Package: kwb.prep (via r-universe)

August 26, 2024

Title Markdown-Documented Data Preparation

Version 0.3.0

Description R Package for Markdown-documented data preparation.

License MIT + file LICENSE

URL https://github.com/KWB-R/kwb.prep

BugReports https://github.com/KWB-R/kwb.prep/issues

Imports archive, data.table, dplyr, fs, knitr, kwb.utils, magrittr, methods, rmarkdown, yaml

Suggests covr, kwb.code, kwb.file, testthat

Remotes github::KWB-R/kwb.code, github::KWB-R/kwb.file, github::KWB-R/kwb.utils

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

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

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

RemoteRef HEAD

RemoteSha 38d3419f9b7a56cc1937d4af9f56669cf5a40c45

Contents

`applyFilter` *Apply Filter Criteria from List*

Description

Apply Filter Criteria from List

Usage

```
applyFilter(data, criteria_list, element, length_column = NULL)
```

Arguments

`data`	data frame
`criteria_list`	list of (named) vectors of character representing filter criteria
`element`	name of list element to be selected fom `criteria_list`
`length_column`	passed to `applyFilterCriteria`

Examples

```
criteria_list <- list(
  apple = c("is red or green" = "colour %in% c('red', 'green')"),
  banana = c("is not straight" = "! straight")
)

fruit_properties <- data.frame(
  colour = c("green", "red", "yellow"),
  straight = c(TRUE, TRUE, FALSE)
  )

applyFilter(fruit_properties, criteria_list, "apple")
applyFilter(fruit_properties, criteria_list, "banana")
```

`applyFilterCriteria` *Filter Rows from Data Frame Matching Criteria*

Description

Details about criteria applied and number of rows matching each criterion is returned in the attribute "details.filter". If a criterion evaluates to `NA`, the corresponding row in the data frame is removed (just as if the criterion evaluated to `FALSE`).

Usage

```
applyFilterCriteria(x, criteria = NULL, lengthColumn = NULL, ...)
```

Arguments

`x`	data frame
`criteria`	vector of character defining filter criteria to be evaluated in x
`lengthColumn`	name of the column containing lengths, e.g. "Length_raw"
`...`	passed to `matchesCriteria`

Examples

```
# Create a very simple data frame
df <- data.frame(value = 1:10, group = rep(c("a", "b"), 5))

# Show the data frame
df

# Filter for rows meeting two criteria
result <- applyFilterCriteria(df, c(
  "value is below or equal to 5" = "value <= 5",
  "group is 'a'" = "group == 'a'"
))

# Show the result
result

# Get the evaluation of each criterion in columns
kwb.utils::getAttribute(result, "matches")
```

apply_filters *Apply Groups of Filter Criteria from Configuration*

Description

Apply Groups of Filter Criteria from Configuration

Usage

```
apply_filters(data, groups, length_column = NULL, id_columns = names(data)[1L])
```

Arguments

data	data frame
groups	names of filter criteria groups defined in list returned by kwb.prep:::read_filter_criteria
length_column	name of column in data containing lengths (to be summed up for the overview that is returned)
id_columns	names of column(s) in data that uniquely identify the records. This column / these columns are returned in order to report about the records that have been removed

Value

data, filtered according to the specified criteria. The returned data frame has an attribute filter_info being a list with as many elements as there are groups. The elements are named according to the values given in groups. Each list element is a list with one element overview (being a data frame with one row per filter criterion) and further elements removed_<i> being data frames with only id_columns that represent the records that have been removed in the according filter step i.

