Package 'kutils'

May 8, 2026

Type Package

Title Project Management Tools

Version 1.73

Date 2023-09-17

Encoding UTF-8

Maintainer Paul Johnson <pauljohn@ku.edu>

Description Tools for data importation, recoding, and inspection.
There are functions to create new project
folders, R code templates, create uniquely named output
directories, and to quickly obtain a visual summary for each
variable in a data frame. The main feature here is the systematic
implementation of the ``variable key'' framework for data
importation and recoding. We are eager to have community feedback
about the variable key and the vignette about it. In version 1.7,
the function 'semTable' is removed. It was deprecated since 1.67.
That is provided in a separate package, 'semTable'.

License GPL-2

Depends R (>= 3.3.0)

Imports stats, utils, methods, foreign, xtable, plyr, openxlsx, RUnit

Suggests rockchalk

RoxygenNote 7.2.3

LazyData TRUE

NeedsCompilation no

Author Paul Johnson [aut, cre],
Benjamin Kite [aut],
Charles Redmon [aut],
Jared Harpole [ctb],
Kenna Whitley [ctb],
Po-Yi Chen [ctb],
Shadi Pirhosseinloo [ctb]

Repository CRAN

Date/Publication 2023-09-17 15:50:02 UTC

Contents

`all.equal.key` *An all.equal method for variable wide keys*

Description

Disregards attributes by defaults. Before comparing the two keys, the values are sorted by `"name_new"`).

Usage

```
## S3 method for class 'key'
all.equal(target, current, ..., check.attributes = FALSE)
```

Arguments

`target`	A wide variable key
`current`	A wide variable key
`...`	Other arguments that are ignored
`check.attributes`	Default FALSE

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

all.equal.keylong *An all.equal method for variable long keys*

Description

Disregards attributes by defaults. Before comparing the two keys, the values are sorted by "name_new").

Usage

```
## S3 method for class 'keylong'
all.equal(target, current, ..., check.attributes = FALSE)
```

Arguments

target	A long variable key
current	A long variable key
...	Other arguments that are ignored
check.attributes	Default FALSE

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

alphaOnly *Keep only alpha-numeric symbols*

Description

From a text string, keep ASCII letters, numbers, as well as "'", " ", "_" "(", ")", "-", and "+". For maximum compatability with the cross-platform file-naming standard. Obliterates all characters that migth be mistaken for shell symbols, like "^", "$", "@" and so forth.

Usage

```
alphaOnly(x, also)
```

Arguments

x	text string, or vector of strings (each of which is processed separately)
also	A named vector of other symbols that the user wants to remove, along with replacements. For example, c(" " = "_", "-" = "", "+" = "") to replace space with underscore and minus and plus signs with nothing.

Details

Removes trailing spaces.

This version allows internal spaces in the string, by default. The also argument can be used to eliminate spaces or other hated symbols.

Value

cleaned text string

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c("[]kansas(city) Missouri", "percent%slash/",
       "\back{squiggle}_under(paren)", "*star-minus+plus")
alphaOnly(x)
alphaOnly(x, also = c(" " = "_", "+" = "_"))
alphaOnly(x, also = c("(" = "[", ")" = "]"))
```

---

anonomize *Create unique anonymous id values*

Description

Obscure participant id values by replacing them with "anon-1" and so forth.

Usage

```
anonomize(x, prefix = "anon")
```

Arguments

x	A column of "confidential" names, possibly with repeats
prefix	Character string to use as prefix in result. Default is "anon"

Details

Caution: the true "confidential" names are used as names in the output vector

Value

Named character vector of anonymized id names.

Author(s)

Paul Johnson <pauljohn@ku.edu> x <- c("bill", "bob", "fred", "bill") (anonomize(x, prefix = "id"))

`assignMissing` *Set missing values*

Description

The missings values have to be carefully written, depending on the type of variable that is being processed.

Usage

```
assignMissing(x, missings = NULL, sep = ";")
```

Arguments

`x`	A variable
`missings`	A string vector of semi-colon separated values, ranges, and/or inequalities. For strings and factors, only an enumeration of values (or factor levels) to be excluded is allowed. For numeric variables (integers or floating point variables), one can specify open and double-sided intervals as well as particular values to be marked as missing. One can append particular values and ranges by "1;2;3;(8,10);[22,24];> 99;< 2". The double-sided interval is represented in the usual mathematical way, where hard bracketes indicate "closed" intervals and parentheses indicate open intervals. 1. "(a,b)" means values of x greater than a and smaller than b will be set as missing. 2. "[a,b]" is a closed interval, one which includes the endpoints, so a <= x <= b will be set as NA 3. "(a,b]" and "[a,b)" are acceptable. 4. "< a" indicates all values smaller than a will be missing 5. "<= a" means values smaller than or equal to a will be excluded 6. "> a" and ">= a" have comparable interpretations. 7. "8;9;10" Mark off specific values by an enumeration. Be aware, however, that this is useful only for integer variables. As demonstrated in the example, for floating point numbers, one must specify intervals. 8. For factors and character variables, the argument missings can be written either as "lo;med;hi" or "c('lo','med','hi')"
`sep`	A separator symbol, ";" (semicolon) by default

Details

Version 0.95 of kutils introduced a new style for specification of missing values.

Value

A cleaned column in which R's NA symbol replaces values that should be missing

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
## 1.  Integers.
x <- seq.int(-2L, 22L, by = 2L)
## Exclude scores 8, 10, 18
assignMissing(x, "8;10;18")
## Specify range, 4 to 12 inclusive
missings <- "[4,12]"
assignMissing(x, missings)
## Not inclusive
assignMissing(x,  "(4,12)")
## Set missing for any value smaller that 7
assignMissing(x, "< 7")
assignMissing(x, "<= 8")
assignMissing(x, "> 11")
assignMissing(x, "< -1;2;4;(7, 9);> 20")


## 2. strings
x <- c("low", "low", "med", "high")
missings <- "low;high"
assignMissing(x, missings)
missings <- "med;doesnot exist"
assignMissing(x, missings)
## Test alternate separator
assignMissing(x, "low|med", sep = "|")

## 3. factors (same as strings, really)
x <- factor(c("low", "low", "med", "high"), levels = c("low", "med", "high"))
missings <- "low;high"
assignMissing(x, missings)
## Previous same as
missings <- c("low", "high")
assignMissing(x, missings)

missings <- c("med", "doesnot exist")
assignMissing(x, missings)
## ordered factor:
x <- ordered(c("low", "low", "med", "high"), levels = c("low", "med", "high"))
missings <- c("low", "high")
assignMissing(x, missings)

## 4. Real-valued variable
set.seed(234234)
x <- rnorm(10)
x
missings <- "< 0"
assignMissing(x, missings)
missings <- "> -0.2"
```

```
assignMissing(x, missings)
## values above 0.1 and below 0.7 are missing
missings <- "(0.1,0.7)"
assignMissing(x, missings)
## Note that in floating point numbers, it is probably
## futile to specify specific values for missings. Even if we
## type out values to 7 decimals, nothing gets excluded
assignMissing(x, "-0.4879708;0.1435791")
## Can mark a range, however
assignMissing(x, "(-0.487971,-0.487970);(0.14357, 0.14358)")
x
```

`assignRecode` *A variable is transformed in an indicated way*

Description

In the variable key framework, the user might request transformations such as the logarithm, exponential, or square root. This is done by including strings in the recodes column, such as "log(x + 1)" or "3 + 1.1 * x + 0.5 * x ^ 2". This function implements the user's request by parsing the character string and applying the indicated re-calculation.

Usage

```
assignRecode(x, recode = NULL)
```

Arguments

`x`	A column to be recoded
`recode`	A character string using placeholder "x". See examples

Details

In the variable key framework, this is applied to the raw data, after missings are imposed.

Value

A new column

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
set.seed(234234)
x <- rpois(100, lambda = 3)
x <- x[order(x)]
str1 <- "log(x + 1)"
xlog <- assignRecode(x, recode = str1)
plot(xlog ~ x, type = "l")
mean(xlog, na.rm = TRUE)
str2 <- "x^2"
xsq <- assignRecode(x, recode = str2)
plot(xsq ~ x, type = "l")
str3 <- "sqrt(x)"
xsrt <- assignRecode(x, recode = str3)
```

`checkCoercion` *Check if values can be safely coerced without introduction of missing values*

Description

This might be named "coercesSafely" or such. If values cannot be coerced into class specified, then values must be incorrect.

Usage

```
checkCoercion(value, targetclass, na.strings = c("\\.", "", "\\s+", "N/A"))
```

Arguments

`value`	Character vector of values, such as value_new or value_old for one variable in a key.
`targetclass`	R class name
`na.strings`	Values that should be interpreted as R NA. These are ignored in the coercion check.

Value

either TRUE, or a vector of values which are not successfully coerced

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x1 <- c("TRUE", "FALSE", FALSE, TRUE, NA, ".", "N/A", " ", "")
checkCoercion(x1, "logical")
x1 <- c(x1, "TRUE.FALSE", "Has a space")
## Should fail:
checkCoercion(x1, "logical")
x2 <- c(4, 5, 6, 9.2, ".", " ")
## Should fail
checkCoercion(x2, "logical")
x3 <- factor(c("bob", "emily", "bob", "jane", "N/A", " ", NA, "NA"))
checkCoercion(x3, "ordered")
checkCoercion(x3, "integer")
## Should fail:
checkCoercion(x3, "logical")
```

---

colnamesReplace *Replace column names with new names from a named vector*

Description

A convenience function to alter column names. Can be called from code cleanup in the variable key system.

Usage

```
colnamesReplace(
  dat,
  newnames,
  oldnames = NULL,
  ...,
  lowercase = FALSE,
  verbose = FALSE
)
```

Arguments

dat	a data frame
newnames	Can be a named vector of the form c(oldname1 = "newname1", oldname2 = "newname") or it may be simply c("newname1", "newname2") to correspond with the oldname vector.
oldnames	Optional. If supplied, must be same length as newnames.
...	Additional arguments that will be passed to R's gsub function, which is used term-by-term inside this function.
lowercase	Default FALSE. Should all column names be converted to lower case.
verbose	Default FALSE. Want diagnostic output about column name changes?

Value

a data frame

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
set.seed(234234)
N <- 200
mydf <- data.frame(x5 = rnorm(N), x4 = rnorm(N), x3 = rnorm(N),
                   x2 = letters[sample(1:24, 200, replace = TRUE)],
                   x1 = factor(sample(c("cindy", "bobby", "marsha",
                                        "greg", "chris"), 200, replace = TRUE)),
                   x11 = 7,
                   x12 = 18,
                   x13 = 33,
                   stringsAsFactors = FALSE)
mydf2 <- colnamesReplace(mydf, newnames = c("x4" = "GLOPPY"))
mydf2 <- colnamesReplace(mydf, newnames = c("x4" = "GLOPPY", "USA" = "Interesting"), verbose = TRUE)
colnames(mydf2)
head(mydf3 <- colnamesReplace(mydf, newnames = c(x11 = "x12", x12 = "x13", x13 = "x20")))
head(mydf4 <- colnamesReplace(mydf, newnames = c(x12 = "x11", x11 = "x99", x13 = "x20")))
```

---

deduper *Removes redundant words from beginnings of character strings*

Description

In Qualtrix data, we sometimes find repeated words in column names. For whatever reason, the variable names have repeated words like "Philadelphia_Philadelphia_3". This function changes a vector c("Philadelphia_Philadelphia_3", "Denver_Denver_4") to c("Philadelphia_3", "Denver_4"). It is non destructive, so that other values will not be altered.

Usage

```
deduper(x, sep = ",_\\s-", n = NULL)
```

Arguments

x	Character vector
sep	Delimiter. A regular expression indicating the point at which to split the strings before checking for duplicates. Default will look for repeat separated by comma, underscore, or one space character.
n	Limit on number of duplicates to remove. Default, NULL, means delete all duplicates at the beginning of a string.

Details

See https://stackoverflow.com/questions/43711240/r-regular-expression-match-omit-several-repeats

Value

Cleaned up vector.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c("Philadelphia_Philadelphia_3", "Denver_Denver_4",
        "Den_Den_Den_Den_Den_Den_Den_5")
deduper(x)
deduper(x, n = 2)
deduper(x, n = 3)
deduper(x, n = 4)
x <- c("Philadelphia,Philadelphia_3", "Denver Denver_4")
## Shows comma also detected by default
deduper(x)
## Works even if delimiter is inside matched string,
## or separators vary
 x <- c("Den_5_Den_5_Den_5,Den_5 Den_5")
deduper(x)
## generate vector
x <- replicate(10, paste(sample(letters, 5), collapse = ""))
n <- c(paste0("_", sample(1:10, 5)), rep("", 5))
x <- paste0(x, "_", x, n, n)
x
deduper(x)
```

---

deleteBogusColumns *Remove columns in which the proportion of missing data exceeds a threshold.*

Description

This is a column version of `deleteBogusRows`. Use the pm argument to set the proportion of missing required before a column is flagged for deletion

Usage

```
deleteBogusColumns(dframe, pm = 0.9, drop = FALSE, verbose = TRUE, n = 25)
```

Arguments

dframe	A data frame or matrix
pm	"proportion missing data" to be tolerated.
drop	Default FALSE: if data frame result is reduced to one column, should R's default drop behavior "demote" this to a column vector.
verbose	Default TRUE. Should a report be printed summarizing information to be delted?
n	Default 25: limit on number of values to print in diagnostic output. If set to NULL or NA, then all of the column values will be printed for the bogus rows.

