lx Package

May 14, 2017

Type Package

Title LX generic utilities

Version 1.6

Date 2013-11-22

Author Alain Viari

Maintainer <alain.viari@inria.fr>

Description generic package used in all other LX packages.

License GPL

Depends methods

Imports parallel

Suggests xtable

RoxygenNote 5.0.1

R topics documented:

`HELP.FILE.HANDLE` *lx binary file handle*

Description

a file `handle` (called handle for short) is an R object (see notes) wrapping a binary file connection. (do not confuse with a HELP.FILE.HANDLER)
It is intended to provide a uniform interface for different file types within **lx** and its extensions.

a `handle` contains four mandatory fields:

- filename : a character string containing the file name (or url)
- type : a character string containing the file content type (default is "none")
- mode : a character string containing the file opening mode in lx.open
- connect : an R object containing the file physical connection, the type depends upon file type
- handler : a file handler object. see HELP.FILE.HANDLER

and some optional fields:

- header : file specific information (header, e.g. index table)

`handles` are openened and closed thru lx.open and lx.close functions.

files are opened as binary and lx provides the following binary IO functions to read fixed length bytes from them.

- lx.read.int8 : 8 bits integer(s)
- lx.read.int16 : 16 bits integer(s)
- lx.read.int32 : 32 bits integer(s)
- lx.read.int64 : 64 (actually 53) bits integer(s)
- lx.read.string : basta string

Note

in practice a FILE.HANDLE is currently implemented as an R environment

no write operations yet... time is short.

HELP.FILE.HANDLER *lx binary file handler*

Description

a file handler (called handler for short) is an R object (see notes) wrapping binary file IO operations (do not confuse with a HELP.FILE.HANDLE).

a handler should implement all the following operations:

- accept : tell if file can be handled by this handler
- open : open file
- close : close file
- seek : tell/seek into file
- read : read n bytes from file

API for these functions:

```
accept(filename, mode)
  filename: character string, name of file (see base::open)
  mode: character string, open mode (see base::open)
returns logical TRUE if the file can be handled by this handler

open(filename, mode)
  filename: character string, name of file (see base::open)
  mode: character string, open mode (see base::open)
returns physical connection

close(con)
  con: physical connection
returns (ignored)

seek(con, ...)
  con: physical connection
  ...: optional position (see lx::lx.seek)
returns current position in file as a (numeric) byte offset from the origin

read(con, n=1L)
  con: physical connection
  n: integer The (maximal) number of bytes to be read
returns a raw vector of at most n bytes (less if EOF reached)
```

Note

handlers have to be registered thru lx.register.file.handler.
lx maintain a stack of handlers that are used from top to bottom when opening a file (thru lx.open). the first handler accepting the filename argument (thru accept) is selected.
by default, a R binary file (accepting anything) is registered at the lowest stack position with name "default".

in practice a FILE.HANDLER is currently implemented as an R environment

HELP.LX.OPTIONS *lx options*

Description

Lx options: the following options can be accessed and modified by lx.options

name	descr.	default
verbose	be verbose	TRUE
use.threads	use multithreading	FALSE
mc.cores	number of cores to use	parallel::detectCores()
pg.verbose	lx.lapply use progress bar	TRUE
pg.options.style	progress bar style	1
pg.options.time	add system.time in progress bar	TRUE
tex.dir	temporary dir for tex files	tmp
tex.docclass	class of tex document	report
tex.packages	latex packages	tmp
tex.graphics.driver	tex graphics driver	pdf
tex.driver.options	graphics driver options	NULL
tex.graphics.ext	tex graphics file extension	NULL

note: `tex.driver.options` depends upon `tex.graphics.driver` you should usually change them together. e.g. with `tex.graphics.driver=='jpeg'` you may set `tex.driver.options` to `list(units="in", res=600`

lx *LX utilities*

Description

Generic utilities used in LX packages.

These utilities are currently subdivised in different subpackages:

General programming utilities: functions are prefixed by 'lx'
see `??lx` for more

Latex/pdf reports: functions are prefixed by 'tex'
see `??tex` for more

Details

Package:	lx
Type:	Package
Version:	1.0
Date:	2013-11-22
License:	GPL

Author(s)

Alain Viari

`lx.apply` *apply wrapper*

Description

`lx.apply` will call different apply flavors depending upon the `pg.verbose`.argument:
if pg.verbose is TRUE then
__use apply with progress bar
else
__use apply without progress bar

Usage

```
lx.apply(X, MARGIN, FUN, ..., pg.verbose = lx.options("pg.verbose"))
```

Arguments

`X`	an array, including a matrix. see apply.
`MARGIN`	a vector giving the subscripts which the function will be applied over. see apply.
`FUN`	the function to be applied. see apply.
`...`	anything passed to FUN. see apply.
`pg.verbose`	use progress bar

Note

default value for `pg.verbose` is taken from lx.options and may therefore be controlled globally.

there is no multithread flavor yet, see lx.lapply

Examples

```
n <- 100
lx.options(pg.verbose=TRUE)
system.time(x <- lx.apply(matrix(1:n, ncol=2), 1, function(x) sum(rnorm(5000*sum(x)))))
lx.options(pg.verbose=FALSE)
system.time(x <- lx.apply(matrix(1:n, ncol=2), 1, function(x) sum(rnorm(5000*sum(x)))))
```

`lx.args` *get function actual arguments*

Description

get list of actual arguments of function call. this is mostly useful to 1) retrieve arguments from function with undefined number of arguments and 2) to retrieve actual arguments as symbols (instead of values). this should be called within a function (not from top level). returned arguments are not evaluated but are returned as symbols, constants or language expression.

Usage

```
lx.args(pos = NULL, up.frame = 0L)
```

Arguments

`pos`	position of argument to extract. if NULL returns all arguments, including function name at pos=1. you can retrieve all arguments but function name by using pos=-1.
`up.frame,`	number of frame to go up to get caller. default (up=0) means to retrieve arguments from the function calling `lx.args`; up=1 means the function calling the function calling `lx.args` and so on up to top level.

Value

named list of actual (unevaluated) arguments
or NULL if called from top-level

Note

pos is the position of argument i.e as declared in the function definition not in the function call.

avoid calling `lx.args` as an argument of another function unless you are sure when/where it will actually be evaluated.

don't confuse this function with lx.getargs that handle command line arguments.

Examples

```
foo <- function(...) {
  args <- lx.args(-1)
  print(args)
}
foo(first=1, second=anything, "third", FUN=function(x) x+1)

foo <- function() {
  caller <- lx.args(1, 1)[[1]]
  cat("foo caller is:", caller, "\n")
}
bar <- function() foo()
bar()
```

lx.barplot *barplot with values indicated on top of bars*

Description

barplot with values indicated on top of bars

Usage

```
lx.barplot(x, col.text = "black", pos.text = 1, cex.text = 1, ...)
```

Arguments

x	vector or matrix of values
col.text	text color
pos.text	text position (see text)
cex.text	text magnification
...	arguments to barplot

Value

same as barplot

Examples

```
## Not run:
lx.barplot(c(apple=10, orange=5, banana=15))
lx.barplot(c(apple=10, orange=5, banana=15), col.text='white', col=lx.COLORS)

## End(Not run)
```

lx.binsum *binned sum*

Description

compute the jumping sum of k consecutive elements in vector. this is equivalent to `sapply(seq.int(1,length(x),by=k` but is quicker.

Usage

```
lx.binsum(x, k, drop = FALSE)
```

Arguments

x	vector
k	number of consecutive elements to sum (k >= 1)
drop	drop the last element of result if length(x) is not a multiple of k.

Value

vector of size ceiling(length(x) / k) if drop=FALSE or floor(length(x) / k) if drop=TRUE, containing the sum of k consecutive elements.

Note

if drop=FALSE and length(x) is not a multiple of k the last element will sum up less than k elements.

See Also

lx.rollsum for a rolling (instead of jumping) version

Examples

```
lx.binsum(1:10, 2)                # => 3 7 11 15 19
lx.binsum(rep(1,10), 3)           # => 3 3 3 1
lx.binsum(rep(1,10), 3, drop=TRUE) # => 3 3 3
```

---

lx.close *close file handle*

Description

close file handle

Usage

```
lx.close(handle)
```

Arguments

handle	file handle (opened by lx.open)

Value

handle (modified)

Note

you should reassign handle on return since some internal fields have been modified

lx.color.change *change color hue, saturation or value*

Description

change color hue, saturation or value

Usage

```
lx.color.change(color, dh = 0, ds = 0, dv = 0)
```

Arguments

color	any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or a positive integer i meaning palette()[i]
dh	change in hue (-1 <= dh <= 1)
ds	change in saturation (-1 <= ds <= 1)
dv	change in value (-1 <= dv <= 1)

Value

modified color (as hexadecimal string)

Examples

```
lx.color.change('#ff0000', ds=-0.5)  # => #ff8080
lx.color.change('red', ds=-0.5)      # => #ff8080
```

lx.color.light *lighten or darken color*

Description

lighten or darken color

Usage

```
lx.color.light(color, delta)
```

Arguments

color	any of the three kinds of R color specifications, i.e., either a color name (as listed by colors()), a hexadecimal string of the form "#rrggbb" or "#rrggbbaa" (see rgb), or a positive integer i meaning palette()[i]
delta	amount in delta > 0 : lighten delta < 0 : darken

Value

modified color (as hexadecimal string)

Examples

```
lx.color.light('red', 50)  # => #ff8080
lx.color.light('red', -50) # => #800000
```

`lx.COLORS` *lx global colors*

Description

Lx global colors:

name	descr.
lx.COLORS	a nice looking color scale from d3.scale.category10
lx.RED	semi-transparent red color for plots
lx.GREEN	semi-transparent green color for plots
lx.BLUE	semi-transparent blue color for plots
lx.GREY	semi-transparent grey color for plots
lx.LGREY	semi-transparent lightgrey color for plots
lx.TRANSP	semi-transparent white color for plots

See Also

lx.rainbow

`lx.crossprod` *generalized matrix cross-product*

Description

compute cross-product of conformable matrices using any function of row and col.

Usage

```
lx.crossprod(x, y = NULL, FUN, ...)
```

Arguments

`x`	nxm numerical matrix
`y`	mxn numerical matrix (if NULL then y = t(x))
`FUN`	function called for each row and col as `FUN(row, col,...)`
`...`	other parameters to FUN

Value

nxn numerical matrix. where entry (i,j) is `FUN(x[i,], y[,j],...)`

Note

usual matrix crosproduct corresponds to `FUN=function(x, y) sum(x*y)` (see examples)

See Also

crossprod

Examples

```
x <- matrix(1:10, ncol=2)
# usual crossproduct (not efficient)
lx.crossprod(t(x), x, function(x, y) sum(x*y))
# this is the same as (more efficient):
t(x) %*% x
# and the same as (more efficient):
crossprod(x)
# cross product with max
lx.crossprod(t(x), FUN=function(x, y) max(x*y))
```

`lx.default.file.handler` *create a default file handler*

Description

the default file handler handles usual R binary files.

