Package 'R.utils'

February 14, 2019

Version 2.8.0

Depends R (>= 2.14.0), R.oo (>= 1.22.0)

Imports methods, utils, tools, R.methodsS3 (>= 1.7.1)

Suggests digest (>= 0.6.10)

Title Various Programming Utilities

Author Henrik Bengtsson [aut, cre, cph]

Maintainer Henrik Bengtsson <henrikb@braju.com>

Description Utility functions useful when programming and developing R packages.

License LGPL (>= 2.1)

LazyLoad TRUE

URL https://github.com/HenrikBengtsson/R.utils

BugReports https://github.com/HenrikBengtsson/R.utils/issues

RoxygenNote 6.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2019-02-14 21:42:21 UTC

R topics documented:

`R.utils-package` *Package R.utils*

Description

Utility functions useful when programming and developing R packages.

Warning: The Application Programming Interface (API) of the classes and methods in this package may change. Classes and methods are considered either to be stable, or to be in beta or alpha (pre-beta) stage. See list below for details.

The main reason for publishing this package on CRAN although it lacks a stable API, is that its methods and classes are used internally by other packages on CRAN that the author has published.

For package history, see `showHistory(R.utils)`.

Requirements

This package requires the **R.oo** package [1].

Installation and updates

To install this package do:

```
install.packages("R.utils")
```

To get started

- Arguments[alpha] Methods for common argument processing.
- Assert[alpha] Methods for assertion of values and states.
- GString[alpha] A character string class with methods for simple substitution.
- Java[beta] Reads and writes Java streams.
- Options[alpha] Tree-structured options queried in a file-system like manner.

- Settings[alpha] An Options class for reading and writing package settings.
- ProgressBar[beta] Text-based progress bar.
- FileProgressBar[beta] A ProgressBar that reports progress as file size.
- System[alpha] Methods for access to system.
- Verbose[alpha] A class for verbose and log output. Utilized by the VComments and LComments classes.
- SmartComments, VComments, LComments[alpha] Methods for preprocessing source code comments of certain formats into R code.

In addition to the above, there is a large set of function for file handling such as support for reading/following Windows Shortcut links, but also other standalone utility functions. See package index for a list of these. These should also be considered to be in alpha or beta stage.

How to cite this package

Whenever using this package, please cite [1] as

```
Bengtsson, H. The R.oo package - Object-Oriented Programming with References Using
Standard R Code, Proceedings of the 3rd International Workshop on Distributed
Statistical Computing (DSC 2003), ISSN 1609-395X, Hornik, K.; Leisch, F. & Zeileis,
A. (ed.), 2003
```

Wishlist

Here is a list of features that would be useful, but which I have too little time to add myself. Contributions are appreciated.

- Write a TclTkProgressBar class.
- Improve/stabilize the GString class.
- Mature the SmartComments classes. Also add AComments and PComments for assertion and progress/status comments.

If you consider implement some of the above, make sure it is not already implemented by downloading the latest "devel" version!

License

The releases of this package is licensed under LGPL version 2.1 or newer.

The development code of the packages is under a private licence (where applicable) and patches sent to the author fall under the latter license, but will be, if incorporated, released under the "release" license above.

References

1 H. Bengtsson, *The R.oo package - Object-Oriented Programming with References Using Standard R Code*, In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003), March 20-22, Vienna, Austria. http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/

Author(s)

Henrik Bengtsson

`addFinalizerToLast` *Modifies .Last() to call 'finalizeSession()*

Description

Modifies .Last() to call 'finalizeSession() *before* calling the default `.Last()` function.

Note that `.Last()` is *not* guaranteed to be called when the R session finished. For instance, the user may quit R by calling `quit(runLast=FALSE)` or run R in batch mode.

Note that this function is called when the R.utils package is loaded.

Usage

```
## Default S3 method:
addFinalizerToLast(...)
```

Arguments

`...`	Not used.

Value

Returns (invisibly) `TRUE` if `.Last()` was modified, otherwise `FALSE`.

Author(s)

Henrik Bengtsson

See Also

`onSessionExit()`.

Arguments *Static class to validate and process arguments*

Description

Package: R.utils
Class Arguments

```
Object
~~|
~~+--Arguments
```

Directly known subclasses:

public static class **Arguments**
extends Object

Fields and Methods

Methods:

getCharacter	-
getCharacters	Coerces to a character vector and validates.
getDirectory	-
getDouble	-
getDoubles	Coerces to a double vector and validates.
getEnvironment	Gets an existing environment.
getFilename	Gets and validates a filename.
getIndex	-
getIndices	Coerces to a integer vector and validates.
getInstanceOf	Gets an instance of the object that is of a particular class.
getInteger	-
getIntegers	Coerces to a integer vector and validates.
getLogical	-
getLogicals	Coerces to a logical vector and validates.
getNumeric	-
getNumerics	Coerces to a numeric vector and validates.
getReadablePath	-
getReadablePathname	Gets a readable pathname.
getReadablePathnames	Gets a readable pathname.
getRegularExpression	Gets a valid regular expression pattern.
getVector	Validates a vector.
getVerbose	Coerces to Verbose object.

getWritablePath	-
getWritablePathname	Gets a writable pathname.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

---

as.character.binmode *Converts a binary/octal/hexadecimal number into a string*

Description

Converts a binary/octal/hexadecimal number into a string.

Usage

```
 ## S3 method for class 'binmode'
as.character(x, ...)
```

Arguments

x	Object to be converted.
...	Not used.

Value

Returns a character.

Author(s)

Henrik Bengtsson

See Also

as.character.octmode(), cf. octmode. intToBin() (incl. intToOct() and intToHex()).

`Assert` *The Assert class*

Description

Package: R.utils
Class Assert

```
Object
~~|
~~+--Assert
```

Directly known subclasses:

public static class **Assert**
extends Object

Usage

```
Assert(...)
```

Arguments

`...`	Not used.

Fields and Methods

Methods:

`check`	Static method asserting that a generic condition is true.
`inherits`	-
`inheritsFrom`	Static method asserting that an object inherits from of a certain class.
`isMatrix`	Static method asserting that an object is a matrix.
`isScalar`	Static method asserting that an object is a single value.
`isVector`	Static method asserting that an object is a vector.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

`attachLocally.list` *Assigns an objects elements locally*

Description

Assigns an objects elements locally.

Usage

```
## S3 method for class 'list'
attachLocally(object, fields=NULL, excludeFields=NULL, overwrite=TRUE,
  envir=parent.frame(), ...)
```

Arguments

`object`	An object with named elements such as an `environment`, a `list`, or a `data.frame`.
`fields`	A `character vector` specifying elements to be copied. If `NULL`, all elements are considered.
`excludeFields`	A `character vector` specifying elements not to be copied. This has higher priority than `fields`.
`overwrite`	If `FALSE`, fields that already exists will not be copied.
`envir`	The `environment` where elements are copied to.
`...`	Not used.

Value

Returns (invisibly) a `character vector` of the fields copied.

Author(s)

Henrik Bengtsson

See Also

`attachLocally()` of class Object. `attach`().

Examples

```
foo <- function(object) {
  cat("Local objects in foo():\n")
  print(ls())

  attachLocally(object)

  cat("\nLocal objects in foo():\n")
  print(ls())

  for (name in ls()) {
    cat("\nObject '", name, "':\n", sep="")
    print(get(name, inherits=FALSE))
  }
}

a <- "A string"
l <- list(a=1:10, msg="Hello world", df=data.frame(a=NA, b=2))
foo(l)
print(a)
```

`callHooks` *Call hook functions by hook name*

Description

Call hook functions by hook name.

Usage

```
## Default S3 method:
callHooks(hookName, ..., removeCalledHooks=FALSE)
```

Arguments

`hookName`	A `character` string of the hook name.
`...`	Argument passed to each hook function.
`removeCalledHooks`	If `TRUE`, called hook functions are removed, otherwise not.

Value

Returns (invisibly) whatever `callHooks.list()` returns.

Author(s)

Henrik Bengtsson

See Also

Internally, after retrieving hook functions, callHooks.list() is called.

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example 1
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# First, clean up if called more than once
setHook("myFunction.onEnter", NULL, action="replace")
setHook("myFunction.onExit", NULL, action="replace")

runConference <- function(...) {
  callHooks("myFunction.onEnter")
  cat("Speaker A: Hello there...\n")
  callHooks("myFunction.onExit")
}

setHook("myFunction.onEnter", function(...) {
  cat("Chair: Welcome to our conference.\n")
})

setHook("myFunction.onEnter", function(...) {
  cat("Chair: Please welcome Speaker A!\n")
})

setHook("myFunction.onExit", function(...) {
  cat("Chair: Please thanks Speaker A!\n")
})

runConference()


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example 2
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setHook("randomNumber", NULL, action="replace")
setHook("randomNumber", rnorm)      # By function
setHook("randomNumber", "rexp")     # By name
setHook("randomNumber", "runiff")   # Non-existing name
setHook("randomNumber", .GlobalEnv) # Not a function

res <- callHooks("randomNumber", n=1)
str(res)
cat("Number of hooks: ", length(res), "\n");
isErroneous <- unlist(lapply(res, FUN=function(x) !is.null(x$exception)));
cat("Erroneous hooks: ", sum(isErroneous), "\n");
```

callHooks.function *Call hook functions*

Description

Call hook functions.

Usage

```
## S3 method for class 'function'
callHooks(hooks, ...)
```

Arguments

hooks	A function or a list of hook functions or names of such.
...	Argument passed to each hook function.

Value

Returns (invisibly) a list that is named with hook names, if possible. Each element in the list is in turn a list with three element: fcn is the hook function called, result is its return value, and exception is the exception caught or NULL.

Author(s)

Henrik Bengtsson

See Also

See callHooks() to call hook function by name.

capitalize *Capitalizes/decapitalizes each character string in a vector*

Description

Capitalizes/decapitalized (making the first letter upper/lower case) of each character string in a vector.

Usage

```
  ## Default S3 method:
capitalize(str, ...)
  ## Default S3 method:
decapitalize(str, ...)
```

Arguments

str	A vector of character strings to be capitalized.
...	Not used.

Value

Returns a vector of character strings of the same length as the input vector.

Author(s)

Henrik Bengtsson

See Also

toCamelCase.

Examples

```
words <- strsplit("Hello wOrld", " ")[[1]];
cat(paste(toupper(words), collapse=" "), "\n")      # "HELLO WORLD"
cat(paste(tolower(words), collapse=" "), "\n")      # "hello world"
cat(paste(capitalize(words), collapse=" "), "\n")   # "Hello WOrld"
cat(paste(decapitalize(words), collapse=" "), "\n") # "hello wOrld"

# Sanity checks
stopifnot(paste(toupper(words), collapse=" ") == "HELLO WORLD")
stopifnot(paste(tolower(words), collapse=" ") == "hello world")
stopifnot(paste(capitalize(words), collapse=" ") == "Hello WOrld")
stopifnot(paste(decapitalize(words), collapse=" ") == "hello wOrld")
```

---

captureOutput *Evaluate an R expression and captures the output*

Description

Evaluate an R expression and captures the output.

Usage

```
captureOutput(expr, file=NULL, append=FALSE, collapse=NULL, envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
file	A file name or a connection to where the output is directed. Alternatively, if NULL the output is captured to and returned as a character vector.
append	If TRUE, the output is appended to the file or the (unopened) connection, otherwise it overwrites.
collapse	A character string used for collapsing the captured rows. If NULL, the rows are not collapsed.
envir	The environment in which the expression is evaluated.

Details

This method imitates capture.output with the major difference that it captures strings via a raw connection rather than via internal strings. The latter becomes exponentially slow for large outputs [1,2].

Value

Returns captured output as a character vector.

Author(s)

Henrik Bengtsson

References

[1] R-devel thread 'capture.output(): Using a rawConnection() [linear] instead of textConnection() [exponential]?', 2014-02-03. https://stat.ethz.ch/pipermail/r-devel/2014-February/068349.html [2] JottR blog post 'PERFORMANCE: captureOutput() is much faster than capture.output()', 2015-05-26. http://www.jottr.org/2014/05/captureOutput.html

See Also

Internally, eval() is used to evaluate the expression. and capture.output to capture the output.

Examples

```
# captureOutput() is much faster than capture.output()
# for large outputs when capturing to a string.
for (n in c(10e3, 20e3, 30e3, 40e3)) {
  printf("n=%d\n", n)

  x <- rnorm(n)

  t0 <- system.time({
    bfr0 <- capture.output(print(x))
  })
  print(t0)
```

```
  t1 <- system.time({
    bfr <- captureOutput(print(x))
  })
  print(t1)
  print(t1/t0)

  bfr2n <- captureOutput(print(x), collapse="\n")
  bfr2r <- captureOutput(print(x), collapse="\r")

  stopifnot(identical(bfr, bfr0))
} # for (n ...)
```

---

cmdArgs *Simple access to parsed command-line arguments*

Description

Simple access to parsed command-line arguments.

Usage

```
cmdArgs(args=NULL, names=NULL, unique=TRUE, ..., .args=NULL)
cmdArg(...)
```

Arguments

args	A named list of arguments.
names	A character vector specifying the arguments to be returned. If NULL, all arguments are returned.
unique	If TRUE, only unique arguments are returned.
...	For cmdArgs(), additional arguments passed to commandArgs(), e.g. defaults and always. For cmdArg(), named arguments name and default, where name must be a character string and default is an optional default value (if not given, it's NULL). Alternatively, name and default can be given as a named argument (e.g. n=42).
.args	(advanced/internal) A named list of parsed command-line arguments.

Value

cmdArgs() returns a named list with command-line arguments. cmdArg() return the value of the requested command-line argument.

Coercing to non-character data types

The value of each command-line argument is returned as a character string, unless an argument share name with ditto in the (optional) arguments always and default in case the retrieved value is coerced to that of the latter. Finally, remaining character string command-line arguments are coerced to numerics (via as.numeric()), if possible, that is unless the coerced value becomes NA.

Author(s)

Henrik Bengtsson

See Also

Internally, commandArgs() is used.

Examples

```
args <- cmdArgs()
cat("User command-line arguments used when invoking R:\n")
str(args)

# Retrieve command line argument 'n', e.g. '-n 13' or '--n=13'
n <- cmdArg("n", 42L)
printf("Argument n=%d\n", n)

# Short version doing the same
n <- cmdArg(n=42L)
printf("Argument n=%d\n", n)
```

---

colClasses *Creates a vector of column classes used for tabular reading*

Description

Creates a vector of column classes used for tabular reading based on a compact format string.

Usage

```
## Default S3 method:
colClasses(fmt, ...)
```

Arguments

fmt	A character string specifying the column-class format. This string is first translated by sprintf().
...	Optional arguments for the sprintf() translation.

Value

Returns a vector of character strings.

Author(s)

Henrik Bengtsson

See Also

read.table.

Examples

```
# All predefined types
print(colClasses("-?cdfilnrzDP"))
## [1] "NULL"      "NA"        "character" "double"
## [5] "factor"    "integer"   "logical"   "numeric"
## [9] "raw"       "complex"   "Date"      "POSIXct"

# A string in column 1, integers in column 4 and 5, rest skipped
print(colClasses("c--ii----"))
## [1] "character" "NULL"      "NULL"      "integer"
## [5] "integer"   "NULL"      "NULL"      "NULL"
## [9] "NULL"

# Repeats and custom column classes
c1 <- colClasses("3c{MyClass}3{foo}")
print(c1)
## [1] "character" "character" "character" "MyClass"
## [5] "foo"       "foo"       "foo"

# Passing repeats and class names using sprintf() syntax
c2 <- colClasses("%dc{%s}%d{foo}", 3, "MyClass", 3)
stopifnot(identical(c1, c2))

# Repeats of a vector of column classes
c3 <- colClasses("3{MyClass,c}")
print(c3)
## [1] "MyClass"   "character" "MyClass"   "character"
## [4] "MyClass"   "character"

# Large number repeats
c4 <- colClasses("321{MyClass,c,i,d}")
c5 <- rep(c("MyClass", "character", "integer", "double"), times=321)
stopifnot(identical(c4, c5))
```

---

compressFile *Compressing and decompressing files*

Description

Compressing and decompressing files such as gzip:ed and bzip2:ed files.

*NOTE: The default (*remove=TRUE*) behavior is that the input file is removed after that the output file is fully created and closed.*

Usage

```
 ## Default S3 method:
compressFile(filename, destname=sprintf("%s.%s", filename, ext), ext, FUN,
  temporary=FALSE, skip=FALSE, overwrite=FALSE, remove=TRUE, BFR.SIZE=1e+07, ...)
 ## Default S3 method:
decompressFile(filename, destname=gsub(sprintf("[.]%s$", ext), "", filename,
  ignore.case = TRUE), ext, FUN, temporary=FALSE, skip=FALSE, overwrite=FALSE,
  remove=TRUE, BFR.SIZE=1e+07, ...)
 ## Default S3 method:
isCompressedFile(filename, method=c("extension", "content"), ext, fileClass, ...)
 ## Default S3 method:
bzip2(filename, ..., ext="bz2", FUN=bzfile)
 ## Default S3 method:
bunzip2(filename, ..., ext="bz2", FUN=bzfile)
 ## Default S3 method:
gzip(filename, ..., ext="gz", FUN=gzfile)
 ## Default S3 method:
gunzip(filename, ..., ext="gz", FUN=gzfile)
```

Arguments

`filename`	Pathname of input file.
`destname`	Pathname of output file.
`temporary`	If `TRUE`, the output file is created in a temporary directory.
`skip`	If `TRUE` and the output file already exists, the output file is returned as is.
`overwrite`	If `TRUE` and the output file already exists, the file is silently overwritten, otherwise an exception is thrown (unless `skip` is `TRUE`).
`remove`	If `TRUE`, the input file is removed afterward, otherwise not.
`BFR.SIZE`	The number of bytes read in each chunk.
`...`	Passed to the underlying function or alternatively not used.
`method`	A `character` string specifying how to infer whether a file is compressed or not.
`ext, fileClass, FUN`	(internal) Filename extension, file class, and a connection `function` used to read from/write to file.

Details

Internally `bzfile()` and `gzfile()` (see `connections`) are used to read (write) files. If the process is interrupted before completed, the partially written output file is automatically removed.

Value

Returns the pathname of the output file. The number of bytes processed is returned as an attribute.

`isCompressedFile()`, `isGzipped()` and `isBzipped()` return a `logical`. Note that with `method = "extension"` (default), only the filename extension is used to infer whether the file is compressed or not. Specifically, it does not matter whether the file actually exists or not.

Author(s)

Henrik Bengtsson

Examples

```
## bzip2
cat(file="foo.txt", "Hello world!")
print(isBzipped("foo.txt"))
print(isBzipped("foo.txt.bz2"))

bzip2("foo.txt")
print(file.info("foo.txt.bz2"))
print(isBzipped("foo.txt"))
print(isBzipped("foo.txt.bz2"))

bunzip2("foo.txt.bz2")
print(file.info("foo.txt"))

## gzip
cat(file="foo.txt", "Hello world!")
print(isGzipped("foo.txt"))
print(isGzipped("foo.txt.gz"))

gzip("foo.txt")
print(file.info("foo.txt.gz"))
print(isGzipped("foo.txt"))
print(isGzipped("foo.txt.gz"))

gunzip("foo.txt.gz")
print(file.info("foo.txt"))

## Cleanup
file.remove("foo.txt")
```

---

compressPDF *Compresses a PDF (into a new PDF)*

Description

Compresses a PDF (into a new PDF).

Usage

```
## Default S3 method:
compressPDF(filename, path=NULL, outFilename=basename(pathname),
  outPath="compressedPDFs", skip=FALSE, overwrite=FALSE, compression="gs(ebook)+qpdf",
  ...)
```

Arguments

`filename, path`	The filename and (optional) path of the PDF to be compressed.
`outFilename, outPath`	The generated PDF.
`skip`	If `TRUE` and an existing output file, then it is returned.
`overwrite`	If `FALSE`, an error is thrown if the output file already exists, otherwise not.
`compression`	A `character vector` of compression methods to apply. This overrides any low-level arguments passed via `...` that `compactPDF`.
`...`	Additional arguments passed to `compactPDF`, e.g. `gs_quality`.

Value

Returns the pathname of the generated PDF.

Author(s)

Henrik Bengtsson

See Also

Internally `compactPDF` is utilized.

Examples

```
## Not run:
  pathnameZ <- compressPDF("report.pdf")

## End(Not run)
```

---

copyDirectory *Copies a directory*

Description

Copies a directory.

Usage

```
## Default S3 method:
copyDirectory(from, to=".", ..., private=TRUE, recursive=TRUE)
```

Arguments

from	The pathname of the source directory to be copied.
to	The pathname of the destination directory.
...	Additional arguments passed to file.copy(), e.g. overwrite.
private	If TRUE, files (and directories) starting with a period is also copied, otherwise not.
recursive	If TRUE, subdirectories are copied too, otherwise not.