Examples

```
# Define filter criteria
criteria <- list(
  sepal = c(
    "sepal short" = "Sepal.Length < 5",
    "sepal narrow" = "Sepal.Width < 3"
  ),
  petal = c(
    "petal short" = "Petal.Length < 5",
    "petal narrow" = "Petal.Width < 3"
  )
)

# Write criteria to temporary yaml file
tdir <- tempdir()
yaml::write_yaml(criteria, file.path(tdir, "filter_criteria.yml"))

# Set path to temporary "config" folder so that kwb.prep knows about it
```

```
kwb.prep:::set_user_config_dir(tdir)

# Apply filter groups "sepal" and "petal" to the iris dataset
result <- apply_filters(iris, c("sepal", "petal"))

# Have a look at the result
str(result)
```

`assign_objects` *Provide all Objects of kwb.prep in the Global Environment*

Description

Provide all Objects of kwb.prep in the Global Environment

Usage

```
assign_objects()
```

`breaksToIntervalLabels` *Create Interval Labels from Breaks Vector*

Description

Create labels for intervals defined by breaks in different possible styles

Usage

```
breaksToIntervalLabels(breaks, style = 5, ...)
```

Arguments

`breaks`	numeric vector of breaks
`style`	passed to `intervalLabel`
`...`	further arguments passed to `intervalLabel`

checkGrouping *Compare Two Columns of a Data Frame (Raw Vs Regrouped)*

Description

Compare Two Columns of a Data Frame (Raw Vs Regrouped)

Usage

```
checkGrouping(data, column_raw, column_cat)
```

Arguments

data	data frame
column_raw	name of column in data containing original (raw) values
column_cat	name of column in data containing the result of regrouping the raw value into categories

checkNumberOfUnique *Show Number of Unique Values in Selected Columns*

Description

Show Number of Unique Values in Selected Columns

Usage

```
checkNumberOfUnique(data, columns = names(data))
```

Arguments

data	data frame
columns	names of columns in data for which to count unique values

check_table_name *Check for Valid Table Name*

Description

Check if the argument can be used as a table name

Usage

```
check_table_name(table_name)
```

Arguments

table_name	R object to be checked for usage as a table name

Examples

```
try(check_table_name(c("more", "than", "one", "string")))
try(check_table_name("one_is_ok"))
```

check_zip_extension *Stop if File Name Does not End with Zip Extension*

Description

Stop if File Name Does not End with Zip Extension

Usage

```
check_zip_extension(file)
```

Arguments

file	path to file to check for .zip or .7z file name extension

Value

The function does not return anything but stops with a clear error message in case that `file` does not end with something that looks like the file extension of a compressed file.

collect *Collect Elements of Sublists*

Description

Collect Elements of Sublists

Usage

```
collect(x, element, default = NULL)
```

Arguments

x	a list of lists
element	name of list element to be collected from each sublist of x
default	value to be returned for lists that do not have an element called element.

Examples

```
x <- list(
  list(a = 1, b = 2),
  list(c = 3, a = 4),
  list(d = 5, e = 6)
)

collect(x, "a")
collect(x, "a", default = 99)
```

create_text_getter *Create a get_text() Function*

Description

Create a get_text() Function

Usage

```
create_text_getter(raw_strings = NULL, FUN = NULL)
```

Arguments

raw_strings	list of string definitions (key = value) pairs
FUN	function to be called to get the string definitions

Value

a function that can be used to lookup the string constant(s)

Examples

```
get_text <- create_text_getter(
  list(hello_en = "good morning", hello_de = "sch<oe>ne Gr<ue><ss>e")
)

get_text("hello_en")
get_text("hello_de")
#get_text("no_such_key") # error with clear error message
```

---

dataFramesToTextMatrix

Convert List of Data Frames to Character Matrix

Description

In the character matrix the data frames appear one below the other. Each data frame has a header and each data frame is separated from the following data frame by an empty row.

Usage

```
dataFramesToTextMatrix(data_frames)
```

Arguments

data_frames	list of data frames

Examples

```
data_frames <- list(
  data.frame(a = 1:3, b = 2:4),
  data.frame(a = 1:5, b = 2:6, c = 3:7)
)

dataFramesToTextMatrix(data_frames)
```

doRegroupings *Apply Regrouping of Values in a Data Frame*

Description

Apply Regrouping of Values in a Data Frame

Usage

```
doRegroupings(
  Data,
  regroup.actual = kwb.utils::selectElements(settings, "regroup.actual"),
  regroup.config = kwb.utils::selectElements(settings, "regroup.config"),
  settings = NULL,
  checkRemaining = TRUE,
  to.factor = FALSE,
  to.numeric = TRUE,
  dbg = TRUE
)
```