Usage

```
lx.default.file.handler(name)
```

Arguments

`name`	handler name

Details

a default handler (with name "default") is always added at the lowest position in handlers stack. So, usually you don't need to call this function. It is mostly intented for the case where you want to design your own handler, not starting from scratch. A typical case is to create a default R file handler and to override the `accept` function to specify which files to accept.

lx.density *Kernel Density Estimation*

Description

this is a wrapper around density that workarounds a (known) bug with bw="SJ". The density function generates an error when bw="SJ" and length(x) > 46341. The message is:
`Error in bw.SJ(x, method = "ste") : sample is too sparse to find TD`
the workaround consists in sampling 46341 values.

Usage

```
lx.density(x, bw = "nrd0", ...)
```

Arguments

x	the data from which the estimate is to be computed.
bw	the smoothing bandwidth to be used (see density)
...	any other parameter to density

Value

an object of class "density" (see density)

Note

the seed for sampling is always the same and therefore the procedure remains deterministic.

Examples

```
## Not run:
x <- lx.density(rnorm(50000), "SJ")

## End(Not run)
```

lx.doc *minimalist documentation about object*

Description

return a minimalist description of an object (including lx.info if available) and its subfields (if any)

Usage

```
lx.doc(obj, depth = 2L, maxfields = 10L, .name = deparse(substitute(obj)),
  .prefix = ">", .indent = 0L)
```

Arguments

obj	an R object
depth	maximum depth of subfields recursion
maxfields	maximum number of subfields
.name	name to print in header (usually internal)
.prefix	header prefix (usually internal)
.indent	current indentation level (usually internal)

See Also

lx.info

Examples

```
x <- lx.new('people', name='alain', age=10, hobbies=c('scuba', 'guitar'))
lx.info(x) <- 'this is people class'
lx.info(x$age) <- 'people age'
lx.doc(x)
```

---

lx.dup.handle *duplicate file handle*

Description

duplicate file handle with a new physical connection

Usage

```
lx.dup.handle(handle)
```

Arguments

handle	file handle (opened by lx.open)

Details

this is mostly used in multithreading when accessing the same file (in read mode) by multiple threads.

Value

duplicated file handle

lx.erase *erase previously saved or serialized object*

Description

erase previously saved or serialized object

Usage

```
lx.erase(name)
```

Arguments

name	of first object used in lx.save or lx.serialize

See Also

lx.save lx.serialize

Examples

```
data(lx.iris)
tab <- lx.iris
lx.save(tab)
lx.erase('tab')
```

lx.false *always FALSE function*

Description

this function always returns FALSE, whatever the arguments.

Usage

```
lx.false(...)
```

Arguments

...	anything (ignored arguments)

Value

FALSE

lx.file.ext *filename extension*

Description

filename extension

Usage

```
lx.file.ext(path)
```

Arguments

path	filename

Value

file (name) extension (excluding the leading dot)

Note

only purely alphanumeric extensions are recognized

borrowed from package tools

Examples

```
lx.file.ext("dir/file.more.ext")
```

lx.file.no.ext *filename without extension*

Description

filename without extension

Usage

```
lx.file.no.ext(path)
```

Arguments

path	filename

Value

filename without extension (and the leading dot)

Note

only purely alphanumeric extensions are recognized

borrowed from package tools

Examples

```
lx.file.no.ext("dir/file.more.ext")
```

---

lx.Filter *Filter wrapper*

Description

just like Filter but using lx.lapply internally.

Usage

```
lx.Filter(FUN, X, pg.verbose = lx.options("pg.verbose"),
  use.threads = lx.use.threads(), mc.cores = lx.options("mc.cores"),
  mc.preschedule = TRUE, mc.allow.recursive = FALSE)
```

Arguments

FUN	the function to be applied. see lapply.
X	an array, including a matrix. see lapply.
pg.verbose	use progress bar
use.threads	use multithreading
mc.cores	The number of cores to use. see mclapply.
mc.preschedule	perform prescheduling. see mclapply.
mc.allow.recursive	allow recursive call. see mclapply.

Examples

```
lx.options(pg.verbose=TRUE)
lx.Filter(function(x) x%%2==0, 1:10)
lx.options(pg.verbose=FALSE)
lx.Filter(function(x) x%%2==0, 1:10)
```

---

lx.getargs *commandArgs parsing*

Description

return a named list of command line arguments

Usage: call the R script as
./myfile.R –args myarg=something
or
R CMD BATCH –args myarg=something myfile.R

Then in R do
myargs <- getArgs()
to retrieve a named list of arguments

Usage

```
lx.getargs(verbose = FALSE, defaults = NULL)
```

Arguments

verbose	print verbage to screen
defaults	a named list of defaults, optional

Value

a named list

Note

don't confuse this function with lx.args that handle function arguments.

Borrowed from : http://cwcode.wordpress.com/2013/04/16/the-joys-of-rscript/

Author(s)

Chris Wallace

---

lx.gregexpr *gregexpr wrapper*

Description

similar to gregexpr but argument `text` is a single character string and function returns a (possibly empty) integer vector of match positions.

Usage

```
lx.gregexpr(pattern, text, ...)
```

Arguments

pattern	character string containing a regular expression
text	character string where matches are sought
...	any parameter of gregexpr

Value

integer vector of positions where pattern is found or empty vector if no match

Note

the order of parameters is the same as in gregexpr, i.e. pattern first

See Also

lx.strchr

Examples

```
lx.gregexpr('i', 'mississipi')
lx.gregexpr('[imsp]', 'mississipi')
lx.gregexpr('o', 'mississipi')
```

---

lx.happly *lx.lapply with duplicated file handle*

Description

a version of lx.lapply that duplicates the file handle `handle` in each thread. this is useful when `FUN` needs read access to file since the file descriptor will be different in each thread.
warning: don't use for write access since the write position is unpredictable.

Usage

```
lx.happly(X, FUN, handle, ..., pg.verbose = lx.options("pg.verbose"),
  use.threads = lx.use.threads(), mc.cores = lx.options("mc.cores"))
```

Arguments

`X`	an array, including a matrix. see lapply.
`FUN`	the function to be applied. see lapply.
`handle`	a file handle (see HELP.FILE.HANDLE)
`...`	anything passed to FUN. see lapply.
`pg.verbose`	use progress bar
`use.threads`	use multithreading
`mc.cores`	The number of cores to use. see mclapply.

Note

user's function `FUN` has the form `FUN(x, handle, ...)`

---

lx.head *head of vector*

Description

same as head but limited to vectors and quicker

Usage

```
lx.head(x, n = 1L)
```

Arguments

x	vector
n	if positive take n first elements, if negative take all but n last elements

Value

vector of size n if n >= 0, or of size `length(x) - n` if n < 0

See Also

head lx.tail

Examples

```
lx.head(1:10, 2)  # => 1 2
lx.head(1:10, -2) # => 1 2 3 4 5 6 7
```

---

lx.hist *histogram plot with density*

Description

histogram plot with density

Usage

```
lx.hist(x, col.density = "blue", ...)
```

Arguments

x	vector of values for which the histogram is desired
col.density	density line color
...	arguments to hist

Value

same as hist

Examples

```
## Not run:
lx.hist(rnorm(1000))

## End(Not run)
```

lx.in *quicker* %in% *for integer vectors*

Description

works as %in% but much quicker and for integer vectors only.

Usage

```
lx.in(x, table, .threshold = 500)
```

Arguments

x	integer vector: the values to be matched
table	integer vector: the values to be matched against. (better if sorted increasingly)
.threshold	internal threshold on table length to switch between standard %in% and quicker algorithm.

Value

A logical vector, indicating if a match was located for each element of x.

Note

table should be sorted increasingly. if not it will be sorted internally.

Examples

```
lx.in(1:10, 3:5)
lx.in(5e6, 1:1e7)
```

lx.info *set/get information on object*

Description

to get object information:
`lx.info(obj)`
to set object information:
`lx.info(obj) <- 'value'`

Usage

```
lx.info(obj)

lx.info(obj) <- value
```

Arguments

obj	an R object
value	option value

See Also

lx.doc

Examples

```
x <- 1:3
lx.info(x) <- 'my array'
lx.info(x)
lx.doc(x)
```

---

lx.iris *nominal iris dataframe*

Description

this is a version of iris with 3 nominal columns:
species=c('setosa', 'versicolor', 'virginica')
petal=c('small', 'large')
sepal=c('small', 'large')
used for testing and examples.

---

lx.key.trans *translate keys fom dictionnary*

Description

translate keys in vector x to values from dictionnary dict

Usage

```
lx.key.trans(x, dict)
```

Arguments

x	key or vector of keys
dict	dictionnary (list indexed by keys, plus optional default value)

Examples

```
lx.key.trans('red', c(red='rouge', blue='bleu'))
lx.key.trans(c('red', 'blue'), c(red=1, green=2, blue=3))
lx.key.trans(c('red', 'pink'), c(red=1, green=2, blue=3))
lx.key.trans(c('red', 'pink'), c(red=1, green=2, blue=3, 0))
```

`lx.lapply` *lapply wrapper*

Description

`lx.lapply` will call different lapply flavors depending upon the `use.threads` and `pg.verbose`. arguments:
if use.threads is TRUE then
__use mclapply
else if pg.verbose is TRUE then
__use lapply with progress bar
else
__use lapply without progress bar

Usage

```
lx.lapply(X, FUN, ..., pg.verbose = lx.options("pg.verbose"),
  use.threads = lx.use.threads(), mc.cores = lx.options("mc.cores"),
  mc.preschedule = TRUE, mc.allow.recursive = FALSE)
```

Arguments

`X`	an array, including a matrix. see lapply.
`FUN`	the function to be applied. see lapply.
`...`	anything passed to FUN. see lapply.
`pg.verbose`	use progress bar
`use.threads`	use multithreading
`mc.cores`	The number of cores to use. see mclapply.
`mc.preschedule`	perform prescheduling. see mclapply.
`mc.allow.recursive`	allow recursive call. see mclapply.

Note

default value for `use.threads` and `pg.verbose` are taken from lx.options and may therefore be controlled globally.

See Also

lx.happly for a version dulpicating file handle.

Examples

```
n <- 100
lx.options(pg.verbose=TRUE)
system.time(x <- lx.lapply(1:n, function(x) rnorm(5000*x)))
lx.options(pg.verbose=FALSE)
system.time(x <- lx.lapply(1:n, function(x) rnorm(5000*x)))
```

lx.lazy *conditionaly restore or set object*

Description

conditionaly restore or set object depending upon previously saved value

Usage

```
lx.lazy(object, setfun, ..., force = FALSE)
```

Arguments

object	to set (existing or not)
setfun	function to set object
...	arguments of setfun
force	force reexecution of setfun even if dump file does exist

Details

if file <object>.RData does exist then restore value from this file (unless force=TRUE) else execute setfun(...) and save result into dump file for later retrieval.