Value

a data frame, invisibly

Author(s)

Paul Johnson <pauljohn@ku.edu>

See Also

deleteBogusRows

---

deleteBogusRows *Remove rows in which the proportion of missing data exceeds a threshold.*

Description

If cases are mostly missing, delete them. It often happens that when data is imported from other sources, some noise rows exist at the bottom of the input. Anything that is missing in more than, say, 90% of cases is probably useless information. We invented this to deal with problem that MS Excel users often include a marginal note at the bottom of a spread sheet.

Usage

```
deleteBogusRows(dframe, pm = 0.9, drop = FALSE, verbose = TRUE, n = 25)
```

Arguments

dframe	A data frame or matrix
pm	"proportion missing data" to be tolerated.
drop	Default FALSE: if data frame result is reduced to one row, should R's default drop behavior "demote" this to a column vector.
verbose	Default TRUE. Should a report be printed summarizing information to be delted?
n	Default 25: limit on number of values to print in verbose diagnostic output. If set to NULL or NA, then all of the column values will be printed for the bogus rows.

Value

a data frame, invisibly

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
mymat <- matrix(rnorm(10*100), nrow = 10, ncol = 100,
                dimnames = list(1:10, paste0("x", 1:100)))
mymat <- rbind(mymat, c(32, rep(NA, 99)))
mymat2 <- deleteBogusRows(mymat)
mydf <- as.data.frame(mymat)
mydf$someFactor <- factor(sample(c("A", "B"), size = NROW(mydf), replace = TRUE))
mydf2 <- deleteBogusRows(mydf, n = "all")
```

---

dev.create *Create a graphics device*

Description

This is a way to create a graphic device on screen that can display R plots. It is performing the same purpose as R's dev.new, but it overcomes the limitations of RStudio. It is needed because RStudio does not implement fully the functionality of dev.new. This is suitable for Windows, Linux, and Macintosh operating systems.

Usage

```
dev.create(...)
```

Arguments

...	Currently, height and width parameters that would be suitable with dev.new

Details

The argument in dev.new named noRStudioGD seems to be aimed at same purpose. But it does not do what I want and documentation is too sparse.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
if(interactive()) dev.create(height = 7, width = 3)
dev.off()
```

`dir.create.unique` *Create a uniquely named directory. Appends number & optionally date to directory name.*

Description

Checks if the requested directory exists. If so, will create new directory name. My favorite method is to have the target directory with a date-based subdirectory, but set usedate as FALSE if you don't like that. Arguments showWarnings, recursive, and mode are passed along to R's dir.create, which does the actual work here.

Usage

```
dir.create.unique(
  path,
  usedate = TRUE,
  showWarnings = TRUE,
  recursive = TRUE,
  mode = "0777"
)
```

Arguments

`path`	A character string for the base name of the directory.
`usedate`	TRUE or FALSE: Insert YYYYMMDD information?
`showWarnings`	default TRUE. Show warnings? Will be passed on to dir.create
`recursive`	default TRUE. Will be passed on to dir.create
`mode`	Default permissions on unix-alike systems. Will be passed on to dir.create

Details

Default response to dir = "../output/" fixes the directory name like this, "../output/20151118-1/" because usedate is assumed TRUE. If usedate = FALSE, then output names will be like "../output-1/", "../output-2/", and so forth.

Value

a character string with the directory name

Author(s)

Paul E Johnson <pauljohn@ku.edu>

dms *Delete multiple slashes, replace with one*

Description

Sometimes paths end up with "/too//many//slashes". While harmless, this is untidy. Clean it up.

Usage

```
dms(name)
```

Arguments

name	A character string to clean

Author(s)

Paul Johnson <pauljohn@ku.edu>

dts *Delete trailing slash*

Description

This function cleans up a path string by removing the trailing slash. This is necessary on MS Windows, `file.exists(fn)` fails if "/" is on end of file name. Deleting the trailing slash is thus required on Windows and it is not harmful on other platforms.

Usage

```
dts(name)
```

Arguments

name	A path

Details

All usages of `file.exists(fn)` in R should be revised to be multi-platform safe by writing `file.exists(dts(fn))`.

This version also removes repeated adjacent slashes, so that `"/tmp///paul//test/"` becomes `"/tmp/paul/test"`.

Value

Same path with trailing "/" removed.

Author(s)

Paul Johnson <pauljohn@ku.edu>

`escape` *Text that is to be included as content in documents is cleaned (escaped) to prevent errors*

Description

This is for fixing up "untrusted text" that is to be passed into a file as content. It protects against "bad" text strings in 3 contexts, 1) LaTeX documents, 2) HTML documents, or 3) text in a file name. It converts content text to an improved string that will not cause failures in the eventual document.

Usage

```
escape(x, type = "tex")
```

Arguments

`x`	a string, or vector of strings (each of which is processed separately)
`type`	"tex" is default, could be "filename" or "html"

Details

The special in-document LaTeX symbols like percent sign or dollar sign are " session, these will appear as double-backslashed symbols, while in a saved text file, there will only be the one desired slash.

If type = "html", we only clean up <, >, / and &, and quote characters. If document is in unicode, we don't need to do the gigantic set anymore.

If type = "filename", then symbols that are not allowed in file names, such as "\", "*", are replaced. Do not use this on a full path, since it will obliterate path separators.

Value

corrected character vector

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x1 <- c("_asdf&_&$", "asd adf asd_", "^ % & $asdf_")
escape(x1)
x2 <- c("a>b", "a<b", "a < c", 'Paul "pj" Johnson')
escape(x2, type = "tex")
escape(x2, type = "html")
escape(x2, type = "filename")
```

file.backup *Create a backup version of a file by renaming it.*

Description

Inserts the date-time of the most recent modification at the end of the file name, before the extension.

Usage

```
file.backup(name, fullpath = FALSE, keep.old = FALSE, verbose = FALSE)
```

Arguments

name	A character string for the name of the file.
fullpath	Return the full directory path to the file. Default FALSE, return only the file name.
keep.old	If FALSE (default), rename the file. Otherwise, keep old copy.
verbose	If TRUE, output warnings and list the files in the output directory when done.

Details

Return is the new file name that was created, using whatever path information was provided in the file's original name. However, the fullpath argument can be set to TRUE, so a path with the full directory name will be created and returned.

Value

The name of the newly created file.

Author(s)

Shadi Pirhosseinloo <shadi@ku.edu> Paul Johnson <pauljohn@ku.edu>

Examples

```
tdir <- tempdir()
owd <- getwd()

setwd(tdir)
system("touch test.1.txt")
system("touch test.2.txt")
system("touch test.3.txt")
system("touch test.4.txt")
system("touch test.5.txt")
## note: no extension next
system("touch test.6")
list.files()
file.backup("test.1.txt")
```

```
file.backup("test.2.txt", fullpath=TRUE)
list.files()
setwd(owd)
file.backup(file.path(tdir, "test.3.txt"))
## Next should be same path because input had a full path
file.backup(file.path(tdir, "test.4.txt"), fullpath=TRUE)
file.backup(file.path(tdir, "test.5.txt"), fullpath = TRUE, verbose = TRUE)
file.backup(file.path(tdir, "test.6"))
```

`importQualtrics` *Import Qualtrics survey files, apply clean column names*

Description

Defaults are based on most common format received from Qualtrics downloads to CSV or XLSX (MS Excel) formats. We assume that the file has the column names in row 1 and that 3 rows are skipped before the real data begins. If the parameter questrow is used, it designates a row that is interpreted as the survey questions themselves. Often, this is in row 2.

Usage

```
importQualtrics(
  file,
  namerow = 1,
  questionrow = 2,
  importidrow = 3,
  skip = 3,
  dropTEXT = TRUE,
  stringsAsFactors = FALSE
)
```

Arguments

`file`	file name (including path if in another directory) of a CSV or XLSX file from Qualtrics.
`namerow`	Row number for variable names. Default 1, the information to be used as column names (same as HEADER row in R's read.table function)
`questionrow`	Row number to be treated as the questions in the survey. Default is 2. If questions do not seem to be present in this row, there will be a warning.
`importidrow`	Row number to be treated as Qualtrics meta data. Default is 3. Many CSV created by Qualtrics will have row 3 with a character string such as `"{""ImportId"":""QID1303_4""}"`. If importids are not present in this row, there will be a warning.
`skip`	Number of rows that are meta data. Current Qualtrics CSV files will usually have 3 metadata rows, 1 = name, 2 = question, 3 = ImportId. This function will try to guess how many rows of metadata are present. skip should be at least as large as `max(namerow, questions, and importids)`

dropTEXT Default TRUE, columns ending in "_TEXT" are omitted.

stringsAsFactors
Default FALSE, same meaning as R's read.csv. Does not affect importation of Excel files.

Value

Data frame that has attribute "meta"

Author(s)

Paul Johnson <pauljohn@ku.edu>

---

initProject *Create project directories, initialize a git repo, create README.md ChangeLog, and R template file in R directory*

Description

This creates folders for the separate parts of a project. It tries to be clever about which directories are created and where they are placed. Please see details for 3 scenarios for which we have planned. If a directory already exists, it will not be damaged or re-created.

Usage

```
initProject(
  dir = NULL,
  ddir = "data",
  wdir = "workingdata",
  odir = "output",
  tdir = "tmp",
  ldir = "lit",
  writedir = "writeup",
  rdir = "R",
  ...,
  gitArgs = "--shared=group"
)
```

Arguments

dir Default NULL, otherwise a legal directory name to serve as the top level directory for this project

ddir Data directory, place where "read only" unadjusted data files are kept. Default is "data". If user sets it as NA or NULL, the directory will not be created.

wdir Working data directory, where recorded, revised, and cleaned data files are kept. Default is "workingdata".

odir	Output directory. Default is "output".
tdir	Temporary directory, where trash files can be kept for safe keeping. Default is "tmp".
ldir	Literature directory, where material about the project can be stored. Default is "lit".
writedir	The folder where the project writeup will be stored. Default is "writeup".
rdir	The name to be used for the R files. Defaults to "R".
...	A list of other directories that the user would like to create. For example, `adir = "admin"`, `cdir = "client_provided"`, `bdir = "codebooks"`, `sdir = "Stata"`, `mdir = "Mplus"`. These may be grouped in a named vector or list, if user convenience dictates.
gitArgs	This function tries to run "git init" and in our center we add "–shared=group" on a network file server. If that is undesirable in a user's context, put the argument gitArgs as "".

Details

If the dir argument is NULL, as default, then the current working directory will be the place where new directories and the git repository will be created. Assuming the current working directory's base name is not "R", then folders named "R", "data", and so forth will be created in the current working directory.

If one has a current R working directory with a basename "R" (suppose it is `"/tmp/whatever/R"`), and the user runs `initProject()`, something different happens. The function assumes we don't want to create subdirectories inside R. We don't want to end up with `"/tmp/whatever/R/R"`. We don't want `"/tmp/whatever/R/data"` either. Instead, it assumes we want the new directories created on same level as R, so it creates `"/tmp/whatever/data"`, `"/tmp/whatever/workingdata"`, and so forth. From within the R directory, these new directories are seen as `"../data"`, `"../workingdata"`, and so forth. That is, we should end up with directories and a git repo in `"/tmp/whatever"`.

If the `dir` argument is provided by the user, then that is used as the folder in which directories `"R"`, `"data"`, `"workingdate"`, and so forth are created. All materials are created in `dir`, no matter what the current working directory is named (even if it is `"R"`).

The examples demonstrate all three of these scenarios.

Value

Name of project top level directory. Leaves the R working directory unchanged.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
projdir1 <- file.path(tempdir(), "test1")
dir.create(projdir1, recursive = TRUE)
initProject(dir = projdir1)
list.files(projdir1, all.files = TRUE)
```

```
projdir2 <- file.path(tempdir(), "test2")
dir.create(projdir2, recursive = TRUE)
## demonstrate ability to create other directories
initProject(dir = projdir2, admin = "admin", clientfiles = "client")
list.files(projdir2, all.files = TRUE)
## demonstrate ability to nullify standard directories
projdir3 <- file.path(tempdir(), "test3")
dir.create(projdir3, recursive = TRUE)
initProject(projdir3, odir = NA, tdir = NA, writedir = NA)
list.files(projdir3, all.files = TRUE)
unlink(c("projdir1", "projdir2", "projdir3"), recursive = TRUE)
```

`is.data.frame.simple` *Check if a data frame is a simple collection of columns (no lists or matrices within)*

Description

Checks for the existence of dimensions within the data frame. Returns FALSE if any object within dframe has non-null dim value.