Arguments

Data	data frame
regroup.actual	default: settings$regroup.actual
regroup.config	default: settings$regroup.config
settings	list of settings that may contain the elements regroup.actual and regroup.config
checkRemaining	if TRUE (default) it is checked if all values that occurred in a column to be regrouped have been considered in the regrouping
to.factor	if TRUE the new values are converted to factor. The default is FALSE.
to.numeric	(default: TRUE, overrides to.factor!), passed to regroup
dbg	if TRUE (default) debug messages are shown

fieldSummary *Frequency of Value Combinations in Data Frame Columns*

Description

Frequency of Value Combinations in Data Frame Columns

Usage

```
fieldSummary(x, groupBy = names(x)[-1L], lengthColumn = "", na = "Unknown")
```

Arguments

x	data frame
groupBy	vector of character naming the columns (fields) in x to be included in the evaluation. Default: names of all columns in x except the first one (assuming it could be an ID column).
lengthColumn	optional. Name of column in x to be summed up
na	optional. Value to be treated as NA. Default: "Unknown"

Examples

```
n <- 1000L
sample_replace <- function(x, ...) sample(x, size = n, replace = TRUE, ...)
x <- data.frame(
  pipe_id = 1:n,
  material = sample_replace(c("clay", "concrete", "other")),
  age_cat = sample_replace(c("young", "old")),
  length = as.integer(rnorm(n, 50)),
  stringsAsFactors = FALSE
)

fieldSummary(x)
fieldSummary(x, "age_cat")
fieldSummary(x, "material")
fieldSummary(x, "material", lengthColumn = "length")
```

---

fillUpNA *Fill NA in First Vector With Values From Second Vector*

Description

Fill NA in First Vector With Values From Second Vector

Usage

```
fillUpNA(x, y, dbg = TRUE, name_x = NULL, name_y = NULL)
```

Arguments

x	first vector
y	second vector
dbg	if TRUE a debug message is shown
name_x	name of x
name_y	name of y

Value

x with NA replaced by the values in y at the same positions

find_string_constants *Show String Constants Used in R Scripts*

Description

Show String Constants Used in R Scripts

Usage

```
find_string_constants()
```

getChangesOfDuplicates *Get Changes of Rows That Are Duplicated in Selected Columns*

Description

Get Changes of Rows That Are Duplicated in Selected Columns

Usage

```
getChangesOfDuplicates(df, columns, add_columns = columns)
```

Arguments

df	a data frame
columns	names of columns in df in which to look for duplicate value combinations
add_columns	names of additional columns that shall appear in the output even if there are no changes in these columns

Value

list of data frames. The list has as many elements as there are different value combinations in columns that appear more than once in df. Each element is a data frame with all rows from df that have the same value combination in columns. By default the data frame contains the columns given in columns and those columns out of df in which there is at least one change over the values in the different rows.

Examples

```
df <- data.frame(
  id = 1:7,
  name = c("one", "one", "two", "two", "three", "three", "three"),
  type = c("A", "A", "B", "C", "D", "D", "D"),
  size = c(10, 11, 12, 12, 13, 13, 14),
  height = c(1, 1, 2, 3, 4, 4, 5)
)

df

getChangesOfDuplicates(df, "name")
getChangesOfDuplicates(df, c("name", "type"))
```

---

getYearFromColumn *Get Integer Year Number from Column*

Description

Get Integer Year Number from Column

Usage

```
getYearFromColumn(data, column)
```

Arguments

data	data frame
column	representing a date or date and time

Value

vector of integer as long as the number of rows in data

---

get_csv_filenames *Get Names of CSV Files Referenced in Config*

Description

Get Names of CSV Files Referenced in Config

Usage

```
get_csv_filenames(config, keep_empty = FALSE)
```

Arguments

config	configuration object (list) with one entry per "table", each of which is expected to have an entry "file"
keep_empty	logical. Whether or not to keep "file" entries that are empty ("")