Examples

```
foo <- function(n) sum(rnorm(n))  # dummy
n <- 100000000
system.time(lx.lazy(mynorm, foo, n)) # 10 s
system.time(lx.lazy(mynorm, foo, n)) # <1 s
```

lx.list2str *convert named list to character string*

Description

convert named list to character string of the form "key1<sep>value1<collapse>key2<sep>value2...."

Usage

```
lx.list2str(lst, sep = "=", collapse = ", ", quote = NA, ...)
```

Arguments

lst	named list
sep	separator between key and value
collapse	sep separator elements
quote	character to use for quoting character values(use NA to prevent quoting)
...	arguments to format for formating elements

Value

character string

Note

list values should be atomic. vectors are printed as '[x,y,...]'. recursive lists or matrices are not (yet) handled.

Examples

```
ll <- list(a=1, b="you", c=1:3)
lx.list2str(ll)
```

---

lx.loess *smooth data using local polynomial regression*

Description

this is a wrapper around loess

Usage

```
lx.loess(x, y = NULL, span = 0.75, ...)
```

Arguments

x	vector of abscissa if y != NULL or time series values if y == NULL
y	vector of values
span	parameter which controls the degree of smoothing (see loess)
...	other parameters to loess

Details

if just x is provided (i.e. y == NULL) then use x as values and seq_along(x) as abscissa.

Value

named list with following fields
x : the original abscissa (see Details)
y : the smoothed values
loess : raw result from loess

See Also

loess

Examples

```
x <- hist(rnorm(5000), breaks='fd', plot=FALSE)
lx.loess(x$mids, x$counts)
```

lx.Map *Map wrapper*

Description

just like Map but using lx.mapply internally

Usage

```
lx.Map(FUN, ..., use.threads = lx.use.threads())
```

Arguments

FUN	the function to be applied. see mapply.
...	arguments to vectorize over. see mapply.
use.threads	use multithreading

lx.mapply *mapply wrapper*

Description

lx.mapply will call different mapply flavors depending upon the use.threads.argument:
if use.threads is TRUE then
__use mcmapply
else
__use mapply (without progress bar)

Usage

```
lx.mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE,
  use.threads = lx.use.threads(), mc.cores = lx.options("mc.cores"),
  mc.preschedule = TRUE)
```

Arguments

FUN	the function to be applied. see mapply.
...	arguments to vectorize over. see mapply.
MoreArgs	a list of other arguments to FUN. see mapply.
SIMPLIFY	see mapply.
USE.NAMES	see mapply.
use.threads	use multithreading
mc.cores	The number of cores to use. see mclapply.
mc.preschedule	perform prescheduling. see mclapply.

Note

default value for pg.verbose is taken from lx.options and may therefore be controlled globally.

default values for SIMPLIFY and USE.NAMES is TRUE as in mapply, this usually slow down process considerably, you should consider using FALSE instead to speed up computations.

there is no progress bar flavor yet

Examples

```
n <- 100
system.time(x <- lx.mapply(function(x, y) rnorm(5000*(x+y)), 1:n, 1:n))
system.time(x <- lx.mapply(function(x, y) rnorm(5000*(x+y)), 1:n, 1:n, use.threads=TRUE))
```

---

lx.maxima *get maxima of a curve*

Description

get maxima of a curve

Usage

```
lx.maxima(x, span = 0.2, eps = 0.1, ...)
```

Arguments

x	vector of numerical values
span	pre-smoothing parameter (see details)
eps	min ratio between highest and other reported maxima (see details)
...	other parameters to loess

Details

if span > 0 then x is first smoothed using loess with parameter span. Then maxima are searched for. If eps >= 0 then x is expected to be a vector of positive values (typically a distribution) and only maxima with height >= eps * max(height) are returned. If eps < 0 then all maxima are returned (x can be any curve with positive and negative values)

Value

vector of indices of maxima (ordered by decreasing heights)

Note

the technique used to locate extrema is quite simple (change of sign of the derivative) and can easily be fooled by noisy data. this is the reason why data is first smoothed. this function displays better behavior on regular distributions provided by lx.density.

See Also

lx.peaks

Examples

```
x <- lx.density(c(rnorm(100, -1, 0.5), rnorm(100, 1, 0.5)))
x$x[lx.maxima(x$y)]
```

---

lx.mfrow *determine best* par(mfrow=c(x,y)) *layout to fit for n plots*

Description

determine best par(mfrow=c(x,y)) layout to fit for n plots

Usage

```
lx.mfrow(n)
```

Arguments

n	number of plots

See Also

par

Examples

```
opar <- par(no.readonly=TRUE)
par(mfrow=lx.mfrow(10))
# ... plots
par(opar)
```

---

lx.mixin *mixin named lists*

Description

replace key values in lout by key values in lin.

Usage

```
lx.mixin(lout, lin, keys = intersect(names(lout), names(lin)))
```

Arguments

lout	named list
lin	named list
keys	character array of keys to replace. default is intersect(names(lout), names(lin))

Details

this is equivalent to

```
lout[keys] <- lin[keys]
```

and is typically used to override formal arguments by user's arguments to allow specifying only part of a list argument (see example below).

Value

named list

Examples

```
# replace
lx.mixin(list(a=1, c=3), list(a=10, b=20))

# replace and add
lx.mixin(list(a=1, c=3), lin<-list(a=10, b=20), names(lin))

# select
lx.mixin(NULL, lin<-list(a=10, b=20), intersect(names(lin), c("a", "c")))

# typical usage with function args
foo <- function(arg=list(a=1, b="foo")) {
   dft <- formals()
   arg  <- lx.mixin(dft$arg, arg)
   arg
}
foo(list(a=2)) # => list(a = 2, b = "foo")
```

---

lx.napply *mapply with names*

Description

`lx.napply(X, FUN, ...` is functionally equivalent to:
`lx.mapply(FUN, names(X), X, ...)`
and ensures that names(X) is not NULL
FUN should therefore take at least two arguments FUN(name, x, ...)

Usage

```
lx.napply(X, FUN, ..., use.threads = FALSE)
```

Arguments

X	an array
FUN	the function to be applied. see mapply.
...	arguments to vectorize over. see mapply.
use.threads	use multithreading

Note

as an exception in the `lx.?apply` family, the default behavior is no thread

default values for `SIMPLIFY` and `USE.NAMES` is `TRUE` as in lx.mapply, this usually slow down process considerably, you should consider using `FALSE` instead to speed up computations.

Examples

```
ll <- list(a=1, b=2, c=3)
lx.napply(ll, function(nam, val) paste(nam, val, sep="."))
```

`lx.new` *create an S3 instance of given class*

Description

create an S3 instance of given class

Usage

```
lx.new(classname, ...)
```

Arguments

`classname`	string or string vector of classnames
`...`	anything accepted by list

Value

a named list of `...` fields with `classname` class attribute

See Also

class

Examples

```
x <- lx.new('people', name='alain', age=10)
class(x)
x$age
```

lx.open *open (binary) file*

Description

open (binary) file

Usage

```
lx.open(filename, mode = "rb", .handler = NULL)
```

Arguments

filename	file name
mode	open mode (see HELP.FILE.HANDLE)
.handler	force use of given handler name (internal use only)

Value

file handle

Note

if you force handler (thru .handler) the handler$accept test function will not be called.

See Also

HELP.FILE.HANDLE

lx.options *set or get lx options*

Description

set or get lx options

Usage

```
lx.options(..., default = NULL)
```

Arguments

...	arguments of the form key or key=value (see details)
default	default value for retrieving a single key (see details)

Details

Set form: `lx.options(key=value [, key2=value, ...])`
will set option key to value
note: to remove a key then just set its value to NULL (`lx.options(key=NULL)`

Get form: `lx.options(key, [key2, ... ] [, default=value])`
return the value(s) associated to key(s)
note that key is either a symbol or a string, see examples below.
a special form for retrieving a single key with a default value is:
`lx.options(key, default=value)`
finally, if called without argument:
`lx.options()`, then the function returns the list of all current options.

Returned value: for set/get of **single key**: returns the associated value or (in get mode) `default` or `NULL` if key cannot be found
for set/get of **multiple keys**: returns a named list of key:value.
in get mode, if the key is not found, then the corresponding name is `<NA>` and value `NULL`

note that set and get mode can be intermixed (although this is a bit strange to do)

Value

for set/get of **single key**: returns the associated value or (in get mode) `default` or `NULL` if option cannot be found and has no default
for set/get of **multiple keys**: returns a named list of key:value. (note: in get mode if the key is not found, the name is `<NA>` and value `NULL`)

Note

as for options, key is a (uninterpreted) symbol or a character string.
(for developpers: to use a variable value, use a do.call)

see HELP.LX.OPTIONS for a list of current builtin options

Examples

```
lx.options(foo=123, bar=234)
lx.options(foo)
lx.options('foo')
lx.options(foobar, default=456)
lx.options(foo=NULL)
lx.options(foo, default=pi)
lx.options()
```

`lx.out` *printout information on stderr*

Description

prints arguments on stderr if `level >= lx.verbose()`

Usage

```
lx.out(..., level = "info", with.mem = FALSE, with.caller = TRUE,
  up.frame = 0L)
```

Arguments

`...`	zero or more objects which can be coerced to character (and which are pasted together with no separator).
`level`	verbose level.
`with.mem`	if TRUE, additionaly printout memory usage
`with.caller`	if TRUE, additionaly printout caller
`up.frame`	if with.caller==TRUE, caller position in call stack (see details).

Details

if `with.mem` = TRUE this fonction will calls `gc()`, and may therefore slow down process.

the `up.frame` parameter indicates the number of frames to go up to get caller. default (up=0) means to retrieve arguments from the function calling `lx.out`; up=1 means the function calling the function calling `lx.out` and so on up to top level.

the `up.frame` parameter is useful if you called `lx.out` from an error reporting function and want to report the caller function (see examples below)

See Also

lx.verbose lx.warn, lx.warnif, lx.stopif

Examples

```
lx.out("pi=", pi, " and e=", exp(1))
lx.out("pi=", pi, " and e=", exp(1), with.mem=TRUE)

my_report <- function(...) lx.out("my report: ", ..., up.frame=1)
foo <- function() { x <- 1; my_report("x=", x) }
foo()
```

`lx.peaks` *get peaks of a curve*

Description

get peaks position and width of a curve

Usage

```
lx.peaks(x, span = 0.2, eps.x = 0.1, eps.dx = eps.x, na.val = NA, ...)
```

Arguments

`x`	vector of numerical values (typically a distribution)
`span`	pre-smoothing parameter (see details)
`eps.x`	eps parameter for x maxima (see details)
`eps.dx`	eps parameter for x derivative maxima (see details)
`na.val`	default width (see details)
`...`	other parameters to loess

Details

if `span > 0` then `x` is first smoothed using loess with parameter `span`. Then, maxima of x are search with lx.maxima using eps.x parameter. Then, foreach maximum, the closest maxima of x derivative are searched for to compute left and right peak boundaries. if no boundary is found then value is set to maximum.pos +/- `na.val`.