Usage

```
is.data.frame.simple(dframe)
```

Arguments

`dframe`	A data frame

Details

See: http://stackoverflow.com/questions/38902880/data-frame-in-which-elements-are-not-single-columns

Value

Boolean, TRUE or FALSE. An attribute "not_a_simple_column" is created, indicating which of the elements in the dframe have dimensions

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
N <- 100
mydf <- data.frame(x5 = rnorm(N),
                   x4 = rpois(N, lambda = 3),
                   x3 = ordered(sample(c("lo", "med", "hi"),
                   size = N, replace=TRUE)))
```

```
is.data.frame.simple(mydf)
mydf$amatr <- matrix(0, ncol = 2, nrow = NROW(mydf))
is.data.frame.simple(mydf)
mydf$amatr <- NULL
is.data.frame.simple(mydf)
mydf$adf <- mydf
is.data.frame.simple(mydf)
```

---

isNA *Check if values are R NA symbol or any one of the na.strings elements*

Description

A value vector in the key will generally be a character vector. This utility is used to check if the characters are either R missing or values in a list of characters that represent missings.

Usage

```
isNA(x, na.strings = c("\\.", "", "\\s+", "N/A"))
```

Arguments

x	Input data vector
na.strings	Vector of string values to be considered as missing. Defaults will match values that are equal to ., empty string, any number of white space elements, or charcter string N/A. We do not include ‘NA‘ by default because some projects use NA to mean "not appropriate".

Value

Logical vector, TRUE if a value is either NA or in na.strings.

Examples

```
x1 <- c("TRUE", "FALSE", FALSE, TRUE, NA, "NA", ".", "N/A", " ", "")
x1na <- kutils:::isNA(x1)
cbind(x1, x1na)
```

keyApply *Apply variable key to data frame (generate recoded data frame)*

Description

This is the main objective of the variable key system.

Usage

```
keyApply(
  dframe,
  key,
  diagnostic = TRUE,
  safeNumericToInteger = TRUE,
  trimws = "both",
  ignoreCase = TRUE,
  drop = TRUE,
  debug = FALSE
)
```

Arguments

dframe	An R data frame
key	A variable key object, of class either "key" or "keylong"
diagnostic	Default TRUE: Compare the old and new data frames carefully with the keyDiagnostic function.
safeNumericToInteger	Default TRUE: Should we treat values which appear to be integers as integers? If a column is numeric, it might be safe to treat it as an integer. In many csv data sets, the values coded c(1, 2, 3) are really integers, not floats c(1.0, 2.0, 3.0). See safeInteger.
trimws	Default is "both", can change to "left", "right", or set as NULL to avoid any trimming.
ignoreCase	Default TRUE. If column name is capitalized differently than name_old in the key, but the two are otherwise identical, then the difference in capitalization will be ignored.
drop	Default TRUE. True implies drop = c("vars", "vals"). TRUE applies to both variables ("vars") and values ("vals"). "vars" means that a column will be omitted from data if it is not in the key "name_old". Similarly, if anything except "." appears in value_old, then setting drop="vals" means omission of a value from key "value_old" causes observations with those values to become NA. This is the original variable key behavior. The drop argument allows "partial keys", beginning with kutils version 1.12. drop = FALSE means that neither values nor variables are omitted. Rather than TRUE, one can specify either drop = "vars", or drop = "vals".
debug	Default FALSE. If TRUE, emit some warnings.

Value

A new data.frame object, with renamed and recoded variables

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
mydf.key.path <- system.file("extdata", "mydf.key.csv", package = "kutils")
mydf.key <-  keyImport(mydf.key.path)
mydf.path <- system.file("extdata", "mydf.csv", package = "kutils")

mydf <- read.csv(mydf.path, stringsAsFactors = FALSE)
mydf2 <- keyApply(mydf, mydf.key)

nls.keylong.path <- system.file("extdata", "natlongsurv.key_long.csv", package = "kutils")
nls.keylong <- keyImport(nls.keylong.path, long = TRUE)
data(natlongsurv)
nls.dat <- keyApply(natlongsurv, nls.keylong)
```

---

keyCheck *Check a key for consistency of names, values with classes.*

Description

Split the key into blocks of rows defined by "name_new". Within these blocks, Perform these checks: 1. name_old must be homogeneous (identical) within a block of rows. class_old and class_new must also be identical. 2. elements in "value_new" must be consistent with "class_new". If values cannot be coerced to match the class specified by class_new, there must be user error. Same for "value_old" and "class_old".

Usage

```
keyCheck(
  key,
  colname = c("name_new", "class_old", "class_new"),
  na.strings = c("\\.", "", "\\s+", "N/A")
)
```

Arguments

key	A variable key object.
colname	Leave as default to check consistency between classes, values, and names. One can specify a check only on "class_old" or "class_new", for example. But now that all work correctly, I suggest you leave the default.

na.strings	A regular expression of allowed text strings that represent missings. Now it amounts to any of these: ".", "NA", "N/A", or any white space or tab as signified by \s+.

Value

Profuse warnings and a list of failed key blocks.

Author(s)

Paul Johnson <pauljohn@ku.edu> and Ben Kite <bakite@ku.edu>

---

keyCrossRef *keyCrossRef*

Description

Checks a key for dangerous matches of old and new values in a key for different levels.

Usage

```
keyCrossRef(key, ignoreClass = NULL, verbose = FALSE, lowercase = FALSE)
```

Arguments

key	A variable key, ideally a long key. If a wide key is provided it is converted to long.
ignoreClass	Classes that should be excluded from check. Useful when many integer variables are being reverse- coded. Takes a string or vector.
verbose	Should a statement about the number of issues detected be returned? Defaults to FALSE.
lowercase	Should old and new values be passed through tolower function? Defaults to FALSE.

Details

Positions in a long key are referred to as levels. If a value is mismatched at levels 1 and 3, this means that issues are in rows 1 and 3 of the section of the given variable in a long key.

Value

Presents a warning for potentially problematic key sections. Return is dependent on verbose argument.

Author(s)

Ben Kite <bakite@ku.edu>

Examples

```
dat <- data.frame(x1 = sample(c("a", "b", "c", "d"), 100, replace = TRUE),
                  x2 = sample(c("Apple", "Orange"), 100, replace = TRUE),
                  x3 = ordered(sample(c("low", "medium", "high"), 100, replace = TRUE),
                  levels = c("low", "medium", "high")),
                  stringsAsFactors = FALSE)
key <- keyTemplate(dat, long = TRUE)
## No errors with a fresh key.
kutils:::keyCrossRef(key, verbose = TRUE)
key[1:2, "value_new"] <- c("b", "a")
key[5, "value_new"]
key[7:9, "value_new"] <- c("high", "medium", "low")
kutils:::keyCrossRef(key)
kutils:::keyCrossRef(key, ignoreClass = c("ordered", "character"), verbose = TRUE)
```

---

keyDiagnostic *Diagnose accuracy of result from applying variable key to data*

Description

Compare the old and new data frames, checking for accuracy of calculations in various ways.

Usage

```
keyDiagnostic(
  dfold,
  dfnew,
  keylist,
  max.values = 20,
  nametrunc = 18,
  wide = 200,
  confidential = FALSE
)
```

Arguments

dfold	Original data frame
dfnew	The new recoded data frame
keylist	The imported variable key that was used to transform dfold into dfnew.
max.values	Show up to this number of values for the old variable
nametrunc	Truncate column and row names. Needed if there are long factor labels and we want to fit more information on table. Default = 18 for new name, old name is 10 more characters (18 + 10 = 28).
wide	Number of characters per row in printed output. Suggest very wide screen, default = 200.
confidential	Should numbers in table be rounded to nearest "10" to comply with security standard enforced by some American research departments.

Details

CAUTION: This can print WIDE matrices. Because the on-screen output will be WIDE, make the display window WIDE!

Crosstabulate new variable versus old variable to see the coding mismatches. For tables of up to 10 values or so, that will be satisfactory.

For numeric variables, it appears there is no good thing to do except possibly to re-apply any transformations.

Author(s)

Paul Johnson <pauljohn@ku.edu>

---

keyDiff *Show difference between 2 keys*

Description

Show difference between 2 keys

Usage

```
keyDiff(oldkey, newkey)
```

Arguments

oldkey	key, original
newkey	key, possibly created by keyUpdate or by user edits

Value

NULL, or list with as many as 2 key difference data.frames, named "deleted" and "neworaltered"

Author(s)

Ben Kite <bakite@ku.edu> and Paul Johnson <pauljohn@ku.edu>

Examples

```
dat1 <- data.frame("Score" = c(1, 2, 3, 42, 4, 2),
                   "Gender" = c("M", "M", "M", "F", "F", "F"))
## First try with a long key
key1 <- keyTemplate(dat1, long = TRUE)
key1$value_new <- gsub("42", "10", key1$value_new)
key1$value_new[key1$name_new == "Gender"] <-
       mgsub(c("F", "M"), c("female", "male"),
       key1$value_new[key1$name_new == "Gender"])
key1[key1$name_old == "Score", "name_new"] <- "NewScore"
```

```
dat2 <- data.frame("Score" = 7, "Gender" = "other", "Weight" = rnorm(3))
dat2 <- plyr::rbind.fill(dat1, dat2)
dat2 <- dat2[-1,]
key2 <- keyUpdate(key1, dat2, append = TRUE)
(kdiff <- keyDiff(key1, key2))
```

`keyImport` *Import/validate a key object or import/validate a key from a file.*

Description

After the researcher has updated the key by filling in new names and values, we import that key file. This function can import the file by its name, after deducing the file type from the suffix, or it can receive a key object from memory.

Usage

```
keyImport(
  key,
  ignoreCase = TRUE,
  sep = c(character = "\\|", logical = "\\|", integer = "\\|", factor = "\\|",
    ordered = "[\\|<]", numeric = "\\|"),
  na.strings = c("\\.", "", "\\s+", "N/A"),
  missSymbol = ".",
  ...,
  keynames = NULL
)
```

Arguments

`key`	A key object (class key or keylong) or a file name character string (ending in csv, xlsx or rds).
`ignoreCase`	In the use of this key, should we ignore differences in capitalization of the "name_old" variable? Sometimes there are inadvertent misspellings due to changes in capitalization. Columns named "var01" and "Var01" and "VAR01" probably should receive the same treatment, even if the key has name_old equal to "Var01".
`sep`	Character separator in `value_old` and `value_new` strings in a wide key. Default is are "|". It is also allowed to use "<" for ordered variables. Use regular expressions in supplying separator values.
`na.strings`	Values that should be converted to missing data. This is relevant in `name_new` as well as `value_new`. In spreadsheet cells, we treat "empty" cells (the string ""), or values like "." or "N/A", as missing with defaults ".", "", "\s" (white space), and "N/A". Change that if those are not to be treated as missings.
`missSymbol`	Defaults to period "." as missing value indicator.
`...`	additional arguments for read.csv or read.xlsx.

keynames Don't use this unless you are very careful. In our current scheme, the column names in a key should be c("name_old", "name_new", "class_old", "class_new", "value_old", "value_new", "missings", "recodes"). If your key does not use those column names, it is necessary to provide keynames in a format "our_name"="your_name". For example, keynames = c(name_old = "oldvar", name_new = "newname", class_old = "vartype", class_new = "class", value_old = "score", value_new = "val").

Details

This can be either a wide or long format key file.

This cleans up variables in following ways. 1) `name_old` and `name_new` have leading and trailing spaces removed 2) `value_old` and `value_new` have leading and trailing spaces removed, and if they are empty or blank spaces, then new values are set as NA.

Policy change concerning empty "value_new" cells in input keys (20170929).

There is confusion about what ought to happen in a wide key when the user leaves value_new as empty or missing. Literally, this means all values are converted to missing, which does not seem reasonable. Hence, when a key is wide, and value_new is one of the na.strings elements, we assume the value_new is to be copied from value_old. That is to say, if value_new is not supplied, the values remain same as in old data.

In a long key, the behavior is different. Since the user can specify each value for a variable in a separate row, the na.strings appearing in value_new are treated as missing scores in the new data set to be created.

Value

key object, should be same "wide" or "long" as the input Missing symbols in value_old and value_new converted to ".".