Details

Note that this method does *not* use copyFile() to copy the files, but file.copy().

Value

Returns (invisibly) a character vector of pathnames copied.

Author(s)

Henrik Bengtsson

---

countLines *Counts the number of lines in a text file*

Description

Counts the number of lines in a text file by counting the number of occurrences of platform-independent newlines (CR, LF, and CR+LF [1]), including a last line with neither. An empty file has zero lines.

Usage

```
## Default S3 method:
countLines(file, chunkSize=5e+07, ...)
```

Arguments

file	A connection or a pathname.
chunkSize	The number of bytes read in each chunk.
...	Not used.

Details

Both compressed and non-compressed files are supported.

Value

Returns an non-negative integer.

Author(s)

Henrik Bengtsson

References

[1] Page *Newline*, Wikipedia, July 2008. http://en.wikipedia.org/wiki/Newline

Examples

```
pathname <- system.file("NEWS", package="R.utils");
n <- countLines(pathname);
n2 <- length(readLines(pathname));
stopifnot(n == n2);
```

---

createFileAtomically *Creates a file atomically*

Description

Creates a file atomically by first creating and writing to a temporary file which is then renamed.

Usage

```
## Default S3 method:
createFileAtomically(filename, path=NULL, FUN, ..., skip=FALSE, overwrite=FALSE,
  backup=TRUE, verbose=FALSE)
```

Arguments

filename	The filename of the file to create.
path	The path to the file.
FUN	A function that creates and writes to the pathname that is passed as the first argument. This pathname is guaranteed to be a non-existing temporary pathname.
...	Additional arguments passed to pushTemporaryFile() and popTemporaryFile().
skip	If TRUE and a file with the same pathname already exists, nothing is done/written.
overwrite	If TRUE and a file with the same pathname already exists, the existing file is overwritten. This is also done atomically such that if the new file was not successfully created, the already original file is restored. If restoration also failed, the original file remains as the pathname with suffix ".bak" appended.
backup	If TRUE and a file with the same pathname already exists, then it is backed up while creating the new file. If the new file was not successfully created, the original file is restored from the backup copy.
verbose	A logical or Verbose.

Value

Returns (invisibly) the pathname.

Author(s)

Henrik Bengtsson

See Also

Internally, pushTemporaryFile() and popTemporaryFile() are used for working toward a temporary file, and pushBackupFile() and popBackupFile() are used for backing up and restoring already existing file.

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Create a file atomically
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
n <- 10
createFileAtomically("foobar.txt", FUN=function(pathname) {
  cat(file=pathname, "This file was created atomically.\n")
  cat(file=pathname, "Timestamp: ", as.character(Sys.time()), "\n", sep="")
  for (kk in 1:n) {
    cat(file=pathname, kk, "\n", append=TRUE)
    # Emulate a slow process
    if (interactive()) Sys.sleep(0.1)
  }
  cat(file=pathname, "END OF FILE\n", append=TRUE)
}, overwrite=TRUE)

bfr <- readLines("foobar.txt")
cat(bfr, sep="\n")


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Overwrite the file atomically (emulate write failure)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
tryCatch({
  createFileAtomically("foobar.txt", FUN=function(pathname) {
    cat(file=pathname, "Trying to create a new file.\n")
    cat(file=pathname, "Writing a bit, but then an error...\n", append=TRUE)
    # Emulate write error
    stop("An error occured while writing to the new file.")
    cat(file=pathname, "END OF FILE\n", append=TRUE)
  }, overwrite=TRUE)
}, error = function(ex) {
  print(ex$message)
})

# The original file was never overwritten
bfr2 <- readLines("foobar.txt")
```

```
cat(bfr2, sep="\n")
stopifnot(identical(bfr2, bfr))

# The partially temporary file remains
stopifnot(isFile("foobar.txt.tmp"))
bfr3 <- readLines("foobar.txt.tmp")
cat(bfr3, sep="\n")

file.remove("foobar.txt.tmp")
```

`createLink` *Creates a link to a file or a directory*

Description

Creates a link to a file or a directory. This method tries to create a link to a file/directory on the file system, e.g. a symbolic link and Windows Shortcut links. It depends on operating and file system (and argument settings), which type of link is finally created, but all this is hidden internally so that links can be created the same way regardless of system.

Usage

```
## Default S3 method:
createLink(link=".", target, skip=!overwrite, overwrite=FALSE,
  methods=getOption("createLink/args/methods", c("unix-symlink", "windows-ntfs-symlink",
   "windows-shortcut")), ...)
```

Arguments

`link`	The path or pathname of the link to be created. If `"."` (or `NULL`), it is inferred from the `target` argument, if possible.
`target`	The target file or directory to which the shortcut should point to.
`skip`	If `TRUE` and a file with the same name as argument `link` already exists, then the nothing is done.
`overwrite`	If `TRUE`, an existing link file is overwritten, otherwise not.
`methods`	A `character vector` specifying what methods (and in what order) should be tried for creating links.
`...`	Not used.

Value

Returns (invisibly) the path or pathname to the link. If no link was created, `NULL` is returned.

Required privileges on Windows

In order for `method="unix-symlink"` (utilizing `file.symlink()`), `method="windows-ntfs-symlink"` (utilizing executable `mklink`), and/or `method="windows-shortcut"` (utilizing `createWindowsShortcut()`) to succeed on Windows, the client/R session must run with sufficient privileges (it has been reported that Administrative rights are necessary).

Author(s)

Henrik Bengtsson

References

Ben Garrett, *Windows File Junctions, Symbolic Links and Hard Links*, September 2009 [http://goo.gl/R21AC]

See Also

`createWindowsShortcut()` and `file.symlink()`

`createWindowsShortcut` *Creates a Microsoft Windows Shortcut (.lnk file)*

Description

Creates a Microsoft Windows Shortcut (.lnk file).

Usage

```
## Default S3 method:
createWindowsShortcut(pathname, target, overwrite=FALSE, mustWork=FALSE, ...)
```

Arguments

`pathname`	The pathname (with file extension *.lnk) of the link file to be created.
`target`	The target file or directory to which the shortcut should point to.
`overwrite`	If `TRUE`, an existing link file is overwritten, otherwise not.
`mustWork`	If `TRUE`, an error is produced if the Windows Shortcut link is not created, otherwise not.
`...`	Not used.

Value

Returns (invisibly) the pathname.

Required privileges on Windows

In order for this method, which utilizes Windows Script Host a VBScript, to succeed on Windows, the client/R session must run with sufficient privileges (it has been reported that Administrative rights are necessary).

Author(s)

Henrik Bengtsson

References

[1] Create a windows shortcut (.LNK file), SS64.com, http://ss64.com/nt/shortcut.html

See Also

readWindowsShortcut()

Examples

```
# Create Windows Shortcut links to a directory and a file
targets <- list(
  system.file(package="R.utils"),
  system.file("DESCRIPTION", package="R.utils")
)

for (kk in seq_along(targets)) {
  cat("Link #", kk, "\n", sep="")

  target <- targets[[kk]]
  cat("Target: ", target, "\n", sep="")

  # Name of *.lnk file
  pathname <- sprintf("%s.LNK", tempfile())

  tryCatch({
    # Will only work on Windows systems with support for VB scripting
    createWindowsShortcut(pathname, target=target)
  }, error = function(ex) {
    print(ex)
  })

  # Was it created?
  if (isFile(pathname)) {
    cat("Created link file: ", pathname, "\n", sep="")

    # Validate that it points to the correct target
    dest <- filePath(pathname, expandLinks="any")
    cat("Available target: ", dest, "\n", sep="")

    res <- all.equal(tolower(dest), tolower(target))
```

```
    if (!isTRUE(res)) {
     msg <- sprintf("Link target does not match expected target: %s != %s", dest, target)
      cat(msg, "\n")
      warning(msg)
    }

    # Cleanup
    file.remove(pathname)
  }
}
```

---

dataFrame *Allocates a data frame with given column classes*

Description

Allocates a data frame with given column classes.

Usage

```
## Default S3 method:
dataFrame(colClasses, nrow=1, ...)
```

Arguments

colClasses	A character vector of column classes, cf. read.table.
nrow	An integer specifying the number of rows of the allocated data frame.
...	Not used.

Value

Returns an NxK data.frame where N equals nrow and K equals length(colClasses).

See Also

data.frame.

Examples

```
df <- dataFrame(colClasses=c(a="integer", b="double"), nrow=10)
df[,1] <- sample(1:nrow(df))
df[,2] <- rnorm(nrow(df))
print(df)
```

`detachPackage` *Detaches packages by name*

Description

Detaches packages by name, if loaded.

Usage

```
## Default S3 method:
detachPackage(pkgname, ...)
```

Arguments

`pkgname`	A `character` `vector` of package names to be detached.
`...`	Not used.

Value

Returns (invisibly) a named `logical` `vector` indicating whether each package was detached or not.

Author(s)

Henrik Bengtsson

See Also

`detach`().

`dimNA< -` *Sets the dimension of an object with the option to infer one dimension automatically*

Description

Sets the dimension of an object with the option to infer one dimension automatically. If one of the elements in the dimension `vector` is `NA`, then its value is inferred from the length of the object and the other elements in the dimension vector. If the inferred dimension is not an `integer`, an error is thrown.

Usage

```
## Default S3 replacement method:
dimNA(x) <- value
```

Arguments

x	An R object.
value	NULL of a positive numeric vector with one optional NA.

Value

Returns (invisibly) what dim<-() returns (see dim() for more details).

Author(s)

Henrik Bengtsson

See Also

dim().

Examples

```
x <- 1:12
dimNA(x) <- c(2,NA,3)
stopifnot(dim(x) == as.integer(c(2,2,3)))
```

---

displayCode *Displays the contents of a text file with line numbers and more*

Description

Displays the contents of a text file with line numbers and more.

Usage

```
## Default S3 method:
displayCode(con=NULL, code=NULL, numerate=TRUE, lines=-1, wrap=79, highlight=NULL,
  pager=getOption("pager"), ...)
```

Arguments

con	A connection or a character string filename. If code is specified, this argument is ignored.
code	A character vector of code lines to be displayed.
numerate	If TRUE, line are numbers, otherwise not.
lines	If a single numeric, the maximum number of lines to show. If -1, all lines are shown. If a vector of numeric, the lines numbers to display.
wrap	The (output) column numeric where to wrap lines.
highlight	A vector of line number to be highlighted.

pager	If "none", code is not displayed in a pager, but only returned. For other options, see file.show().
...	Additional arguments passed to file.show(), which is used to display the formatted code.

Value

Returns (invisibly) the formatted code as a character string.

Author(s)

Henrik Bengtsson

See Also

file.show().

Examples

```
file <- system.file("DESCRIPTION", package="R.utils")
cat("Displaying: ", file, ":\n", sep="")
displayCode(file)

file <- system.file("NEWS", package="R.utils")
cat("Displaying: ", file, ":\n", sep="")
displayCode(file, numerate=FALSE, lines=100:110, wrap=65)

file <- system.file("NEWS", package="R.utils")
cat("Displaying: ", file, ":\n", sep="")
displayCode(file, lines=100:110, wrap=65, highlight=c(101,104:108))
```

---

doCall *Executes a function call with option to ignore unused arguments*

Description

Executes a function call with option to ignore unused arguments.

Usage

```
## Default S3 method:
doCall(.fcn, ..., args=NULL, alwaysArgs=NULL, .functions=list(.fcn),
  .ignoreUnusedArgs=TRUE, envir=parent.frame())
```

Arguments

.fcn	A function or a character string specifying the name of a function to be called.
...	Named arguments to be passed to the function.
args	A list of additional named arguments that will be appended to the above arguments.
alwaysArgs	A list of additional named arguments that will be appended to the above arguments and that will *never* be ignore.
.functions	A list of function:s or names of functions. This can be used to control which arguments are passed.
.ignoreUnusedArgs	If TRUE, arguments that are not accepted by the function, will not be passed to it. Otherwise, all arguments are passed.
envir	An environment in which to evaluate the call.

Author(s)

Henrik Bengtsson

See Also

do.call().

Examples

```
doCall("plot", x=1:10, y=sin(1:10), col="red", dummyArg=54,
       alwaysArgs=list(xlab="x", ylab="y"),
       .functions=c("plot", "plot.xy"))
```

---

downloadFile.character

Downloads a file

Description

Downloads a file.

Usage

```
## S3 method for class 'character'
downloadFile(url, filename=basename(url), path=NULL, skip=TRUE, overwrite=!skip, ...,
  username=NULL, password=NULL, binary=TRUE, dropEmpty=TRUE, verbose=FALSE)
```

Arguments

url	A character string specifying the URL to be downloaded.
filename, path	(optional) character strings specifying the local filename and the path of the downloaded file.
skip	If TRUE, an already downloaded file is skipped.
overwrite	If TRUE, an already downloaded file is overwritten, otherwise an error is thrown.
...	Additional arguments passed to download.file.
username, password	character strings specifying the username and password for authenticated downloads. The alternative is to specify these via the URL.
binary	If TRUE, the file is downloaded exactly "as is", that is, byte by byte (recommended).
dropEmpty	If TRUE and the downloaded file is empty, the file is ignored and NULL is returned.
verbose	A logical, integer, or a Verbose object.

Details

Currently arguments username and password are only used for downloads via URL protocol 'https'. The 'https' protocol requires that either of 'curl' or 'wget' are available on the system.

Value

Returns the local pathname to the downloaded filename, or NULL if no file was downloaded.

Author(s)

Henrik Bengtsson

See Also

Internally download.file is used. That function may generate an empty file if the URL is not available.

Examples

```
## Not run:
 pathname <- downloadFile("http://www.r-project.org/index.html", path="www.r-project.org/")
 print(pathname)

## End(Not run)
```

egsub *Global substitute of expression using regular expressions*

Description

Global substitute of expression using regular expressions.

Usage

```
egsub(pattern, replacement, x, ..., value=TRUE, envir=parent.frame(), inherits=TRUE)
```

Arguments

pattern	A character string with the regular expression to be matched, cf. gsub().
replacement	A character string of the replacement to use when there is a match, cf. gsub().
x	The expression or a function to be modified.
...	Additional arguments passed to gsub()
value	If TRUE, the value of the replacement itself is used to look up a variable with that name and then using that variables value as the replacement. Otherwise the replacement value is used.
envir, inherits	An environment from where to find the variable and whether the search should also include enclosing frames, cf. get(). Only use if value is TRUE.

Value

Returns an expression.

Author(s)

Henrik Bengtsson

Examples

```
# Original expression
expr <- substitute({
  res <- foo.bar.yaa(2)
  print(res)
  R.utils::use("R.oo")
  x <- .b.
})

# Some predefined objects
foo.bar.yaa <- function(x) str(x)
a <- 2
b <- a
```

```
# Substitute with variable name
expr2 <- egsub("^[.]([a-zA-Z0-9_.]+)[.]$", "\\1", expr, value=FALSE)
print(expr2)
## {
##     res <- foo.bar.yaa(2)
##     print(res)
##     R.utils::use("R.oo")
##     x <- b
## }

# Substitute with variable value
expr3 <- egsub("^[.]([a-zA-Z0-9_.]+)[.]$", "\\1", expr, value=TRUE)
print(expr3)
## {
##     res <- foo.bar.yaa(2)
##     print(res)
##     R.utils::use("R.oo")
##     x <- 2
## }
# Substitute the body of a function
warnifnot <- egsub("stop", "warning", stopifnot, value=FALSE)
print(warnifnot)
warnifnot(pi == 3.14)
```

---

env *Creates a new environment, evaluates an expression therein, and returns the environment*

Description

Creates a new environment, evaluates an expression therein, and returns the environment.

Usage

```
env(..., hash=FALSE, parent=parent.frame(), size=29L)
```

Arguments

`...`	Arguments passed to `evalq`(), particularly a `expression` to be evaluated inside the newly created `environment`.
`hash, parent, size`	Arguments passed to `new.env`().

Value

Returns an `environment`.

Author(s)

Henrik Bengtsson

References

[1] R-devel thread 'Create an environment and assign objects to it in one go?' on March 9-10, 2011.

See Also

Internally new.env() and evalq() are used.

Examples

```
x <- list();

x$case1 <- env({
 # Cut'n'pasted from elsewhere
 a <- 1;
 b <- 2;
});

x$case2 <- env({
 # Cut'n'pasted from elsewhere
 foo <- function(x) x^2;
 a <- foo(2);
 b <- 1;
 rm(foo); # Not needed anymore
});

# Turn into a list of lists
x <- lapply(x, FUN=as.list);

str(x);
```

---

extract.array *Extract a subset of an array, matrix or a vector with unknown dimensions*

Description

Extract a subset of an array, matrix or a vector with unknown dimensions.

This method is useful when you do not know the number of dimensions of the object your wish to extract values from, cf. example.

Usage

```
## S3 method for class 'array'
extract(x, ..., indices=list(...), dims=names(indices), drop=FALSE)
```

Arguments

x	An array or a matrix.
...	These arguments are by default put into the indices list.
indices	A list of index vectors to be extracted.
dims	An vector of dimensions - one per element in indices - which will be coerced to integers. If NULL, it will default to seq_along(indices).
drop	If TRUE, dimensions of length one are dropped, otherwise not.

Value

Returns an array.

Author(s)

Henrik Bengtsson

See Also

slice.index()

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example using an array with a random number of dimensions
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
maxdim <- 4
dim <- sample(3:maxdim, size=sample(2:maxdim, size=1), replace=TRUE)
ndim <- length(dim)
dimnames <- list()
for (kk in 1:ndim)
  dimnames[[kk]] <- sprintf("%s%d", letters[kk], 1:dim[kk])
x <- 1:prod(dim)
x <- array(x, dim=dim, dimnames=dimnames)

cat("\nArray 'x':\n")
print(x)


cat("\nExtract 'x[2:3,...]':\n")
print(extract(x, "1"=2:3))

cat("\nExtract 'x[3,2:3,...]':\n")
print(extract(x, "1"=3,"2"=2:3))

cat("\nExtract 'x[...,2:3]':\n")
print(extract(x, indices=2:3, dims=length(dim(x))))



# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
# Assertions
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
y <- array(1:24, dim=c(2,3,4))
yA <- y[,,2:3]
yB <- extract(y, indices=list(2:3), dims=length(dim(y)))
stopifnot(identical(yB, yA))

yA <- y[,2:3,2]
yB <- extract(y, indices=list(2:3,2), dims=c(2,3), drop=TRUE)
stopifnot(identical(yB, yA))
```

`fileAccess` *Checks the permission of a file or a directory*

Description

Checks the permission of a file or a directory.

Usage

```
## Default S3 method:
fileAccess(pathname, mode=0, safe=TRUE, ...)
```

Arguments

`pathname`	A `character` string of the file or the directory to be checked.
`mode`	An `integer` (0,1,2,4), cf. `file.access`().
`safe`	If `TRUE`, the permissions are tested more carefully, otherwise `file.access`() is used.
`...`	Not used.

Details

In R there is `file.access`() for checking whether the permission of a file. Unfortunately, that function cannot be 100% trusted depending on platform used and file system queried, cf. [1].

Value

Returns an `integer`; 0 if the permission exists, -1 if not.

Symbolic links

This function follows symbolic links (also on Windows) and returns a value based on the link target (rather than the link itself).

Author(s)

Henrik Bengtsson

References

[1] R-devel thread *file.access() on network (mounted) drive on Windows Vista?* on Nov 26, 2008. https://stat.ethz.ch/pipermail/r-devel/2008-December/051461.html
[2] Filesystem permissions, Wikipedia, 2010. http://en.wikipedia.org/wiki/Filesystem_permissions

See Also

file.access()

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Current directory
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
path <- "."

# Test for existence
print(fileAccess(path, mode=0))
# Test for execute permission
print(fileAccess(path, mode=1))
# Test for write permission
print(fileAccess(path, mode=2))
# Test for read permission
print(fileAccess(path, mode=4))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# A temporary file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
pathname <- tempfile()
cat(file=pathname, "Hello world!")