Value

vector of character with the file names referenced in config

Examples

```
config <- list(
  table_a = list(file = "table-a.csv"),
  table_b = list(file = "table-b.csv")
)
get_csv_filenames(config)
```

---

get_lower_extension *Lower Case Extension of a File*

Description

Lower Case Extension of a File

Usage

```
get_lower_extension(file)
```

Arguments

file	file path or file name

Examples

```
get_lower_extension("abc.XYZ")
```

get_path *Resolve Path from Path Dictionary in Config Folder*

Description

Resolve Path from Path Dictionary in Config Folder

Usage

```
get_path(x = NULL, ...)
```

Arguments

x	key to be looked up in the path dictionary
...	possible key = value assignments to be used to replace \<placeholders\> in the path that was looked up

get_renamings *Get List Defining Renamings from Data Frame*

Description

Get list defining renamings in the form of `from = to` assignments from a data frame read by a function that may be specified.

Usage

```
get_renamings(from, to = "column", data = NULL, reader = read_csv_file, ...)
```

Arguments

from	name of column of data to take the "from" values from
to	name of column of data to take the "to" values from
data	data frame defining renamings
reader	reader function providing data. Default: kwb.prep:::read_csv_file
...	arguments passed to the reader function

Value

list defining renamings as e.g. expected by renameColumns

get_renamings_from_config *Get List of Renamings from Configuration*

Description

Get List of Renamings from Configuration

Usage

```
get_renamings_from_config(config, table_name, all = TRUE)
```

Arguments

config	list with one element per table/csv file
table_name	name of list element within config
all	if FALSE only the fields with property "required = TRUE" are considered

Value

list with original names as names and internal names as values. The list can be used in a call to renameColumns

get_selection *Get a Set of Column Names from a Data Frame Defining Selections*

Description

Get a Set of Column Names from a Data Frame Defining Selections

Usage

```
get_selection(
  number = 1,
  data = NULL,
  reader = read_csv_file,
  ...,
  column = paste0("select.", number),
  target = "column"
)
```

Arguments

number	number of the selection group, default: 1
data	data frame defining groups of columns
reader	reader function providing data. Default: kwb.prep:::read_csv_file
...	arguments passed to the reader function
column	name of column in data containing numbers to indicate which columns to select in which order
target	name of column in data containing the column names

Value

vector of column names

---

get_text *Get Text Constant*

Description

Get Text Constant

Usage

```
get_text(key = NULL, ..., raw_strings = get_raw_strings())
```

Arguments

key	identifier
...	additional arguments passed to sprintf
raw_strings	list with raw string definitions as key = value pairs

Value

if key is NULL) a list with all text constants or the text constant looked up for the given key

---

get_user_strings *Get List of User-Defined Text Constants*

Description

Get List of User-Defined Text Constants

Usage

```
get_user_strings()
```

get_zipped_paths *List Files in Zip Archive*

Description

List Files in Zip Archive

Usage

```
get_zipped_paths(zip_file, include_dirs = FALSE)
```

Arguments

zip_file	path to zip archive
include_dirs	if TRUE directory paths are also returned. The default is FALSE, i.e. only the paths to files are returned.

Value

paths to files contained in zip archive

groupByBreaks *Group Values Belonging to Intervals*

Description

Group values together that belong to the same intervals being defined by breaks

Usage

```
groupByBreaks(
  x,
  breaks,
  values = breaksToIntervalLabels(breaks),
  right = TRUE,
  add.Inf.limits = TRUE,
  to.factor = FALSE,
  columns = NULL,
  keyFields = NULL
)
```

Arguments

x	vector of values or a data frame. If x is a data frame, the function is applied to each column given in columns (all numeric columns by default)
breaks	vector of breaks
values	values to be assigned
right	if TRUE the intervals are right-closed, else left-closed.
add.Inf.limits	if TRUE (default), -Inf and Inf are added to the left and right, respectively, of breaks
to.factor	if TRUE the new values are converted to factor. The default is FALSE.
columns	NULL or vector of column names (if x is a data frame)
keyFields	NULL or vector of column names (if x is a data frame). If not NULL, a data frame with these columns coming first and the interval labels in the last column is returned.