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
mydf.key.path <- system.file("extdata", "mydf.key.csv", package = "kutils")
mydf.key <-  keyImport(mydf.key.path)
## Create some dupes
mydf.key <- rbind(mydf.key, mydf.key[c(1,7), ])
mydf.key2 <- keyImport(mydf.key)
mydf.key2
## create some empty value_new cells
mydf.key[c(3, 5, 7) , "value_new"] <- ""
mydf.key3 <- keyImport(mydf.key)
mydf.key3
mydf.keylong.path <- system.file("extdata", "mydf.key_long.csv", package = "kutils")
mydf.keylong <- keyImport(mydf.keylong.path)

## testDF is a slightly more elaborate version created for unit testing:
```

```
testdf.path <- system.file("extdata", "testDF.csv", package = "kutils")
testdf <- read.csv(testdf.path, header = TRUE)
keytemp <- keyTemplate(testdf, long = TRUE)
## A "hand edited key file"
keyPath <- system.file("extdata", "testDF-key.csv", package="kutils")
key <- keyImport(keyPath)
keydiff <- keyDiff(keytemp, key)
key2 <- rbind(key, keydiff$neworaltered)
key2 <- unique(key)
if(interactive())View(key2)
```

---

keyLookup *Look for old (or new) names in variable key*

Description

Use the key to find the original name of a variable that has been renamed, or find the new name of an original variable. The `get` argument indicates if the `name_old` or `name_new` is desired.

Usage

```
keyLookup(x, key, get = "name_old")
```

Arguments

`x`	A variable name. If `get = "name_old"`, then `x` is a value for `name_new`. If `get = "name_new"`, `x` should be a value for `name_old`.
`key`	Which key should be used
`get`	Either "name_old" (to retrieve the original name) or "name_new" (to get the new name)

Details

If `get = "name_old"`, the return is a character vector, with one element per value of `x`. If there is no match for a value of `x`, the value NA is returned for that value. However, if `get = "name_new"`, the return might be either a vector (one element per value of `x`) or a list with one element for each value of `x`. The list is returned when a value of `x` corresponds to more than one element in `name_old`.

Value

A vector or list of matches between x and either name_new or name_old elements in the key.

Author(s)

Paul Johnson

Examples

```
mydf.key.path <- system.file("extdata", "mydf.key.csv", package = "kutils")
mydf.key <-  keyImport(mydf.key.path)
mydf.key$name_new <- paste0("new_", mydf.key$name_new)
keyLookup("new_x5", mydf.key, get = "name_old")
keyLookup(c("new_x6", "new_x1"), mydf.key, get = "name_old")
keyLookup(c("x6", "x1"), mydf.key, get = "name_new")
keyLookup(c("asdf", "new_x1"), mydf.key, get = "name_old")

mydf.key <- rbind(mydf.key,
                  c("x3", "x3f",  "ordered", "factor", "","","",""))
keyLookup(c("x3"), mydf.key, get = "name_new")
keyLookup(c("x1", "x3", "x5"), mydf.key, get = "name_new")
```

---

keyRead *Read file after deducing file type from suffix.*

Description

If the input is XLSX, sheets named "key" and "varlab" are imported if the exist. If input is CSV, then the key CSV file is imported and another file suffixed with "-varlab" is imported if it exists.

Usage

```
keyRead(file, ..., na.strings = c("\\s+"))
```

Arguments

file	name of file to be imported, including path to file. file name must end in "csv", "xlsx" or "rds"
...	additional arguments for read.csv or read.xlsx.
na.strings	Values to be converted to R missing symbol NA. Default is white space, "\s+".

Details

The variable lables are a named vector saved as an attribute of the key object.

Value

A data frame or matrix.

Author(s)

Paul Johnson <pauljohn@ku.edu>

keySave *Save key as file after deducing type from suffix*

Description

This is specialized to saving of key objects, it is not a general purpose function for saving things. It scans the suffix of the file name and then does the right thing.

Usage

```
keySave(obj, file, na_ = ".", varlab)
```

Arguments

obj	a variable key object
file	file name. must end in "csv", "xlsx" or "rds"
na_	Value to insert to represent a missing score. Default ".".
varlab	FALSE or TRUE. Default is FALSE, no new labels will be created. If a key object has a varlab already, it is saved with the key, always. This parameter controls whether a new varlab template should be created when the object is saved. If TRUE and obj has no varlab attribute, a new varlab template is created by the varlabTemplate function. If TRUE and a varlab attribute currently exists, but some variables are missing labels, then varlabTemplate is called to fill in new variable labels.

Details

In updates 2017-09, a varlab element was introduced. The varlab attribute of the object is saved. The files created incorporate the variable labels object in different ways. 1) XLSX: variable labels a worksheet named "varlab" 2) CSV: variable labels saved in a separate file suffixed "-varlab.csv". 3) RDS: varlab is an attribute of the key object.

Value

NULL if no file is created. Otherwise, a key object with an attribute varlab is returned.

Author(s)

Paul Johnson <pauljohn@ku.edu>

keysPool *Homogenize class values and create a long key by pooling variable keys.*

Description

For long-format keys, this is one way to correct for errors in "class_old" or "class_new" for common variables. For a long key created by stacking together several long keys, or for a list of long keys, this will try to homogenize the classes by using a "highest common denominator" approach. If one key has x1 as a floating point, but another block of rows in the key has x1 as integer, then class must be changed to floating point (numeric). If another section of a key has x1 as a character, then character becomes the class.

Usage

```
keysPool(
  keylong = NULL,
  keysplit = NULL,
  classes = list(c("logical", "integer"), c("integer", "numeric"), c("ordered",
    "factor"), c("factor", "character")),
  colnames = c("class_old", "class_new"),
  textre = "TEXT$"
)
```

Arguments

keylong	A list of long keys, or just one long key, presumably a result of rbinding several long keys.
keysplit	Not often needed for user-level code. A list of key blocks, each of which is to be inspected and homogenized. Not used if a keylong argument is provided.
classes	A list of vectors specifying legal promotions.
colnames	Either c("class_old","class_new), ""class_old", or "class_new". The former is the default.
textre	A regular expression matching a column name to be treated as character. Default matches any variable name ending in "TEXT"

Details

Users might run keyTemplate on several data sets, arriving at keys that need to be combined. The long versions of the keys can be stacked together by a function like rbind. If the values class_old and class_new for a single variable are inconsistent, then the "key stack" will fail the tests in keyCheck. This function automates the process of fixing the class variables by "promoting" classes where possible.

Begin with a simple example. In one data set, the value of x is drawn from integers 1L, 2L, 3L, while in another set it is floating values like 1.1, 2.2. After creating long format keys, and stacking them together, the values of class_old will clash. For x, we will observe both "integer" and "numeric" in

the class_old column. In that situation, the class_old for all of the rows under consideration should be set as "numeric".

The promotion schemes are described by the variable classes, where we have the most conservative changes first. The most destructive change is when variables are converted from integer to character, for example. The conservative conversion strategies are specified in the classes variable, in which the last element in a vector will be used to replace the preceeding classes. For example, c("ordered", "factor", "character") means that the class_old values of "ordered" and "factor" will be replaced by "character".

The conversions specified by classes are tried, in order. 1. logical -> integer 2. integer -> numeric 3. ordered -> factor

If their application fails to homogenize a vector, then class is changed to "character". For example, when the value of class_old observed is c("ordered", "numeric", "character"). In that case, the class is promoted to "character", it is the least common denominator.

Value

A class-corrected version of the same format as the input, either a long key or a list of key elements.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
dat1 <- data.frame(x1 = as.integer(rnorm(100)), x2 = sample(c("Apple", "Orange"),
                   100, replace = TRUE), x3 = ifelse(rnorm(100) < 0, TRUE, FALSE))
dat2 <- data.frame(x1 = rnorm(100), x2 = ordered(sample(c("Apple", "Orange"),
                   100, replace = TRUE)), x3 = rbinom(100, 1, .5),
                   stringsAsFactors = FALSE)
key1 <- keyTemplate(dat1, long = TRUE)
key2 <- keyTemplate(dat2, long = TRUE)
keys2stack <- rbind(key1, key2)
keys2stack.fix <- keysPool(keys2stack)
keys2stack.fix2 <- keysPool(keys2stack.fix, colname = "class_new")
## Sometimes this will not be able to homogenize
dat1 <- data.frame(x1 = as.integer(rnorm(100)),
                   x2 = sample(c("Apple", "Orange"), 100, replace = TRUE))
dat2 <- data.frame(x1 = rnorm(100),
                   x2 = sample(c("Apple", "Orange"), 100, replace = TRUE),
                   stringsAsFactors = FALSE)
key1 <- keyTemplate(dat1, long = TRUE)
key2 <- keyTemplate(dat2, long = TRUE)
## Create a stack of keys for yourself
keys2stack <- rbind(key1, key2)
keys.fix <- keysPool(keys2stack)
## We will create stack of keys for you
keys.fix2 <- keysPool(list(key1, key2))
## View(keys.fix)
## View(keys.fix2)
```

```
## If you have wide keys, convert them with wide2long, either by
key1 <- keyTemplate(dat1)
key2 <- keyTemplate(dat2)
keysstack.wide <- rbind(wide2long(key1), wide2long(key2))
keys.fix <- keysPool(keysstack.wide)
## or
keysPool(list(wide2long(key1), wide2long(key2)))
```

`keysPoolCheck` *Compares keys from different data sets; finds differences classes of variables. This used to check for similarity of keys from various data sets, one precursor to either combining the keys or merging the data sets themselves.*

Description

When several supposedly "equivalent" data sets are used to generate variable keys, there may be trouble. If variables with same name have different classes, keyApply might fail when applied to one of the data sets.

Usage

```
keysPoolCheck(keys, col = "class_old", excludere = "TEXT$")
```

Arguments

`keys`	A list with variable keys.
`col`	Name of key column to check for equivalence. Default is "class_old", but "class_new" can be checked as well.
`excludere`	Exclude variables matching a regular expression (re). Default example shows exclusion of variables that end in the symbol "TEXT".

Details

This reports on differences in classes among keys. By default, it looks for differences in "class_old", because that's where we usually see trouble.

The output here is diagnostic. The keys can be fixed manually, or the function keysPool can implement an automatic correction.

Value

Data.frame summarizing class differences among keys

Author(s)

Paul Johnson

Examples

```
set.seed(234)
dat1 <- data.frame(x1 = rnorm(100),
                   x2 = sample(c("Male", "Female"), 100, replace = TRUE),
                   x3_TEXT = "A", x4 = sample(1:10000, 100))
dat2 <- data.frame(x1 = rnorm(100), x2 = sample(c("Male", "Female"),
                   100, replace = TRUE),
                   x3_TEXT = sample(1:100, 100),
                   stringsAsFactors = FALSE)
key1 <- keyTemplate(dat1)
key2 <- keyTemplate(dat2)
keys <- list(key1, key2)
keysPoolCheck(keys)
## See problem in class_old
keysPoolCheck(keys, col = "class_old")
## problems in class_new
keysPoolCheck(keys, col = "class_new")
keysPoolCheck(keys, excludere = "TEXT$")
```

---

keyTemplate *Create variable key template (in memory or in a file)*

Description

A variable key is a human readable document that describes the variables in a data set. A key can be revised and re-imported by R to recode data. This might also be referred to as a "programmable codebook." This function inspects a data frame, takes notice of its variable names, their classes, and legal values, and then it creates a table summarizing that information. The aim is to create a document that principal investigators and research assistants can use to keep a project well organized. Please see the vignette in this package.

Usage

```
keyTemplate(
  dframe,
  long = FALSE,
  sort = FALSE,
  file = NULL,
  max.levels = 15,
  missings = NULL,
  missSymbol = ".",
  safeNumericToInteger = TRUE,
  trimws = "both",
  varlab = FALSE
)
```

Arguments

dframe	A data frame
long	Default FALSE.
sort	Default FALSE. Should the rows representing the variables be sorted alphabetically? Otherwise, they appear in the order in which they were included in the original dataset.
file	DEFAULT NULL, meaning no file is produced. Choose a file name ending in either "csv" (for comma separated variables), "xlsx" (compatible with Microsoft Excel), or "rds" (R serialization data). The file name will be used to select among the 3 storage formats. XLSX output requires the openxlsx package.
max.levels	How high is the limit on the number of values for discrete (integer, character, and Date) variables? Default = 15. If observed number exceeds max.levels, we conclude the author should not assign new values in the key and only the missing value will be included in the key as a "placeholder". This does not affect variables declared as factor or ordered variables, for which all levels are included in all cases.
missings	Values in exising data which should be treated as missing in the new key. Character string in format acceptable to the `assignMissing` function. Can be a string with several missing indicators"1;2;3;(8,10);[22,24];> 99;< 2".
missSymbol	Default ".". A character string used to represent missing values in the key that is created. Relevant (mostly) for the key's `value_new` column. Default is the period, ".". Because R's symbol `NA` can be mistaken for the character string `"NA"`, we use a different (hopefully unmistakable) symbol in the key.
safeNumericToInteger	Default TRUE: Should we treat values which appear to be integers as integers? If a column is numeric, it might be safe to treat it as an integer. In many csv data sets, the values coded c(1, 2, 3) are really integers, not floats c(1.0, 2.0, 3.0). See `safeInteger`.
trimws	Default is "both", user can change to "left", "right", or set as NULL to avoid any trimming.
varlab	A key can have a companion data structure for variable labels. Default is FALSE, but the value may also be TRUE or a named vector of variable labels, such as `c("x1" = "happiness", "x2" = "wealth")`. The labels become an attribute of the key object. See Details for information on storage of varlabs in saved key files.