Value

list of three components:

```
- pos   :  vector of indices of peaks position (ordered by decreasing heights)
- left  :  vector of indices of peaks left side
- right :  vector of indices of peaks right side
```

Note

if x is a normal distribution then band width is an estimate of 2*sd

the technique used to locate extrema is quite simple (change of sign of the derivative) and can easily be fooled by noisy data. this is the reason why data is first smoothed. this function displays better behavior on regular distributions provided by lx.density.

See Also

lx.maxima

Examples

```
x <- lx.density(c(rnorm(100, -1, 0.3), rnorm(100, 1, 0.3)))
p <- lx.peaks(x$y)
## Not run:
  plot(x)
  abline(v=x$x[p$pos], col=1)
  abline(v=x$x[c(p$left, p$right)], col=2)

## End(Not run)
```

`lx.plot.inset` *add inset plot into current plot*

Description

add a small (inset) plot in current plot

Usage

```
lx.plot.inset(..., inset.pos = c("tr", "tl", "br", "bl"), inset.alpha = 0.5,
  inset.margin = 0.01)
```

Arguments

`...`	same arguments as in generic plot
`inset.pos`	inset position 'tl', 'tr', 'bl', 'br'
`inset.alpha`	inset size factor
`inset.margin`	inset margin

Details

inset position indicate the poistion of the inset as tl:top-left, tr:top-right, bl=bottom-left, br:bottom-right

Examples

```
## Not run:
plot(function(x) {sin(x*10)/x}, col="red", ylab="", main="main-plot")
lx.plot.inset(function(x) {cos(x*20)}, ylab="", xlab="inset", cex.axis=0.5,
              inset.pos='tr', inset.alpha=0.5)

## End(Not run)
```

`lx.prm.beta` *Re-parameterized Beta distribution*

Description

Beta distibution reparameterized with mean and variance

Usage

```
lx.prm.beta(mean, sd)

lx.dbeta(x, mean, sd, ...)

lx.pbeta(x, mean, sd, ...)

lx.rbeta(n, mean, sd, ...)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations
`x`	vector of quantiles
`...`	other parameters to dbeta, pbeta and rbeta
`n`	number of observations

Value

lx.prm.beta returns the standard parameters of R library functions. **lx.dbeta** gives the density, **lx.pbeta** gives the distribution function and **lx.rbeta** generates random deviates.

Functions

- `lx.dbeta`: density function
- `lx.pbeta`: distribution function
- `lx.rbeta`: random deviate

See Also

dbeta, pbeta and rbeta

`lx.prm.binom` *Re-parameterized Binomial distribution*

Description

Binomial distibution reparameterized with mean and variance

Usage

```
lx.prm.binom(mean, sd)

lx.dbinom(x, mean, sd, ...)

lx.pbinom(x, mean, sd, ...)

lx.rbinom(n, mean, sd, ...)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations
`x`	vector of quantiles
`...`	other parameters to dbinom, pbinom and rbinom
`n`	number of observations

Value

lx.prm.binom returns the standard parameters of R library functions. **lx.dbinom** gives the density, **lx.pbinom** gives the distribution function and **lx.rbinom** generates random deviates.

Functions

- `lx.dbinom`: density function
- `lx.pbinom`: distribution function
- `lx.rbinom`: random deviate

See Also

dbinom, pbinom and rbinom

`lx.prm.exp` *Re-parameterized Exponential distribution*

Description

Exponential distibution reparameterized with mean

Usage

```
lx.prm.exp(mean = 1)

lx.dexp(x, mean = 1, ...)

lx.pexp(x, mean = 1, ...)

lx.rexp(n, mean = 1, ...)
```

Arguments

`mean`	vector of means
`x`	vector of quantiles
`...`	other parameters to dexp, pexp and rexp
`n`	number of observations

Value

lx.prm.exp returns the standard parameters of R library functions. **lx.dexp** gives the density, **lx.pexp** gives the distribution function and **lx.rexp** generates random deviates.

Functions

- `lx.dexp`: density function
- `lx.pexp`: distribution function
- `lx.rexp`: random deviate

See Also

dexp, pexp and rexp

`lx.prm.gamma` *Re-parameterized Gamma distribution*

Description

Gamma distibution reparameterized with mean and variance

Usage

```
lx.prm.gamma(mean, sd)

lx.dgamma(x, mean, sd, ...)

lx.pgamma(x, mean, sd, ...)

lx.rgamma(n, mean, sd, ...)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations
`x`	vector of quantiles
`...`	other parameters to dgamma, pgamma and rgamma
`n`	number of observations

Value

lx.prm.gamma returns the standard parameters of R library functions. **lx.dgamma** gives the density, **lx.pgamma** gives the distribution function and **lx.rgamma** generates random deviates.

Functions

- `lx.dgamma`: density function
- `lx.pgamma`: distribution function
- `lx.rgamma`: random deviate

See Also

dgamma, pgamma and rgamma

`lx.prm.nbinom` *Re-parameterized Negative Binomial distribution*

Description

Negative Binomial distibution reparameterized with mean and variance

Usage

```
lx.prm.nbinom(mean, sd)

lx.dnbinom(x, mean, sd, ...)

lx.pnbinom(x, mean, sd, ...)

lx.rnbinom(n, mean, sd, ...)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations
`x`	vector of quantiles
`...`	other parameters to dnbinom, pnbinom and rnbinom
`n`	number of observations

Value

lx.prm.nbinom returns the standard parameters of R library functions. **lx.dnbinom** gives the density, **lx.pnbinom** gives the distribution function and **lx.rnbinom** generates random deviates.

Functions

- `lx.dnbinom`: density function
- `lx.pnbinom`: distribution function
- `lx.rnbinom`: random deviate

See Also

dnbinom, pnbinom and rnbinom

`lx.prm.norm` *Normal distribution*

Description

These functions are equivalent to dnorm, pnorm and rnorm. they are just provided for symetry with other re-parametrized distributions.

Usage

```
lx.prm.norm(mean, sd)

lx.dnorm(x, mean = 0, sd = 1, ...)

lx.pnorm(x, mean = 0, sd = 1, ...)

lx.rnorm(n, mean = 0, sd = 1, ...)
```

Arguments

`mean`	vector of means
`sd`	vector of standard deviations
`x`	vector of quantiles
`...`	other parameters to dnorm, pnorm and rnorm
`n`	number of observations

Value

lx.prm.norm returns the standard parameters of R library functions. **lx.dnorm** gives the density, **lx.pnorm** gives the distribution function and **lx.rnorm** generates random deviates.

Functions

- `lx.dnorm`: density function
- `lx.pnorm`: distribution function
- `lx.rnorm`: random deviate

See Also

dnorm, pnorm and rnorm

`lx.prm.pois` *Poisson distribution*

Description

These functions are equivalent to dpois, ppois and rpois. they are just provided for symetry with other re-parametrized distributions.

Usage

```
lx.prm.pois(mean)

lx.dpois(x, mean, ...)

lx.ppois(x, mean, ...)

lx.rpois(n, mean, ...)
```

Arguments

`mean`	vector of (non-negative) means
`x`	vector of quantiles
`...`	other parameters to dpois, ppois and rpois
`n`	number of observations

Value

lx.prm.pois returns the standard parameters of R library functions. **lx.dpois** gives the density, **lx.ppois** gives the distribution function and **lx.rpois** generates random deviates.

Functions

- `lx.dpois`: density function
- `lx.ppois`: distribution function
- `lx.rpois`: random deviate

See Also

dpois, ppois and rpois

lx.rainbow *rainbow color scale*

Description

similar to rainbow except that colors do not close circularly (i.e. start and end colors are not the same).

Usage

```
lx.rainbow(n, skip = 1/6, ...)
```

Arguments

n	the number of required colors
skip	end portion of color circle to remove (default=1/6)
...	any argument of rainbow

Value

a character vector of colors

Examples

```
lx.rainbow(10)
```

lx.read.char *read character(s)*

Description

read characters from file handle stopping or not on NULL chars.

Usage

```
lx.read.char(handle, n = 1L, skip.null = FALSE, .raw = FALSE,
  .chunk.size = 100L)
```

Arguments

handle	file handle (opened by lx.open)
n	integer, (maximum) number of chars to read
skip.null	logical, skip over NULL (see details)
.raw	logical, skip processing (see details)
.chunk.size	(internal parameter) size of chunks to use when skip.null==FALSE and n < 0

Details

```
if skip.null==FALSE (default)
  if (n >= 0) then input will stop either when
              n chars have been read or a NULL character is found
              or EOF has been reached.
  if (n < 0) then input will stop on first NULL character
             (or EOF).
if skip.null==TRUE
   read will skip over NULL chars until either
   n chars have been read (not counting NULLs)
   or EOF has been reached.

.raw == TRUE assume that the input stream contains
exactly n bytes without embedded NULLs. this will skip
all checking and processing (ignoring skip.null parameter).
this is intended for high speed reading when you know exactly
what is currently in stream.
```

Value

character string

Note

n is an integer, therefore limited to 2,147,483,647

See Also

lx.read.int8 to read bytes (including NULL), lx.read.string as an alternative form.

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
s <- lx.read.char(fh, n=40)
nchar(s)  # 5
lx.rewind(fh)
s <- lx.read.char(fh, n=40, skip=TRUE)
nchar(s, type="bytes")  # 40
lx.close(fh)
```

---

lx.read.int16 *read binary 16 bits integer(s)*

Description

read binary 16 bits integer(s)

Usage

```
lx.read.int16(handle, n = 1L, signed = FALSE, little.endian = TRUE)
```

Arguments

handle	file handle (opened by lx.open)
n	number of integers to read
signed	read signed integer(s)
little.endian	integer encoding is little endian

Value

(vector of) integer(s)

Note

n is an integer, therefore limited to 2,147,483,647

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
x <- lx.read.int16(fh, n=2)
lx.close(fh)
```

---

lx.read.int32 *read binary 32 bits integer(s)*

Description

read binary 32 bits integer(s)

Usage

```
lx.read.int32(handle, n = 1L, signed = FALSE, little.endian = TRUE)
```

Arguments

handle	file handle (opened by lx.open)
n	number of integers to read
signed	read signed integer(s)
little.endian	integer encoding is little endian

Value

(vector of) integer(s) if signed == TRUE else (vector of) double(s) without loss of precision

Note

n is an integer, therefore limited to 2,147,483,647

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
cod  <- lx.read.int32(fh, n=1)
nseq <- lx.read.int32(fh, n=1)
s1 <- lx.read.string(fh)
siz1 <- lx.read.int64(fh, n=1) # 24
crc1 <- lx.read.int32(fh, n=1)
s2 <- lx.read.string(fh)
siz2 <- lx.read.int64(fh, n=1) # 32
crc2 <- lx.read.int32(fh, n=1)
ss1 <- lx.read.char(fh, n=siz1) # "ACGTACGTACGTAAAAACGTACGT"
ss2 <- lx.read.char(fh, n=siz2) # "GCGCGCGCGCGCGCGCGCGCGCGCTTTTTATATATATAX"
lx.close(fh)
```

`lx.read.int64` *read binary 64 (actually 53) bits integer*

Description

read binary 64 (actually 53) bits integer

Usage