Examples

```
groupByBreaks(1:10, breaks = 5, values = c("<= 5", "> 5"))
groupByBreaks(1:10, breaks = 5, right = FALSE, values = c("< 5", ">= 5"))

# Prepare a simple data frame
x <- kwb.utils::noFactorDataFrame(
  id = c("A", "B", "C"),
  value = c(10, 20, 30)
)

# Keep the ID column of the data frame
groupByBreaks(x, breaks = 20, keyFields = "id")
```

---

has_zip_extension *Does a File have a Zip Extension (.zip, .7z)?*

Description

Does a File have a Zip Extension (.zip, .7z)?

Usage

```
has_zip_extension(file, expected = c("zip", "7z"))
```

Arguments

file	path(s) to file(s) to be checked for zip extension
expected	expected file name extensions. Default: c("zip", "7z")

Value

vector of logical

Examples

```
all(has_zip_extension(c("a.zip", "b.ZIP", "c.Zip", "d.7z", "e.7Z"))) # TRUE
has_zip_extension("a.txt") # FALSE
```

---

import_db *Import CSV Files from ZIP File*

Description

The function stops with an error message if the `file` does not have the file extension ".zip" or if the zip file does not contain the expected csv files or if a csv file does not contain all expected fields (columns). Expected file names and field names are provided `config`). If everything looks ok, the csv files in the zip file are extracted into a (new) folder in the app's "run" directory. The app directory is provided in the environment variable SEMA_BERLIN_PREP_APP_DIR.

Usage

```
import_db(zip_file, config, base_name = basename(zip_file))
```

Arguments

`zip_file`	path to zip file containing csv files
`config`	configuration object (list) describing the csv files
`base_name`	base name of the folder to be created. The current date will also be encoded in the folder name. By default the base name of the zip file (file name without file extension) is used.

---

intervalLabel *Create Interval Label from Upper and Lower Boundary*

Description

Create a label for the interval defined by the upper boundary `a` and the lower boundary `b`

Usage

```
intervalLabel(a, b, right = TRUE, style = 1, sep = ",", space = " ")
```

Arguments

a	upper boundary
b	lower boundary
right	if TRUE (default) the interval is closed at the upper boundary
style	integer number between 1 and 5 indicating one of five possible styles to name the interval between a and b. See examples below.
sep	separator to be used between lower and upper boundary
space	space between comparison operators and boundary values.

Examples

```
# Labels of different styles for right closed intervals (right = TRUE is the
# default)
intervalLabel(1, 10, style = 1) # "(1,10]"
intervalLabel(1, 10, style = 2) # "<= 10"
intervalLabel(1, 10, style = 3) # "> 1"
intervalLabel(1, 10, style = 4) # "<= " "> 1" (vector of two elements!)
intervalLabel(1, 10, style = 5) # "<= 10" "> " (vector of two elements!)

# The same with left closed intervals:
right <- FALSE
intervalLabel(1, 10, right, style = 1) # "[1,10)"
intervalLabel(1, 10, right, style = 2) # "< 10"
intervalLabel(1, 10, right, style = 3) # ">= 1"
intervalLabel(1, 10, right, style = 4) # "< " ">= 1" (vector of two elements!)
intervalLabel(1, 10, right, style = 5) # "< 10" ">= " (vector of two elements!)
```

---

kable_no_rows *Print Data Frame as Markdown Table (Without Row Names by Default)*

Description

Print Data Frame as Markdown Table (Without Row Names by Default)

Usage

```
kable_no_rows(..., row.names = FALSE)
```

Arguments

...	passed to kable
row.names	passed to kable, default: FALSE

kable_translated *Rename Data Frame Columns and Print as Markdown*

Description

Rename Data Frame Columns and Print as Markdown

Usage

```
kable_translated(x, ...)
```

Arguments

x	x
...	passed to translate_columns

logicalToYesNo *Convert Vector of Logical to Vector of "Ja"/"Nein"*

Description

Convert Vector of Logical to Vector of "Ja"/"Nein"