Details

The variable key can be created in two formats, wide and long. The original style of the variable key, wide, has one row per variable. It has a style for compact notation about current values and required recodes. That is more compact, probably easier for experts to read, but perhaps more difficult to edit. The long style variable key has one row per value per variable. Thus, in a larger project, the long key can have many rows. However, in a larger project, the long style key is easier to edit with a spread sheet program.

After a key is created, it should be re-imported into R with the `kutils::keyImport` function. Then the key structure can guide the importation and recoding of the data set.

Concerning the varlab attribute. Run `attr(key, "varlab"` to review existing labels, if any.

Storing the variable labels in files requires some care because the `rds`, `xlsx`, and `csv` formats have different capabilities. The `rds` storage format saves all attributes without difficulty. In contrast, because `csv` and `xlsx` do not save attributes, the varlabs are stored as separate character matrices. For `xlsx` files, the varlab object is saved as a second sheet in `xlsx` file, while in `csv` a second file suffixed "-varlab.csv" is created.

Value

A key in the form of a data frame. May also be saved on disk if the file argument is supplied. The key may have an attribute "varlab", variable labels.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
set.seed(234234)
N <- 200
mydf <- data.frame(x5 = rnorm(N),
                   x4 = rpois(N, lambda = 3),
                   x3 = ordered(sample(c("lo", "med", "hi"),
                   size = N, replace=TRUE),
                   levels = c("med", "lo", "hi")),
                   x2 = letters[sample(c(1:4,6), N, replace = TRUE)],
                   x1 = factor(sample(c("cindy", "bobby", "marcia",
                                        "greg", "peter"), N,
                   replace = TRUE)),
                   x7 = ordered(letters[sample(c(1:4,6), N, replace = TRUE)]),
                   x6 = sample(c(1:5), N, replace = TRUE),
                   stringsAsFactors = FALSE)
mydf$x4[sample(1:N, 10)] <- 999
mydf$x5[sample(1:N, 10)] <- -999

## Note: If we change this example data, we need to save a copy in
## "../inst/extdata" for packacing
dn <- tempdir()
write.csv(mydf, file = file.path(dn, "mydf.csv"), row.names = FALSE)
mydf.templ <- keyTemplate(mydf, file = file.path(dn, "mydf.templ.csv"),
                          varlab = TRUE)
mydf.templ_long <- keyTemplate(mydf, long = TRUE,
                            file = file.path(dn, "mydf.templlong.csv"),
                            varlab = TRUE)

mydf.templx <- keyTemplate(mydf, file = file.path(dn, "mydf.templ.xlsx"),
                            varlab = TRUE)
mydf.templ_longx <- keyTemplate(mydf, long = TRUE,
                             file = file.path(dn, "mydf.templ_long.xlsx"),
                             varlab = TRUE)
## Check the varlab attribute
attr(mydf.templ, "varlab")
```

```
mydf.tmpl2 <- keyTemplate(mydf,
                          varlab = c(x5 = "height", x4 = "age",
                          x3 = "intelligence", x1 = "Name"))
## Check the varlab attribute
attr(mydf.tmpl2, "varlab")

## Try with the national longitudinal study data
data(natlongsurv)
natlong.templ <- keyTemplate(natlongsurv,
                           file = file.path(dn, "natlongsurv.templ.csv"),
                           max.levels = 15, varlab = TRUE, sort = TRUE)

natlong.templlong <- keyTemplate(natlongsurv, long = TRUE,
                    file = file.path(dn, "natlongsurv.templ_long.csv"), sort = TRUE)
if(interactive()) View(natlong.templlong)
natlong.templlong2 <- keyTemplate(natlongsurv, long = TRUE,
                       missings = "<0", max.levels = 50, sort = TRUE,
                       varlab = TRUE)
if(interactive()) View(natlong.templlong2)

natlong.templwide2 <- keyTemplate(natlongsurv, long = FALSE,
                       missings = "<0", max.levels = 50, sort = TRUE)
if(interactive()) View(natlong.templwide2)

all.equal(wide2long(natlong.templwide2), natlong.templlong2)

head(keyTemplate(natlongsurv, file = file.path(dn, "natlongsurv.templ.xlsx"),
             max.levels = 15, varlab = TRUE, sort = TRUE), 10)
head(keyTemplate(natlongsurv, file = file.path(dn, "natlongsurv.templ.xlsx"),
             long = TRUE, max.levels = 15, varlab = TRUE, sort = TRUE), 10)

list.files(dn)
```

`keyTemplateSPSS` *Import an SPSS file, create a key representing the numeric -> factor transition*

Description

This is a way to keep track of the scores that are used in the SPSS file. It is also an easy way to start a new variable key that makes it convenient to work on the value_new column with R text functions.

Usage

```
keyTemplateSPSS(dat, long = TRUE)
```

Arguments

`dat`	A character string path to the SPSS file
`long`	TRUE returns a long key, otherwise a wide key

Value

A variable key (long or wide)

Author(s)

Paul Johnson <pauljohn@ku.edu>

`keyTemplateStata` *Import a Stata (version 12 or lower) file, create a key representing the numeric -> factor transition*

Description

This is a way to keep track of the scores that are used in the Stata file. It is also an easy way to start a new variable key that makes it convenient to work on the value_new column with R text functions.

Usage

```
keyTemplateStata(dat, long = TRUE)
```

Arguments

`dat`	A character string path to the Stata file
`long`	TRUE returns a long key, otherwise a wide key

Value

A variable key (long or wide)

Author(s)

Paul Johnson <pauljohn@ku.edu>

`keyUpdate` *Update a key in light of a new data frame (add variables and values)*

Description

The following chores must be handled. 1. If the data.frame has variables which are not currently listed in the variable key's "name_old" variable, then new variables are added to the key. 2. If the data.frame has new values for the previously existing variables, then those values must be added to the keys. 3. If the old key has "name_new" or "class_new" designated for variables, those MUST be preserved in the new key for all new values of those variables.

Usage

```
keyUpdate(key, dframe, append = TRUE, safeNumericToInteger = TRUE)
```

Arguments

key	A variable key
dframe	A data.frame object.
append	If long key, should new rows be added to the end of the updated key? Default is TRUE. If FALSE, new rows will be sorted with the original values.
safeNumericToInteger	Default TRUE: Should we treat variables which appear to be integers as integers? In many csv data sets, the values coded `c(1, 2, 3)` are really integers, not floats `c(1.0, 2.0, 3.0)`. See `safeInteger`. ## Need to consider implementing this: ## @param ignoreCase

Details

This function will not alter key values for "class_old", "value_old" or "value_new" for variables that have no new information.

This function deduces if the key provided is in the wide or long format from the class of the object.

Value

Updated variable key.

Author(s)

Ben Kite <bakite@ku.edu>

Examples

```
## Original data frame has 2 variables
dat1 <- data.frame("Score" = c(1, 2, 3, 42, 4, 2),
                   "Gender" = c("M", "M", "M", "F", "F", "F"))
## New data has all of original dat1, plus a new variable "Weight"
#and has new values for "Gender" and "Score"
dat2 <- plyr::rbind.fill(dat1, data.frame("Score" = 7,
           "Gender" = "other", "Weight" = rnorm(3)))
## Create a long key for the original data, specify some
## recodes for Score and Gender in value_new
key1.long <- keyTemplate(dat1, long = TRUE, varlab = TRUE)

key1.long$value_new <- gsub("42", "10", key1.long$value_new)
key1.long$value_new[key1.long$name_new == "Gender"] <-
       mgsub(c("F", "M"), c("female", "male"),
       key1.long$value_new[key1.long$name_new == "Gender"])
key1.long[key1.long$name_old == "Score", "name_new"] <- "NewScore"
keyUpdate(key1.long, dat2, append = TRUE)
## Throw away one row, make sure key still has Score values
```

```
dat2 <- dat2[-1,]
(key1.long.u <- keyUpdate(key1.long, dat2, append = FALSE))
## Key change Score to character variable
key1.longc <- key1.long
key1.longc[key1.longc$name_old == "Score", "class_new"] <- "character"
keyUpdate(key1.longc, dat2, append = TRUE)
str(dat3 <- keyApply(dat2, key1.longc))
## Now try a wide key
key1.wide <- keyTemplate(dat1)
## Put in new values, same as in key1.long
key1.wide[key1.wide$name_old == "Score", c("name_new", "value_new")] <-
                           c("NewScore", "1|2|3|4|10|.")
key1.wide[key1.wide$name_old == "Gender", "value_new"] <- "female|male|."
## Make sure key1.wide equivalent to key1.long:
## If this is not true, it is a fail
all.equal(long2wide(key1.long), key1.wide, check.attributes = FALSE)
(key1.wide.u <- keyUpdate(key1.wide, dat2))
key1.long.to.wide <- long2wide(key1.long.u)
all.equal(key1.long.to.wide, key1.wide.u, check.attributes = FALSE)
str(keyApply(dat2, key1.wide.u))

mydf.key.path <- system.file("extdata", "mydf.key.csv", package = "kutils")
mydf.key <-  keyImport(mydf.key.path)
##'
set.seed(112233)
N <- 20
## The new Jan data arrived!
mydf2 <- data.frame(x5 = rnorm(N),
                    x4 = rpois(N, lambda = 3),
                    x3 = ordered(sample(c("lo", "med", "hi"),
                                        size = N, replace=TRUE),
                                 levels = c("med", "lo", "hi")),
                    x2 = letters[sample(c(1:4,6), N, replace = TRUE)],
                    x1 = factor(sample(c("jan"), N, replace = TRUE)),
                    x7 = ordered(letters[sample(c(1:4,6), N, replace = TRUE)]),
                    x6 = sample(c(1:5), N, replace = TRUE),
                    stringsAsFactors = FALSE)
mydf.key2 <- keyUpdate(mydf.key, mydf2)
mydf.key2
mydf.key2["x1", "value_old"] <- "cindy|bobby|jan|peter|marcia|greg|."
mydf.key2["x1", "value_new"] <- "Cindy<Bobby<Jan<Peter<Marcia<Greg<."
##'
mydf.key.path <- system.file("extdata", "mydf.key.csv", package = "kutils")
mydf.path <- system.file("extdata", "mydf.csv", package = "kutils")
mydf <- read.csv(mydf.path, stringsAsFactors=FALSE)
mydf3 <- rbind(mydf, mydf2)
## Now recode with revised key
mydf4 <- keyApply(mydf3, mydf.key2)
rockchalk::summarize(mydf4)
```

---

likert *Percentage tables for Likert Scale variables*

Description

Creates a table with columns for allowed values and rows for variables.

Usage

```
likert(data, vlist, columnlabels, valuelabels, rows = FALSE, digits = 2, ...)
```

Arguments

data	A data frame. Function will try to include all variables in data, unless vlist is provided.
vlist	A vector of column names in data that should be displayed
columnlabels	Column labels, optional to beautify variable names. If not supplied, column names will be used as column labels. Provide either 1) A named vector that replaces one or more columns, `c(oldname1 = "newlabel1", oldname2 = "newlabel2")` where oldnames are in colnames(data), or 2) a vector of same length as vlist (or data if vlist is not supplied) that will replace them one for one.
valuelabels	A vector of values to beautify existing levels. If not supplied, factor levels will be used as row labels
rows	Should output be transposed. This may help if there are many variables that need to fit on the page. Percentages will appear on the rows, rather than columns.
digits	Number of decimals to display in percentages
...	Arguments to pass to R's table function. We suggest `useNA = "always"` to add missing value information and `exclude = original.value.label` to exclude values observed. Currently, `useNA = "ifany"` does not work as expected, the number of missings will be displayed, even if there are none.

Value

A list, including a frequency table (called "freqTab"), column counts ("counts"), column sums ("sums"), and column percents ("pcts").

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
vvector <- c("Strongly Disagree", "Disagree", "Neutral",
             "Agree", "Strongly Agree")
set.seed(2342234)
N <- 28
scales <-
    data.frame(Vegas = factor(sample(1:5, N, replace = TRUE),
        levels = 1:5, labels = vvector),
               NewYork = factor(sample(1:5, N, replace = TRUE),
        levels = 1:5, labels = vvector),
               Paris = factor(sample(1:5, N, replace = TRUE),
```

```
        levels = 1:5, labels = vvector),
               Berlin = factor(sample(1:5, N, replace = TRUE),
        levels = 1:5, labels = vvector))

likert(scales)

likert(scales, exclude = "Disagree")

likert(scales, exclude = "Strongly Disagree", useNA = "ifany")

(mySummary1 <- likert(data = scales, vlist = c("Vegas", "NewYork", "Paris")))
mySummary1[["pcts"]]

(mySummary2 <- likert(scales, vlist = c("Vegas", "NewYork", "Paris"),
                    valuelabels = c("SD", "D", "N", "A", "SA")))
(mySummary3 <- likert(scales, vlist = c("Vegas", "NewYork", "Paris"),
                    valuelabels = c("Strongly Disagree" = "Strong Disagreement")))

(mySummary5 <- likert(scales, vlist = c("Vegas", "NewYork", "Paris"),
      valuelabels = c("SD", "D", "N", "A", "SA"),
      columnlabels = c("Vegas" = "Sin City"), rows = TRUE))

 ## Example of how one might write this in a file.
 ## print(xtable::xtable(mySummary1[[1]], digits = 1),
 ##       type = "html", file = "varCount-1.html")
```

`long2wide` *convert a key object from long to wide format*

Description

##' This is not flexible, assumes columns are named in our canonical style, which means the columns are named c("name_old", "name_new", "class_old", "class_new", "value_old", "value_new").