# Test for existence
print(fileAccess(pathname, mode=0))
# Test for execute permission
print(fileAccess(pathname, mode=1))
# Test for write permission
print(fileAccess(pathname, mode=2))
# Test for read permission
print(fileAccess(pathname, mode=4))

file.remove(pathname)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# The 'base' package directory
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
path <- system.file(package="base")

# Test for existence
```

```
print(fileAccess(path, mode=0))
# Test for execute permission
print(fileAccess(path, mode=1))
# Test for write permission
print(fileAccess(path, mode=2))
# Test for read permission
print(fileAccess(path, mode=4))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# The 'base' package DESCRIPTION file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
pathname <- system.file("DESCRIPTION", package="base")

# Test for existence
print(fileAccess(pathname, mode=0))
# Test for execute permission
print(fileAccess(pathname, mode=1))
# Test for write permission
print(fileAccess(pathname, mode=2))
# Test for read permission
print(fileAccess(pathname, mode=4))
```

`filePath` *Construct the path to a file from components and expands Windows Shortcuts along the pathname from root to leaf*

Description

Construct the path to a file from components and expands Windows Shortcuts along the pathname from root to leaf. This function is backward compatible with `file.path`() when argument `removeUps=FALSE` and `expandLinks="none"`, except that a (character) `NA` is return if any argument is NA.

This function exists on all platforms, not only Windows systems.

Usage

```
## Default S3 method:
filePath(..., fsep=.Platform$file.sep, removeUps=TRUE,
  expandLinks=c("none", "any", "local", "relative", "network"), unmap=FALSE,
  mustExist=FALSE, verbose=FALSE)
```

Arguments

`...`	Arguments to be pasted together to a file path and then be parsed from the root to the leaf where Windows shortcut files are recognized and expanded according to argument `which` in each step.
`fsep`	the path separator to use.

removeUps	If TRUE, relative paths, for instance "foo/bar/../" are shortened into "foo/", but also "./" are removed from the final pathname, if possible.
expandLinks	A character string. If "none", Windows Shortcut files are ignored. If "local", the absolute target on the local file system is used. If "relative", the relative target is used. If "network", the network target is used. If "any", first the local, then the relative and finally the network target is searched for.
unmap	If TRUE, paths on mapped Windows drives are "followed" and translated to their corresponding "true" paths.
mustExist	If TRUE and if the target does not exist, the original pathname, that is, argument pathname is returned. In all other cases the target is returned.
verbose	If TRUE, extra information is written while reading.

Details

If expandLinks != "none", each component, call it *parent*, in the absolute path is processed from the left to the right as follows: 1. If a "real" directory of name *parent* exists, it is followed. 2. Otherwise, if Microsoft Windows Shortcut file with name *parent.lnk* exists, it is read. If its local target exists, that is followed, otherwise its network target is followed. 3. If no valid existing directory was found in (1) or (2), the expanded this far followed by the rest of the pathname is returned quietly. 4. If all of the absolute path was expanded successfully the expanded absolute path is returned.

Value

Returns a character string.

On speed

Internal file.exists() is call while expanding the pathname. This is used to check if there exists a Windows shortcut file named 'foo.lnk' in 'path/foo/bar'. If it does, 'foo.lnk' has to be followed, and in other cases 'foo' is ordinary directory. The file.exists() is unfortunately a bit slow, which is why this function appears slow if called many times.

Author(s)

Henrik Bengtsson

See Also

readWindowsShellLink(). readWindowsShortcut(). file.path().

Examples

```
# Default
print(file.path("foo", "bar", "..", "name")) # "foo/bar/../name"

# Shorten pathname, if possible
print(filePath("foo", "bar", "..", "name"))  # "foo/name"
print(filePath("foo/bar/../name"))           # "foo/name"
```

```
# Recognize Windows Shortcut files along the path, cf. Unix soft links
filename <- system.file("data-ex/HISTORY.LNK", package="R.utils")
print(filename)
filename <- filePath(filename, expandLinks="relative")
print(filename)
```

FileProgressBar *A progress bar that sets the size of a file accordingly*

Description

Package: R.utils
Class FileProgressBar

```
Object
~~|
~~+--ProgressBar
~~~~~~~|
~~~~~~~+--FileProgressBar
```

Directly known subclasses:

public static class **FileProgressBar**
extends ProgressBar

Usage

```
FileProgressBar(pathname=NULL, ...)
```

Arguments

pathname	The pathname of the output file.
...	Other arguments accepted by the ProgressBar constructor.

Details

A progress bar that sets the size of a file accordingly. This class useful to check the progress of a batch job by just querying the size of a file, for instance, via ftp.

Fields and Methods

Methods:

cleanup	Removes the progress file for a file progress bar.
update	Updates file progress bar.

Methods inherited from ProgressBar:
as.character, getBarString, increase, isDone, reset, setMaxValue, setProgress, setStepLength, setTicks, setValue, update

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
  ## Not run:

# Creates a progress bar (of length 100) that displays it self as a file.
pb <- FileProgressBar("~/progress.simulation")
reset(pb)
while (!isDone(pb)) {
  x <- rnorm(3e4)
  increase(pb)
  # Emulate a slow process
  if (interactive()) Sys.sleep(0.1)
  Sys.sleep(0.01)
}


## End(Not run)
```

---

finalizeSession *Function to call for finalizing the R session*

Description

Function to call for finalizing the R session. When called, all registered "onSessionExit" hooks (functions) are called. To define such hooks, use the onSessionExit() function.

This method should not be used by the user.

Usage

```
## Default S3 method:
finalizeSession(...)
```

Arguments

...	Not used.

Value

Returns (invisibly) the hooks successfully called.

Author(s)

Henrik Bengtsson

See Also

`onSessionExit`().

---

findSourceTraceback *Finds all 'srcfile' objects generated by source() in all call frames*

Description

Finds all 'srcfile' objects generated by source() in all call frames. This makes it possible to find out which files are currently scripted by `source`().

Usage

```
## Default S3 method:
findSourceTraceback(...)
```

Arguments

...	Not used.

Value

Returns a named list of `srcfile`() objects and/or `character` strings. The names of the list entries corresponds to the 'filename' value of each corresponding 'srcfile' object. The returned list is empty if `source`() was not called.

Author(s)

Henrik Bengtsson

See Also

See also `sourceutils`.

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Create two R script files where one source():s the other
# and both lists the traceback of filenames source():d.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
path <- tempdir();
pathnameA <- Arguments$getWritablePathname("foo.R", path=path);
pathnameB <- Arguments$getWritablePathname("bar.R", path=path);

code <- 'cat("BEGIN foo.R\n")';
code <- c(code, 'print(findSourceTraceback());');
code <- c(code, sprintf('source("%s");', pathnameB));
code <- c(code, 'cat("END foo.R\n")');
code <- paste(code, collapse="\n");
cat(file=pathnameA, code);

code <- 'cat("BEGIN bar.R\n")';
code <- c(code, 'x <- findSourceTraceback();');
code <- c(code, 'print(x);');
code <- c(code, 'cat("END bar.R\n")');
code <- paste(code, collapse="\n");
cat(file=pathnameB, code);


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Source the first file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
source(pathnameA, echo=TRUE);
```

---

gcat *Parses, evaluates and outputs a GString*

Description

Parses, evaluates and outputs a GString.

Usage

```
## Default S3 method:
gcat(..., file="", append=FALSE, envir=parent.frame())
```

Arguments

`...`	`character` strings passed to `gstring`().
`file`	A `connection`, or a pathname where to direct the output. If `""`, the output is sent to the standard output.
`append`	Only applied if `file` specifies a pathname If `TRUE`, then the output is appended to the file, otherwise the files content is overwritten.
`envir`	The `environment` in which the `GString` is evaluated.

Value

Returns (invisibly) a character string.

Author(s)

Henrik Bengtsson

See Also

gstring().

---

gcDLLs *Identifies and removes DLLs of packages already unloaded*

Description

Identifies and removes DLLs of packages already unloaded. When packages are unloaded, they are ideally also unloading any DLLs (also known as a dynamic shared object or library) they have loaded. Unfortunately, not all package do this resulting in "stray" DLLs still being loaded and occupying R's limited registry. These functions identifies and removes such DLLs.

Usage

```
gcDLLs(gc=TRUE, quiet=TRUE)
```

Arguments

gc	If TRUE, if there are stray DLLs, then the garbage collector is run before unloading those DLLs. This is done in order to trigger any finalizers, of which some may need those DLLs, to be called.
quiet	If FALSE, a message is outputted for every stray DLL that is unloaded.

Details

If a library fails to unload, an informative warning is generated.

Value

Returns (invisibly) the set of stray DLLs identified.

How to unload DLLs in package (for package developers)

To unload a package DLL whenever the package in unloaded, add the following to your package:

```
.onUnload <- function(libpath) {
    ## (1) Force finalizers to be called before removing the DLL
    ##     in case some of them need the DLL.
    gc()

    ## (2) Unload the DLL for this package
    library.dynam.unload(.packageName, libpath)
}
```

Author(s)

Henrik Bengtsson

See Also

getLoadedDLLs().

---

getAbsolutePath *Gets the absolute pathname string*

Description

Gets the absolute pathname string.

Usage

```
## Default S3 method:
getAbsolutePath(pathname, workDirectory=getwd(), expandTilde=FALSE, ...)
```

Arguments

pathname	A character string of the pathname to be converted into an absolute pathname.
workDirectory	A character string of the current working directory.
expandTilde	If TRUE, tilde (~) is expanded to the corresponding directory, otherwise not.
...	Not used.

Details

This method will replace replicated slashes ('/') with a single one, except for the double forward slashes prefixing a Microsoft Windows UNC (Universal Naming Convention) pathname.

Value

Returns a character string of the absolute pathname.

Author(s)

Henrik Bengtsson

See Also

isAbsolutePath().

---

getParent *Gets the string of the parent specified by this pathname*

Description

Gets the string of the parent specified by this pathname. This is basically, by default the string before the last path separator of the absolute pathname.

Usage

```
## Default S3 method:
getParent(pathname, depth=1L, fsep=.Platform$file.sep, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
depth	An integer specifying how many generations up the path should go.
fsep	A character string of the file separator.
...	Not used.

Value

Returns a character string if the parent exists, otherwise NULL.

Author(s)

Henrik Bengtsson

getRelativePath *Gets the relative pathname relative to a directory*

Description

Gets the relative pathname relative to a directory.

Usage

```
## Default S3 method:
getRelativePath(pathname, relativeTo=getwd(), caseSensitive=NULL, ...)
```

Arguments

pathname	A character string of the pathname to be converted into an relative pathname.
relativeTo	A character string of the reference pathname.
caseSensitive	If TRUE, the comparison is case sensitive, otherwise not. If NULL, it is decided from the relative path.
...	Not used.

Details

In case the two paths are on different file systems, for instance, C:/foo/bar/ and D:/foo/, the method returns pathname as is.

Value

Returns a character string of the relative pathname.

Non-case sensitive comparison

If caseSensitive == NULL, the relative path is used to decide if the comparison should be done in a case-sensitive mode or not. The current check is if it is a Windows path or not, that is, if the relative path starts with a device letter, then the comparison is non-case sensitive.

Author(s)

Henrik Bengtsson

See Also

getAbsolutePath(). isAbsolutePath().

Examples

```
getRelativePath("foo", "foo")                       # "."
getRelativePath("foo/bar", "foo")                   # "bar"
getRelativePath("foo/bar", "foo/bar/yah")           # ".."
getRelativePath("foo/bar/cool", "foo/bar/yah/sub/") # "../../cool"
getRelativePath("/tmp/bar/", "/bar/foo/")           # "../../tmp/bar"

# Windows
getRelativePath("C:/foo/bar/", "C:/bar/")           # "../foo/bar"
getRelativePath("C:/foo/bar/", "D:/bar/")           # "C:/foo/bar"
```

GString *Character string with advanced substitutions*

Description

Package: R.utils
Class GString

```
character
~~|
~~+--GString
```

Directly known subclasses:

public static class **GString**
extends character

Usage

```
GString(..., sep="")
```

Arguments

...	one or more objects, to be coerced to character vectors.
sep	A character string to separate the terms.

Fields and Methods

Methods:

as.character	Gets the processed character string.
evaluate	Parses and evaluates a GString.
gcat	-

getBuiltinDate	Gets the current date.
getBuiltinDatetime	Gets the current date and time.
getBuiltinHostname	Gets the hostname of the system running R.
getBuiltinOs	Gets the operating system of the running machine.
getBuiltinPid	Gets the process id of the current R session.
getBuiltinRhome	Gets the path where R is installed.
getBuiltinRversion	Gets the current R version.
getBuiltinTime	Gets the current time.
getBuiltinUsername	Gets the username of the user running R.
getRaw	Gets the unprocessed GString.
getVariableValue	Gets a variable value given a name and attributes.
gstring	-
parse	Parses a GString.
print	Prints the processed GString.

Methods inherited from character:
Ops,nonStructure,vector-method, Ops,structure,vector-method, Ops,vector,nonStructure-method, Ops,vector,structure-method, all.equal, as.Date, as.POSIXlt, as.data.frame, as.raster, coerce,ANY,character-method, coerce,character,SuperClassM method, coerce,character,signature-method, coerce<-,ObjectsWithPackage,character-method, coerce<-,signature,character-method, downloadFile, formula, getDLLRegisteredRoutines, isOpen, toAsciiRegExprPattern, toFileListTree, uses

Author(s)

Henrik Bengtsson

See Also

For convenience, see functions gstring() and gcat().

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# First example
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
who <- "world"

# Compare this...
cat(as.character(GString("Hello ${who}\n")))

# ...to this.
cat(GString("Hello ${who}\n"))

# Escaping
cat(as.character(GString("Hello \${who}\n")))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Looping over vectors
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
x <- 1:5
y <- c("hello", "world")
cat(as.character(GString("(x,y)=(${x},${y})")), sep=", ")
cat("\n")

cat(as.character(GString("(x,y)=(${x},$[capitalize]{y})")), sep=", ")
cat("\n")


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Predefined ("builtin") variables
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cat(as.character(GString("Hello ${username} on host ${hostname} running ",
"R v${rversion} in process #${pid} on ${os}. R is installed in ${rhome}.")))


# Other built-in variables/functions...
cat(as.character(GString("Current date: ${date}\n")))
cat(as.character(GString("Current date: $[format='%d/%m/%y']{date}\n")))
cat(as.character(GString("Current time: ${time}\n")))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Evaluating inline R code
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cat(as.character(GString("Simple calculation: 1+1=${`1+1`}\n")))
cat(as.character(GString("Alternative current date: ${`date()`}\n")))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Function values
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Call function rnorm with arguments n=1, i.e. rnorm(n=1)
cat(as.character(GString("Random normal number: $[n=1]{rnorm}\n")))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Global search-replace feature
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Replace all '-' with '.'
cat(as.character(GString("Current date: ${date/-/.}\n")))
# Another example
cat(as.character(GString("Escaped string: 12*12=${`12*12`/1/}\n")))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Defining new "builtin" function values
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Define your own builtin variables (functions)
setMethodS3("getBuiltinAletter", "GString", function(object, ...) {
  base::letters[runif(1, min=1, max=length(base::letters))]
})
```

```
cat(as.character(GString("A letter: ${aletter}\n")))
cat(as.character(GString("Another letter: ${aletter}\n")))


# Another example
setMethodS3("getBuiltinGstring", "GString", function(object, ...) {
  # Return another GString.
  GString("${date} ${time}")
})

cat(as.character(GString("Advanced example: ${gstring}\n")))


# Advanced example
setMethodS3("getBuiltinRunif", "GString", function(object, n=1, min=0, max=1, ...) {
  formatC(runif(n=n, min=min, max=max), ...)
})

cat(as.character(GString("A random number: ${runif}\n")))
n <- 5
cat(as.character(GString("${n} random numbers: ")))
cat(as.character(GString("$[n=n, format='f']{runif}")))
cat("\n")


# Advanced options.
# Options are parsed as if they are elements in a list, e.g.
#   list(n=runif(n=1,min=1,max=5), format='f')
cat(as.character(GString("$Random number of numbers: ")))
cat(as.character(GString("$[n=runif(n=1,min=1,max=5), format='f']{runif}")))
cat("\n")
```

`gstring` *Parses and evaluates a GString into a regular string*

Description

Parses and evaluates a GString into a regular string.

Usage

```
## Default S3 method:
gstring(..., file=NULL, path=NULL, envir=parent.frame())
```

Arguments

`...`	`character` strings.
`file, path`	Alternatively, a file, a URL or a `connection` from with the strings are read. If a file, the `path` is prepended to the file, iff given.
`envir`	The `environment` in which the `GString` is evaluated.

Value

Returns a character string.

Author(s)

Henrik Bengtsson

See Also

gcat().

`hasUrlProtocol` *Checks if one or several pathnames has a URL protocol*

Description

Checks if one or several pathnames has a URL protocol.

Usage

```
## Default S3 method:
hasUrlProtocol(pathname, ...)
```

Arguments

`pathname`	A character vector.
`...`	Not used.

Value

Returns a logical vector.

Author(s)

Henrik Bengtsson

`hpaste` *Concatenating vectors into human-readable strings*

Description

Concatenating vectors into human-readable strings such as "1, 2, 3, ..., 10".

Usage

```
## Default S3 method:
hpaste(..., sep="", collapse=", ", lastCollapse=NULL,
  maxHead=if (missing(lastCollapse)) 3 else Inf,
  maxTail=if (is.finite(maxHead)) 1 else Inf, abbreviate="...")
```

Arguments

`...`	Arguments to be pasted.
`sep`	A `character` string used to concatenate the arguments in `...`, if more than one.
`collapse, lastCollapse`	The `character` strings to collapse the elements together, where `lastCollapse` is specifying the collapse string used between the last two elements. If `lastCollapse` is `NULL` (default), it is corresponds to using the default collapse.
`maxHead, maxTail, abbreviate`	Non-negative `integer`s (also `Inf`) specifying the maximum number of elements of the beginning and then end of the vector to be outputted. If `n = length(x)` is greater than `maxHead+maxTail+1`, then `x` is truncated to consist of `x[1:maxHead]`, `abbreviate`, and `x[(n-maxTail+1):n]`.

Details

`hpaste(..., sep=" ", maxHead=Inf)` corresponds to `paste(..., sep=" ", collapse=", ")`.

Value

Returns a `character` string.

Author(s)

Henrik Bengtsson

See Also

Internally `paste`() is used.

Examples

```
# Some vectors
x <- 1:6
y <- 10:1
z <- LETTERS[x]

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Abbreviation of output vector
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
printf("x = %s.\n", hpaste(x))
## x = 1, 2, 3, ..., 6.

printf("x = %s.\n", hpaste(x, maxHead=2))
## x = 1, 2, ..., 6.

printf("x = %s.\n", hpaste(x), maxHead=3) # Default
## x = 1, 2, 3, ..., 6.

# It will never output 1, 2, 3, 4, ..., 6
printf("x = %s.\n", hpaste(x, maxHead=4))
## x = 1, 2, 3, 4, 5, 6.

# Showing the tail
printf("x = %s.\n", hpaste(x, maxHead=1, maxTail=2))
## x = 1, ..., 5, 6.

# Turning off abbreviation
printf("y = %s.\n", hpaste(y, maxHead=Inf))
## y = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

## ...or simply
printf("y = %s.\n", paste(y, collapse=", "))
## y = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Adding a special separator before the last element
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Change last separator
printf("x = %s.\n", hpaste(x, lastCollapse=", and "))
## x = 1, 2, 3, 4, 5, and 6.



# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Backward compatibility with paste()
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
s1 <- hpaste(x, maxHead=Inf)
s2 <- paste(x, collapse=", ")
printf("s = %s.\n", s1);
stopifnot(identical(s1, s2))
```

```
s1 <- hpaste('<', x, '>', maxHead=Inf)
s2 <- paste('<', x, '>', sep="", collapse=", ")
printf("s = %s.\n", s1);
stopifnot(identical(s1, s2))

s1 <- hpaste(x, y, z, sep="/", maxHead=Inf)
s2 <- paste(x, y, z, sep="/", collapse=", ")
printf("s = %s.\n", s1);
stopifnot(identical(s1, s2))

s1 <- hpaste(x, collapse=NULL, maxHead=Inf)
s2 <- paste(x, collapse=NULL)
stopifnot(identical(s1, s2))
```

`inAnyInterval.numeric` *Checks if a set of values are inside one or more intervals*

Description

Checks if a set of values are inside one or more intervals.

Usage

```
## S3 method for class 'numeric'
inAnyInterval(...)
```

Arguments

`...`	Arguments passed to `*mapToIntervals()`.

Value

Returns a `logical vector`.

Author(s)

Henrik Bengtsson

See Also

`mapToIntervals()`.

insert *Insert values to a vector at certain positions*

Description

Insert values to a vector at certain positions.

Usage

```
## Default S3 method:
insert(x, ats, values=NA, useNames=TRUE, ...)
```

Arguments

x	The vector of data values.
ats	The indices of x where the values should be inserted.
values	A list or a vector of the values to be inserted. Should be of same length as ats, unless if a single value when it is automatically extended without a warning.
useNames	If FALSE, the names attribute is dropped/ignored, otherwise not. Only applied if argument x is named.
...	Not used.

Author(s)

Henrik Bengtsson

See Also

append() takes argument after (a scalar). For example, append(x, y, after=after) == insert(x, values=y, ats=aft
Contrary to append(), insert() accepts a vector of insert indices.

Examples

```
# Insert NAs (default) between all values
y <- c(a=1, b=2, c=3)
print(y)
x <- insert(y, ats=2:length(y))
Ex <- c(y[1], NA_real_, y[2], NA_real_, y[3])
print(x)
stopifnot(identical(x,Ex))

# Insert at first position
y <- c(a=1, b=2, c=3)
print(y)
x <- insert(y, ats=1, values=rep(NA_real_,2))
Ex <- c(NA_real_,NA_real_,y)
print(x)
```

```
stopifnot(identical(x,Ex))

x <- insert(y, ats=1, values=rep(NA_real_,2), useNames=FALSE)
print(x)

# Insert at last position (names of 'values' are ignored
# because input vector has no names)
x <- insert(1:3, ats=4, values=c(d=2, e=1))
Ex <- c(1:3,2,1)
print(x)
stopifnot(identical(x,Ex))


# Insert in the middle of a vector
x <- insert(c(1,3,2,1), ats=2, values=2)
print(x)
stopifnot(identical(as.double(x),as.double(Ex)))


# Insert multiple vectors at multiple indices at once
x0 <- c(1:4, 8:11, 13:15)

x <- insert(x0, at=c(5,9), values=list(5:7,12))
print(x)
Ex <- 1:max(x)
stopifnot(identical(as.double(x),as.double(Ex)))

x <- insert(x0, at=c(5,9,12), values=list(5:7,12,16:18))
print(x)
Ex <- 1:max(x)
stopifnot(identical(as.double(x),as.double(Ex)))


# Insert missing indices
Ex <- 1:20
missing <- setdiff(Ex, x0)
x <- x0
for (m in missing)
  x <- insert(x, ats=m, values=m)
print(x)
stopifnot(identical(as.double(x),as.double(Ex)))
```

`installPackages` *Install R packages by name or URL*

Description

Install R packages by name or URL.

Usage

```
## Default S3 method:
installPackages(pkgs, types="auto", repos=getOption("repos"), ..., destPath=".",
  cleanup=TRUE)
```

Arguments

pkgs	A character vector specifying the names and/or the URLs of the R packages to be installed.
types	A character vector of corresponding package types.
repos	A character vector of package repository URLs.
...	Additional arguments passed to install.packages.
destPath	Path where any downloaded files are saved.
cleanup	If TRUE, downloaded and successfully installed package files are removed, otherwise not.

Value

Returns nothing.

Limitations

This method cannot install any packages that are already in use. Certain packages are always in use when calling this method, e.g. **R.methodsS3**, **R.oo**, and **R.utils**.

Author(s)

Henrik Bengtsson

Examples

```
## Not run:
 installPackages("R.rsp")
 installPackages("http://cran.r-project.org/src/contrib/Archive/R.rsp/R.rsp_0.8.2.tar.gz")
 installPackages("http://cran.r-project.org/bin/windows/contrib/r-release/R.rsp_0.9.17.zip")

## End(Not run)
```

intervalsToSeq.matrix *Generates a vector of indices from a matrix of intervals*

Description

Generates a vector of indices from a matrix of intervals.

Usage

```
## S3 method for class 'matrix'
intervalsToSeq(fromTo, sort=FALSE, unique=FALSE, ...)
```

Arguments

fromTo	An Nx2 integer matrix.
sort	If TRUE, the returned indices are ordered.
unique	If TRUE, the returned indices are unique.
...	Not used.

Author(s)

Henrik Bengtsson

See Also

seqToIntervals().

Examples

```
## Not run: See example(seqToIntervals)
```

intToBin *Converts an integer to a binary/octal/hexadecimal number*

Description

Converts an integer to a binary/octal/hexadecimal number.

Usage

```
intToBin(x)
intToOct(x)
intToHex(x)
```

Arguments

x	A numeric vector of integers to be converted.

Details

For length(x) > 1, the number of characters in each of returned elements is the same and driven by the x element that requires the highest number of character - all other elements are padded with zeros (or ones for negative values). This is why we for instance get intToHex(15) == "f" but intToHex(15:16) == c("0f", "10").

The supported range for intToHex(), intToOct(), and intToBin() is that of R integers, i.e. [-.Machine$integer.max, +.Machine$integer.max] where. .Machine$integer.max is 2^31-1. This limitation is there such that negative values can be converted too.

Value

Returns a character string of length length(x). For coercions out of range, NA_character_ is returned for such elements.

Author(s)

Henrik Bengtsson

---

isAbsolutePath *Checks if this pathname is absolute*

Description

Checks if this pathname is absolute.

Usage

```
## Default S3 method:
isAbsolutePath(pathname, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
...	Not used.

Value

Returns a TRUE if the pathname is absolute, otherwise FALSE.

Author(s)

Henrik Bengtsson

isDirectory *Checks if the file specification is a directory*

Description

Checks if the file specification is a directory.

Usage

```
## Default S3 method:
isDirectory(pathname, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
...	Not used.

Value

Returns TRUE if the file specification is a directory, otherwise FALSE is returned.

Symbolic links

This function follows symbolic links (also on Windows) and returns a value based on the link target (rather than the link itself).

Author(s)

Henrik Bengtsson

See Also

To check if it is a file see isFile(). Internally file.info() is used. See also file_test.

isEof.connection *Checks if the current file position for a connection is at the 'End of File'*

Description

Checks if the current file position for a connection is at the 'End of File'.

Usage

```
## S3 method for class 'connection'
isEof(con, ...)
```

Arguments

con	A connection.
...	Not used.

Details

Internally seek() is used, which according to to the R help is discouraged on Windows. However, after many years of large-scale testing on various Windows versions and file systems we have yet to experience issues with using seek() on Windows.

Value

Returns a logical.

Author(s)

Henrik Bengtsson

See Also

For more information see connection.

---

isFile *Checks if the file specification is a file*

Description

Checks if the file specification is a file.

Usage

```
## Default S3 method:
isFile(pathname, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
...	Not used.

Value

Returns TRUE if the file specification is a file, otherwise FALSE is returned.

Symbolic links

This function follows symbolic links (also on Windows) and returns a value based on the link target (rather than the link itself).

Author(s)

Henrik Bengtsson

See Also

To check if it is a directory see isDirectory(). Internally file.info() is used. See also file_test.

---

isOpen.character *Checks if there is an open connection to a file*

Description

Checks if there is an open connection to a file.

Usage

```
## S3 method for class 'character'
isOpen(pathname, rw=c("read", "write"), ...)
```

Arguments

pathname	An character vector.
rw	A character vector. If "read", a file is considered to be open if there exist an open connection that can read from that file. If "write", a file is considered to be open if there exist an open connection that can write to that file. Both these values may be specified.
...	Not used.

Value

Returns a logical vector indicating for each file whether there exists an open file connection or not.

Author(s)

Henrik Bengtsson

See Also

See isOpen() in connections. showConnections().

isPackageInstalled *Checks if a package is installed or not*

Description

Checks if a package is installed or not.

Usage

```
## Default S3 method:
isPackageInstalled(package, ...)
```

Arguments

package	A character vector of package names.
...	Not used.

Value

Returns a logical vector.

Author(s)

Henrik Bengtsson

See Also

isPackageLoaded().

isPackageLoaded *Checks if a package is loaded or not*

Description

Checks if a package is loaded or not. Note that, contrary to require(), this function does not load the package if not loaded.

Usage

```
## Default S3 method:
isPackageLoaded(package, version=NULL, ...)
```

Arguments

package	The name of the package.
version	A character string specifying the version to test for. If NULL, any version is tested for.
...	Not used.

Value

Returns a logical.

Author(s)

Henrik Bengtsson

See Also

To check if a package is installed or not, see isPackageInstalled().

---

isReplicated *Identifies all entries with replicated values*

Description

Identifies all entries with replicated values, that is, with values that exist more than once.

Usage

```
isReplicated(x, ...)
replicates(x, ...)
```

Arguments

x	A vector of length K.
...	Additional arguments passed to duplicated().

Details

Let `reps <- isReplicated(x)`. Then it always holds that:

- `reps == rev(isReplicated(rev(x)))`
- `reps == duplicated(x) | duplicated(x, fromLast=TRUE)`
- `reps == !is.element(x, setdiff(x, unique(x[duplicated(x)])))`

Value

A logical vector of length K, where TRUE indicates that the value exists elsewhere, otherwise not.

Author(s)

Henrik Bengtsson

See Also

Internally duplicated() is used. See also isSingle().

Examples

```
x <- c(1,1,2,3,4,2,1)
x <- base::letters[x]
print(x)

# Identify entries with replicated values
reps <- isReplicated(x)
print(x[reps])
stopifnot(x[reps] == replicates(x))

# Identify entries with unique values
print(x[!reps])
stopifnot(x[!reps] == singles(x))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Validation
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
x <- c(1,1,2,3,4,2,1)
x <- base::letters[x]
reps <- isReplicated(x)

stopifnot(all(table(x[reps]) > 1))
stopifnot(all(table(x[!reps]) == 1))
stopifnot(all(reps == rev(isReplicated(rev(x)))))
stopifnot(all(reps == duplicated(x) | duplicated(x, fromLast=TRUE)))
stopifnot(all(reps == !is.element(x, setdiff(x, unique(x[duplicated(x)])))))
stopifnot(all(sort(c(singles(x), replicates(x))) == sort(x)))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Benchmarking singles()
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
set.seed(0xBEEF)
n <- 1e6
x <- sample(1:(n/2), size=n, replace=TRUE)
t <- system.time({
  s <- isSingle(x)
})
print(sum(s))

t0 <- system.time({
  s0 <- !(x %in% x[duplicated(x)]);
```

```
})
print(t/t0)
stopifnot(all(s == s0))
```

`isSingle` *Identifies all entries that exists exactly once*

Description

Identifies all entries that exists exactly once.

Usage

```
isSingle(x, ...)
singles(x, ...)
```

Arguments

`x`	A `vector` of length K.
`...`	Additional arguments passed to `isReplicated`().

Value

A `logical` `vector` of length K, indicating whether the value is unique or not.

Author(s)

Henrik Bengtsson

See Also

Internally `isReplicated`() is used.

`isUrl` *Checks if one or several pathnames is URLs*

Description

Checks if one or several pathnames is URLs.

Usage

```
## Default S3 method:
isUrl(pathname, ...)
```

Arguments

pathname	A character vector.
...	Not used.

Value

Returns a logical vector of either TRUE or FALSE.

Author(s)

Henrik Bengtsson

---

isZero *Checks if a value is (close to) zero or not*

Description

Checks if a value (or a vector of values) is (close to) zero or not where "close" means if the absolute value is less than neps*eps. *Note that* x == 0 *will not work in all cases.*

By default eps is the smallest possible floating point value that can be represented by the running machine, i.e. .Machine$double.eps and neps is one. By changing neps it is easy to adjust how close to zero "close" means without having to know the machine precision (or remembering how to get it).

Usage

```
## Default S3 method:
isZero(x, neps=1, eps=.Machine$double.eps, ...)
```

Arguments

x	A vector of values.
eps	The smallest possible floating point.
neps	A scale factor of eps specifying how close to zero "close" means. If eps is the smallest value such that 1 + eps != 1, i.e. .Machine$double.eps, neps must be greater or equal to one.
...	Not used.

Value

Returns a logical vector indicating if the elements are zero or not.

Author(s)

Henrik Bengtsson

See Also

all.equal(). Comparison. .Machine.

Examples

```
x <- 0
print(x == 0)      # TRUE
print(isZero(x))   # TRUE

x <- 1
print(x == 0)      # FALSE
print(isZero(x))   # FALSE

x <- .Machine$double.eps
print(x == 0)      # FALSE
print(isZero(x))   # FALSE

x <- 0.9*.Machine$double.eps
print(x == 0)      # FALSE
print(isZero(x))   # TRUE

# From help(Comparisions)
x1 <- 0.5 - 0.3
x2 <- 0.3 - 0.1
print(x1 - x2)
print(x1 == x2)                           # FALSE on most machines
print(identical(all.equal(x1, x2), TRUE)) # TRUE everywhere
print(isZero(x1-x2))                      # TRUE everywhere
```

Java *Static class for Java related methods*

Description

Package: R.utils
Class Java

Object
~~|
~~+--Java

Directly known subclasses:

public static class **Java**
extends Object

Static class that provides methods for reading and writing Java data types. Currently the following data types are supported: byte, short and int. R character strings can be written as UTF-8 formatted strings, which can be read by Java. Currently on Java String's that contain ASCII characters can be imported into R. The reason for this is that other characters are translated into non-eight bits data, e.g. 16- and 24-bits, which the readChar() method currently does not support.

Furthermore, the Java class defines some static constants describing the minimum and maximum value of some of the common Java data types: `BYTE.MIN`, `BYTE.MAX` `SHORT.MIN`, `SHORT.MAX` `INT.MIN`, `INT.MAX` `LONG.MIN`, and `LONG.MAX`.

Usage

```
Java()
```

Fields and Methods

Methods:

asByte	Converts a numeric to a Java byte.
asInt	Converts an numeric to a Java integer.
asLong	Converts a numeric to a Java long.
asShort	Converts a numeric to a Java short.
readByte	Reads a Java formatted byte (8 bits) from a connection.
readInt	Reads a Java formatted int (32 bits) from a connection.
readShort	Reads a Java formatted short (16 bits) from a connection.
readUTF	Reads a Java (UTF-8) formatted string from a connection.
writeByte	Writes a byte (8 bits) to a connection in Java format.
writeInt	Writes a integer (32 bits) to a connection in Java format.
writeShort	Writes a short (16 bits) to a connection in Java format.
writeUTF	Writes a string to a connection in Java format (UTF-8).

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
pathname <- tempfile()

# Open the temporary file for writing
out <- file(pathname, open="wb")
b <- -128:127
```

```
Java$writeByte(out, b)
s <- -32768:32767
Java$writeShort(out, s)
i <- c(-2147483648, -2147483647, -1, 0, +1, 2147483646, 2147483647);
Java$writeInt(out, i)
str <- c("This R string was written (using the UTF-8 format) using",
         "the static methods of the Java class in the R.io package.")
str <- paste(str, collapse="\n")
Java$writeUTF(out, str)
close(out)

# Open the temporary file for reading
inn <- file(pathname, open="rb")

bfr <- Java$readByte(inn, n=length(b))
cat("Read ", length(bfr), " bytes.\n", sep="")
if (!identical(bfr, b))
  throw("Failed to read the same data that was written.")

bfr <- Java$readShort(inn, n=length(s))
cat("Read ", length(bfr), " shorts.\n", sep="")
if (!identical(bfr, s))
  throw("Failed to read the same data that was written.")

bfr <- Java$readInt(inn, n=length(i))
cat("Read ", length(bfr), " ints.\n", sep="")
if (!identical(bfr, i))
  throw("Failed to read the same data that was written.")

bfr <- Java$readUTF(inn)
cat("Read ", nchar(bfr), " UTF characters:\n", "'", bfr, "'\n", sep="")

close(inn)

file.remove(pathname)
```

`lastModified` *Gets the time when the file was last modified*

Description

Gets the time when the file was last modified. The time is returned as a `POSIXct` object.

Usage

```
## Default S3 method:
lastModified(pathname, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
...	Not used.

Value

Returns POSIXct object specifying when the file was last modified. If the file does not exist or it is a directory, 0 is returned.

Symbolic links

This function follows symbolic links (also on Windows) and returns a value based on the link target (rather than the link itself).

Author(s)

Henrik Bengtsson

See Also

Internally file.info() is used.

LComments *The LComments class*

Description

Package: R.utils
Class LComments

```
Object
~~|
~~+--SmartComments
~~~~~~~|
~~~~~~~+--VComments
~~~~~~~~~~~~|
~~~~~~~~~~~~+--LComments
```

Directly known subclasses:

public static class **LComments**
extends VComments

The LComments class.

This class, is almost identical to the super class, except that the constructor has different defaults.

Usage

```
LComments(letter="L", verboseName="log", ...)
```

Arguments

letter	The smart letter.
verboseName	The name of the verbose object.
...	Not used.

Fields and Methods

Methods:
No methods defined.

Methods inherited from VComments:
convertComment, reset, validate

Methods inherited from SmartComments:
compile, convertComment, parse, reset, validate

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

---

listDirectory *Gets the file names in the directory*

Description

Gets the file names in the directory.

Contrary to `list.files()`, this method guarantees to work recursively. Moreover, when subdirectories are processed recursively, directory names are also returned.

Usage

```
## Default S3 method:
listDirectory(path=".", pattern=NULL, recursive=FALSE, allNames=FALSE, fullNames=FALSE,
  ...)
```

Arguments

path	A path to be listed.
pattern	A character string of the filename pattern passed. See list.files() for more details.
recursive	If TRUE, subdirectories are recursively processed, and not if FALSE. Alternatively, the maximum recursive depth can be specified as a non-negative numeric, where FALSE corresponds to 0L depth and TRUE corresponds +Inf depth.
allNames	If TRUE, also files starting with a period are returned.
fullNames	If TRUE, the full path names are returned.
...	Not used.

Value

Returns a vector of file names.

Recursive searching

Recursive searching of directory structure is done breath-first in a lexicographic order.

Author(s)

Henrik Bengtsson

See Also

Internally list.files() is used.

---

loadObject *Method to load object from a file or a connection*

Description

Method to load object from a file or a connection, which previously have been saved using saveObject().

Usage

```
## Default S3 method:
loadObject(file, path=NULL, format=c("auto", "xdr", "rds"), ...)
```

Arguments

file	A filename or connection to read the object from.
path	The path where the file exists.
format	File format.
...	Not used.

Details

The main difference from this method and load() in the **base** package, is that this one returns the object read rather than storing it in the global environment by its default name. This makes it possible to load objects back using any variable name.

Value

Returns the saved object.

Author(s)

Henrik Bengtsson

See Also

saveObject() to save an object to file. Internally load() is used. See also loadToEnv(). See also saveRDS().

---

mapToIntervals.numeric

Maps values to intervals

Description

Maps values to intervals by returning an index vector specifying the (first) interval that each value maps to, if any.

Usage

```
## S3 method for class 'numeric'
mapToIntervals(x, intervals, includeLower=TRUE, includeUpper=TRUE, ...)
```

Arguments

x	A numeric vector of K values to be matched.
intervals	The N intervals to be matched against. If an Nx2 numeric matrix, the first column should be the lower bounds and the second column the upper bounds of each interval. If a numeric vector of length 2N, each consecutive pair should be the lower and upper bounds of an interval.
includeLower, includeUpper	If TRUE, the lower (upper) bound of *each* interval is included in the test, otherwise not.
...	Not used.

Value

Returns an integer vector of length K. Values that do not map to any interval have return value NA.

Author(s)

Henrik Bengtsson

See Also

inAnyInterval(). match(). findInterval(). cut().

---

mergeIntervals.numeric

Merges intervals

Description

Merges intervals by returning an index vector specifying the (first) interval that each value maps to, if any.

Usage

```
## S3 method for class 'numeric'
mergeIntervals(intervals, ...)
```

Arguments

intervals	The N intervals to be merged. If an Nx2 numeric matrix, the first column should be the lower bounds and the second column the upper bounds of each interval. If a numeric vector of length 2N, each consecutive pair should be the lower and upper bounds of an interval.
...	Not used.

Details

The upper and lower bounds are considered to be inclusive, that is, all intervals are interpreted to be of form [a,b]. There is currently no way to specify intervals with open bounds, e.g. (a,b].

Furthermore, the bounds are currently treated as real values. For instance, merging [0,1] and [2,3] will return the same intervals. Note, if integer intervals were treated specially, we would merge these intervals to integer interval [0,3] == {0,1,2,3}.

Value

Returns a matrix (or a vector) of M intervals, where M <= N. The intervals are ordered by their lower bounds. The @mode of the returned intervals is the same as the mode of the input intervals.

Author(s)

Henrik Bengtsson

See Also

inAnyInterval(). match().

---

mkdirs *Creates a directory including any necessary but nonexistent parent directories*

Description

Creates a directory including any necessary but nonexistent parent directories.

Usage

```
## Default S3 method:
mkdirs(pathname, mustWork=FALSE, maxTries=5L, ...)
```

Arguments

pathname	A character string of the pathname to be checked.
mustWork	If TRUE and the directory does not already exists or is failed to be created, an error is thrown, otherwise not.
maxTries	A positive integer specifying how many times the method should try to create a missing directory before giving up.
...	Not used.

Value

Returns TRUE if the directory was successfully created, otherwise FALSE. Note that if the directory already exists, FALSE is returned.