Usage

```
logicalToYesNo(x, yesno = c("Ja", "Nein"))
```

Arguments

x	vector of logical
yesno	vector of character of length two giving the strings to be used for TRUE and FALSE, respectively

Value

vector of character

Examples

```
logicalToYesNo(c(TRUE, FALSE, TRUE))
logicalToYesNo(c(TRUE, FALSE, TRUE), yesno = c("Yeah!", "Oh no!"))
```

md_header *Print Markdown Section Header*

Description

Print Markdown Section Header

Usage

```
md_header(
  level,
  caption_key = "key?",
  caption = NULL,
  print = TRUE,
  msg = TRUE
)
```

Arguments

level	level
caption_key	caption_key
caption	caption
print	print
msg	msg

overwriteIfNotNA *Use Non-NA Values from Source Column in Target Column*

Description

Overwrite the values in the target column with the values in the source column at indices where the values in the source column are not NA

Usage

```
overwriteIfNotNA(data, target_column, source_column)
```

Arguments

data	data frame
target_column	name of target column
source_column	name of source column

print.data_frame_diff *Print Result of Data Frame Comparison*

Description

Print Result of Data Frame Comparison

Usage

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

Arguments

x	object of class "data_frame_diff"
...	currently not used

printNumberOfNA *Print Number of NA Values in Given Column*

Description

Print Number of NA Values in Given Column

Usage

```
printNumberOfNA(data, column, name = NULL)
```

Arguments

data	data frame
column	column name
name	name of data

printTableForColumn *Print Result of table() for Given Column*

Description

Print Result of table() for Given Column

Usage

```
printTableForColumn(data, column, name = NULL)
```

Arguments

data	data frame
column	column name
name	name of data

read_actual_regrouping *Read and Filter "regroup_actual.csv"*

Description

Read and Filter "regroup_actual.csv"

Usage

```
read_actual_regrouping(
  name_actual,
  group = NULL,
  columns = NULL,
  as_list = TRUE
)
```

Arguments

name_actual	Base name of file in config folder, default: "regroup_actual". The file specifies: which regroupings are arcually to be applied? What are the names of input and output columns?
group	Name of column in <name_actual>.csv containing non-empty fields for rows that are to be considered. NULL (the default) means that all rows of the file are considered.

columns	names of (input) columns that are to be regrouped. Only those regroupings are performed that work on these columns or columns that are created during the regrouping. By default columns = NULL all regroupings are used (unless group is given).
as_list	it TRUE (the default) the actual regrouping configuration is returned as a list (as required by doRegroupings), otherwise as a data frame.

---

read_csv_file *Read Data Frame From CSV File*

Description

Read Data Frame From CSV File

Usage

```
read_csv_file(
  file,
  sep = get_column_separator(),
  dec = ",",
  encoding = "UTF-8",
  na.strings = "",
  ...,
  remove_comments = TRUE,
  set_empty_string_to_na = FALSE,
  dbg = 1L
)
```

Arguments

file	path to csv file
sep	Column separator character. Default: semicolon ";"
dec	Decimal separator character. Default: comma ","
encoding	file encoding string. Default: "UTF-8". Possible other value: "unknown"
na.strings	strings occurring in the files representing NA (not available). Default: ""
...	further arguments passed to fread
remove_comments	Should rows starting with "#" be removed (the default)?
set_empty_string_to_na	if TRUE (the default is FALSE) empty strings in character columns are replaced with NA
dbg	if TRUE debug messages are shown

regroup *Assign Values to Groups of Values*

Description

Assign Values to Groups of Values

Usage

```
regroup(
  x,
  assignments,
  ignore.case = NULL,
  to.factor = FALSE,
  to.numeric = TRUE
)
```

Arguments

x	vector of values
assignments	list of assignments of the form \<key\> = \<values\> with \<values\> being a vector of elements to be looked up in x and to be replaced with \<key\> in the output vector
ignore.case	if TRUE the case is ignored when comparing values
to.factor	if TRUE the new values are converted to factor. The default is FALSE.
to.numeric	if TRUE (the default!) and independent of to.factor (!) the returned values are converted to numeric values if all assigned (even though string) values "look like" numeric values, such as "1", "2", "3.4", "5.67".