Usage

```
long2wide(
  keylong,
  na.strings = c("\\.", "", "\\s+", "N/A"),
  missSymbol = "."
)
```

Arguments

`keylong`	A variable key in the long format
`na.strings`	Strings to be treated as missings in value_new
`missSymbol`	Default is ".", character to insert in value when R NA is found.

Value

A wide format variable key

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
mydf.path <- system.file("extdata", "mydf.csv", package = "kutils")
mydf <- read.csv(mydf.path, stringsAsFactors=FALSE)
## A wide key we are trying to match:
mydf.key <- keyTemplate(mydf, long = FALSE, sort = TRUE)
mydf.key["x4", "missings"] <- "999"
## A long key we will convert next
mydf.keylong <- keyTemplate(mydf, long = TRUE, sort = TRUE)
mydf.keylong[mydf.keylong[ , "name_old"] == "x4" &
    mydf.keylong[ , "value_old"] == "999", "missings"] <- "999"
mydf.long2wide <- long2wide(mydf.keylong)
all.equal(mydf.key, mydf.long2wide)

mydf.keylong.path <- system.file("extdata", "mydf.key_long.csv", package = "kutils")
mydf.keylong <- keyImport(mydf.keylong.path)
mydf.keywide <- long2wide(mydf.keylong)
mydf.keylong2 <- wide2long(mydf.keywide)
## Is error if following not TRUE
all.equal(mydf.keylong2, mydf.keylong)
```

---

mergeCheck *First draft of function to diagnose problems in merges and key variables*

Description

This is a first effort. It works with 2 data frames and 1 key variable in each. It does not work if the by parameter includes more than one column name (but may work in future). The return is a list which includes full copies of the rows from the data frames in which trouble is observed.

Usage

```
mergeCheck(
  x,
  y,
  by,
  by.x = by,
  by.y = by,
  incomparables = c(NULL, NA, NaN, Inf, "\\s+", "")
)
```

Arguments

x	data frame
y	data frame
by	Commonly called the "key" variable. A column name to be used for merging (common to both x and y)
by.x	Column name in x to be used for merging. If not supplied, then by.x is assumed to be same as by.
by.y	Column name in y to be used for merging. If not supplied, then by.y is assumed to be same as by.
incomparables	values in the key (by) variable that are ignored for matching. We default to include these values as incomparables: c(NULL, NA, NaN, Inf, "\s+", ""). Note this is a larger list of incomparables than assumed by R merge (which assumes only NULL).

Value

A list of data structures that are displayed for keys and data sets. The return is `list(keysBad, keysDuped, unmatched)`. `unmatched` is a list with 2 elements, the unmatched cases from `x` and `y`.

Author(s)

Paul Johnson

Examples

```
df1 <- data.frame(id = 1:7, x = rnorm(7))
df2 <- data.frame(id = c(2:6, 9:10), x = rnorm(7))
mc1 <- mergeCheck(df1, df2, by = "id")
## Use mc1 objects mc1$keysBad, mc1$keysDuped, mc1$unmatched
df1 <- data.frame(id = c(1:3, NA, NaN, "", " "), x = rnorm(7))
df2 <- data.frame(id = c(2:6, 5:6), x = rnorm(7))
mergeCheck(df1, df2, by = "id")
df1 <- data.frame(idx = c(1:5, NA, NaN), x = rnorm(7))
df2 <- data.frame(idy = c(2:6, 9:10), x = rnorm(7))
mergeCheck(df1, df2, by.x = "idx", by.y = "idy")
```

---

mgsub *apply a vector of replacements, one after the other.*

Description

This is multi-gsub. Use it when it is necessary to process many patterns and replacements in a given order on a vector.

Usage

```
mgsub(pattern, replacement, x, ...)
```

Arguments

pattern	vector of values to be replaced. A vector filled with patterns as documented in the gsub pattern argument
replacement	vector of replacements, otherwise same as gsub. Length of replacement must be either 1 or same as pattern, otherwise an error results.
x	the vector in which elements are to be replaced, same as gsub
...	Additional arguments to be passed to gsub

Value

vector with pattern replaced by replacement

Author(s)

Jared Harpole <jared.harpole@gmail.com> and Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c("Tom", "Jerry", "Elmer", "Bugs")
pattern <- c("Tom", "Bugs")
replacement <- c("Thomas", "Bugs Bunny")
(y <- mgsub(pattern, replacement, x))
x[1] <- "tom"
(y <- mgsub(pattern, replacement, x, ignore.case = TRUE))
(y <- mgsub(c("Elmer", "Bugs"), c("Looney Characters"), x, ignore.case = TRUE))
```

---

modifyVector *Use new information to update a vector. Similar in concept to R's modify list*

Description

Original purpose was to receive 2 named vectors, x and y, and copy "updated" named values from y into x. If x or y are not named, however, this will do something useful.

- Both vectors are named: values in x for which y names match will be updated with values from y. If augment is true, then named values in y that are not present in x will be added to x.
- If neither vector is named: returns a new vector with x as the values and y as the names. Same as returning names(x) <- y.
- If x is not named, y is named: replaces elements in x with values of y where suitable (x matches names(y)). For matches, returns x = y[x] if names(y) include x.
- If x is named, y is not named: returns y, but with names from x. Lengths of x and y must be identical.
- If y is NULL or not provided, x is returned unaltered.

Usage

```
modifyVector(x, y, augment = FALSE, warnings = FALSE)
```

Arguments

x	vector to be updated, may be named or not.
y	possibly a named vector. If unnamed, must match length of x. If named, and length is shorter than x, then name-value pairs in x will be replaced with name-value pairs with y. If names in y are not in x, the augment argument determines the result.
augment	If TRUE, add new items in x from y. Otherwise, ignore named items in y that are not in x.
warnings	Defaults as FALSE. Show warnings about augmentation of the target vector.

Value

an updated vector

Author(s)

Paul Johnson

Examples

```
x <- c(a = 1, b = 2, c = 3)
y <- c(b = 22)
modifyVector(x, y)
y <- c(c = 7, a = 13, e = 8)
## If augment = TRUE, will add more elements to x
modifyVector(x, y, augment = TRUE)
modifyVector(x, y)
x <- c("a", "b", "c")
y <- c("income", "education", "sei")
## Same as names(x) <- y
modifyVector(x, y)
x <- c("a", "b", "c")
y <- c(a = "happy")
modifyVector(x, y)
y <- c(a = "happy", g = "glum")
## Will give error unless augment = TRUE
modifyVector(x, y, augment = TRUE)
```

n2NA *Convert nothing to R missing(NA).*

Description

By "nothing", we mean white space or other indications of nothingness. Goal is to find character strings that users might insert in a key to indicate missing values. Those things, which are given default values in the argument nothings, will be changed to NA.

Usage

```
n2NA(x, nothings = c("\\.", "\\s"), zapspace = TRUE)
```

Arguments

x	A character vector. If x is not a character vector, it is returned unaltered without warning.
nothings	A vector of values to be matched by regular expressions as missing. The default vector is c("\.", "\s"), where "\." means a literal period (backslashes needed to escape the symbol which would otherwise match anything in a regular expression).
zapspace	Should leading and trailing white space be ignored, so that, for example " . " and "." are both treated as missing.

Details

Using regular expression matching, any value that has nothing except for the indicated "nothing" values is converted to NA. The "nothing" values included by default are a period by itself (A SAS missing value), an empty string, or white space, meaning " ", or any number of spaces, or a tab.

Value

A vector with "nothing" values replaced by R's NA symbol. Does not alter other values in the vector. Previous version had applied zapspace to non-missing values, but it no longer does so.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
gg <- c("", " ", "   ", "\t", "\t some", "some\t", " space first", ".",
       " . ")
n2NA(x = gg)
n2NA(x = gg, zapspace = FALSE)
n2NA(x = gg, nothings = c("\\s"), zapspace = FALSE)
n2NA(x = gg, nothings = c("\\."), zapspace = TRUE)
n2NA(x = gg, nothings = c("\\."), zapspace = FALSE)
```

`natlongsurv` *Smoking, Happiness, and other survey responses*

Description

An idiosyncratic selection of 29 variables from the Original Cohort-Young Women 1968-2003 edition of the US National Longitudinal Survey. This originally included 5159 rows, but subset includes only 2867 rows, so sample frequencies will not match the values listed in the codebook. A snapshot of the codebook, "natlongsurv.cdb.txt", which we have trimmed down, is included in the package.

Usage

```
data(natlongsurv)
```

Format

A data frame with 2867 rows and 29 variables:

- R0000100 IDENTIFICATION CODE
- R0003300 MARITAL STATUS, 1968
- R0005700 AGE WHEN STOPPED ATTENDING SCHOOL, 1968
- R0060300 IQ SCORE, 1968
- R1051600 HIGHEST GRADE COMPLETED
- R1302000 SMOKING - DOES R SMOKE, 1991
- R1302100 SMOKING - NUMBER OF CIGARETTES R SMOKES PER DAY, 91 (PRESENT SMOKER)
- R6235600 HIGHEST GRADE COMPLETED
- R6502300 IS RESIDENCE/LIVING QUARTERS HOME/APARTMENT/OTHER?
- R6513700 HOUSEHOLD RECORD - HOUSEHOLD MEMBER - AGE CALCULATED FROM BIRTH DATE
- R6516200 CURRENT MARITAL STATUS
- R6520300 HIGHEST GRADE COMPLETED OF HUSBAND
- R6553600 HIGHEST GRADE COMPLETED OF PARTNER
- R7289200 SMOKING - CURRENTLY SMOKE CIGARETTES
- R7289400 ALCOHOL USE - HAS R CONSUMED ANY ALCOHOLIC BEVERAGES IN PAST MONTH?
- R7293430 YOUNG WOMEN 20-ITEM CES-D ITEM RESPONSE SCORE
- R7312300 INCOME FROM WAGES/SALARY IN PAST YEAR
- R7329900 INCOME ADEQUACY: R OPINION OF HER HAPPINESS WITH HER/FAMILY INCOME

- R7330000 INCOME ADEQUACY: R OPINION OF AMOUNT NEEDED TO MAKE ENDS MEET $ AMOUNT
- R7337600 R HAS ATTENDED/COMPLETED TWO/MORE YEARS OF COLLEGE
- R7344600 ATTITUDE TOWARD FEELINGS OVERALL
- R7344700 DID R DO ANY UNPAID VOLUNTEER WORK IN PAST YEAR?
- R7347500 ATTITUDE TOWARD SOCIAL SECURITY - PERCENT WOULD INVEST IN STOCKS? 2004
- R7347600 ATTITUDE TOWARD SOCIAL SECURITY - PERCENT WOULD INVEST IN BONDS OF PRIVATE COMPANIES? 2004
- R7347700 ATTITUDE TOWARD SOCIAL SECURITY - PERCENT WOULD INVEST IN U.S. GOVERNMENT BONDS? 2004
- R7477700 TOTAL CHILDREN IN ROSTER
- R7477800 COUNT ELIGIBLE HOUSEHOLD CHILDREN
- R7610300 REGION OF RESIDENCE

Details

All variables are for the 2003 year, except where otherwise noted.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Source

National Longitudinal Surveys public-use data set (Bureau of Labor Statistics, 2018).

References

Bureau of Labor Statistics. 2018. NLS Original Cohort: Mature and Young Women, US National Longitudinal Surveys Public Use Data Sets (www.bls.gov/nls/home.htm).

Examples

```
data(natlongsurv)
peek(natlongsurv, ask = FALSE, file = paste0(tempdir(), "/","peek.pdf"))
```

padW0 *Insert 0's in the front of existing digits or characters so that all elements of a vector have the same number of characters.*

Description

The main purpose was to correct ID numbers in studies that are entered as integers with leading 0's as in 000001 or 034554. R's default coercion of integers will throw away the preceding 0's, and reduce that to 1 or 34554. The user might want to re-insert the 0's, thereby creating a character vector with values "000001" and "045665".

Usage

```
padW0(x, n = 0)
```

Arguments

x	vector to be converted to a character variable by inserting 0's at the front. Should be integer or character, floating point numbers will be rounded down. All other variable types will return errors.
n	Optional parameter. The desired final length of character vector. This parameter is a guideline to determine how many 0's must be inserted. This will be ignored if n is smaller than the number of characters in the longest element of x.

Details

If x is an integer or a character vector, the result is the more-or-less expected outcome (see examples). If x is numeric, but not an integer, then x will be rounded to the lowest integer.

The previous versions of this function failed when there were missing values (NA) in the vector x. This version returns NA for those values.

One of the surprises in this development was that sprintf() in R does not have a known consequence when applied to a character variable. It is platform-dependent (unredictable). On Ubuntu Linux 16.04, for example `sprintf("%05s", 2)` gives back `" 2"`, rather than (what I expected) `"00002"`. The problem is mentioned in the documentation for `sprintf`. The examples show this does work now, but please test your results.

Value

A character vector

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x1 <- c(0001, 0022, 3432, NA)
padW0(x1)
padW0(x1, n = 10)
x2 <- c("1", "22", "3432", NA)
padW0(x2)
## There's been a problem with strings, but this works now.
## It even preserves non-leading spaces. Hope that's what you want.
x3 <- c("1", "22 4", "34323  42", NA)
padW0(x3)
x4 <- c(1.1, 334.52, NA)
padW0(x4)
```

---

peek *Show variables, one at a time, QUICKLY and EASILY.*

Description

This makes it easy to quickly scan through all of the columns in a data frame to spot unexpected patterns or data entry errors. Numeric variables are depicted as histograms, while factor and character variables are summarized by the R table function and then presented as barplots. This is most useful with a large screen graphic device (try running the function provided with this package, `dev.create(height=7, width=7)`) or any other method you prefer to create a large device.

