the remaining columns will appear in the data frame y of the returned list.

Value

list with two elements x and y each of which are data frames containing at least the columns given in by.

Examples

```
z <- data.frame(
  name = c("peter", "peter", "paul", "mary", "paul", "mary"),
  age = c(42, 42, 31, 28, 31, 28),
  height = c(181, 181, 178, 172, 178, 172),
  subject = c("maths", "bio", "bio", "bio", "chem", "maths"),
  year = c(2016, 2017, 2017, 2017, 2015, 2016),
  mark = c("A", "B", "B", "A", "C", "b")
)

# What fields seem to be properties of objects identified by name?
# -> Age and height are fix properties of the persons identified by name
(result1 <- unmerge(z, "name"))

# What fields seem to be properties of objects identified by subject?
# -> It seems that the subjects have been tested in different years
(result2 <- unmerge(z, "subject"))

# Test if merge(result$x, result$y) results in z
y1 <- merge(result1$x, result1$y)
y2 <- merge(result2$x, result2$y)

columns <- sort(names(z))

identical(fullySorted(z[, columns]), fullySorted(y1[, columns])) # TRUE
identical(fullySorted(z[, columns]), fullySorted(y2[, columns])) # TRUE
```

---

user *Name of the Current User*

Description

Get the name of the current user from the environment variables

Usage

```
user(osType = .OStype())
```

Arguments

osType	Optional. Type of operating system, one of "unix", "windows"

Value

character string represenging the name of the current user

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)

warningDeprecated *Create Warning About a Deprecated Function*

Description

Create Warning About a Deprecated Function

Usage

```
warningDeprecated(old_name, new_name, parentheses = TRUE)
```

Arguments

old_name	name of deprecated function
new_name	name of new function to be used instead
parentheses	logical. Should parentheses be printed after the names in the output? Default: TRUE.

Examples

```
## Not run:
warningDeprecated("old_function()", "new_function()")

## End(Not run)
```

windowsPath *convert to MS Windows-compatible path*

Description

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

Usage

```
windowsPath(path)
```

Arguments

path	vector of character representing file paths

writeDictionary *Write a Dictionary (List) to a Text File*

Description

Write a Dictionary (List) to a Text File

Usage

```
writeDictionary(dictionary, file)
```

Arguments

dictionary	list of character vectors of length one defining key = value pairs forming a dictionary.
file	full path to the file to which dictionary is to be written

See Also

readDictionary, readDictionaries

writeText *Write Text Lines to a File*

Description

Write text to a file using writeLines and output a debug message by default

Usage

```
writeText(x, file, type = "", dbg = TRUE, ...)
```

Arguments

x	vector of character representing the lines to be written to file, passed to writeLines
file	path to file to be written, passed to writeLines
type	character string to be included in the debug message: "Writing <type>'file-path' ..."
dbg	if TRUE, debug messages are shown
...	further arguments passed to writeLines

Value

This function invisibly returns the path to the output file.

Examples

```
# Define text to be written to file
x <- c("Hello", "world")

# Write text to a temporary file and catch the file path
file <- writeText(x, tempfile(fileext = ".txt"))

# Make the debug message more informative
writeText(x, file, type = "welcome file")

# Read lines back and show on the console
catLines(readLines(file))
```

Index