Value

vector with as many elements as there are elements in x. The vector contains \<key\> at positions where the elements in x appeared in the vector \<values\> of a \<key\> = \<values\> assignment of assignments

Examples

```
regroup(c("A", "B", "C", "D"), assignments = list(
  "AB" = c("A", "B"),
  "CD" = c("C", "D")
))


x <- c("A", "B", "C", "D", "E", "A")
assignments <- list(
 "1" = c("A", "B"),
 "2" = c("C", "D")
```

```
)

regroup(x, assignments)

# to.factor is ignored...
regroup(x, assignments, to.factor = TRUE)

# ... unless to.numeric is FALSE!
regroup(x, assignments, to.factor = TRUE, to.numeric = FALSE)
```

`regroupedValues` *Regroup Values According to Configuration*

Description

Regroup Values According to Configuration

Usage

```
regroupedValues(
  values,
  config = NULL,
  labels = "labels1",
  to.factor = FALSE,
  to.numeric = TRUE,
  dbg = TRUE
)
```

Arguments

`values`	vector of values
`config`	configuration (list) describing how to regroup. If the list contains an element `breaks` the function `groupByBreaks` is called to group values together that belong to the same intervals that are defined by the breaks. Otherwise the list must contain an element `values` and an element of the name given in `labels` (default: "labels1"). These are given to the function `regroup` that performs a "value to label"-regrouping.
`labels`	default: "labels1"
`to.factor`	if `TRUE` the new values are converted to `factor`. The default is `FALSE`.
`to.numeric`	(default: `TRUE`, overrides `to.factor`!), passed to `regroup`
`dbg`	if `TRUE` (default) debug messages are shown

regrouping_is_used *Which actual regroupings would be used?*

Description

Which of the actual regroupings would be used if `columns` were available in a data frame

Usage

```
regrouping_is_used(columns, actuals)
```

Arguments

`columns`	vector of column names for which to check if they are subject to regrouping
`actuals`	list of elements `from` and `to`, as returned by `kwb.prep:::read_actual_regrouping`

Value

vector of logical as long as `actuals`. Attribute `column`: which columns would the data frame have after the regrouping?

removeRowsThatAreNaInColumn *Remove Rows That are NA in Given Column*

Description

Remove Rows That are NA in Given Column

Usage

```
removeRowsThatAreNaInColumn(data, column, dbg = TRUE)
```

Arguments

`data`	data frame
`column`	column name
`dbg`	it `TRUE` debug messages are shown

Value

`data` with rows removed that are `NA` in `data[[column]]`

Examples

```
df <- data.frame(a = c(1, NA, 3), b = c(11, 22, NA))
df

removeRowsThatAreNaInColumn(df, "a")
removeRowsThatAreNaInColumn(df, "b")
```

`replaceByCondition` *Replace Values in Column in Rows Matching Condition*

Description

Replace Values in Column in Rows Matching Condition

Usage

```
replaceByCondition(df, file = NULL, group = NULL, config = NULL, dbg = TRUE)
```

Arguments

`df`	data frame in which to do substitutions
`file`	path to CSV file with columns `group`, `target_column`, `condition`, `replacement`
`group`	group name. If given, only rows in `file` that have this group name in column `group` are considered.
`config`	optional. Data frame containing the configuration as being read from `file`.
`dbg`	if `TRUE` debug messages are shown

Examples

```
# Create a very simple data frame
df <- data.frame(a = 1:3)

# Create a very simple configuration
config <- read.table(sep = ",", header = TRUE, text = c(
  "group,target,condition,replacement",
  "g1,a,a==2,22",
  "g2,a,a==3,33"
))

# Write the configuration to a temporary file
file <- tempfile()
write.csv(config, file)

# Apply all replacements configured in the file ...
replaceByCondition(df, file)

# ... or in the configuration
```

```
replaceByCondition(df, config = config)

# Apply selected replacements
replaceByCondition(df, file, group = "g1")
replaceByCondition(df, file, group = "g2")
```