```
lx.read.int64(handle, n = 1L, signed = FALSE, little.endian = TRUE)
```

Arguments

`handle`	file handle (opened by lx.open)
`n`	number of integers to read
`signed`	read signed integer(s)
`little.endian`	integer encoding is little endian

Value

(vector of) double(s) with possible precision loss.

Note

since R double have 53 bits precision the return values are only exact for values in range `[-2^53, 2^53] = [-900719925`
and `[0, 2^53] = [0, 9007199254740992] for unsigned long`

n is an integer, therefore limited to 2,147,483,647

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
cod  <- lx.read.int32(fh, n=1)
nseq <- lx.read.int32(fh, n=1)
s1 <- lx.read.string(fh)
siz1 <- lx.read.int64(fh, n=1)
crc1 <- lx.read.int32(fh, n=1)
s2 <- lx.read.string(fh)
```

```
siz2 <- lx.read.int64(fh, n=1)
crc2 <- lx.read.int32(fh, n=1)
ss1 <- lx.read.char(fh, n=siz1)
ss2 <- lx.read.char(fh, n=siz2)
lx.close(fh)
```

---

lx.read.int8 *read binary 8 bits integer(s)*

Description

read binary 8 bits integer(s)

Usage

```
lx.read.int8(handle, n = 1L, signed = FALSE)
```

Arguments

handle	file handle (opened by lx.open)
n	number of integers to read
signed	read signed integer(s)

Value

(vector of) integer(s)

Note

n is an integer, therefore limited to 2,147,483,647

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
x <- lx.read.int8(fh, n=4)
lx.close(fh)
```

---

lx.read.string *read binary C-style or basta string*

Description

a C-style string is null terminated character array
a basta string is binary encoded in the following way

- int32 : string size
- chars+NULL : null terminated (C style) character array

Usage

```
lx.read.string(handle, with.size = TRUE)
```

Arguments

handle	file handle (opened by lx.open)
with.size	if true basta string else simple C-style string

Value

character string

Note

if with.size==FALSE this function is equivalent to `lx.read.char(handle, n=-1L, skip=FALSE)`

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
cod  <- lx.read.int32(fh, n=1)
nseq <- lx.read.int32(fh, n=1)
s1 <- lx.read.string(fh)
siz1 <- lx.read.int64(fh, n=1)
crc1 <- lx.read.int32(fh, n=1)
s2 <- lx.read.string(fh)
siz2 <- lx.read.int64(fh, n=1)
crc2 <- lx.read.int32(fh, n=1)
ss1 <- lx.read.char(fh, n=siz1)
ss2 <- lx.read.char(fh, n=siz2)
lx.close(fh)
```

---

lx.recycle *expand by recycling or shrink vector to given size*

Description

given a vector x either shrink or expand by recycling to size n

Usage

```
lx.recycle(x, n)
```

Arguments

x	vector
n	positive integer

Value

vector of size n

See Also

lx.rotate

Examples

```
lx.recycle(1:3, 6)
lx.recycle(1:10, 6)
lx.recycle(1:10, 0)
```

`lx.regex.quote` *quote special characters used in regex*

Description

quote special characters used in regex

Usage

```
lx.regex.quote(s)
```

Arguments

`s`	string or string vector

Examples

```
lx.regex.quote('the (lazy) dog')
```

`lx.register.file.handler` *register handler*

Description

register handler at topmost position in handlers stack

Usage

```
lx.register.file.handler(handler)
```

Arguments

`handler`	: file handler

Note

if a handler with same name already exists in stack it is first removed and the new handler is added topmost.

lx.remove.file.handler *remove file handler(s)*

Description

remove handler from handlers stack

Usage

```
lx.remove.file.handler(name)
```

Arguments

name	: name(s) of handler(s) to remove

lx.require *check and optionally install package*

Description

check if package is present and install it if necessary

Usage

```
lx.require(package, install = TRUE, load = FALSE, tarball = NULL,
  silent = FALSE, ...)
```

Arguments

package	package name (as string or symbol)
install	perform installation if not installed
load	additionnaly load package
tarball	local tarball where to find package (in that case you should also specify **repos=NULL** in ... parameters
silent	do not print any message (see notes)
...	any parameter of install.packages

Details

first check if package is present in installed.packages. if not and if install is TRUE then install it using install.packages with provided arguments.

Value

invisible(TRUE) if package is (now) available else invisible(FALSE)

Note

`silent` will not silent the (possible) installation phase because its a bad idea to download and install anything without notification. to force complete silence see install.packages options (quietly and verbose).

you may silently check the existence of a package by `lx.require(package, install=FALSE, load=FALSE, silent=T`

---

lx.restore *restore objects from dump file*

Description

restore objects from dump file

Usage

```
lx.restore(name)
```

Arguments

`name`	of first object used in `lx.save`

See Also

`lx.save`

Examples

```
data(lx.iris)
tab1 <- tab2 <- lx.iris
lx.save(tab1, tab2)
remove(tab1, tab2)
lx.restore('tab1')
```

---

lx.rev.dict *reverse dictionnary*

Description

reverse dictionnary

Usage

```
lx.rev.dict(dict)
```

Arguments

`dict`	dictionnary : a list indexed by keys (plus optional default value that is ignored)

Value

reverse dictionary : a list indexed by values

Examples

```
dict <- list(warm=c('red'), cold=c('blue', 'green'), basic=c('red', 'blue', 'green'), 'white')
lx.rev.dict(dict)
# => list(red=c('warm', 'basic'), blue=c('cold', 'basic'), green=c('cold', 'basic'))
```

---

lx.rewind *rewind file handle*

Description

rewind file handle opened by lx.open

Usage

```
lx.rewind(handle)
```

Arguments

handle	file handle (opened by lx.open)

Value

handle (invisible)

Note

this is a shorthand for `lx.seek(handle, 0)`

---

lx.rnd.pop *restore current global random seed*

Description

pop current global random seed (.Random.seed) from internal stack this has to be used (in pairs with lx.rnd.push) to save current random seed and restore it later

Usage

```
lx.rnd.pop()
```

Value

(invisible) restored random seed

Examples

```
set.seed(1)
x <- sample(letters, 3)
set.seed(1)
lx.rnd.push() # same as
set.seed(2)   # lx.rnd.push(2)
y <- sample(letters, 3)
lx.rnd.pop()
z <- sample(letters, 3)
identical(x, z)
```

`lx.rnd.push` *save current global random seed*

Description

push current global random seed (.Random.seed) into internal stack this has to be used (in pairs with lx.rnd.pop) to save current random seed and restore it later

Usage

```
lx.rnd.push(seed)
```

Arguments

`seed`	**optional** parameter, if specified then additionnaly set.seed(seed)

Value

(invisible) current random seed

Examples

```
set.seed(1)
x <- sample(letters, 3)
set.seed(1)
lx.rnd.push() # same as
set.seed(2)   # lx.rnd.push(2)
y <- sample(letters, 3)
lx.rnd.pop()
z <- sample(letters, 3)
identical(x, z)
```

lx.rollsum *rolled sum*

Description

compute the rolling sum of k consecutive elements in vector. this is equivalent to `sapply(seq_along(x), function(i)` but is much quicker since it runs in `O(n)`.

Usage

```
lx.rollsum(x, k, drop = FALSE)
```

Arguments

x	vector
k	number of consecutive elements to sum (k >= 1)
drop	drop the last (k-1) elements of result

Value

vector of size length(x) if (drop==F) or length(x)-k+1 if (drop==F), containing the sum of k consecutive elements.

Note

with drop==F the last (k-1) elements will sum up less than k elements.

See Also

lx.binsum for a jumping (instead of rolling) version

Examples

```
lx.rollsum(1:5, 2)          # => 3 5 7 9 5
lx.rollsum(rep(1,10), 3, TRUE) # => 3 3 3 3 3 3 3 3
```

lx.rotate *rotate vector circularly n times*

Description

rotate vector circularly n times

Usage

```
lx.rotate(x, n = 1L)
```

Arguments

x	vector
n	number of rotations. if > 0 rotate to left else rotate to right

Value

rotated vector

Examples

```
lx.rotate(1:5, 2)  # => 3 4 5 1 2
lx.rotate(1:5, -2) # => 4 5 1 2 3
```

---

lx.rowMaxs *get maximum in rows*

Description

find the maximum for each row of a matrix

Usage

```
lx.rowMaxs(x, ties.method = c("random", "first", "last"), what = c("value",
  "index"))
```

Arguments

x	numerical matrix
ties.method	a character string specifying how ties are handled. (see max.col)
what	a character string specifying what should be returned (see Value)

Value

- if `what == "value"` return a maximal value for each row
- if `what == "index"` return index of a maximal value for each row

Note

with `what == "index"` this function is identical to max.col

See Also

lx.rowMins

lx.rowMins *get minimum in rows*

Description

find the minimum for each row of a matrix

Usage

```
lx.rowMins(x, ties.method = c("random", "first", "last"), what = c("value",
  "index"))
```

Arguments

x	numerical matrix
ties.method	a character string specifying how ties are handled. (see max.col)
what	a character string specifying what should be returned (see Value)

Value

- if `what == "value"` return a minimum value for each row
- if `what == "index"` return index of a minimum value for each row

Note

with `what == "index"` this function is identical to max.col(-x)

See Also

lx.rowMaxs

lx.sample *base::sample wrapper*

Description

this function calls `base::sample(x, size, prob, replace)` except for the case where `prob != NULL and replace ==`
In this case `base::sample` is very slow (actually quadratic in the size of `x` and of `size`. This is replaced by the 'reservoir' technique of Efraimidis & Spirakis (see note)

Usage

```
lx.sample(x, size, replace = FALSE, prob = NULL)
```

Arguments

x	Either a vector of one or more elements from which to choose, or a positive integer. (see sample)
size	a non-negative integer giving the number of items to choose.
replace	should sampling be with replacement
prob	a vector of probability weights for obtaining the elements of the vector being sampled.

Value

a vector of length size with elements drawn from either x or from the integers 1:x

Note

reference: P. Efraimidis and P.Spirakis, Information Processing Letters, 97, 181-185 (2006).

---

lx.sapply *sapply wrapper*

Description

lx.sapply will call different sapply flavors depending upon the pg.verbose.argument:
if pg.verbose is TRUE then
__use sapply with progress bar
else
__use sapply without progress bar

Usage

```
lx.sapply(X, FUN, ..., pg.verbose = lx.options("pg.verbose"))
```

Arguments

X	an array, including a matrix. see lapply.
FUN	the function to be applied. see lapply.
...	anything passed to FUN. see lapply.
pg.verbose	use progress bar

Note

default value for pg.verbose is taken from lx.options and may therefore be controlled globally.

there is no multithread flavor yet, see lx.lapply

Examples

```
n <- 100
lx.options(pg.verbose=TRUE)
system.time(x <- lx.sapply(1:n, function(x) rnorm(5000*x)))
lx.options(pg.verbose=FALSE)
system.time(x <- lx.sapply(1:n, function(x) rnorm(5000*x)))
```

lx.save *save objects into dump file*

Description

save objects into dump file

Usage

```
lx.save(...)
```

Arguments

...	list of objects to save

Details

the dump file name is <obj1>.RData where obj1 is the first object in ...