Slow file systems

On very rare occasions, we have observed on a large shared file system that if one tests for the existence of a directory immediately after creating it with dir.create(), it may appear not to be created. We believe this is due to the fact that there is a short delay between creating a directory and that information being fully propagated on the file system. To minimize the risk for such false assertions on "slow" file systems, this method tries to create a missing directory multiple times (argument maxTries) (while waiting a short period of time between each round) before giving up.

Author(s)

Henrik Bengtsson

See Also

Internally dir.create() is used.

mout *Miscellaneous functions for outputting via message()*

Description

Miscellaneous functions for outputting via message(). These "m*" methods work analogously to their corresponding "*" methods print(), cat(), show, str, and printf() but uses message() to output the content, which in turn outputs to standard error. The mout() method can be used for all other output methods, e.g. mout(write(x, file=stdout())).

Usage

```
mout(..., appendLF=FALSE)
```

Arguments

...	Arguments passed to the underlying output method.
appendLF	A logical specifying whether to append a newline at the end or not.

Value

Returns what the message() returns.

Author(s)

Henrik Bengtsson

Examples

```
print(letters[1:8])
mprint(letters[1:8])

cat(c(letters[1:8], "\n"))
mcat(c(letters[1:8], "\n"))

str(letters[1:8])
mstr(letters[1:8])

printf("x=%d\n", 1:3)
mprintf("x=%d\n", 1:3)
```

mpager *A \"pager\" function that outputs to standard error*

Description

A \"pager\" function that outputs to standard error and is compatible with file.show().

Usage

```
mpager(files, header=NULL, title="R Information", delete.file=FALSE)
```

Arguments

files	A character vector of K pathnames.
header	A character vector of K headers.
title	A character string.
delete.file	If TRUE, the files are deleted after displayed, otherwise not.

Value

Returns nothing.

Author(s)

Henrik Bengtsson

See Also

file.show() and argument pager.

nullfile *Gets the pathname or a connection to the NULL device on the current platform*

Description

Gets the pathname or a connection to the NULL device on the current platform.

Usage

```
nullfile()
nullcon()
```

Value

`nullfile()` returns a `character` string, which is `"/dev/null"` except on Windows where it is `"nul:"`. `nullcon()` returns a *newly opened* (binary) `connection` to the NULL device - make sure to close it when no longer needed.

Author(s)

Henrik Bengtsson

See Also

In R (>= 3.6.0), there exists `base::nullfile()`, which is identical to `R.utils::nullfile()`.

`NullVerbose` *A Verbose class ignoring everything*

Description

Package: R.utils
Class NullVerbose

```
Object
~~|
~~+--Verbose
~~~~~~~|
~~~~~~~+--NullVerbose
```

Directly known subclasses:

public static class **NullVerbose**
extends Verbose

A Verbose class ignoring everything.

Usage

```
NullVerbose(...)
```

Arguments

`...`	Ignored.

Fields and Methods

Methods:

cat	-
enter	-
evaluate	-
exit	-
header	-
isOn	Checks if the output is on.
isVisible	Checks if a certain verbose level will be shown or not.
newline	-
print	-
printf	-
ruler	-
str	-
summary	-
writeRaw	All output methods.

Methods inherited from Verbose:
as.character, as.double, as.logical, capture, cat, enter, enterf, equals, evaluate, exit, getThreshold, getTimestampFormat, header, isOn, isVisible, less, more, newline, off, on, popState, print, printf, pushState, ruler, setDefaultLevel, setThreshold, setTimestampFormat, str, summary, timestamp, timestampOff, timestampOn, warnings, writeRaw

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
verbose <- Verbose()
cat(verbose, "A verbose messages")

verbose <- NullVerbose()
cat(verbose, "A verbose messages")   # Ignored
```

---

onGarbageCollect *Registers a function to be called when the R garbage collector is (detected to be) running*

Description

Registers a function to be called when the R garbage collector is (detected to be) running.

Usage

```
## Default S3 method:
onGarbageCollect(fcn, action=c("prepend", "append", "replace"), ...)
```

Arguments

fcn	A function to be called without argument.
action	A character string specifying how the hook function is added to list of hooks.
...	Not used.

Value

Returns (invisibly) the hooks successfully called.

Author(s)

Henrik Bengtsson

Examples

```
## Not run:
  onGarbageCollect(function(...) {
    message("The R garbage collector is running!")
  })

## End(Not run)
```

---

onSessionExit *Registers a function to be called when the R session finishes*

Description

Registers a function to be called when the R session finishes.

Usage

```
## Default S3 method:
onSessionExit(fcn, action=c("prepend", "append", "replace"), ...)
```

Arguments

fcn	A function to be called without argument.
action	A character string specifying how the hook function is added to list of hooks.
...	Not used.

Details

Functions registered this way are called when finalizeSession() is called. Moreover, when this package is loaded, the .Last() function is modified such that finalizeSession() is called. However, note that .Last() is *not* guaranteed to be called when the R session finished. For instance, the user may quit R by calling quit(callLast=FALSE). Moreover, when R is run in batch mode, .Last() is never called.

Value

Returns (invisibly) the hooks successfully called.

Author(s)

Henrik Bengtsson

See Also

.Last(). finalizeSession().

Examples

```
## Not run:
  onSessionExit(function(...) {
    message("Bye bye world!")
  })

  quit()

## End(Not run)
```

Options *The Options class*

Description

Package: R.utils
Class Options

Object
~~|
~~+--Options

Directly known subclasses:
Settings

public static class **Options**
extends Object

A class to set and get either options stored in a `list` tree structure.

Each option has a pathname. The format of a pathname is similar to a (Unix) filesystem pathname, e.g. "graphics/cex". See examples for more details.

Usage

```
Options(options=list(), ...)
```

Arguments

`options`	A tree `list` structure of options.
`...`	Not used.

Details

Note, this class and its methods do *not* operate on the global options structure defined in R (options).

Value

The constructor returns an Options object.

Fields and Methods

Methods:

`as.character`	Returns a character string version of this object.
`as.list`	Gets a list representation of the options.
`equals`	Checks if this object is equal to another Options object.
`getLeaves`	Gets all (non-list) options in a flat list.
`getOption`	Gets an option.
`hasOption`	Checks if an option exists.
`names`	Gets the full pathname of all (non-list) options.
`nbrOfOptions`	Gets the number of options set.
`setOption`	Sets an option.
`str`	Prints the structure of the options.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
local <- Options()

# Query a missing option
cex <- getOption(local, "graphics/cex")
cat("graphics/cex =", cex, "\n")  # Returns NULL

# Query a missing option with default value
cex <- getOption(local, "graphics/cex", defaultValue=1)
cat("graphics/cex =", cex, "\n")  # Returns NULL

# Set option and get previous value
oldCex <- setOption(local, "graphics/cex", 2)
cat("previous graphics/cex =", oldCex, "\n")  # Returns NULL

# Set option again and get previous value
oldCex <- setOption(local, "graphics/cex", 3)
cat("previous graphics/cex =", oldCex, "\n")  # Returns 2

# Query a missing option with default value, which is ignored
cex <- getOption(local, "graphics/cex", defaultValue=1)
cat("graphics/cex =", cex, "\n")  # Returns 3

# Query multiple options with multiple default values
multi <- getOption(local, c("graphics/cex", "graphics/pch"), c(1,2))
print(multi);

# Check existance of multiple options
has <- hasOption(local, c("graphics/cex", "graphics/pch"))
print(has);

# Get a subtree of options
graphics <- getOption(local, "graphics")
print(graphics)

# Get the complete tree of options
all <- getOption(local)
print(all)
```

---

patchCode *Patches installed and loaded packages and more*

Description

Patches installed and loaded packages and more.

Usage

```
## Default S3 method:
patchCode(paths=NULL, recursive=TRUE, suppressWarnings=TRUE,
  knownExtensions=c("R", "r", "S", "s"), verbose=FALSE, ...)
```

Arguments

paths	The path to the directory (and subdirectories) which contains source code that will patch loaded packages. If NULL, the patch path is given by the option R_PATCHES, If the latter is not set, the system environment with the same name is used. If neither is given, then ~/R-patches/ is used.
recursive	If TRUE, source code in subdirectories will also get loaded.
suppressWarnings	If TRUE, warnings will be suppressed, otherwise not.
knownExtensions	A character vector of filename extensions used to identify source code files. All other files are ignored.
verbose	If TRUE, extra information is printed while patching, otherwise not.
...	Not used.

Details

The method will look for source code files (recursively or not) that match known filename extensions. Each found source code file is then source()d.

If the search is recursive, subdirectories are entered if and only if either (1) the name of the subdirectory is the same as a *loaded* (and installed) package, or (2) if there is no installed package with that name. The latter allows common code to be organized in directories although it is still not assigned to packages.

Each of the directories given by argument paths will be processed one by one. This makes it possible to have more than one file tree containing patches.

To set an options, see options(). To set a system environment, see Sys.setenv(). The character ; is interpreted as a separator. Due to incompatibility with Windows pathnames, : is *not* a valid separator.

Value

Returns (invisibly) the number of files sourced.

Author(s)

Henrik Bengtsson

See Also

source(). library().

Examples

```
## Not run:
  # Patch all source code files in the current directory
  patchCode(".")

  # Patch all source code files in R_PATCHES
  options("R_PATCHES"="~/R-patches/")
  # alternatively, Sys.setenv("R_PATCHES"="~/R-patches/")
  patchCode()

## End(Not run)
```

---

popBackupFile *Drops a backup suffix from the backup pathname*

Description

Drops a backup suffix from the backup pathname and, by default, restores an existing backup file accordingly by renaming it.

Usage

```
## Default S3 method:
popBackupFile(filename, path=NULL, suffix=".bak", isFile=TRUE,
  onMissing=c("ignore", "error"), drop=TRUE, ..., verbose=FALSE)
```

Arguments

filename	The filename of the backup file.
path	The path of the file.
suffix	The suffix of the filename to be dropped.
isFile	If TRUE, the backup file must exist and will be renamed. If FALSE, it is only the pathname string that will be modified. For details, see below.
onMissing	A character string specifying what to do if the backup file does not exist.
drop	If TRUE, the backup file will be dropped in case the original file already exists or was successfully restored.
...	Not used.
verbose	A logical or Verbose.

Value

Returns the pathname with the backup suffix dropped.

Author(s)

Henrik Bengtsson

See Also

See pushBackupFile() for more details and an example.

---

popTemporaryFile *Drops a temporary suffix from the temporary pathname*

Description

Drops a temporary suffix from the temporary pathname and, by default, renames an existing temporary file accordingly.

Usage

```
## Default S3 method:
popTemporaryFile(filename, path=NULL, suffix=".tmp", isFile=TRUE, ..., verbose=FALSE)
```

Arguments

filename	The filename of the temporary file.
path	The path of the temporary file.
suffix	The suffix of the temporary filename to be dropped.
isFile	If TRUE, the temporary file must exist and will be renamed. If FALSE, it is only the pathname string that will be modified. For details, see below.
...	Not used.
verbose	A logical or Verbose.

Details

If isFile is FALSE, the pathname where the suffix of the temporary pathname has been dropped is returned. If isFile is TRUE, the temporary file is renamed. Then, if the temporary file does not exists or it was not successfully renamed, an exception is thrown.

Value

Returns the pathname with the temporary suffix dropped.

Author(s)

Henrik Bengtsson

See Also

See pushTemporaryFile() for more details and an example.

printf *C-style formatted output*

Description

C-style formatted output.

Usage

```
## Default S3 method:
printf(fmt, ..., sep="", file="")
```

Arguments

fmt	A character vector of format strings. See same argument for sprintf().
...	Additional arguments sprintf().
sep	A character vector of strings to append after each element.
file	A connection, or a character of a file to print to. See same argument for cat().

Value

Returns nothing.

Author(s)

Henrik Bengtsson

See Also

For C-style formatting of character strings, see sprintf().

Examples

```
cat("Hello world\n")
printf("Hello world\n")

x <- 1.23
cat(sprintf("x=%.2f\n", x))
printf("x=%.2f\n", x)

y <- 4.56
cat(sprintf(c("x=%.2f\n", "y=%.2f\n"), c(x,y)), sep="")
printf(c("x=%.2f\n", "y=%.2f\n"), c(x,y))
```

ProgressBar *Provides text based counting progress bar*

Description

Package: R.utils
Class ProgressBar

Object
~~|
~~+--ProgressBar

Directly known subclasses:
FileProgressBar

public static class **ProgressBar**
extends Object

Usage

```
ProgressBar(max=100, ticks=10, stepLength=1, newlineWhenDone=TRUE)
```

Arguments

max	The maximum number of steps.
ticks	Put visual "ticks" every ticks step.
stepLength	The default length for each increase.
newlineWhenDone	If TRUE, a newline is outputted when bar is updated, when done, otherwise not.

Fields and Methods

Methods:

as.character	Gets a string description of the progress bar.
getBarString	Gets the progress bar string to be displayed.
increase	Increases (steps) progress bar.
isDone	Checks if progress bar is completed.
reset	Reset progress bar.
setMaxValue	Sets maximum value.
setProgress	Sets current progress.
setStepLength	Sets default step length.
setTicks	Sets values for which ticks should be visible.
setValue	Sets current value.

update Updates progress bar.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
# A progress bar with default step length one.
pb <- ProgressBar(max=42)
reset(pb)
while (!isDone(pb)) {
  x <- rnorm(3e4)
  increase(pb)
  # Emulate a slow process
  if (interactive()) Sys.sleep(0.02)
}
cat("\n")

# A "faster" progress bar with default step length 1.4.
pb <- ProgressBar(max=42, stepLength=1.4)
reset(pb)
while (!isDone(pb)) {
  x <- rnorm(3e4)
  increase(pb)
  # Emulate a slow process
  if (interactive()) Sys.sleep(0.02)
}

cat("\n")
```

---

pushBackupFile *Appends a backup suffix to the pathname*

Description

Appends a backup suffix to the pathname and, optionally, renames an existing file accordingly.

In combination with popBackupFile(), this method is useful for creating a backup of a file and restoring it.

Usage

```
## Default S3 method:
pushBackupFile(filename, path=NULL, suffix=".bak", isFile=TRUE,
  onMissing=c("ignore", "error"), copy=FALSE, overwrite=TRUE, ..., verbose=FALSE)
```

Arguments

filename	The filename of the file to backup.
path	The path of the file.
suffix	The suffix to be appended.
isFile	If TRUE, the file must exist and will be renamed on the file system. If FALSE, it is only the pathname string that will be modified. For details, see below.
onMissing	A character string specifying what to do if the file does not exist.
copy	If TRUE, an existing original file remains after creating the backup copy, otherwise it is dropped.
overwrite	If TRUE, any existing backup files are overwritten, otherwise an exception is thrown.
...	Not used.
verbose	A logical or Verbose.

Value

Returns the pathname with the suffix appended.

Author(s)

Henrik Bengtsson

See Also

popBackupFile().

Examples

```
# Create a file
pathname <- "foobar.txt";
cat(file=pathname, "File v1\n");


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# (a) Backup and restore a file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Turn it into a backup file
pathnameB <- pushBackupFile(pathname, verbose=TRUE);
print(pathnameB);

# Restore main file from backup
pathnameR <- popBackupFile(pathnameB, verbose=TRUE);
```

```
print(pathnameR);


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# (b) Backup, create a new file and frop backup file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Turn it into a backup file
pathnameB <- pushBackupFile(pathname, verbose=TRUE);
print(pathnameB);

# Create a new file
cat(file=pathname, "File v2\n");

# Drop backup because a new main file was successfully created
pathnameR <- popBackupFile(pathnameB, verbose=TRUE);
print(pathnameR);
```

`pushTemporaryFile` *Appends a temporary suffix to the pathname*

Description

Appends a temporary suffix to the pathname and, optionally, renames an existing file accordingly.

In combination with `popTemporaryFile()`, this method is useful for creating a file/writing data to file *atomically*, by first writing to a temporary file which is the renamed. If for some reason the generation of the file was interrupted, for instance by a user interrupt or a power failure, then it is only the temporary file that is incomplete.

Usage

```
## Default S3 method:
pushTemporaryFile(filename, path=NULL, suffix=".tmp", isFile=FALSE, ..., verbose=FALSE)
```

Arguments

`filename`	The filename of the file.
`path`	The path of the file.
`suffix`	The suffix to be appended.
`isFile`	If `TRUE`, the file must exist and will be renamed on the file system. If `FALSE`, it is only the pathname string that will be modified. For details, see below.
`...`	Not used.
`verbose`	A `logical` or `Verbose`.

Details

If `isFile` is `FALSE`, the pathname where the suffix of the temporary pathname has been added is returned. If `isFile` is `TRUE`, the file is also renamed. Then, if the file does not exists or it was not successfully renamed, an exception is thrown.

Value

Returns the pathname with the suffix appended.

Author(s)

Henrik Bengtsson

See Also

`popTemporaryFile()`.

Examples

```
createAtomically <- function(pathname, ...) {
  cat("Pathname: ", pathname, "\n", sep="");

  # Generate a file atomically, i.e. the file will either be
  # complete or not created at all.  If interrupted while
  # writing, only a temporary file will exist/remain.
  pathnameT <- pushTemporaryFile(pathname);
  cat("Temporary pathname: ", pathnameT, "\n", sep="");

  cat(file=pathnameT, "This file was created atomically:\n");
  for (kk in 1:10) {
    cat(file=pathnameT, kk, "\n", append=TRUE);
    # Emulate a slow process
    if (interactive()) Sys.sleep(0.1)
  }
  cat(file=pathnameT, "END OF FILE\n", append=TRUE);

  # Rename the temporary file
  pathname <- popTemporaryFile(pathnameT);

  pathname;
} # createAtomically()


pathname <- tempfile();

tryCatch({
  # Try to interrupt the process while writing...
  pathname <- createAtomically(pathname);
}, interrupt=function(intr) {
  str(intr);
})
```

```
# ...and this will throw an exception
bfr <- readLines(pathname);
cat(bfr, sep="\n");
```

---

queryRCmdCheck *Gets the on R CMD check if the current R session was launched by it*

Description

Gets the on R CMD check if the current R session was launched by it.

Usage

```
queryRCmdCheck(...)
```

Arguments

`...`	Not used.

Value

Returns `character` string `"checkingTests"` if 'R CMD check' runs one one of the package tests, and `"checkingExamples"` if it runs one of the package examples. If the current R session was not launched by 'R CMD check', then `"notRunning"` is returned.

Limitations

This function only works if the working directory has not been changed.

Author(s)

Henrik Bengtsson

Examples

```
status <- queryRCmdCheck()
if (status != "notRunning") {
  cat("The current R session was launched by R CMD check. Status: ", status, "\n")
} else {
  cat("The current R session was not launched by R CMD check.\n")
}

# Display how R was launched
print(base::commandArgs())

# Display loaded packages etc.
print(search())
```

```
# Display current working directory
print(getwd())
```

---

readBinFragments *Reads binary data from disjoint sections of a connection or a file*

Description

Reads binary data from disjoint sections of a connection or a file.

Usage

```
## Default S3 method:
readBinFragments(con, what, idxs=1, origin=c("current", "start"), size=NA, ...,
  verbose=FALSE)
```

Arguments

con	A connection or the pathname of an existing file.
what	A character string or an object specifying the the data type (mode()) to be read.
idxs	A vector of (non-duplicated) indices or a Nx2 matrix of N from-to index intervals specifying the elements to be read. Positions are either relative to the start or the current location of the file/connection as given by argument origin.
origin	A character string specify whether the indices in argument idxs are relative to the "start" or the "current" position of the file/connection.
size	The size of the data type to be read. If NA, the natural size of the data type is used.
...	Additional arguments passed to readBin().
verbose	A logical or a Verbose object.

Value

Returns a vector of the requested mode().

Author(s)

Henrik Bengtsson

See Also

writeBinFragments().

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Create a data file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
data <- 1:255
size <- 2
pathname <- tempfile("exampleReadBinFragments")
writeBin(con=pathname, data, size=size)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Read and write using index vectors
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cat("Read file...\n")
# Read every 16:th byte in the file
idxs <- seq(from=1, to=255, by=16)
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs)
stopifnot(identical(x, data[idxs]))
print(x)
# Read every 16:th byte in a connection starting with the 6th.
idxs <- idxs + 5L;
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs)
stopifnot(identical(x, data[idxs]))
print(x)
cat("Read file...done\n")

cat("Write file...\n")
# Update every 16:th byte in the file
idxs <- seq(from=1, to=255, by=16)
x0 <- data[idxs]
writeBinFragments(pathname, idxs=idxs, rev(x0), size=size)
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs)
print(x)
stopifnot(identical(rev(x0), x))

# Update every 16:th byte in the file
idxs <- seq(from=1, to=255, by=16)
writeBinFragments(pathname, idxs=idxs, rev(x), size=size)
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs)
print(x)
stopifnot(identical(x0, x))

# Assert everything is as expected
# Read the complete file
x <- readBin(pathname, what="integer", size=size, signed=FALSE, n=length(data))
stopifnot(identical(x, data))
cat("Write file...done\n")


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Ditto but via a connection
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
cat("Read connection...\n")
# Read every 16:th byte in a connection
con <- file(pathname, open="rb")
idxs <- seq(from=1, to=255, by=16)
x <- readBinFragments(con, what="integer", size=size, signed=FALSE, idxs=idxs)
stopifnot(identical(x, data[idxs]))
print(x)

# Read every 16:th byte in a connection starting with the 6th.
idxs <- idxs + 5L;
x <- readBinFragments(con, what="integer", size=size, signed=FALSE, idxs=idxs, origin="start")
stopifnot(identical(x, data[idxs]))
print(x)
close(con)
cat("Read connection...done\n")


# Update every 16:th byte in a connection
cat("Write connection...\n")
con <- file(pathname, open="r+b")
idxs <- seq(from=1, to=255, by=16)
x0 <- data[idxs]
writeBinFragments(pathname, idxs=idxs, rev(x0), size=size)
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs)
print(x)
stopifnot(identical(rev(x0), x))

# Update every 16:th byte in the file
idxs <- seq(from=1, to=255, by=16)
writeBinFragments(pathname, idxs=idxs, rev(x), size=size)
x <- readBinFragments(pathname, what="integer", size=size, signed=FALSE, idxs=idxs, origin="start")
print(x)
stopifnot(identical(x0, x))

close(con)

# Assert everything is as expected
# Read the complete file
x <- readBin(pathname, what="integer", size=size, signed=FALSE, n=length(data))
stopifnot(identical(x, data))
cat("Write connection...done\n")


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Clean up
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
file.remove(pathname)
```

---

readRdHelp *Reads one or more Rd help files in a certain format*

Description

Reads one or more Rd help files in a certain format.