Usage

```
peek(
  dat,
  sort = TRUE,
  file = NULL,
  textout = FALSE,
  ask,
  ...,
  xlabstub = "kutils peek: ",
  freq = FALSE,
  histargs = list(probability = !freq),
  barargs = list(horiz = TRUE, las = 1)
)
```

Arguments

`dat`	An R data frame or something that can be coerced to a data frame by `as.data.frame`
`sort`	Default TRUE. Do you want display of the columns in alphabetical order?
`file`	Should output go in file rather than to the screen. Default is NULL, meaning show on screen. If you supply a file name, we will write PDF output into it.
`textout`	If TRUE, counts from histogram bins and tables will appear in the console.

ask	As in the old style R par(ask = TRUE): should keyboard interaction advance to the next plot. Will default to false if the file argument is non-null. If file is null, setting ask = FALSE will cause graphs to whir bye without pausing.
...	Additional arguments for the pdf, histogram, table, or barplot functions. Please see Details below.
xlabstub	A text stub that will appear in the x axis label. Currently it includes advertising for this package.
freq	As in the histogram frequency argument. Should graphs show counts (freq = TRUE) or proportions (AKA densities) (freq = FALSE)
histargs	A list of arguments to be passed to the hist function.
barargs	A list of arguments to be passed to the barplot function.

Value

A vector of column names that were plotted

Try the Defaults

Every effort has been made to make this simple and easy to use. Please run the examples as they are before becoming too concerned about customization. This function is intended for getting a quick look at each variable, one-by-one, it is not intended to create publication quality histograms. For sake of the fastidious users, a lot of settings can be adjusted. Users can control the parameters for presentation of histograms (parameters for hist) and barplots (parameters for barplot). The function also can create frequency tables (which users can control by providing additional named arguments).

Style

The histograms are standard, upright histograms. The barplots are horizontal. I chose to make the bars horizontal because long value labels are more easily accomodated on the left axis. The code measures the length (in inches) for strings and the margin is increased accordingly. The examples have a demonstration of that effect.

Dealing with Dots

additional named arguments, . . . , are inspected and sorted into groups intended to control use of R functions hist, barplot, table and pdf.

The parameters c("exclude", "dnn", "useNA", "deparse.level") and will go to the table function, which is used to make barplots for factor and character variables. These named arguments are extracted and sent to the pdf function: c("width", "height", "onefile", "family", "title", "fonts", "version", "paper", "encoding", "bg", "fg", "pointsize", "pagecentre", "colormodel", "useDingbats", "useKerning", "fillOddEven", "compress"). Any other arguments that are unique to hist or barplot are sorted out and sent only to those functions.

Any other arguments, including graphical parameters will be sent to both the histogram and barplot functions, so it is a convenient way to obtain uniform appearance. Additional arguments that are common to barplot and hist will work, and so will any graphics parameters (named arguments of

par, for example). However, if one wants to target some arguments to hist, but not barplot, then the histargs list argument should be used. Similarly, barargs should be used to send argument to the barplot function. Warning: the defaults for histargs and barargs include some settings that are needed for the existing design. If new lists for histargs or barargs are supplied, the previously specified defaults are lost. Hence, users should include the existing members of those lists, possibly with revised values.

All of this argument sorting effort is done in order to reduce a prolific number of warnings that were observed in previous editions of this function.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
set.seed(234234)
N <- 200
mydf <- data.frame(x5 = rnorm(N), x4 = rnorm(N), x3 = rnorm(N),
                   x2 = letters[sample(1:24, 200, replace = TRUE)],
                   x1 = factor(sample(c("cindy", "bobby", "marsha",
                                        "greg", "chris"), 200, replace = TRUE)),
                   stringsAsFactors = FALSE)
## Insert 16 missings
mydf$x1[sample(1:150, 16,)] <- NA
mydf$adate <- as.Date(c("1jan1960", "2jan1960", "31mar1960", "30jul1960"), format = "%d%b%y")
peek(mydf)
peek(mydf, sort = FALSE)
## Demonstrate the dot-dot-dot usage to pass in hist params
peek(mydf, breaks = 30, ylab = "These are Counts, not Densities", freq = TRUE)
## Not Run: file output
## peek(mydf, sort = FALSE, file = "three_histograms.pdf")
## Use some objects from the datasets package
library(datasets)
peek(cars, xlabstub = "R cars data: ")
peek(EuStockMarkets, xlabstub = "Euro Market Data: ")
peek(EuStockMarkets, xlabstub = "Euro Market Data: ", breaks = 50,
     freq = TRUE)
## Not run: file output
## peek(EuStockMarkets, breaks = 50, file = "myeuro.pdf",
##      height = 4, width=3, family = "Times")
## peek(EuStockMarkets, breaks = 50, file = "myeuro-%d3.pdf",
##      onefile = FALSE, family = "Times", textout = TRUE)
## xlab goes into "..." and affects both histograms and barplots
peek(mydf, breaks = 30, ylab = "These are Counts, not Densities",
    freq = TRUE)
## xlab is added in the barargs list.
peek(mydf, breaks = 30, ylab = "These are Counts, not Densities",
    freq = TRUE, barargs = list(horiz = TRUE, las = 1, xlab = "I'm in barargs"))
peek(mydf, breaks = 30, ylab = "These are Counts, not Densities", freq = TRUE,
     barargs = list(horiz = TRUE, las = 1, xlim = c(0, 100),
     xlab = "I'm in barargs, not in histargs"))
```

```
levels(mydf$x1) <- c(levels(mydf$x1), "arthur philpot smythe")
mydf$x1[4] <- "arthur philpot smythe"
mydf$x2[1] <- "I forgot what letter"
peek(mydf, breaks = 30,
     barargs = list(horiz = TRUE, las = 1))
```

`print.keycheck` *Print out the result of mergeCheck function.*

Description

This is a placeholder for a more elaborate print method to be prepared in the future. Please advise us what might be most helpful.

Usage

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

Arguments

`x`	keycheck output from mergeCheck
`...`	Other arguments

Value

None, side effect if print to screen

Author(s)

Paul Johnson

`print.keyDiff` *Print a keyDiff object*

Description

Print a keyDiff object

Usage

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

Arguments

x	A keyDiff object
...	Other arguments passed through to print

Author(s)

Ben Kite <bakite@ku.edu>

---

print.likert *print method for likert tables*

Description

Nothing fancy here. cat is called on first item in list

Usage

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

Arguments

x	likert object, 1st item will be printed
...	Arguments passed to print method

Value

Nothing

Author(s)

Paul Johnson <pauljohn@ku.edu>

---

qualtricsBlockStack *Create meta data frame to align identical questions*

Description

Qualtrics returns a data frame that has vertical "blocks", one for each "treatment condition" in an experimental condition. Researchers often want to align the questions from the blocks vertically, essentially converting the Qualtrics "wide" format to a "long" format. This is a helper function that identifies questions that may need to be stacked together. The input is a meta data structure (can be retrieved as an attribute from importQualtrics). It will find out which questions are identical and prepare to re-align ("stack") the columns.

Usage

```
qualtricsBlockStack(meta, questionname = "question")
```

Arguments

meta	A meta data structure retrieved from importQualtrics
questionname	Character string for name of column in meta data that holds the questions

Value

A new meta data table that horizontally aligns equivalent questions.

Author(s)

Paul Johnson

---

removeMatches *Remove elements if they are in a target vector, possibly replacing with NA*

Description

If a vector has c("A", "b", "c") and we want to remove "b" and "c", this function can do the work. It can also replace "b" and "c" with the NA symbol.

Usage

```
removeMatches(x, y, padNA = FALSE)
```

Arguments

x	vector from which elements are to be removed
y	shorter vector of elements to be removed
padNA	Default FALSE, Should removed items be replaced with NA values?

Details

If elements in y are not members of x, they are silently ignored.

The code for this is not complicated, but it is difficult to remember. Here's the recipe to remove elements y from x: `x <- x[!x %in% y[y %in% x]]`. It is easy to get that wrong when in a hurry, so we use this function instead. The padNA was an afterthought, but it helps sometimes.

Value

a vector with elements in y removed

Author(s)

Ben Kite <bakite@ku.edu> and Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c("e", "a")
removeMatches(x, y)
y <- c("q", "r", "s")
removeMatches(x, y)
```

---

reverse *Reverse the levels in a factor*

Description

Simple literal reversal. Will stop with an error message if x is not a factor (or ordered) variable.

Usage

```
reverse(x, eol = c("Skip", "DNP"))
```

Arguments

x	a factor variable
eol	values to be kept at the end of the list. Does not accept regular expresssions, just literal text strings representing values.

Details

Sometimes people want to reverse some levels, excluding others and leaving them at the end of the list. The "eol" argument sets aside some levels and puts them at the end of the list of levels.

The use case for the eol argument is a factor with several missing value labels, as appears in SPSS. With up to 18 different missing codes, we want to leave them at the end. In the case for which this was designed, the researcher did not want to designate those values as missing before inspecting the pattern of observed values.

Value

a new factor variable with reversed values

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
## Consider alphabetication of upper and lower
x <- factor(c("a", "b", "c", "C", "a", "c"))
levels(x)
xr1 <- reverse(x)
xr1
## Keep "C" at end of list, after reverse others
xr2 <- reverse(x, eol = "C")
xr2
y <- ordered(x, levels = c("a", "b", "c", "C"))
yr1 <- reverse(y)
class(yr1)[1] == "ordered"
yr1
## Hmm. end of list amounts to being "maximal".
## Unintended side-effect, but interesting.
yr2 <- reverse(y, eol = "C")
yr2
## What about a period as a value (SAS missing)
z <- factor(c("a", "b", "c", "b", "c", "."))
reverse(z)
z <- factor(c(".", "a", "b", "c", "b", "c", "."))
reverse(z)
## How about R NA's
z <- factor(c(".", "a", NA, "b", "c", "b", NA, "c", "."))
z
reverse(z)
z <- ordered(c(".", "a", NA, "b", "c", "b", NA, "c", "."))
z
str(z)
## Put "." at end of list
zr <- reverse(z, eol = ".")
zr
str(zr)
z <- ordered(c(".", "c", NA, "e", "a", "c", NA, "e", "."),
         levels = c(".", "c", "e", "a"))
reverse(z, eol = ".")
reverse(z, eol = c("a", "."))
```

`safeInteger` *If a numeric variable has only integer values, then make it an integer.*

Description

Users often accidentally create floating point numeric variables when they really mean integers, such as c(1, 2, 3), when they should have done c(1L, 2L, 3L). Before running `as.integer()` to coerce the variable, we'd rather be polite and ask the variable "do you mind being treated as if you are an integer?" This function checks to see if the variable is "close enough" to being an integer, and then coerces as integer. Otherwise, it returns NULL. And issues a warning.

Usage

```
safeInteger(
  x,
  tol = .Machine$double.eps,
  digits = 7,
  vmax = .Machine$integer.max,
  verbose = FALSE
)
```

Arguments

x	a numeric variable
tol	Tolerance value. Defaults to Machine$double.eps. See details.
digits	Digits value passed to the zapsmall function. Defaults to 7.
vmax	Maximum value allowed for an integer. Defaults to Machine$integer.max.
verbose	Default FALSE: print warnings about x

Details

First, calculate absolute value of differences between x and `as.integer(x)`. Second, find out if the sum of those differences is smaller than `tol`. If so, then x can reasonably be coerced to an integer.

Be careful with the return. The correct return value for variables that should not be coerced as integer is uncertain at this point. We've tested various strategies, sometimes returning FALSE, NULL, or just the original variable.

Value

Either an integer vector or the original variable

Author(s)

Paul Johnson <pauljohn@ku.edu> and Ben Kite <bakite@ku.edu>

Examples

```
x1 <- c(1, 2, 3, 4, 5, 6)
is.integer(x1)
is.double(x1)
is.numeric(x1)
(x1int <- safeInteger(x1))
is.integer(x1int)
is.double(x1int)
is.numeric(x1int)
x2 <- rnorm(100)
x2int <- safeInteger(x2)
head(x2int)
x3 <- factor(x1, labels = c(LETTERS[1:6]))
x3int <- safeInteger(x3)
```

shorten *Reduce each in a vector of strings to a given length*

Description

This is a simple "chop" at k characters, no fancy truncation at spaces or such. Optionally, this will make unique the resulting truncated strings. That way, truncation at character 4 of "Washington" and "Wash" and "Washingham" will not result in 3 values of "Wash", but rather "Wash", "Wash.1", and "Wash.2"

Usage

```
shorten(x, k = 20, unique = FALSE)
```

Arguments

x	character string
k	integer limit on length of string. Default is 20
unique	Default FALSE

Value

vector of character variables no longer than k

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c("Washington", "Washingham", "Washmylaundry")
shorten(x, 4)
shorten(x, 4, unique = TRUE)
```

starsig *How many stars would we need for this p value?*

Description

Regression table makers need to know how many stars to attach to parameter estimates. This takes p values and a vector which indicates how many stars are deserved. It returns a required number of asterixes. Was named "stars" in previous version, but renamed due to conflict with R base function stars

Usage

```
starsig(pval, alpha = c(0.05, 0.01, 0.001), symbols = c("*", "**", "***"))
```

Arguments

pval	P value
alpha	alpha vector, defaults as c(0.05, 0.01, 0.001).
symbols	The default is c("*", "**", "***"), corresponding to mean that p values smaller than 0.05 receive one star, values smaller than 0.01 get two stars, and so forth. Must be same number of elements as alpha. These need not be asterixes, could be any character strings that users desire. See example.