See Also

lx.restore

Examples

```
data(lx.iris)
tab1 <- tab2 <- lx.iris
lx.save(tab1, tab2)
remove(tab1, tab2)
lx.restore('tab1')
```

lx.saved *tells if object has been previously saved*

Description

tells if object has been previously saved

Usage

```
lx.saved(name)
```

Arguments

name	of first object used in lx.save

See Also

lx.save

Examples

```
data(lx.iris)
tab <- lx.iris
lx.save(tab)
lx.saved('tab')
```

---

lx.scale *linearly rescale numeric vector to specified range.*

Description

rescale numeric vector x to x' from range [f1, f2] to range [t1, t2] according to:
`x' = a.x + b`
where a=(t2-t1)/(f2-f1) and b = t1 - a.f1

Usage

```
lx.scale(x, to = c(0, 1), from = range(x, na.rm = TRUE), zerob = FALSE)
```

Arguments

`x`	numeric vector
`to`	output range (numeric vector of length two)
`from`	input range (numeric vector of length two). if not specified, use the range of x
`zerob`	force b=0. this is useful for rescaling differences

Value

rescaled vector

Examples

```
lx.scale(1:10)
lx.scale(1:10, c(1,10))
```

---

lx.seek *tell/seek file*

Description

reposition file handle opened by lx.open

Usage

```
lx.seek(handle, ...)
```

Arguments

handle	file handle (opened by lx.open)
...	optional position in file

Details

if ... is absent then returns current position (ftell) else set the position to (numeric) argument.

Value

current position (before any move) as a (numeric) byte offset from the origin.

Note

for default (R) handler, position may exceed 32 bit integer.

Examples

```
fh <- lx.open(lx.system.file('samples/test_bin.bst'))
cod  <- lx.read.int32(fh, n=1)
pos <- lx.seek(fh)
nseq <- lx.read.int32(fh, n=1)
lx.seek(fh, 0)
cod  <- lx.read.int32(fh, n=1)
nseq <- lx.read.int32(fh, n=1)
lx.seek(fh, pos)
nseq <- lx.read.int32(fh, n=1)
lx.close(fh)
```

---

lx.serialize *serialize single object into file*

Description

serialize single object into file

Usage

```
lx.serialize(object, name = NULL, ...)
```

Arguments

object	object to serialize
name	file name (without extension) if NULL then use symbol object
...	additionnal options to saveRDS

Details

object is serialized into file named name.rds

See Also

lx.unserialize

Examples

```
data(lx.iris)
tab <- lx.iris
lx.serialize(tab)  # create file \code{tab.rds}
duptab <- lx.unserialize('tab')
```

---

lx.serialized *tells if serialized object does exist*

Description

tells if serialized object does exist

Usage

```
lx.serialized(name)
```

Arguments

name	of object used in lx.serialized

See Also

lx.serialize

Examples

```
data(lx.iris)
tab <- lx.iris
lx.serialize(tab)  # create file tab.Rdata
lx.serialized('tab')
```

---

lx.shift *shift vector n times and pad with fill value*

Description

shift vector n times and pad with fill value

Usage

```
lx.shift(x, n = 1L, fill = NA)
```

Arguments

x	vector
n	number of shifts. if >= 0 shift to left else shift to right
fill	value to fill

Value

shifted vector

Examples

```
lx.shift(1:5, 2, fill=0)  # => 3 4 5 0 0
lx.shift(1:5, -2, fill=0) # => 0 0 1 2 3
```

---

lx.smooth.median *Tukey's median smoothing*

Description

smooth data using iterative Tukey's median smoothing. this is typically intented for piecewise constant data (step function) to smooth signal whuile preserving edges. see http://dx.doi.org/10.1007/BFb0057597

Usage

```
lx.smooth.median(x, k = 3L, tol = 0)
```

Arguments

x	numeric vector, the *dependent* variable to be smoothed
k	vector of successive smoothing windows
tol	tolerance for smoothing iterations (see details)

Details

the procedure runs as follows:

```
foreach window.size in k
  repeat
    x0 <- x
    x  <- runmed(x0, window.size)
  while (rss(x0,x) < tol)
end
```

set tol to Inf to prevent iterations and to 0 (default) for full iterations.

Value

vector of smoothed values of the same length as x

See Also

lx.loess, runmed

Examples

```
x <- c(rnorm(100, -1, 0.1), rnorm(100, 1, 0.1))
sx <- lx.smooth.median(x, c(3, 5, 15, 35))
## Not run:
plot(x)
lines(sx, col=2)

## End(Not run)
```

`lx.stack.is.empty` *tell if stack is empty*

Description

tell if stack is empty

Usage

```
lx.stack.is.empty(stk)
```

Arguments

`stk`	Stack

Value

TRUE if stack is empty

Note

this implementation uses lists and is not efficient. this is mostly intended for storing small number of elements. if you need a better implementation, use the 'rstack' library

Examples

```
stk <- lx.stack.new()
stk <- lx.stack.push(stk, 1)
stk <- lx.stack.push(stk, "two")
stk <- lx.stack.push(stk, NULL)
stk <- lx.stack.push(stk, 1:4)
while (! lx.stack.is.empty(stk)) {
 stk <- lx.stack.pop(stk)
 val <- lx.stack.value(stk)
 lx.out(typeof(val), ": ", val)
}
```

lx.stack.new *new (FILO) stack*

Description

new (FILO) stack

Usage

```
lx.stack.new()
```

Value

empty stack

Note

this implementation uses lists and is not efficient. this is mostly intended for storing small number of elements. if you need a better implementation, use the 'rstack' library

Examples

```
stk <- lx.stack.new()
stk <- lx.stack.push(stk, 1)
stk <- lx.stack.push(stk, "two")
stk <- lx.stack.push(stk, NULL)
stk <- lx.stack.push(stk, 1:4)
while (! lx.stack.is.empty(stk)) {
 stk <- lx.stack.pop(stk)
 val <- lx.stack.value(stk)
 lx.out(typeof(val), ": ", val)
}
```

lx.stack.pop *pop value from stack*

Description

pop top element from stack and store it in stk->value

Usage

```
lx.stack.pop(stk)
```

Arguments

stk	Stack

Value

modified stack

Note

no error is raised if stack is empty

this implementation uses lists and is not efficient. this is mostly intended for storing small number of elements. if you need a better implementation, use the 'rstack' library

Examples

```
stk <- lx.stack.new()
stk <- lx.stack.push(stk, 1)
stk <- lx.stack.push(stk, "two")
stk <- lx.stack.push(stk, NULL)
stk <- lx.stack.push(stk, 1:4)
while (! lx.stack.is.empty(stk)) {
 stk <- lx.stack.pop(stk)
 val <- lx.stack.value(stk)
 lx.out(typeof(val), ": ", val)
}
```

---

lx.stack.push *push value into stack*

Description

push value into stack

Usage

```
lx.stack.push(stk, value)
```

Arguments

stk	Stack
value	any R object (including NULL)

Value

modified stack

Note

this implementation uses lists and is not efficient. this is mostly intended for storing small number of elements. if you need a better implementation, use the 'rstack' library

Examples

```
stk <- lx.stack.new()
stk <- lx.stack.push(stk, 1)
stk <- lx.stack.push(stk, "two")
stk <- lx.stack.push(stk, NULL)
stk <- lx.stack.push(stk, 1:4)
while (! lx.stack.is.empty(stk)) {
 stk <- lx.stack.pop(stk)
```

```
 val <- lx.stack.value(stk)
 lx.out(typeof(val), ": ", val)
}
```

`lx.stack.value` *get last popped value from stack*

Description

get last popped value from stack

Usage

```
lx.stack.value(stk)
```

Arguments

`stk`	Stack

Value

last popped value

Note

this implementation uses lists and is not efficient. this is mostly intended for storing small number of elements. if you need a better implementation, use the 'rstack' library

Examples

```
stk <- lx.stack.new()
stk <- lx.stack.push(stk, 1)
stk <- lx.stack.push(stk, "two")
stk <- lx.stack.push(stk, NULL)
stk <- lx.stack.push(stk, 1:4)
while (! lx.stack.is.empty(stk)) {
 stk <- lx.stack.pop(stk)
 val <- lx.stack.value(stk)
 lx.out(typeof(val), ": ", val)
}
```

lx.stopif *stop execution if condition is true*

Description

stop execution if condition is true

Usage

```
lx.stopif(condition, ..., level = "error", up.frame = 0L, trace = TRUE)
```

Arguments

condition	boolean value
...	anything accepted by lx.out
level	see lx.out
up.frame	see lx.out
trace	printout traceback

Value

invisible(condition)

See Also

lx.out, lx.warn, lx.warnif

Examples

```
## Not run:
x <- -1
lx.stopif(x < 0, 'x=', x, ' is negative')

## End(Not run)
```

lx.strchr *find occurences of character(s) in string*

Description

find occurences of character(s) in string

Usage

```
lx.strchr(s, chr = "")
```

Arguments

s	string where matches are sought
chr	string containing any character to find

Value

integer vector of positions where any char from chr is found or empty vector if no match

See Also

lx.gregexpr

Examples

```
lx.strchr('mississipi', 'i')
lx.strchr('mississipi', 'imsp')
lx.strchr('mississipi', 'o')
```

---

lx.strrev *reverse string*

Description

reverse string

Usage

```
lx.strrev(s)
```

Arguments

s	string or string vector

Value

vector of reversed strings

Examples

```
lx.strrev('the lazy dog')
```

lx.strsplit *split string around regexp*

Description

this is awrapper around strsplit to simplify call when both s and split are simple strings.

Usage

```
lx.strsplit(s, split = "( |\t)+", trim = TRUE, ...)
```

Arguments

s	string (or string vector)
split	string (or string vector) containing regular expression(s) (unless you specify `fixed = TRUE` in ...) to use for splitting. see strsplit
trim	also trim results using lx.strtrim
...	other argument to strsplit

Details

the type of result depends upon the length of s. if s is a vector (length > 1) then this function behave exactly as strsplit and returns a list of string vectors. if s is a simple string (as this is often the case) then this function returns a simple string vector.