Usage

```
## Default S3 method:
readRdHelp(..., format=c("text", "html", "latex", "rd"), drop=TRUE)
```

Arguments

...	Arguments passed to help.
format	A character string specifying the return type.
drop	If FALSE or more than one help entry is found, the result is returned as a list.

Value

Returns a list of character strings or a single character string.

Author(s)

Henrik Bengtsson

---

readTable *Reads a file in table format*

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

WARNING: This method is very much in an alpha stage. Expect it to change.

This method is an extension to the default read.table function in R. It is possible to specify a column name to column class map such that the column classes are automatically assigned from the column header in the file.

In addition, it is possible to read any subset of rows. The method is optimized such that only columns and rows that are of interest are parsed and read into R's memory. This minimizes memory usage at the same time as it speeds up the reading.

Usage

```
## Default S3 method:
readTable(file, colClasses=NULL, isPatterns=FALSE, defColClass=NA, header=FALSE, skip=0,
  nrows=-1, rows=NULL, col.names=NULL, check.names=FALSE, path=NULL, ...,
  stripQuotes=TRUE, method=c("readLines", "intervals"), verbose=FALSE)
```

Arguments

file	A connection or a filename. If a filename, the path specified by path is added to the front of the filename. Unopened files are opened and closed at the end.
colClasses	Either a named or an unnamed character vector. If unnamed, it specified the column classes just as used by read.table. If it is a named vector, names(colClasses) are used to match the column names read (this requires that header=TRUE) and the column classes are set to the corresponding values.
isPatterns	If TRUE, the matching of names(colClasses) to the read column names is done by regular expressions matching.
defColClass	If the column class map specified by a named colClasses argument does not match some of the read column names, the column class is by default set to this class. The default is to read the columns in an "as is" way.
header	If TRUE, column names are read from the file.
skip	The number of lines (commented or non-commented) to skip before trying to read the header or alternatively the data table.
nrows	The number of rows to read of the data table. Ignored if rows is specified.
rows	An row index vector specifying which rows of the table to read, e.g. row one is the row following the header. Non-existing rows are ignored. Note that rows are returned in the same order they are requested and duplicated rows are also returned.
col.names	Same as in read.table().
check.names	Same as in read.table(), but default value is FALSE here.
path	If file is a filename, this path is added to it, otherwise ignored.
...	Arguments passed to read.table used internally.
stripQuotes	If TRUE, quotes are stripped from values before being parse. This argument is only effective when method=="readLines".
method	If "readLines", (readLines()) is used internally to first only read rows of interest, which is then passed to read.table(). If "intervals", contiguous intervals are first identified in the rows of interest. These intervals are the read one by one using read.table(). The latter methods is faster and especially more memory efficient if the intervals are not too many, where as the former is preferred if many "scattered" rows are to be read.
verbose	A logical or a Verbose object.

Value

Returns a data.frame.

Author(s)

Henrik Bengtsson

See Also

readTableIndex(). read.table. colClasses().

readTableIndex *Reads a single column from file in table format*

Description

Reads a single column from file in table format, which can then be used as a index-to-row (look-up) map for fast access to a subset of rows using readTable().

Usage

```
## Default S3 method:
readTableIndex(..., indexColumn=1, colClass="character", verbose=FALSE)
```

Arguments

indexColumn	An single integer of the index column.
colClass	A single character specifying the class of the index column.
...	Arguments passed to readTable() used internally.
verbose	A logical or a Verbose object.

Value

Returns a vector.

Author(s)

Henrik Bengtsson

See Also

readTable().

Examples

```
## Not run:
   # File containing data table to be access many times
   filename <- "somefile.txt"

   # Create a look-up index
   index <- readTableIndex(filename)

   # Keys of interest
   keys <- c("foo", "bar", "wah")

   # Read only those keys and do it fast
   df <- readTable(filename, rows=match(keys, index))

## End(Not run)
```

readWindowsShortcut *Reads a Microsoft Windows Shortcut (.lnk file)*

Description

Reads a Microsoft Windows Shortcut (.lnk file).

Usage

```
## Default S3 method:
readWindowsShortcut(con, verbose=FALSE, ...)
```

Arguments

con	A connection or a character string (filename).
verbose	If TRUE, extra information is written while reading.
...	Not used.

Details

The MIME type for a Windows Shortcut file is application/x-ms-shortcut.

Value

Returns a list structure.

Author(s)

Henrik Bengtsson

References

[1] Wotsit's Format, http://www.wotsit.org/, 2005.
[2] Hager J, *The Windows Shortcut File Format* (as reverse-engineered by), version 1.0.
[3] Microsoft Developer Network, *IShellLink Interface*, 2008. http://msdn2.microsoft.com/en-us/library/bb774950.aspx
[4] Andrews D, *Parsing Windows Shortcuts (lnk) files in java*, comp.lang.java.help, Aug 1999. http://groups.google.com/group/comp.lang.java.help/browse_thread/thread/a2e147b07d5480a2/
[5] Multiple authors, *Windows shell links* (in Tcl), Tcler's Wiki, April 2008. http://wiki.tcl.tk/1844
[6] Daniel S. Bensen, *Shortcut File Format (.lnk)*, Stdlib.com, April 24, 2009. https://web.archive.org/web/20110817051855/http://www.stdlib.com/art6-Shortcut-File-Format-lnk.html (was http://www.stdlib.com/art6-Shortcut-File-Format-lnk.html)
[7] [MS-SHLLINK]: Shell Link (.LNK) Binary File Format, Microsoft Inc., September 25, 2009.

See Also

createWindowsShortcut() and filePath()

Examples

```
pathname <- system.file("data-ex/HISTORY.LNK", package="R.utils")
lnk <- readWindowsShortcut(pathname)

# Print all information
print(lnk)

# Get the relative path to the target file
history <- file.path(dirname(pathname), lnk$relativePath)

# Alternatively, everything in one call
history <- filePath(pathname, expandLinks="relative")
```

---

removeDirectory *Removes a directory*

Description

Removes a directory, and if requested, also its contents.

Usage

```
## Default S3 method:
removeDirectory(path, recursive=FALSE, mustExist=TRUE, ...)
```

Arguments

path	A character string specifying the directory to be removed.
recursive	If TRUE, subdirectories and files are also removed. If FALSE, and directory is non-empty, an exception is thrown.
mustExist	If TRUE, and the directory does not exist, an exception is thrown.
...	Not used.

Value

Returns (invisibly) TRUE, the directory was successfully removed, otherwise FALSE, unless an exception is thrown.

Symbolic links

This function can also be used to remove symbolic links to directories without removing the target. Note that neither file.remove() nor unlink() is capable of remove symbolic *directory* links on Windows.

Author(s)

Henrik Bengtsson

See Also

Internally unlink() is used.

---

resample *Sample values from a set of elements*

Description

Sample values from a set of elements. Contrary to sample(), this function also works as expected when there is only one element in the set to be sampled, cf. [1]. This function originates from the example code of sample() as of R v2.12.0.

Usage

```
## Default S3 method:
resample(x, ...)
```

Arguments

x	A vector of any length and data type.
...	Additional arguments passed to sample.int().

Value

Returns a sampled vector of the same data types as argument x.

Author(s)

Henrik Bengtsson

References

[1] Henrik Bengtsson, *Using sample() to sample one value from a single value?*, R-devel mailing list, 2010-11-03.

See Also

Internally sample.int() is used.

saveObject *Saves an object to a file or a connection*

Description

Saves an object to a file or a connection.

Usage

```
## Default S3 method:
saveObject(object, file=NULL, path=NULL, format=c("auto", "xdr", "rds"), compress=TRUE,
  ..., safe=TRUE)
```

Arguments

object	The object to be saved.
file	A filename or connection where the object should be saved. If NULL, the filename will be the hash code of the object plus ".xdr".
path	Optional path, if file is a filename.
format	File format.
compress	If TRUE, the file is compressed to, otherwise not.
...	Other arguments accepted by save() in the base package.
safe	If TRUE and file is a file, then, in order to lower the risk for incomplete files, the object is first written to a temporary file, which is then renamed to the final name.

Value

Returns (invisibly) the pathname or the connection.

Author(s)

Henrik Bengtsson

See Also

loadObject() to load an object from file. digest for how hash codes are calculated from an object. See also saveRDS().

seqToHumanReadable *Gets a short human readable string representation of an vector of indices*

Description

Gets a short human readable string representation of an vector of indices.

Usage

```
## Default S3 method:
seqToHumanReadable(idx, tau=2L, delimiter="-", collapse=", ", ...)
```

Arguments

idx	A vector of integer indices.
tau	A non-negative integer specifying the minimum span of of a contiguous sequences for it to be collapsed to <from>-<to>.
delimiter	A character string delimiter.
collapse	A character string used to collapse subsequences.
...	Not used.

Author(s)

Henrik Bengtsson

See Also

Internally, seqToIntervals() is used.

Examples

```
print(seqToHumanReadable(1:2))                 # "1, 2"
print(seqToHumanReadable(1:2, tau=1))          # "1-2"
print(seqToHumanReadable(1:10))                # "1-10"
print(seqToHumanReadable(c(1:10, 15:18, 20)))  # "1-10, 15-18, 20"
```

seqToIntervals *Gets all contiguous intervals of a vector of indices*

Description

Gets all contiguous intervals of a vector of indices.

Usage

```
## Default S3 method:
seqToIntervals(idx, ...)
```

Arguments

idx	A vector of N integer indices.
...	Not used.

Value

An Nx2 integer matrix.

Author(s)

Henrik Bengtsson

See Also

*intervalsToSeq(). To identify sequences of *equal* values, see rle().

Examples

```
x <- 1:10
y <- seqToIntervals(x)
print(y)  # [1 10]

x <- c(1:10, 15:18, 20)
y <- seqToIntervals(x)
print(y)  # [1 10; 15 18; 20 20]

z <- intervalsToSeq(y)
print(z)
stopifnot(all.equal(x,z))
```

setOption *Sets a option in R*

Description

Sets a option in R by specifying its name as a character string.

Usage

```
## Default S3 method:
setOption(x, value, ...)
```

Arguments

x	The name of the option to be set.
value	The new value of the option.
...	Not used.

Value

Returns (invisibly) the previous value of the option.

Author(s)

Henrik Bengtsson

See Also

See getOption() and "base::options".

Settings *Class for applicational settings*

Description

Package: R.utils
Class Settings

```
Object
~~|
~~+--Options
~~~~~~~|
~~~~~~~+--Settings
```

Directly known subclasses:

public static class **Settings**
extends Options

Class for applicational settings.

Usage

```
Settings(basename=NULL, ...)
```

Arguments

basename	A character string of the basename of the settings file.
...	Arguments passed to constructor of superclass Options.

Fields and Methods

Methods:

findSettings	Searches for the settings file in one or several directories.
getLoadedPathname	Gets the pathname of the settings file loaded.
isModified	Checks if settings has been modified compared to whats on file.
loadAnywhere	Loads settings from file.
promptAndSave	Prompt user to save modified settings.
saveAnywhere	Saves settings to file.

Methods inherited from Options:
as.character, as.list, equals, getLeaves, getOption, hasOption, names, nbrOfOptions, setOption, str

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Load settings with package and save on exit

Here is a generic .First.lib() function for loading settings with package. It also (almost) assures that the package is detached when R finishes. See onSessionExit() why it is not guaranteed!

The almost generic .Last.lib() function, which will prompt user to save settings, is called when a package is detached.

It is custom to put these functions in a file named zzz.R.

.First.lib():

```
.First.lib <- function(libname, pkgname) {
```

```
  # Write a welcome message when package is loaded
  pkg <- Package(pkgname)
  assign(pkgname, pkg, pos=getPosition(pkg))

  # Read settings file ".<pkgname>Settings" and store it in package
  # variable '<pkgname>Settings'.
  varname <- paste(pkgname, "Settings")
  basename <- paste(".", varname, sep="")
  settings <- Settings$loadAnywhere(basename, verbose=TRUE)
  if (is.null(settings))
    settings <- Settings(basename)
  assign(varname, settings, pos=getPosition(pkg))

  # Detach package when R finishes, which will save package settings too.
  onSessionExit(function(...) detachPackage(pkgname))

  packageStartupMessage(getName(pkg), " v", getVersion(pkg),
      " (", getDate(pkg), ") successfully loaded. See ?", pkgname,
      " for help.\n", sep="")
} # .First.lib()
```

.Last.lib():

```
.Last.lib <- function(libpath) {
  pkgname <- "<package name>"

  # Prompt and save package settings when package is detached.
  varname <- paste(pkgname, "Settings", sep="")
  if (exists(varname)) {
    settings <- get(varname)
    if (inherits(settings, "Settings"))
      promptAndSave(settings)
  }
} # .Last.lib()
```

Author(s)

Henrik Bengtsson

Examples

```
# Load settings from file, or create default settings
basename <- "some.settings"
settings <- Settings$loadAnywhere(basename)
if (is.null(settings))
  settings <- Settings(basename)

# Set default options, if missing.
```

```
setOption(settings, "graphics/verbose", TRUE, overwrite=FALSE)
setOption(settings, "io/verbose", Verbose(threshold=-1), overwrite=FALSE)

# Save and reload settings
path <- tempdir()
saveAnywhere(settings, path=path)
settings2 <- Settings$loadAnywhere(basename, paths=path)

# Clean up
file.remove(getLoadedPathname(settings2))

# Assert correctness
stopifnot(equals(settings, settings2))
```

`shell.exec2` *Open a file or URL using Windows File Associations*

Description

Open a file or URL using Windows File Associations using `shell.exec()` but makes some tweaks to filenames to make them more likely to be opened properly.

This function is only applicable on Windows systems.

Usage

```
shell.exec2(file)
```

Arguments

`file`	A `character` string specifying a file or an URL.

Details

Before passing a *file* on the file system to `shell.exec()`, this function: (i) unmaps any mapped drive letters used in the pathname (e.g. 'X:/foo.bar.html' to 'C:/Users/Joe/bar.html'), (ii) and replaces any forward slashed with backward ones (e.g. 'C:/Users/Joe/bar.html' to 'C:\Users\Joe\bar.html'). URLs are passed as is to `shell.exec()`.

The reason for (i) is that some web browsers (e.g. Google Chrome) will not open files on mapped drives. The reason for (ii) is that if forward slashes are used, then `shell.exec()` will give an error that the file was not found (at least with the default Windows shell).

Value

Returns nothing.

Setting on startup

The intended usage of this function is to set it as the default browser for `browseURL`. Just add the following to your `.Rprofile` file:

```
if (.Platform$OS.type == "windows")
  options(browser=function(...) R.utils::shell.exec2(...))
```

This will only load (not attach) the **R.utils** package when the browser function is actual used.

Author(s)

Henrik Bengtsson

SmartComments *Abstract class SmartComments*

Description

Package: R.utils
Class SmartComments

```
Object
~~|
~~+--SmartComments
```

Directly known subclasses:
LComments, VComments

public abstract static class **SmartComments**
extends Object

Abstract class SmartComments.

Usage

```
SmartComments(letter=NA, ...)
```

Arguments

`letter`	A single `character`.
`...`	Not used.

Details

A "smart" source-code comment is an R comment, which start with a '\#', but is followed by a single letter, then a single symbol and a second '\#' and then an option character string, and there must not be any code before the comment on the same line. In summary, a smart comment line has format: `<white spaces>#<letter><symbol># <some text>`.

Example code with two smart comments (VComments):

```
x <- 2
#V1# threshold=-1
#Vc# A v-comment log message
cat("Hello world")
```

which after compilation becomes

```
x <- 2
verbose <- Verbose(threshold=-1)
if (verbose) { cat(verbose, "A v-comment log message"); }
cat("Hello world")
```

Fields and Methods

Methods:

compile	Preprocess a vector of code lines.
convertComment	Converts a single smart comment to R code.
parse	Parses one single smart comment.
reset	Resets a SmartComments compiler.
validate	Validates the compiled lines.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

See Also

VComments.

`sourceDirectory` *Sources files recursively to either local or global environment*

Description

Sources files recursively to either local or global environment.

Usage

```
## Default S3 method:
sourceDirectory(path, pattern=".*[.](r|R|s|S|q)([.](lnk|LNK))*$", recursive=TRUE,
  envir=parent.frame(), onError=c("error", "warning", "skip"), modifiedOnly=TRUE, ...,
  verbose=FALSE)
```

Arguments

`path`	A path to a directory to be sourced.
`pattern`	A regular expression file name pattern to identify source code files.
`recursive`	If `TRUE`, subdirectories are recursively sourced first, otherwise not.
`envir`	An `environment` in which the code should be evaluated.
`onError`	If an error occurs, the error may stop the job, give a warning, or silently be skipped.
`modifiedOnly`	If `TRUE`, only files that are modified since the last time they were sourced are sourced, otherwise regardless.
`...`	Additional arguments passed to `sourceTo`().
`verbose`	A `logical` or a `Verbose` object.

Value

Returns a `vector` of the full pathnames of the files sourced.

Details

Subdirectories and files in each (sub-)directory are sourced in lexicographic order.

Hooks

This method does not provide hooks, but the internally used `sourceTo`() does.

Author(s)

Henrik Bengtsson

See Also

`sourceTo`() and compare to `source`().

sourceTo *Parses and evaluates code from a file or a connection*

Description

Parses and evaluates code from a file or a connection. This has the same effect as if `source(..., local=TRUE)` would have been called from within the given environment. This is useful when setting up a new local working environment.

Usage

```
## Default S3 method:
sourceTo(file, path=NULL, chdir=FALSE, ..., local=TRUE, envir=parent.frame(),
  modifiedOnly=FALSE)
```

Arguments

`file`	A `connection` or a `character` string giving the pathname of the file or URL to read from.
`path`	An optional `character` string giving the path to the file. Ignored if `file` is a connection.
`chdir`	If `TRUE` and `file` is a pathname, the R working directory is temporarily changed to the directory containing `file` for evaluating.
`...`	Arguments to `source`(). If argument `file` is not explicitly given, the first argument is assumed to be the `file` argument. This argument is converted into a string by `as.character()`.
`local`	If `FALSE`, evaluation is done in the global environment, otherwise in the calling environment.
`envir`	An `environment` in which `source`() should be called. If `NULL`, the global environment is used.
`modifiedOnly`	If `TRUE`, the file is sourced only if modified since the last time it was sourced, otherwise regardless.

Value

Return the result of `source`().

Hooks

This methods recognizes the hook `sourceTo/onPreprocess`, which is called after the lines in file has been read, but before they have been parsed by the R parser, cf. `parse`(). An `onPreprocess` hook function should take a `character vector` of code lines and return a `character vector` of code lines. This can for instance be used to pre-process R source code with special directives such as `VComments`.

Note that only one hook function can be used for this function, otherwise an error is generated.

Author(s)

Henrik Bengtsson

See Also

sourceDirectory(). sys.source() and source().