`replaceUnlessNA` *Use Elements of Substitute at Indices Where Substitutes Are Not NA*

Description

Use Elements of Substitute at Indices Where Substitutes Are Not NA

Usage

```
replaceUnlessNA(x, substitute)
```

Arguments

`x`	vector in which to substitute
`substitute`	vector containing substitutions

`reportNA` *Count NA in a Column and Give a Message about It*

Description

Count NA in a Column and Give a Message about It

Usage

```
reportNA(data, column, subject = "in data")
```

Arguments

`data`	data frame
`column`	name of column in `data`
`subject`	value for placeholder *subject* in output: "NAs *subject*: *count_NA*"

set_column *Set Column*

Description

Set Column

Usage

```
set_column(
  df,
  column,
  value = NULL,
  indices = NULL,
  from = NULL,
  must_exist = TRUE
)
```

Arguments

df	data frame
column	column
value	value
indices	row indices
from	name of source column, optional
must_exist	is column assumed to exist?

set_user_strings *Get List of User-Defined Text Constants*

Description

Get List of User-Defined Text Constants

Usage

```
set_user_strings(x)
```

Arguments

x	list of key = (character) value assignments

stopIfNotIn *Stop with Info If Element Is Not in Expected Set*

Description

Stop with info message if element is not in expected set of elements

Usage

```
stopIfNotIn(
  element,
  elements,
  singular = "option",
  plural = paste0(singular, "s"),
  do_stop = TRUE
)
```

Arguments

element	element to be looked for in elements
elements	vector of possible elements
singular	name of object to appear in error message. Default: "option"
plural	name of object (plural) to appear in error message. Default: paste0(singular, "s")
do_stop	if FALSE (the default is TRUE) program execution does not stop. Instead a message is shown.

stopOnDuplicates *Stop If There Are Duplicates over given Columns*

Description

Stop If There Are Duplicates over given Columns

Usage

```
stopOnDuplicates(data, columns = names(data), name = NULL)
```

Arguments

data	data frame
columns	names of columns over which to look for duplicates. By default, all columns in data are used.
name	name of data

stop_text *Stop with Error Message Looked Up by Keyword*

Description

Stop with Error Message Looked Up by Keyword

Usage

```
stop_text(...)
```

Arguments

...	arguments passed to get_text

unzip_archive *Unzip Archive*

Description

Unzip Archive

Usage

```
unzip_archive(zip_file, target_dir = tempdir(), flatten = TRUE, dbg = TRUE)
```

Arguments

zip_file	path to archive file
target_dir	path to target directory
flatten	if TRUE (the default) all files in the archive are unzipped directly into the target directory, independent from possible folder structures within the archive
dbg	whether or not to show debug messages

writeStandardCsv *Write CSV File in a Standardised Manner*

Description

Write CSV File in a Standardised Manner

Usage

```
writeStandardCsv(x, file, ...)
```

Arguments

x	data frame
file	path to CSV file to be written
...	additional arguments passed to write.table

write_filter_info *Write Information on Filtering to CSV files*

Description

Write Information on Filtering to CSV files

Usage

```
write_filter_info(x, target_dir, prefix = deparse(substitute(x)), dbg = TRUE)
```

Arguments

x	data frame as returned by apply_filters, with attribute filter_info set.
target_dir	path to directory into which to write csv files
prefix	string by which to prefix all files
dbg	whether or not to show debug messages

Value

x, unchanged, invisibly

`write_markdown_chapter` *Write a Markdown Chapter*

Description

Write a Markdown Chapter

Usage

```
write_markdown_chapter(x, caption_key = "key?", level = 3L, caption = NULL)
```

Arguments

`x`	x
`caption_key`	caption_key
`level`	level
`caption`	caption

Index