Value

vector of strings (or list of vectors of strings, see Details)

Examples

```
lx.strsplit('the lazy dog')
# is same as
strsplit('the lazy dog', ' ')[[1]]
lx.strsplit(c('the lazy', 'black dog'))
lx.strsplit('the lazy dog', 'y')
lx.strsplit('the lazy dog', 'y', fixed=TRUE)
```

lx.strtrim *trim leading or trailing blanks (spaces or tabs) in string*

Description

trim leading or trailing blanks (spaces or tabs) in string

Usage

```
lx.strtrim(s, side = c("both", "left", "right"))
```

Arguments

s	string or string vector
side	c('left', 'right', 'both') which end to trim

Examples

```
paste('>', lx.strtrim(' the lazy dog  '), '<', sep='')
```

---

lx.strtrunc *truncate string*

Description

truncate long string as "abcd...wxyz", usually for pretty printing

Usage

```
lx.strtrunc(s, width, between = "...")
```

Arguments

s	string
width	maximum width
between	between string

Examples

```
lx.strtrunc("the lazy dog", 6)
```

---

lx.summary *a prettier* summary *for dataframes*

Description

a prettier summary for dataframes

Usage

```
lx.summary(df)
```

Arguments

df	a dataframe

Note

only works with numeric columns

Examples

```
lx.summary(iris[,-5])
```

lx.sysinfo *wrapper to Sys.info*

Description

get system and user information

Usage

```
lx.sysinfo(...)
```

Arguments

...	0 or more keys to Sys.info (if empty, then return all entries)

Value

a (possibly empty) vector of entries of Sys.info

Note

result may be NULL if Sys.info is not implemented

Examples

```
x <- lx.sysinfo()
x <- lx.sysinfo('user', 'machine')
```

lx.system.file *find names of R system files*

Description

this is nearly identical to `system.file` but does not check at all for file existence.

Usage

```
lx.system.file(filename = "", package = "lx")
```

Arguments

filename	filename to append to path
package	package name

Value

<packageRootDir>/[filename]

lx.table — *count or contingency table with specified levels*

Description

same as table but force levels. (call factor on ... then call table)

Usage

```
lx.table(..., levels = NULL)
```

Arguments

...	same as table
levels	levels to force

Note

table is equivalent to `lx.table(...)`

See Also

lx.tables

Examples

```
x <- round(runif(10, 1, 10))
lx.table(x, levels=1:10)
y <- round(runif(10, 1, 10))
lx.table(x, y, levels=1:10)
```

lx.table.bycols — *k-dimensional contingency table(s) from columns*

Description

k-dimensional contingency table(s) from columns

Usage

```
lx.table.bycols(tab, by = colnames(tab))
```

Arguments

tab	a table (matrix or dataframe) of nominal values
by	a list of column indices or names (of length k >= 2)

Value

k-dimensional contingency table, defined as the contigency table of the first two columns stratified on other columns

Note

this is the same as `table(tab[,by])`

See Also

lx.tables

Examples

```
data(lx.iris)
tab <- lx.iris
# default: stratify by last dimensions (here sepal)
lx.table.bycols(tab)
# stratify by species (column index 1)
lx.table.bycols(tab, c(2,3,1))
# same as
lx.table.bycols(tab, c('petal', 'sepal', 'species'))
```

`lx.table.bycoocs` *cross table of co-occurences in sets*

Description

cross table of co-occurences in sets

Usage

```
lx.table.bycoocs(lsets, levels = NULL)
```

Arguments

`lsets`	a list of sets Si
`levels`	elements to keep (if NULL keeps all elements)

Value

n x n table where n = |union(Si)| and entry T[i,j] equals the number of co-occurences of elements i and j in sets. diagonal elements are the number of occurences of each element.

See Also

lx.tables

Examples

```
l <- list(c('a'), c('a', 'b'), c('a', 'b', 'c'))
lx.table.bycoocs(l)
```

lx.table.byfacts *make contingency table(s) by factors in columns*

Description

make contingency table(s) by factors in columns

Usage

```
lx.table.byfacts(tab, by = colnames(tab))
```

Arguments

tab	a table (matrix or dataframe) of nominal values
by	a list of column indices or names (of length k >= 1)

Value

a 2-dimensional contingency table T of size f x k, where f = number of levels in `tab[,by]` and T[i,j] equals to the number of rows with factor i in column j

See Also

lx.tables

Examples

```
data(lx.iris)
tab <- lx.iris
lx.table.byfacts(tab)
```

lx.table.bymsets *cross table of multisets intersections*

Description

cross table of multisets intersections

Usage

```
lx.table.bymsets(lsets, as.card = FALSE)
```

Arguments

lsets	a list of n named multisets Mi each multiset can be specified as a list with repeated elements e.g. c('a', 'a', 'b', 'c', 'b') or, with as.card=TRUE, as a numeric list e.g. c(a=2, b=2, c=1)
as.card	use cardinality in multiset definition

Value

n x m table T, where m = number of different elements in all Mi and T [i, j] equals to the number of elements j in multiset i

See Also

lx.tables

Examples

```
lsets <- list(A=c('a', 'a', 'b'), B=c('a'), C=c('a', 'b', 'b', 'c'), D=c('c'))
lx.table.bymsets(lsets)
lsets <- list(A=c(a=2, b=1), B=c(a=1), C=c(a=1, b=2, c=1), D=c(c=1))
lx.table.bymsets(lsets, as.card=TRUE)
```

---

lx.table.bypairs *pairwise contingency table(s) from columns*

Description

pairwise contingency table(s) from columns

Usage

```
lx.table.bypairs(tab, by = colnames(tab))
```

Arguments

tab	a table (matrix or dataframe) of nominal values
by	a list of column indices or names (of length k >= 2)

Value

list of C(k, n) contingency tables crossing all pairs of columns in by

See Also

lx.tables

Examples

```
data(lx.iris)
tab <- lx.iris
lx.table.bypairs(tab)
```

lx.table.bysets *cross table of sets intersections*

Description

cross table of sets intersections

Usage

```
lx.table.bysets(lsets)
```

Arguments

lsets	a list of n named sets Si

Value

n x n table where entry T[i,j] equals the size of Si inter Sj

Note

aka `outer(lsets, lsets, function(x,y) { length(intersect(x,y)) })` but `outer` does not work for this

See Also

lx.tables

Examples

```
lx.table.bysets(list(a=c(1,2), b=c(1,3), c=c(2,3,4)))
# this the same as
lx.table.bysets(list(a=c(1,2,2), b=c(1,3), c=c(2,3,4,3)))
# since a, b, c are considered to as sets
```

lx.table.margins *add margins (total) to a 2-dimensional contingency table*

Description

add margins (total) to a 2-dimensional contingency table

Usage

```
lx.table.margins(ctab, margin = c(1, 2))
```

Arguments

ctab	2-dimensional contingency table
margin	which margin to add 1 (total by rows), 2 (total by columns) c(1,2) both margins 0 do nothing

See Also

lx.tables

Examples

```
data(lx.iris)
lx.table.margins(lx.table.bycols(lx.iris, by=1:2))
```

`lx.tables` *cross tabulation and contigency tables*

Description

there are several lx function to built contingency tables.

- lx.table : count or contingency table with full levels
- lx.table.bycols : k-dimensional contingency table(s) from columns
- lx.table.bypairs : pairwise contingency table(s) from columns
- lx.table.byfacts : count or contingency table by factors in columns
- lx.table.bysets : cross table of sets intersections
- lx.table.bymsets : cross table of multisets intersections
- lx.table.bycoocs : cross table of co-occurences in sets
- lx.table.margins : add margins (total) to a 2-dimensional contingency table

---

`lx.tail` *tail of vector*

Description

same as tail but limited to vectors and quicker

Usage

```
lx.tail(x, n = 1L)
```

Arguments

x	vector
n	if positive take n last elements, if negative take all but n first elements

Value

vector of size n if n >= 0, or of size `length(x) - n` if n < 0

See Also

tail lx.head

Examples

```
lx.tail(1:10, 2)  # => 9 10
lx.tail(1:10, -2) # => 3 4 5 6 7 8 9 10
```

---

lx.tapply *tapply-like with specified levels*

Description

similar to tapply but force levels and call FUN on empty subsets.

Usage

```
lx.tapply(x, by, FUN, levels = NULL, ...)
```

Arguments

x	an atomic object, typically a vector
by	a vector of factors (or anything that can be converted to factors) of same length as x
FUN	the function to be applied
levels	levels to force
...	optional arguments to FUN

Value

a vector of the same length as levels (or levels(by) if levels==NULL), each value is the result of calling FUN on the subset of x at given level.

See Also

lx.tables

Examples

```
xval  <- c( 1,   2,   3,   0,   1 )
group <- c('a', 'b', 'c', 'a', 'a')
lx.tapply(xval, by=group, length)
lx.tapply(xval, by=group, sum)
lx.tapply(xval, by=group, length, levels=c('a', 'b', 'd'))
```

lx.tobase *get number representation*

Description

get base representation of a positive numeric (accurate up to 53 bits)

Usage

```
lx.tobase(num, base = 16L, prefix = "")
```

Arguments

num	positive (unsigned) numeric value
base	representation base (2 <= base <= 36)
prefix	prefix to add (e.g. '0x' for hexadecimal)

Value

string

Examples

```
lx.tobase(123, prefix="0x")  # 0x7b
```

lx.traceback *print call traceback*

Description

print call traceback on stderr

Usage

```
lx.traceback(level = "debug", up.frame = 0L)
```

Arguments

level	see lx.out
up.frame	see lx.out

Value

invisible(traceback list from traceback)

lx.true *always TRUE function*

Description

this function always returns TRUE, whatever the arguments.

Usage

```
lx.true(...)
```

Arguments

...	anything (ignored arguments)

Value

TRUE

lx.unserialize *unserialize object from* name.rds

Description

unserialize object from name.rds

Usage

```
lx.unserialize(name, ...)
```

Arguments

name	of object used in lx.serialized
...	additionnal options to readRDS

See Also

lx.serialize

Examples

```
data(lx.iris)
tab <- lx.iris
lx.serialize(tab)  # create file tab.Rdata
duptab <- lx.unserialize('tab')
```

lx.upper2vect *get matrix upper triangle as vector*

Description

get matrix upper triangle as vector

Usage

```
lx.upper2vect(x, diag = FALSE, bycol = TRUE)
```

Arguments

x	nxn square numerical matrix
diag	logical should the diagonal be included?
bycol	logical order result by columns

Value

numerical vector of matrix upper triangle

Note

if matrix is not square, a warning is emitted but result (on the smaller dimension) is still returned

See Also

upper.tri, lx.vect2upper

Examples

```
x <- matrix(1:16, ncol=4)
y <- lx.upper2vect(x, diag=TRUE)
xx <- lx.vect2upper(y, diag=TRUE)
```

lx.use.threads *tell if lx can use multithreading*

Description

multithreading is on when lx.options(use.threads) is TRUE and lx.options(mc.cores) > 1

Usage

```
lx.use.threads()
```

Note

threads are implemented in the `parallel` library by using `fork`, and will run parallel processes even if lx.options(mc.cores) is greater than the actual number of cores. In this case the overall process may slow donw considerably.