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example 1
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cat("=== Example 1 ================================================\n")

foo <- function(file, ...) {
  cat("Local objects before calling sourceTo():\n")
  print(ls())

  res <- sourceTo(file, ...)

  cat("Local objects after calling sourceTo():\n")
  print(ls())
}

cat("Global objects before calling foo():\n")
lsBefore <- NA
lsBefore <- ls()
foo(file=textConnection(c('a <- 1', 'b <- 2')))

cat("Global objects after calling foo():\n")
stopifnot(length(setdiff(ls(), lsBefore)) == 0)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example 2 - with VComments preprocessor
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cat("=== Example 2 ================================================\n")

preprocessor <- function(lines, ...) {
  cat("----------------------------------------\n")
  cat("Source code before preprocessing:\n")
  displayCode(code=lines, pager="console")
  cat("----------------------------------------\n")
  cat("Source code after preprocessing:\n")
  lines <- VComments$compile(lines)
  displayCode(code=lines, pager="console")
  cat("----------------------------------------\n")
  lines
}

oldHooks <- getHook("sourceTo/onPreprocess")
setHook("sourceTo/onPreprocess", preprocessor, action="replace")
```

```
code <- c(
 'x <- 2',
 '#V1# threshold=-1',
 '#Vc# A v-comment log message',
 'print("Hello world")'
)
fh <- textConnection(code)
sourceTo(fh)
setHook("sourceTo/onPreprocess", oldHooks, action="replace")
```

---

splitByPattern *Splits a single character string by pattern*

Description

Splits a single character string by pattern. The main difference compared to strsplit() is that this method also returns the part of the string that matched the pattern. Also, it only takes a single character string.

Usage

```
## Default S3 method:
splitByPattern(str, pattern, ...)
```

Arguments

str	A single character string to be split.
pattern	A regular expression character string.
...	Not used.

Value

Returns a named character vector with names equal to "TRUE" if element is a pattern part and "FALSE" otherwise.

Author(s)

Henrik Bengtsson

See Also

Compare to strsplit().

Examples

```
rspCode <- "<body>Hello <%=\"world\"%></body>"
rspParts <- splitByPattern(rspCode, pattern="<%.*%>")
cat(rspCode, "\n")
print(rspParts)
```

stext *Writes text in the margin along the sides of a plot*

Description

Writes text in the margin along the sides of a plot.

Usage

```
## Default S3 method:
stext(text, side=1, line=0, pos=0.5, margin=c(0.2, 0.2),
  charDim=c(strwidth("M", cex = cex), strheight("M", cex = cex)), cex=par("cex"), ...)
```

Arguments

text	The text to be written. See mtext for details.
side	An integer specifying which side to write the text on. See mtext for details.
line	A numeric specifying on which line to write on.
pos	A numeric, often in [0,1], specifying the position of the text relative to the left and right edges.
margin	A numeric vector length two specifying the text margin.
charDim	A numeric vector length two specifying the size of a typical symbol.
cex	A numeric specifying the character expansion factor.
...	Additional arguments passed to mtext.

Value

Returns what mtext returns.

Author(s)

Henrik Bengtsson

See Also

Internally mtext is used.

`subplots` *Creates a grid of subplots*

Description

Creates a grid of subplots in the current figure. If arguments `nrow` and `ncol` are given a `nrow`-by-`ncol` grid of subplots are created. If only argument `n` is given then a r-by-s grid is created where |r-s| <= 1, i.e. a square or almost a square of subplots is created. If `n` and `nrow` is given then a grid with `nrow` rows and at least `n` subplots are created. Similar if `n` and `ncol` is given. The argument `byrow` specifies if the order of the subplots should be rowwise (`byrow=TRUE`) or columnwise.

Usage

```
## Default S3 method:
subplots(n=1, nrow=NULL, ncol=NULL, byrow=TRUE, ...)
```

Arguments

`n`	If given, the minimum number of subplots.
`nrow`	If given, the number of rows the grid of subplots should contain.
`ncol`	If given, the number of columns the grid of subplots should contain.
`byrow`	If `TRUE`, the panels are ordered row by row in the grid, otherwise column by column.
`...`	Not used.

Value

Returns the `matrix` containing the order of plots.

Author(s)

Henrik Bengtsson

See Also

`layout` and `layout.show()`.

Examples

```
subplots(nrow=2, ncol=3)  # 2-by-3 grid of subplots
subplots(n=6, nrow=2)     # 2-by-3 grid of subplots
subplots(n=5, ncol=2)     # 3-by-2 grid of subplots
subplots(1)               # (Reset) to a 1-by-1 grid of subplots
subplots(2)               # 1-by-2 grid of subplots
subplots(3)               # 2-by-2 grid of subplots
l <- subplots(8)          # 3-by-3 grid of subplots
layout.show(length(l))
```

System *Static class to query information about the system*

Description

Package: R.utils
Class System

```
Object
~~|
~~+--System
```

Directly known subclasses:

public static class **System**
extends Object

The System class contains several useful class fields and methods. It cannot be instantiated.

Fields and Methods

Methods:

currentTimeMillis	Get the current time in milliseconds.
findGhostscript	Searches for a Ghostview executable on the current system.
findGraphicsDevice	Searches for a working PNG device.
getHostname	Retrieves the computer name of the current host.
getMappedDrivesOnWindows	-
getUsername	Retrieves the name of the user running R.
mapDriveOnWindows	-
openBrowser	Opens an HTML document using the OS default HTML browser.
parseDebian	Parses a string, file or connection for Debian formatted parameters.
unmapDriveOnWindows	-

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

systemR *Launches another R process from within R*

Description

Launches another R process from within R via system() by automatically locating the R executable, cf [1].

Usage

```
## Default S3 method:
systemR(command="", ..., Rcommand="R", verbose=FALSE)
```

Arguments

command	A character string be appended to the system() call. If a vector, then the strings are concatenated separated with a space.
...	Additional arguments passed to system().
Rcommand	A character string specifying the basename of the R executable.
verbose	A logical or a Verbose object.

Value

Returns what system() returns.

Author(s)

Henrik Bengtsson

References

[1] R-devel thread 'Best way to locate R executable from within R?', May 22, 2012.

See Also

The R executable is located using R.home(), which is then launched using system().

Examples

```
res <- systemR("--slave -e cat(runif(1))", intern=TRUE)
cat("A random number: ", res, "\n", sep="")
```

TextStatusBar *A status bar at the R prompt that can be updated*

Description

Package: R.utils
Class TextStatusBar

```
Object
~~|
~~+--TextStatusBar
```

Directly known subclasses:

public static class **TextStatusBar**
extends Object

A status bar at the R prompt that can be updated.

Usage

```
TextStatusBar(fmt=paste("%-", getOption("width") - 1, "s", sep = ""), ...)
```

Arguments

fmt	A character format string to be used by sprintf(). Default is a left-aligned string of full width.
...	Named arguments to be passed to sprintf() together with the format string.

Details

A label with name hfill can be used for automatic horizontal filling. It must be numeric and be immediate before a string label such that a hfill label and the following string label together specifies an sprintf format such as "%*-s". The value of hfill will be set such that the resulting status bar has width equal to getOption("width")-1 (the reason for the -1 is to prevent the text status bar from writing into the next line). If more than one hfill label is used their widths will be uniformly distributed. Left over spaces will be distributed between hfill labels with initial values of one.

Fields and Methods

Methods:

flush	Flushes the output.

getLabel	Gets the current value of a label.
newline	Writes a newline.
popMessage	Adds a message above the status bar.
setLabel	Sets the value of a label.
setLabels	Sets new values of given labels.
update	Updates the status bar (visually).
updateLabels	Sets the new values of given labels and updates the status bar.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Read all HTML files in the base package
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
path <- system.file(package="base")
files <- list.files(path, recursive=TRUE, full.names=TRUE)
files <- files[sapply(files, FUN=isFile)]
nfiles <- length(files)

cat(sprintf("Reading %d files in %s:\n", nfiles, path))

# Create a status bar with four labels
sb <- TextStatusBar("File: %-*s [%3.0f%% %7.0f bytes %-8s]",
                hfill=1, file="", progress=0, nbytes=0L, time="")

nbytes <- 0L
for (kk in seq_len(nfiles)) {
  file <- files[kk]

  # Update the status bar
  if (sb) {
    setLabel(sb, "progress", 100*kk/nfiles)
    if (kk %% 10 == 1 || kk == nfiles)
      setLabel(sb, "file", substr(basename(file), 1, 44))

    size <- file.info(file)$size
    # popMessage() calls update() too
    popMessage(sb, sprintf("Processing %s (%.2fkB)",
                                       basename(file), size/1024))
    flush(sb)
  }
```

```
  # Read the file
  bfr <- readBin(file, what="raw", n=size)
  nbytes <- nbytes + size

  # Emulate a slow process
  if (interactive()) Sys.sleep(rexp(1, rate=60))

  # Update the status bar
  if (sb) {
    setLabel(sb, "nbytes", nbytes)
    setLabel(sb, "time", format(Sys.time(), "%H:%M:%S"))
    update(sb)
  }
}
setLabel(sb, "file", "<done>")
update(sb)
cat("\n")
```

`TimeoutException` *TimeoutException represents timeout errors*

Description

Package: R.utils
Class TimeoutException

```
Object
~~|
~~+--try-error
~~~~~~~|
~~~~~~~+--condition
~~~~~~~~~~~~|
~~~~~~~~~~~~+--error
~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~+--simpleError
~~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~~+--Exception
~~~~~~~~~~~~~~~~~~~~~~~~~~~|
~~~~~~~~~~~~~~~~~~~~~~~~~~~+--TimeoutException
```

Directly known subclasses:

public static class **TimeoutException**
extends Exception

TimeoutException represents timeout errors occurring when a set of R expressions executed did not finish in time.

Usage

```
TimeoutException(..., cpu=NA, elapsed=NA)
```

Arguments

`...`	Any arguments accepted by `Exception`.
`cpu, elapsed`	The maximum time the R expressions were allowed to be running before the timeout occurred as measured in CPU time and (physically) elapsed time.

Fields and Methods

Methods:

`getMessage`	Gets the message of the exception.

Methods inherited from Exception:
as.character, getCall, getCalls, getLastException, getMessage, getStackTrace, getWhen, print, printStackTrace, throw

Methods inherited from error:
as.character, throw

Methods inherited from condition:
abort, as.character, conditionCall, conditionMessage, print

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

See Also

For detailed information about exceptions see `Exception`.

`touchFile` *Updates the timestamp of a file*

Description

Updates the timestamp of a file. Currently, it is only possible to change the timestamp specifying when the file was last modified, and time can only be set to the current time.

Usage

```
## Default S3 method:
touchFile(pathname, ...)
```

Arguments

pathname	A character vector specifying files to be updated.
...	Not used.

Value

Returns (invisibly) a vector of the old timestamps.

Author(s)

Henrik Bengtsson

References

[1] R-devel mailing list thread *Unix-like touch to update modification timestamp of file?*, started on 2008-02-26. http://stat.ethz.ch/pipermail/r-devel/2008-February/048542.html

See Also

Internally, Sys.setFileTime() (iff available) and file.info() are utilized.

Examples

```
# 1. Create a file
pathname <- tempfile()
cat(file=pathname, "Hello world!")
md5a <- digest::digest(pathname, file=TRUE)

# 2. Current time stamp
ta <- file.info(pathname)$mtime
print(ta)

# 3. Update time stamp
Sys.sleep(1.2)
touchFile(pathname)
tb <- file.info(pathname)$mtime
print(tb)

# 4. Verify that the timestamp got updated
stopifnot(tb > ta)

# 5. Verify that the contents did not change
md5b <- digest::digest(pathname, file=TRUE)
stopifnot(identical(md5a, md5b))
```

toUrl *Converts a pathname into a URL*

Description

Converts a pathname into a URL starting with `file://`.

Usage

```
## Default S3 method:
toUrl(pathname, safe=TRUE, ...)
```

Arguments

`pathname`	A `character vector` of pathnames to be made into URLs.
`safe`	If `TRUE`, certain "unsafe" characters are escaped.
`...`	Not used.

Value

Returns a `character vector`.

Author(s)

Henrik Bengtsson

See Also

`URLencode`.

unwrap.array *Unwrap an array, matrix or a vector to an array of more dimensions*

Description

Unwrap an array, matrix or a vector to an array of more dimensions. This is done by splitting up each dimension into several dimension based on the names of that dimension.

Usage

```
## S3 method for class 'array'
unwrap(x, split=rep("[.]", length(dim(x))), drop=FALSE, ...)
```

Arguments

x	An array or a matrix.
split	A list or a character vector. If a list, it should contain functions that takes a character vector as the first argument and optional ... arguments. Each function should split the vector into a list of same length and where all elements contains the same number of parts. If a character vector, each element split[i] is replaced by a function call function(names, ...) strsplit(names, split=spli
drop	If TRUE, dimensions of of length one are dropped, otherwise not.
...	Arguments passed to the split functions.

Details

Although not tested thoroughly, unwrap() should be the inverse of wrap() such that identical(unwrap(wrap(x)), x) holds.

Value

Returns an array.

Author(s)

Henrik Bengtsson

See Also

*wrap().

Examples

```
## Not run: See ?wrap.array for an example
```

---

useRepos *Sets package repositories*

Description

Sets package repositories.

Usage

```
useRepos(repos=NULL, where=c("before", "after", "replace"), unique=TRUE, fallback=TRUE,
  ...)
```

Arguments

repos	A character vector of repositories to use. If NULL, nothing is replaced.
where	A character string specifying how to add them to the current set of repositories.
unique	If TRUE, only unique repositories are set.
fallback	If TRUE, any remaining non-specified repository value of format '...@' (e.g. '@CRAN@') than could not be recovered by other means, will be assigned to a pre-defined known value, if possible. If so, then an informative warning is given.
...	Not used.

Value

Returns a list with element 'repos' reflecting `options("repos")` as the options where prior to calling this function.

Author(s)

Henrik Bengtsson

See Also

withRepos().

VComments *The VComments class*

Description

Package: R.utils
Class VComments

```
Object
~~|
~~+--SmartComments
~~~~~~~|
~~~~~~~+--VComments
```

Directly known subclasses:
LComments

public static class **VComments**
extends *SmartComments*

The VComments class.

Usage

```
VComments(letter="V", verboseName="verbose", ...)
```

Arguments

letter	The smart letter.
verboseName	The name of the verbose object.
...	Not used.

Details

The 'v' in VComments stands for 'verbose', because of its relationship to the Verbose class.

Here is a list of VComments and the R code that replaces each of them by the compiler:

Constructors

- \#V0\#[<args>] - NullVerbose(<args>)
- \#V1\#[<args>] - Verbose(<args>)

Controls

- \#V=\#[<variable>] - Sets the name of the <verbose> object. Default is 'verbose'.
- \#V\^\#<threshold> - setThreshold(<verbose>, <threshold>)
- \#V?\#<expression> - if (isVisible(<verbose>)) { <expression> }
- \#V@\#<level> - setDefaultLevel(<verbose>, <level>)
- \#Vm\#<method> <args> - <method>(<verbose>, <args>)

Enters and exits

- \#V+\#[<message>] - enter(<verbose>, <message>)
- \#V-\#[<message>] - exit(<verbose>, <message>)
- \#V!\#[<message>] - pushState(<verbose>)
  on.exit(popState(<verbose>))
  If <message>, enter(<verbose>, <message>)

Simple output

- \#Vn\#<ignored> - newline(<verbose>)
- \#Vr\#<ignored> - ruler(<verbose>)
- \#Vt\#<ignored> - timestamp(<verbose>)
- \#Vw\#[<title>] - warnings(<verbose>, <title>)

Output messages

- \#Vc\#[<message>] - cat(<verbose>, <message>)
- \#Ve\#<expression> - eval(<verbose>, <expression>)
- \#Vh\#<message> - header(<verbose>, <message>)
- \#Vp\#<object> - print(<verbose>, <object>)
- \#Vs\#<object> - summary(<verbose>, <object>)
- \#Vz\#<object> - str(<verbose>, <object>)

Fields and Methods

Methods:

convertComment	Converts a verbose comment to R code.
reset	Resets a VComments compiler.
validate	Validates the compiled lines.

Methods inherited from SmartComments:
compile, convertComment, parse, reset, validate

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Author(s)

Henrik Bengtsson

Examples

```
filename <- system.file("data-ex/exampleVComments.R", package="R.utils")
lines <- readLines(filename)

cat("Code before preprocessing:\n")
displayCode(code=lines, pager="console")

lines <- VComments$compile(lines)

cat("Code after preprocessing:\n")
displayCode(code=lines, pager="console")
```

Verbose *Class to writing verbose messages to a connection or file*

Description

Package: R.utils
Class Verbose

Object
~~|
~~+--Verbose

Directly known subclasses:
MultiVerbose, NullVerbose

public static class **Verbose**
extends Object

Class to writing verbose messages to a connection or file.

Usage

```
Verbose(con=stderr(), on=TRUE, threshold=0, asGString=TRUE, timestamp=FALSE,
  removeFile=TRUE, core=TRUE, ...)
```

Arguments

con	A connection or a character string filename.
on	A logical indicating if the writer is on or off.
threshold	A numeric threshold that the level argument of any write method has to be equal to or larger than in order to the message being written. Thus, the lower the threshold is the more and more details will be outputted.
timestamp	If TRUE, each output is preceded with a timestamp.
removeFile	If TRUE and con is a filename, the file is first deleted, if it exists.
asGString	If TRUE, all messages are interpreted as GString before being output, otherwise not.
core	Internal use only.
...	Not used.

Fields and Methods

Methods:

as.character	Returns a character string version of this object.
as.double	Gets a numeric value of this object.
as.logical	Gets a logical value of this object.
capture	Captures output of a function.
cat	Concatenates and prints objects if above threshold.

enter	Writes a message and indents the following output.
enterf	-
equals	Checks if this object is equal to another.
evaluate	Evaluates a function and prints its results if above threshold.
exit	Writes a message and unindents the following output.
getThreshold	Gets current verbose threshold.
getTimestampFormat	Gets the default timestamp format.
header	Writes a header.
isOn	Checks if the output is on.
isVisible	Checks if a certain verbose level will be shown or not.
less	Creates a cloned instance with a higher threshold.
more	Creates a cloned instance with a lower threshold.
newline	Writes one or several empty lines.
off	Turn off the output.
on	Turn on the output.
popState	-
print	Prints objects if above threshold.
printf	Formats and prints object if above threshold.
pushState	Pushes the current indentation state of the Verbose object.
ruler	Writes a ruler.
setDefaultLevel	Sets the current default verbose level.
setThreshold	Sets verbose threshold.
setTimestampFormat	Sets the default timestamp format.
str	Prints the structure of an object if above threshold.
summary	Generates a summary of an object if above threshold.
timestamp	Writes a timestamp.
timestampOff	-
timestampOn	Turns automatic timestamping on and off.
warnings	Outputs any warnings recorded.
writeRaw	Writes objects if above threshold.

Methods inherited from Object:
$, $<-, [[, [[<-, as.character, attach, attachLocally, clearCache, clearLookupCache, clone, detach, equals, extend, finalize, getEnvironment, getFieldModifier, getFieldModifiers, getFields, getInstantiationTime, getStaticInstance, hasField, hashCode, ll, load, names, objectSize, print, save

Output levels

As a guideline, use the following levels when outputting verbose/debug message using the Verbose class. For a message to be shown, the output level must be greater than (not equal to) current threshold. Thus, the lower the threshold is set, the more messages will be seen.

- <= -100Only for debug messages, i.e. messages containing all necessary information for debugging purposes and to find bugs in the code. Normally these messages are so detailed so they will be a pain for the regular user, but very useful for bug reporting and bug tracking by the developer.

- -99 – -11Detailed verbose messages. These will typically be useful for the user to understand what is going on and do some simple debugging fixing problems typically due to themselves and not due to bugs in the code.
- -10 – -1Verbose messages. For example, these will typically report the name of the file to be read, the current step in a sequence of analysis steps and so on. These message are not very useful for debugging.
- 0Default level in all output methods and default threshold. Thus, by default, messages at level 0 are not shown.
- >= +1Message that are always outputted (if threshold is kept at 0). We recommend not to output message at this level, because methods should be quiet by default (at the default threshold 0).

A compatibility trick and a speed-up trick

If you want to include calls to Verbose in a package of yours in order to debug code, but not use it otherwise, you might not want to load R.utils all the time, but only for debugging. To achieve this, the value of a reference variable to a Verbose class is always set to `TRUE`, cf. typically an Object reference has value `NA`. This makes it possible to use the reference variable as a first test before calling Verbose methods. Example:

```
foo <- function(..., verbose=FALSE) {
  # enter() will never be called if verbose==FALSE, thus no error.
  verbose && enter(verbose, "Loading")
}
```

Thus, R.utils is not required for `foo()`, but for `foo(verbose==Verbose(level=-1))` it is.