Details

Recently, we have requests for different symbols. Some people want a "+" symbol if the p value is smaller than 0.10 but greater than 0.05, while some want tiny smiley faces if p is smaller than 0.001. We accomodate that by allowing a user specified vector of symbols, which defaults to c("*", "**", "***")

Value

a character vector of symbols (eg asterixes), same length as pval

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
starsig(0.06)
starsig(0.021)
starsig(0.001)
alpha.ex <- c(0.10, 0.05, 0.01, 0.001)
symb.ex <- c("+", "*", "**", ":)!")
starsig(0.07, alpha = alpha.ex, symbols = symb.ex)
starsig(0.04, alpha = alpha.ex, symbols = symb.ex)
starsig(0.009, alpha = alpha.ex, symbols = symb.ex)
starsig(0.0009, alpha = alpha.ex, symbols = symb.ex)
```

---

statdatKey — *keyFactors: private function that does work for keyTemplateSPSS and key template Stata*

Description

keyFactors: private function that does work for keyTemplateSPSS and key template Stata

Usage

```
statdatKey(datf, datn, long = TRUE)
```

Arguments

datf	Data frame with factors
datn	Numeric data frame
long	Should the result be a long or wide key

---

stringbreak — *Insert "\n" after the k'th character in a string. This IS vectorized, so can receive just one or many character strings in a vector.*

Description

If a string is long, insert linebreak "\n"

Usage

```
stringbreak(x, k = 20)
```

Arguments

x	Character string.
k	Number of characters after which to insert "\n". Default is 20

Details

If x is not a character string, x is returned without alteration. And without a warning

Value

Character with "\n" inserted

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- "abcdef ghijkl mnopqrs tuvwxyz abc def ghi jkl mno pqr stv"
stringbreak(x, 10)
stringbreak(x, 20)
stringbreak(x, 25)
x <- c("asdf asdfjl asfdjkl asdfjklasdfasd", "qrweqwer qwerqwerjklqw erjqwe")
stringbreak(x, 5)
```

`truncsmart` *Cuts a string at a specified linewidth, trying to align cut with a separator*

Description

Some strings are simply too long. We don't want to chop them exactly at, say, 40 characters, if we could allow 42 and chop on a space or other separator. We'd rather chop at 37 if there is a separator, rather than terminate a word exactly at 40. This function shortens them and attempt to cut at a separator, allowing for a user specified fudge-factor (the tol parameter).

Usage

```
truncsmart(
  x,
  target = 20,
  tol = c(5, 3),
  separators = c(" ", "_", ";", ","),
  capwidth = 1
)
```

Arguments

`x`	character or vector of characters
`target`	Goal for result length, in characters
`tol`	number of characters forward/back to check; if single value then only backwards checking
`separators`	characters at which truncation is preferred, such as space or underscore.
`capwidth`	penalty for capital characters

Details

The default capwidth value is 1, so the calculations treat all letters equally. In practice, we notice trouble when some strings are written in ALL CAPS and they are longer than the same information in lower case letters. We have decided to allow a user-specified penalty for capital letters. If each capital counts for, say 1.2 ordinary letters, then we may end up truncating the string on an earlier separator.

There's some approximation here. The capital-penalized widths are calculated for all characters and then we left-shift the target value so that it is equal to the last penalized value that is under the target length. Then the "look to the left" and "look to the right" logic begins. That looking logic ignores the capital letter penalty, it is treating all letters the same.

Value

shorter string truncated at acceptable separators when found

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- "Aasdf asdIasdf fW_asd asd aasjdf_as fasdasdfasdfasd"
truncsmart(x, target = 10, tol = c(5, 2))
truncsmart(x, target = 10, tol = c(1, 4))
truncsmart(x, target = 10, tol = c(5, 2), capwidth = 1.2)
truncsmart(x, target = 20, tol = c(5, 2))
truncsmart(x, target = 20, tol = c(10,10), capwidth = 2)
truncsmart(x, target = 20, tol = c(10,10), capwidth = 3)
truncsmart(x, target = 20, tol = c(10,10), capwidth = 4)
truncsmart(x, target = 20, tol = c(10,10), capwidth = 6)
```

---

updatePackages *Update packages, spot new dependencies, and install them*

Description

Addresses the problem that updates for previously installed R packages may insert new dependencies. R's update.packages does not trigger the installation of packages that are added as new requirements in existing packages.

Usage

```
updatePackages(
  ask = FALSE,
  checkBuilt = TRUE,
  dependencies = c("Depends", "Imports", "LinkingTo"),
  libnew = "/usr/share/R/library/",
  repos = options("repos"),
  ...
)
```

Arguments

ask	If TRUE, asks user to select packages to update
checkBuilt	If TRUE, packages built under earlier versions of R are to be considered 'old'
dependencies	A vector specifying which type of dependencies need to be taken into account. We default to c("Depends", "Imports", "LinkingTo").
libnew	The R library folder into which the new packages must be installed. Defaults to "/usr/share/R/library", which is where EL7 likes those things. To install packages in personal user directory, put libnew = NULL.
repos	A vector of repositories on which to search for packages. Same definition as in R's install.packages or install.packages.
...	additional arguments passed to update.packages and install.packages

Details

This function checks for existence of updates, ascertains whether those packages impose new requirements, and installs any new requirements. Then it conducts the update.

This function is valuable in system maintenance because sometimes existing packages adopt new requirements and the update.packages function does not notice. Another possible case would be that a user accidentally deletes some packages without realizing other packages depend on them.

If this is run as the root/administrator privileged, then base R packages may be updated, but if user is not root/administrator, there will be a warning that packages were not updated because permissions were lacking. For example

"Warning: package 'boot' in library '/usr/lib/R/library' will not be updated.

This warning does not interfere with rest of purpose of this function, since the new dependencies can be installed in a place where the user has privileges, either by specifying libnew as a full directory name or by setting it to NULL, which puts new packages in $R_LIBS_USER

Value

A vector of new packages being installed

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
## Not run:
myrepos <- c("http://rweb.crmda.ku.edu/cran",
             "http://www.bioconductor.org/packages/3.3/bioc")
updatePackages(repos = myrepos)
## libnew defaults to "/usr/share/R/library". Specify NULL
## so that new packages will go to user's directory
updatePackages(libnew = NULL)

## End(Not run)
```

---

varlabTemplate *Create Variable Label Template*

Description

Receive a key, create a varlab object, with columns `name_old` `name_new`, and `varlab`.

Usage

```
varlabTemplate(obj, varlab = TRUE)
```

Arguments

obj	A variable key
varlab	Default NULL, function will start from clean slate, a set of column labels that match name_new. User can specify values by providing a named vector of labels, e.g., c("x1" = "happiness", "x2" = "wealth"), where the names are values to be matched against "name_new" in key.

Details

If not specified, a matrix is created with empty variable labels.

Value

Character matrix with columns name_new and varlab.

Author(s)

Paul Johnson

Examples

```
mydf.path <- system.file("extdata", "mydf.csv", package = "kutils")
mydf <- read.csv(mydf.path, stringsAsFactors=FALSE)
mydf.keywide1 <- keyTemplate(mydf, long = FALSE, sort = FALSE,
                    varlab = TRUE)
attr(mydf.keywide1, "varlab")
mydf.keywide2 <- keyTemplate(mydf, long = FALSE, sort = FALSE,
                    varlab = c("x3" = "fun"))
attr(mydf.keywide2, "varlab")
attr(mydf.keywide2, "varlab") <- varlabTemplate(mydf.keywide2,
                  varlab = c("x5" = "wealth", "x10" = "happy"))
attr(mydf.keywide2, "varlab")
attr(mydf.keywide2, "varlab") <- varlabTemplate(mydf.keywide2,
                  varlab = TRUE)
attr(mydf.keywide2, "varlab")
## Target we are trying to match:
mydf.keylong <- keyTemplate(mydf, long = TRUE, sort = FALSE, varlab = TRUE)
attr(mydf.keylong, "varlab")
attr(mydf.keylong, "varlab") <- NULL
varlabTemplate(mydf.keylong)
attr(mydf.keylong, "varlab") <- varlabTemplate(mydf.keylong,
                   varlab = c("x3" = "wealth", "x10" = "happy"))
attr(mydf.keylong, "varlab")
attr(mydf.keylong, "varlab") <- varlabTemplate(mydf.keylong, varlab = TRUE)
attr(mydf.keylong, "varlab")
```

wide2long *Convert a key object from wide to long format*

Description

This is not flexible, assumes columns are named in our canonical style, which means the columns are named c("name_old", "name_new", "class_old", "class_new", "value_old", "value_new").

Usage

```
wide2long(
  key,
  sep = c(character = "\\|", logical = "\\|", integer = "\\|", factor = "\\|",
    ordered = "[\\|<]", numeric = "\\|")
)
```

Arguments

key	A variable key in the wide format
sep	Default separator is the pipe, "|" for most variables, while ordered accepts pipe or less than, "|<". If the key did not follow those customs, other sep values may be specified for each variable class.

Value

A long format variable key

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
mydf.path <- system.file("extdata", "mydf.csv", package = "kutils")
mydf <- read.csv(mydf.path, stringsAsFactors=FALSE)
## Target we are trying to match:
mydf.keylong <- keyTemplate(mydf, long = TRUE, sort = FALSE)

mydf.key <- keyTemplate(mydf)
mydf.keywide2long <- wide2long(mydf.key)

## rownames not meaningful in long key, so remove in both versions
row.names(mydf.keywide2long) <- NULL
row.names(mydf.keylong) <- NULL
all.equal(mydf.keylong, mydf.keywide2long)
```

writeCSV *Write CSV files with quotes same as MS Excel 2013 or newer*

Description

R's write.csv inserts quotes around all elements in a character vector (if quote = TRUE). In contrast, MS Excel CSV export no longer inserts quotation marks on all elements in character variables, except when the cells include commas or quotation marks. This function generates CSV files that are, so far as we know, exactly the same "quoted style" as MS Excel CSV export files.

Usage

```
writeCSV(x, file, row.names = FALSE)
```

Arguments

x	a data frame
file	character string for file name
row.names	Default FALSE for row.names

Details

This works by manually inserting quotation marks where necessary and turning FALSE R's own method to insert quotation marks.

Value

the return from write.table, using revised quotes

Author(s)

Paul Johnson

Examples

```
set.seed(234)
x1 <- data.frame(x1 = c("a", "b,c", "b", "The \"Washington, DC\""),
      x2 = rnorm(4), stringsAsFactors = FALSE)
x1
fn <- tempfile(pattern = "testcsv", fileext = ".csv")
writeCSV(x1, file = fn)
readLines(fn)
x2 <- read.table(fn, sep = ",", header = TRUE, stringsAsFactors = FALSE)
all.equal(x1,x2)
```

zapspace *Convert leading or trailing white space and tab characters to nothing.*

Description

This eliminates any characters matched by the regular expression '\s' if they appear at the beginning of a string or at its end. It does not alter spaces in the interior of a string. This was created when I was not aware of R's `trimws` and the purpose is the same. On our TODO list, we intend to eliminate this function and replace its use with `trimws`

Usage

```
zapspace(x)
```

Arguments

x	A character vector

Value

If x is a character vector, return is a character vector with leading and trailing white space values removed. If x is not a character vector, an unaltered x is returned.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <-  c("", " ", "   ", "\t", "\t some", "some\t", " space first")
zapspace(x)
```

Index