`lx.vect2upper` *make matrix from upper triangle vector*

Description

this is the inverse of lx.upper2vect

Usage

```
lx.vect2upper(x, diag = FALSE, bycol = TRUE, symmetrize = TRUE,
  na.val = NA)
```

Arguments

`x`	numerical vector
`diag`	logical is the diagonal included in x? (see lx.upper2vect)
`bycol`	logical is x ordered by columns? (see lx.upper2vect)
`symmetrize`	logical should result be symmetrized
`na.val`	numerical value for NA (see note)

Value

numerical matrix

Note

undefined values (i.e. diagonal if diag=FALSE and lower matrix if symmetrize=FALSE) are set to na.val

See Also

lx.upper2vect

Examples

```
x <- matrix(1:16, ncol=4)
y <- lx.upper2vect(x, diag=TRUE)
xx <- lx.vect2upper(y, diag=TRUE)
```

lx.verbose *set/get verbose level*

Description

set mode: `lx.verbose(level)` where `level` is either an integer value or one of "debug" (=0); "info" (=10); "warning" (=20); "error" (=100)
get mode: `lx.verbose()`

Usage

```
lx.verbose(level)
```

Arguments

`level`	verbose level (see description)

Value

in get mode: return current verbose level
in set mode return previous verbose level (i.e. before set)

Examples

```
p <- lx.verbose("debug") # set to debug and return previous
lx.verbose(p)            # restore to previous value
```

lx.warn *emit warning*

Description

prints warning on stderr

Usage

```
lx.warn(..., level = "warning", up.frame = 0L)
```

Arguments

`...`	anything accepted by lx.out
`level`	see lx.out
`up.frame`	see lx.out

See Also

lx.out, lx.warnif, lx.stopif

Examples

```
lx.warn('pi=', pi, with.mem=TRUE)
```

lx.warnif *emit warning if condition is true*

Description

emit warning if condition is true

Usage

```
lx.warnif(condition, ..., level = "warning", up.frame = 0L)
```

Arguments

condition	boolean value
...	anything accepted by lx.out
level	see lx.out
up.frame	see lx.out

Value

invisible(condition)

See Also

lx.out, lx.warn, lx.stopif

Examples

```
x <- -1
lx.warnif(x < 0, 'x=', x, ' is negative')
```

lx.wt.mean *weighted mean*

Description

weighted mean

Usage

```
lx.wt.mean(x, w = NULL, na.rm = TRUE)
```

Arguments

x	a numeric vector
w	a numeric vector of (positive) weights
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds

Value

weighted mean of x

Examples

```
lx.wt.mean(1:2, 1:2)
```

`lx.wt.var` *weighted variance*

Description

weighted variance

Usage

```
lx.wt.var(x, w = NULL, as.repeat = TRUE, na.rm = TRUE)
```

Arguments

`x`	a numeric vector
`w`	a numeric vector of (positive) weights
`as.repeat`	a logical value indicating whether weights should be interpreted as repeated measures or not (see details).
`na.rm`	a logical value indicating whether NA values should be stripped before the computation proceeds

Details

there is no single accepted formula for the weighted variance. It depends whether weights should be interpreted as repeats of the same measure (in which case they usually are integers summing up to the total number of measures) (`as.repeat=TRUE`) or if thet should be interpreted as a 'confidence' attached to each measure (`as.repeat=FALSE`).

Value

weighted variance of x

Examples

```
lx.wt.var(1:2, 1:2)
var(c(1, 2, 2)) # should be the same as previous
lx.wt.var(1:2, 1:2, as.repeat=FALSE) # but this is different
lx.wt.var(1:2, rep(1,2))
var(1:2) # should be the same as previous
```

`lx.zero` *One dimensional Root (Zero) finding*

Description

look for a zero (root) of a continuous monotonic function by dichotomic search
i.e. return value x such that `fun(x) = 0`.
this function is similar to uniroot but additionally returns a lower and upper bound of root (see details).

Usage

```
lx.zero(fun, xmin = 0, xmax = 1, xtol = 10 * .Machine$double.eps,
  ytol = 10 * .Machine$double.eps, maxiter = 10000L, with.interval = TRUE,
  ..., .errorBound = TRUE)
```

Arguments

`fun`	monotonic function on interval xmin, xmax
`xmin`	minimum value of x
`xmax`	maximum value of x
`xtol`	tolerance on x (see details)
`ytol`	tolerance on y=0 (see details)
`maxiter`	max number of iterations (see details)
`with.interval`	logical, provide lower and upper bounds for root
`...`	additional arguments to fun
`.errorBound`	logical, raise error if zero is not between xmin and xmax boundaries.

Details

`fun` should be monotonic (either increasing or decreasing) if not, the result will (probably) be wrong. Since the zero must be located between `xmin` and `xmax`, `fun(xmin)` and `fun(xmax)` should be of opposite sign else an error is raised, unless `.errorBound = FALSE` in which case the closest boundary is returned (and niter is set to `NA`)

the algorithm proceeds by dichotomic search (bissection) and convergence is achieved when at least one of the following criterion is satisfied:

- `fun(x) = 0 +/- ytol`
- the change in x for one step is less than `xtol`
- the maximum number of iterations is reached

if `with.interval == TRUE` then a lower and upper bound for root are additionally provided. Unless the maximum number of iterations has been reached, the result guarantees that `[lower,upper]` is the largest interval such that `for all x in [lower+/-xtol, upper+/-xtol] fun(x)=0 +/- ytol`

Value

a named list with 4 components:

- root : the root value
- lower: root lower bound (see details)
- upper: root upper bound (see details)
- niter: number of iterations

Note

the algorithm is far from beeing optimal and is usually slower than uniroot. So, if you don't care about lower and upper bounds, you better have to use to uniroot instead.

if zero is reached at xmin (resp. xmax) boundary, then the lower (resp. upper) bound is truncated.

either `xmin` or `xmax` may be a pole (i.e. `fun(x) = Inf`) but not both.

there is no need to set `xtol` or `ytol` below `.Machine$double.eps`.

Examples

```
# simple functions
lx.zero(function(x) x^2 - 0.2^2, ytol=1e-3)
lx.zero(function(x) log(x) - log(0.3), ytol=1e-3)
lx.zero(function(x) x - pi, xmax=10, ytol=1e-3)
lx.zero(function(x) pi - x, xmax=10, ytol=1e-3)
# out of bounds
tryCatch(lx.zero(function(x) x - 2), error=function(e) NA)
lx.zero(function(x) x - 2, .errorBound=FALSE)
# interval
lx.zero(function(x) x^5, ytol=1e-3) # interval is truncated
lx.zero(function(x) x^5, ytol=1e-3, xmin=-1)
# poles
lx.zero(function(x) (x-0.5)/x/(2-x))
```

---

print.LXFileHandle *print method for LXFileHandle*

Description

print method for LXFileHandle

Usage

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

Arguments

x	file handle (opened by lx.open)
...	further arguments passed to or from other methods.

Value

(invisible) x

`print.Stack` *print method for Stack*

Description

print method for Stack

Usage

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

Arguments

`x`	Stack
`...`	further arguments passed to or from other methods.

Value

(invisible) stk

`tex.close` *close tex report file*

Description

close tex report file

Usage

```
tex.close(tex, compile = TRUE, ...)
```

Arguments

`tex`	handle (as returned by tex.open)
`compile`	if TRUE, call tex.compile to produce pdf
`...`	any argument to tex.compile (see note)

Value

invisible(tex)

Note

a useful argument is `clean=TRUE` that remove all auxiliary files created during the conversion

tex.compile *compile tex report file*

Description

compile tex report file

Usage

```
tex.compile(tex, ...)
```

Arguments

tex	handle (as returned by tex.open)
...	any argument passed to tools::texi2pdf

Value

invisible(tex)

tex.fig.off *end tex graphics*

Description

end tex graphics

Usage

```
tex.fig.off(tex, ...)
```

Arguments

tex	handle (as returned by tex.fig.on)
...	any attribute to latex:includegraphics

Value

tex

Note

it is important to reaffect tex handle upon return since the internal components have been modified

tex.fig.on *start new tex graphics*

Description

start new tex graphics

Usage

```
tex.fig.on(tex, ...)
```

Arguments

tex	handle (as returned by tex.open)
...	any argument to graphics driver

Value

tex

Note

it is important to reaffect tex handle upon return since the internal components have been modified

this function opens the graphic driver defined in `lx.options('tex.graphics.driver')` (default to 'pdf') with options `lx.options('tex.driver.options')`.
with 'jpeg', 'png' and 'tiff' drivers, you may set the 'units' and 'res' options in `lx.options('tex.driver.options')` (typical values are units='in' and res=300).

tex.installed *check if latex is properly installed*

Description

check if latex is properly installed

Usage

```
tex.installed(cmd = "pdflatex", warn = TRUE)
```

Arguments

cmd	latext command to check
warn	issue warning on error

Value

logical

Note

this function uses `system('which command')` and therefore only works on Unix

tex.open *open new tex report file*

Description

open new tex report file

Usage

```
tex.open(name = "report", title = name, author = lx.sysinfo("user"),
  prolog = NULL)
```

Arguments

name	file basename
title	report title
author	report author
prolog	function to add user's specific latex commands in document prolog. called as prolog(texHandle)

Value

tex handle

Note

the actual tex file is lx.options(tex.dir)/name.tex and the final pdf file is name.pdf

tex.out *output line(s) to tex report file*

Description

output line(s) to tex report file

Usage

```
tex.out(tex, ..., lf = FALSE)
```

Arguments

tex	handle (as returned by tex.open)
...	any argument supported by cat
lf	add a latex linefeed \\ at end of line

Value

invisible(tex)

See Also

tex.print for formatted output

`tex.print` *print formatted argument to tex report file*

Description

print formatted argument to tex report file

Usage

```
tex.print(tex, x, ...)
```

Arguments

`tex`	handle (as returned by tex.open)
`x`	an object to print (see details)
`...`	any argument supported by print variant

Details

if `x` is of class `xtable` then just call the S3 method `print` on `x`.
else, first format x using `xtable` then call print

x should be of a type handled by `xtable`. see tex.out for simple output.

Value

invisible(tex)

Note

this requires the `xtable` package that will be internally installed if not already present (using internal distribution)

See Also

tex.out for simple output

tex.section *start new tex section*

Description

start new tex section

Usage

```
tex.section(tex, name, numbered = TRUE)
```

Arguments

tex	handle (as returned by tex.open)
name	section name
numbered	should section be numbered

Value

invisible(tex)

tex.subsection *start new tex subsection*

Description

start new tex subsection

Usage

```
tex.subsection(tex, name, numbered = TRUE)
```

Arguments

tex	handle (as returned by tex.open)
name	section name
numbered	should section be numbered

Value

invisible(tex)

tex.tag *output tex tag to tex report file*

Description

output tex tag as
\<tag>{<value>}[<attr>] or \<tag>[<attr>]{<value>} depending upon the prepend argument.

Usage

```
tex.tag(tex, tag = NULL, value = NULL, attr = NULL, prepend = FALSE)
```

Arguments

tex	handle (as returned by tex.open)
tag	tag name
value	tag value
attr	tag attribute(s)
prepend	if TRUE append attributes before value. default is FALSE, append attributes after value.

Value

invisible(tex)

Index