Moreover, if using the `NullVerbose` class for ignoring all verbose messages, the above trick will indeed speed up the code, because the value of a NullVerbose reference variable is always `FALSE`.

Extending the Verbose class

If extending this class, make sure to output messages via `*writeRaw()` or one of the other output methods (which in turn all call the former). This guarantees that `*writeRaw()` has full control of the output, e.g. this makes it possible to split output to standard output and to file.

Author(s)

Henrik Bengtsson

See Also

`NullVerbose`.

Examples

```
verbose <- Verbose(threshold=-1)

header(verbose, "A verbose writer example", padding=0)

enter(verbose, "Analysis A")
for (kk in 1:10) {
  printf(verbose, "step %d\n", kk)
  if (kk == 2) {
    cat(verbose, "Turning ON automatic timestamps")
    timestampOn(verbose);
  } else if (kk == 4) {
    timestampOff(verbose);
    cat(verbose, "Turned OFF automatic timestamps")
    cat(verbose, "Turning OFF verbose messages for steps ", kk, "-6")
    off(verbose)
  } else if (kk == 6) {
    on(verbose)
    cat(verbose, "Turned ON verbose messages just before step ", kk+1)
  }

  if (kk %in% c(5,8)) {
    enterf(verbose, "Sub analysis #%d", kk)
    for (jj in c("i", "ii", "iii")) {
      cat(verbose, "part ", jj)
    }
    exit(verbose)
  }
}
cat(verbose, "All steps completed!")
exit(verbose)

ruler(verbose)
cat(verbose, "Demo of some other methods:")
str(verbose, c(a=1, b=2, c=3))
print(verbose, c(a=1, b=2, c=3))
summary(verbose, c(a=1, b=2, c=3))
evaluate(verbose, rnorm, n=3, mean=2, sd=3)

ruler(verbose)
newline(verbose)
```

`withCapture` *Evaluates an expression and captures the code and/or the output*

Description

Evaluates an expression and captures the code and/or the output.

Usage

```
withCapture(expr, replace=getOption("withCapture/substitute", ".x."), code=TRUE,
  output=code, ..., max.deparse.length=getOption("max.deparse.length", 10000), trim=TRUE,
  newline=getOption("withCapture/newline", TRUE), collapse="\n", substitute=replace,
  envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
replace	An optional named list used for substituting symbols with other strings.
code	If TRUE, the deparsed code of the expression is echoed.
output	If TRUE, the output of each evaluated subexpression is echoed.
...	Additional arguments passed to sourceTo which in turn passes arguments to source().
max.deparse.length	A positive integer specifying the maximum length of a deparsed expression, before truncating it.
trim	If TRUE, the captured rows are trimmed.
newline	If TRUE and collapse is non-NULL, a newline is appended at the end.
collapse	A character string used for collapsing the captured rows. If NULL, the rows are not collapsed.
substitute	(to be deprecated) use replace instead.
envir	The environment in which the expression is evaluated.

Value

Returns a character string class 'CapturedEvaluation'.

Author(s)

Henrik Bengtsson

See Also

Internally, eval() is used to evaluate the expression.

Examples

```
print(withCapture({
 n <- 3;
 n;

 for (kk in 1:3) {
   printf("Iteration #%d\n", kk);
 }

 print(Sys.time());
```

```
 type <- "horse";
 type;
}))

## > n <- 3
## > n
## [1] 3
## > for (kk in 1:3) {
## +     printf("Iteration #%d\n", kk)
## + }
## Iteration #1
## Iteration #2
## Iteration #3
## > print(Sys.time())
## [1] "2011-11-06 11:06:32 PST"
## > type <- "horse"
## > type
## [1] "horse"


# Automatic "variable" substitute
# (disable with relabel=NULL)
a <- 2
b <- "Hello world!"

print(withCapture({
 x <- .a.
 s <- .b.
 x
 s
}))

## > x <- 2
## > s <- "Hello world!"
## > x
## [1] 2
## > s
## [1] "Hello world!"
```

`withLocale` *Evaluate an R expression with locale set temporarily*

Description

Evaluate an R expression with locale set temporarily.

Usage

```
withLocale(expr, category, locale, ..., substitute=TRUE, envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
category	A character string specifying the category to use.
locale	character vector specifying the locale to used. The first successfully set one will be used.
...	Not used.
substitute	If TRUE, argument expr is substitute():ed, otherwise not.
envir	The environment in which the expression should be evaluated.

Value

Returns the results of the expression evaluated.

Author(s)

Henrik Bengtsson

See Also

Internally, eval() is used to evaluate the expression. and Sys.setlocale() to set locale.

Examples

```
# Vector
cat("Original vector:\n")
x <- c(letters[1:8], LETTERS[1:8])
print(x)

cat("Sorting with 'C' locale:\n")
y1 <- withLocale(sort(x), "LC_COLLATE", "C")
print(y1)

cat("Sorting with an 'English' locale:\n")
y2 <- withLocale(sort(x), "LC_COLLATE", c("en_US", "en_US.UTF8", "English_United States.1252"))
print(y2)
```

---

withOptions *Evaluate an R expression with options set temporarily*

Description

Evaluate an R expression with options set temporarily.

Usage

```
withOptions(expr, ..., args=list(), substitute=TRUE, envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
...	Named options to be used.
args	(optional) Additional named options specified as a named list.
substitute	If TRUE, argument expr is substitute():ed, otherwise not.
envir	The environment in which the expression should be evaluated.

Details

Upon exit (also on errors), this function will reset *all* options to the state of options available upon entry. This means any options *modified* but also those *added* when evaluating expr will also be undone upon exit.

Value

Returns the results of the expression evaluated.

Author(s)

Henrik Bengtsson

See Also

Internally, eval() is used to evaluate the expression. and options() to set options.

Examples

```
print(pi)

# Same, i.e. using default
withOptions({
  print(pi)
})

# Printing with two digits
withOptions({
  print(pi)
}, digits=2)

# Printing with two digits then with three more
withOptions({
  print(pi)
  withOptions({
    print(pi)
  }, digits=getOption("digits")+3)
}, digits=2)

# Still printing with the default
print(pi)
```

withRepos *Evaluate an R expression with repositories set temporarily*

Description

Evaluate an R expression with repositories set temporarily.

Usage

```
withRepos(expr, repos="[[mainstream]]", ..., substitute=TRUE, envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
repos	A character vector of repositories to use.
...	Additional arguments passed to useRepos().
substitute	If TRUE, argument expr is substitute():ed, otherwise not.
envir	The environment in which the expression should be evaluated.

Value

Returns the results of the expression evaluated.

Author(s)

Henrik Bengtsson

See Also

Internally, eval() is used to evaluate the expression. See also options() and install.packages.

Examples

```
## Not run:
  # Install from BioC related repositories only
  withRepos(install.packages("edgeR"), repos="[[BioC]]")

  # Install from CRAN or BioC related repositories only
  withRepos(install.packages("edgeR"), repos=c("CRAN", "[[BioC]]"))

  # Install from mainstream repositories only (same as previous)
  withRepos(install.packages("edgeR"), repos="[[mainstream]]")

  # Install from R-Forge and mainstream repositories only
  withRepos(install.packages("R.utils"), repos="[[R-Forge]]")

  # Update only CRAN packages
  withRepos(update.packages(ask=FALSE), repos="[[CRAN]]")
```

```
  # Update only Bioconductor packages
  withRepos(update.packages(ask=FALSE), repos="[[BioC]]")

## End(Not run)
```

---

withSeed *Evaluate an R expression with a temporarily set random set*

Description

Evaluate an R expression with a temporarily set random set.

Usage

```
withSeed(expr, seed, ..., substitute=TRUE, envir=parent.frame())
```

Arguments

expr	The R expression to be evaluated.
seed, ...	Arguments passed to set.seed().
substitute	If TRUE, argument expr is substitute():ed, otherwise not.
envir	The environment in which the expression should be evaluated.

Details

Upon exit (also on errors), this function will restore .Random.seed in the global environment to the value it had upon entry. If it did not exist, it will be removed.

Value

Returns the results of the expression evaluated.

Author(s)

Henrik Bengtsson

See Also

Internally, set.seed() is used to set the random seed.

Examples

```
# Generate a random number
y0 <- runif(1)
print(y0)

# Generate a random number using the same seed over and over
yp <- NULL
for (ii in 1:10) {
  y <- withSeed({
    runif(1)
  }, seed=0x42)
  print(y)
  # Assert identical
  if (!is.null(yp)) stopifnot(identical(y, yp))
  yp <- y
}

# Generate a random number
y <- runif(1)
print(y)
```

`withSink` *Evaluate an R expression while temporarily diverting output*

Description

Evaluate an R expression while temporarily diverting output.

Usage

```
withSink(expr, file, append=FALSE, type=c("output", "message"), substitute=TRUE,
  envir=parent.frame())
```

Arguments

`expr`	The R expression to be evaluated.
`file`	A writable `connection` or a `character` string naming the file to write to.
`append`	If `TRUE`, the diverted output is appended to the file, otherwise not.
`type`	A `character` string specifying whether to divert output sent to the standard output or the standard error. See `sink()` for details.
`substitute`	If `TRUE`, argument `expr` is `substitute()`:ed, otherwise not.
`envir`	The `environment` in which the expression should be evaluated.

Details

Upon exit (also on errors), this function will close the requested "sink". If additional sinks (of any type) where also opened during the evaluation, those will also be closed with a warning.

Value

Returns the results of the expression evaluated.

Author(s)

Henrik Bengtsson

See Also

Internally, sink() is used to divert any output.

Examples

```
# Divert standard output
pathname <- tempfile(fileext=".output.txt")
res <- withSink(file=pathname, {
  print(letters)
})
mcat(readLines(pathname), sep="\n")


# Divert standard error/messages
pathname <- tempfile(fileext=".message.txt")
res <- withSink(file=pathname, type="message", {
  mprint(LETTERS)
})
mcat(readLines(pathname), sep="\n")
```

---

withTimeout *Evaluate an R expression and interrupts it if it takes too long*

Description

Evaluate an R expression and interrupts it if it takes too long.

Usage

```
withTimeout(expr, substitute=TRUE, envir=parent.frame(), timeout, cpu=timeout,
  elapsed=timeout, onTimeout=c("error", "warning", "silent"), ...)
```

Arguments

expr	The R expression to be evaluated.
substitute	If TRUE, argument expr is substitute():ed, otherwise not.
envir	The environment in which the expression should be evaluated.

`timeout, cpu, elapsed`
: A `numeric` specifying the maximum number of seconds the expression is allowed to run before being interrupted by the timeout. The `cpu` and `elapsed` arguments can be used to specify whether time should be measured in CPU time or in wall time.

`onTimeout`
: A `character` specifying what action to take if a timeout event occurs.

`...`
: Not used.

Details

This method utilizes `setTimeLimit`() by first setting the timeout limits, then evaluating the expression that may or may not timeout. The method is guaranteed to reset the timeout limits to be infinitely long upon exiting, regardless whether it returns normally or preemptively due to a timeout or an error.

Value

Returns the results of the expression evaluated. If timed out, `NULL` is returned if `onTimeout` was `"warning"` or `"silent"`. If `"error"` a `TimeoutException` is thrown.

Non-supported cases

In order to understand when this function works and when it does not, it is useful to know that it utilizes R's built-in time-out mechanism, which sets the limits on what is possible and not. From `setTimeLimit`(), we learn that:

"Time limits are checked whenever a user interrupt could occur. This will happen frequently in R code and during Sys.sleep(*), but only at points in compiled C and Fortran code identified by the code author."

More precisely, if a function is implemented in native code (e.g. C) and the developer of that function does not check for user interrupts, then you cannot interrupt that function neither via a user interrupt (e.g. Ctrl-C) *nor via the built-in time out mechanism*. To change this, you need to contact the developer of that piece of code and ask them to check for R user interrupts in their native code.

Furthermore, it is not possible to interrupt/break out of a "readline" prompt (e.g. `readline`() and `readLines`()) using timeouts; the timeout exception will not be thrown until after the user completes the prompt (i.e. after pressing ENTER).

System calls via `system`() and `system2()` cannot be timed out via the above mechanisms. However, in R (>= 3.5.0) these functions have argument `timeout` providing their own independent timeout mechanism.

Other examples of calls that do *not* support timeout are "atomic" calls that may take very long such as large object allocation and `rnorm(n)` where `n` is very large.

(*) Note that on Unix and macOS, `Sys.sleep(time)` will signal a timeout error only *after* `time` seconds passed, regardless of `timeout` limit (< `time`).

Author(s)

Henrik Bengtsson

References

[1] R help thread 'Time out for a R Function' on 2010-12-06. http://www.mail-archive.com/r-help@r-project.org/msg119344.html

See Also

Internally, eval() is used to evaluate the expression and setTimeLimit() is used to control for timeout events.

Examples

```
# - - - - - - - - - - - - - - - - - - - - - - - - - -
# Function that takes "a long" time to run
# - - - - - - - - - - - - - - - - - - - - - - - - - -
foo <- function() {
  print("Tic")
  for (kk in 1:100) {
    print(kk)
    Sys.sleep(0.1)
  }
  print("Tac")
}


# - - - - - - - - - - - - - - - - - - - - - - - - - -
# Evaluate code, if it takes too long, generate
# a timeout by throwing a TimeoutException.
# - - - - - - - - - - - - - - - - - - - - - - - - - -
res <- NULL
tryCatch({
  res <- withTimeout({
    foo()
  }, timeout = 1.08)
}, TimeoutException = function(ex) {
  message("Timeout. Skipping.")
})


# - - - - - - - - - - - - - - - - - - - - - - - - - -
# Evaluate code, if it takes too long, generate
# a timeout returning NULL and generate a warning.
# - - - - - - - - - - - - - - - - - - - - - - - - - -
res <- withTimeout({
  foo()
}, timeout = 1.08, onTimeout = "warning")


# The same using an expression object
expr <- quote(foo())
res <- withTimeout(expr, substitute = FALSE,
                   timeout = 1.08, onTimeout = "warning")
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - -
# Evaluate code, if it takes too long, generate
# a timeout, and return silently NULL.
# - - - - - - - - - - - - - - - - - - - - - - - - - -
res <- withTimeout({
  foo()
}, timeout = 1.08, onTimeout = "silent")
```

`wrap.array` *Reshape an array or a matrix by permuting and/or joining dimensions*

Description

Reshape an array or a matrix by permuting and/or joining dimensions.

A useful application of this is to reshape a multidimensional `array` to a `matrix`, which then can be saved to file using for instance `write.table()`.

Usage

```
## S3 method for class 'array'
wrap(x, map=list(NA), sep=".", ...)
```

Arguments

`x`	An `array` or a `matrix`.
`map`	A `list` of length equal to the number of dimensions in the reshaped array. Each element should be an `integer` `vector`s specifying the dimensions to be joined in corresponding new dimension. One element may equal `NA` to indicate that that dimension should be a join of all non-specified (remaining) dimensions. Default is to wrap everything into a `vector`.
`sep`	A `character` pasting joined dimension names.
`...`	Not used.

Details

If the indices in `unlist(map)` is in a non-increasing order, aperm() will be called, which requires reshuffling of array elements in memory. In all other cases, the reshaping of the array does not require this, but only fast modifications of attributes `dim` and `dimnames`.

Value

Returns an `array` of `length(map)` dimensions, where the first dimension is of size `prod(map[[1]])`, the second `prod(map[[2]])`, and so on.

Author(s)

Henrik Bengtsson

See Also

*unwrap(). See aperm().

Examples

```
# Create a 3x2x3 array
dim <- c(3,2,3)
ndim <- length(dim)
dimnames <- list()
for (kk in 1:ndim)
  dimnames[[kk]] <- sprintf("%s%d", letters[kk], 1:dim[kk])
x <- 1:prod(dim)
x <- array(x, dim=dim, dimnames=dimnames)


cat("Array 'x':\n")
print(x)


cat("\nReshape 'x' to its identity:\n")
y <- wrap(x, map=list(1, 2, 3))
print(y)
# Assert correctness of reshaping
stopifnot(identical(y, x))


cat("\nReshape 'x' by swapping dimensions 2 and 3, i.e. aperm(x, perm=c(1,3,2)):\n")
y <- wrap(x, map=list(1, 3, 2))
print(y)
# Assert correctness of reshaping
stopifnot(identical(y, aperm(x, perm=c(1,3,2))))


cat("\nWrap 'x' to a matrix 'y' by keeping dimension 1 and joining the others:\n")
y <- wrap(x, map=list(1, NA))
print(y)
# Assert correctness of reshaping
for (aa in dimnames(x)[[1]]) {
  for (bb in dimnames(x)[[2]]) {
    for (cc in dimnames(x)[[3]]) {
      tt <- paste(bb, cc, sep=".")
      stopifnot(identical(y[aa,tt], x[aa,bb,cc]))
    }
  }
}
```

```
cat("\nUnwrap matrix 'y' back to array 'x':\n")
z <- unwrap(y)
print(z)
stopifnot(identical(z,x))


cat("\nWrap a matrix 'y' to a vector and back again:\n")
x <- matrix(1:8, nrow=2, dimnames=list(letters[1:2], 1:4))
y <- wrap(x)
z <- unwrap(y)
print(z)
stopifnot(identical(z,x))


cat("\nWrap and unwrap a randomly sized and shaped array 'x2':\n")
maxdim <- 5
dim <- sample(1:maxdim, size=sample(2:maxdim, size=1))
ndim <- length(dim)
dimnames <- list()
for (kk in 1:ndim)
  dimnames[[kk]] <- sprintf("%s%d", letters[kk], 1:dim[kk])
x2 <- 1:prod(dim)
x2 <- array(x, dim=dim, dimnames=dimnames)

cat("\nArray 'x2':\n")
print(x)

# Number of dimensions of wrapped array
ndim2 <- sample(1:(ndim-1), size=1)

# Create a random map for joining dimensions
splits <- NULL;
if (ndim > 2)
  splits <- sort(sample(2:(ndim-1), size=ndim2-1))
splits <- c(0, splits, ndim);
map <- list();
for (kk in 1:ndim2)
  map[[kk]] <- (splits[kk]+1):splits[kk+1];

cat("\nRandom 'map':\n")
print(map)

cat("\nArray 'y2':\n")
y2 <- wrap(x2, map=map)
print(y2)

cat("\nArray 'x2':\n")
z2 <- unwrap(y2)
print(z2)

stopifnot(identical(z2,x2))
```

writeBinFragments *Writes binary data to disjoint sections of a connection or a file*

Description

Writes binary data to disjoint sections of a connection or a file.

Usage

```
## Default S3 method:
writeBinFragments(con, object, idxs, size=NA, ...)
```

Arguments

con	A connection or the pathname of an existing file.
object	A vector of objects to be written.
idxs	A vector of (non-duplicated) indices or a Nx2 matrix of N from-to index intervals specifying the elements to be read. Positions are always relative to the start of the file/connection.
size	The size of the data type to be read. If NA, the natural size of the data type is used.
...	Additional arguments passed to writeBin().

Value

Returns nothing.

Author(s)

Henrik Bengtsson

See Also

readBinFragments().

Examples

```
## Not run: # See example(readBinFragments.connection)
```

`writeDataFrame.data.frame` *Writes a data.frame to tabular text file*

Description

Writes a data.frame to tabular text file with an optional header.

Usage

```
## S3 method for class 'data.frame'
writeDataFrame(data, file, path=NULL, sep="\t", quote=FALSE, row.names=FALSE,
  col.names=!append, ..., header=list(), createdBy=NULL,
  createdOn=format(Sys.time(), format = "%Y-%m-%d %H:%M:%S %Z"),
  nbrOfRows=nrow(data), headerPrefix="# ", headerSep=": ", append=FALSE, overwrite=FALSE)
```

Arguments

`data`	A `data.frame`.
`file`	A `connection` or a filename to write to.
`path`	The directory where the file will be written.
`sep, quote, row.names, col.names, ...`	Additional arguments passed to `write.table`.
`header`	An optional named `list` of header rows to be written at the beginning of the file. If `NULL`, no header will be written.
`createdBy, createdOn, nbrOfRows`	If non-`NULL`, common header rows to be added to the header.
`headerPrefix`	A `character` string specifying the prefix of each header row.
`headerSep`	A `character` string specifying the character separating the header name and header values.
`append`	If `TRUE`, the output is appended to an existing file.
`overwrite`	If `TRUE`, an existing file is overwritten.

Value

Returns (invisibly) the pathname to the file written (or the `connection` written to).

Author(s)

Henrik Bengtsson

See Also

`write.table`. `readTable`().

Index