Package ‘toscutil’

June 30, 2022

Title Utility Functions

Version 2.5.0

Description Base R sometimes requires verbose statements for simple, often recurring tasks, such as printing text without trailing space, ending with newline. This package aims at providing shorthands for such tasks.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.0

Imports methods, utils, rlang

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Tobias Schmidt [aut, cre]

Maintainer Tobias Schmidt <`tobias.schmidt331@gmail.com`>

Repository CRAN

Date/Publication 2022-06-30 10:10:02 UTC

R topics documented:

`caller` *Return Name of Calling Function*

Description

Returns the name of a calling function as string, i.e. if function `g` calls function `f` and function `f` calls caller(2)`, then string "g" is returned.

Usage

```
caller(n = 1)
```

Arguments

`n`	How many frames to go up in the call stack

Details

Be careful when using `caller(n)` as input to other functions. Due to R's non-standard-evaluation (NES) mechanism it is possible that the function is not executed directly by that function but instead passed on to other functions, i.e. the correct number of frames to go up cannot be predicted a priori. Solutions are to to evaluate the function first and store the result in a variable and then pass the variable to the function or to just try out the required number of frames to go up in an interactive session. For further examples see section Examples.

Value

Name of calling function

Examples

```
# Here we want to return a list of all variables created inside a function
f <- function(a = 1, b = 2) {
  x <- 3
  y <- 4
  return(locals(without = formalArgs(caller(4))))
  # We need to go 4 frames up, to catch the formalArgs of `f`, because the
  # `caller(4)` argument is not evaluated directly be `formalArgs`.
}
all.equal(setdiff(f(), list(x = 3, y = 4)), list())

# The same result could have been achieved as follows
f <- function(a = 1, b = 2) {
  x <- 3
  y <- 4
  func <- caller(1)
  return(locals(without = c("func", formalArgs(func))))
}
all.equal(setdiff(f(), list(x = 3, y = 4)), list())
```

`cat2` *Concatenate and Print*

Description

Same as `cat` but with an additional argument `end`, which gets printed after all other elements. Inspired by pythons `print` command.

Deprecation note: all cat aliases, i.e., everything except cat2 are deprecated and should not be used any more!

Usage

```
cat2(..., sep = " ", end = "\n")

cat0(..., sep = "", end = "")

catn(..., sep = " ", end = "\n")

cat0n(..., sep = "", end = "\n")

catsn(..., sep = " ", end = "\n")

catnn(..., sep = "\n", end = "\n")
```

Arguments

`...`	objects passed on to cat
`sep`	a character vector of strings to append after each element
`end`	a string to print after all other elements

Value

No return value, called for side effects

Examples

```
cat0("hello", "world") # prints "helloworld" (without newline)
catn("hello", "world") # prints "hello world\n"
cat0n("hello", "world") # prints "helloworld\n"
catsn("hello", "world") # prints "hello world\n"
catnn("hello", "world") # prints "hello\nworld\n"
```

`catf` *Format and Print*

Description

Same as `cat2(sprintf(fmt, ...))`

Usage

```
catf(
  fmt,
  ...,
  end = "",
  file = "",
  sep = " ",
  fill = FALSE,
  labels = NULL,
  append = FALSE
)

catfn(
  fmt,
  ...,
  end = "\n",
  file = "",
  sep = " ",
  fill = FALSE,
  labels = NULL,
  append = FALSE
)
```

Arguments

`fmt`	passed on to `base::sprintf()`
`...`	passed on to `base::sprintf()`
`end`	passed on to `cat2()`

`file`	passed on to `cat2()` (which passes it on to `base::cat()`)
`sep`	passed on to `cat2()` (which passes it on to `base::cat()`)
`fill`	passed on to `cat2()` (which passes it on to `base::cat()`)
`labels`	passed on to `cat2()` (which passes it on to `base::cat()`)
`append`	passed on to `cat2()` (which passes it on to `base::cat()`)

Value

No return value, called for side effects

Examples

```
catf("A%dB%sC", 2, "asdf") # prints "A2BasdfC"
catfn("A%dB%sC", 2, "asdf") # prints "A2BasdfC\n"
```

`config_dir` *Return Normalized Configuration Directory Path of a Program*

Description

`config_dir` returns the absolute, normalized path to the configuration directory of a program/package/app based on an optional app-specific commandline argument, an optional app-specific environment variable and the XDG Base Directory Specification

Usage

```
config_dir(
  app_name,
  cl_arg = {
     commandArgs()[grep("--config-dir", commandArgs()) + 1]
 },
  env_var = Sys.getenv(toupper(paste0(app_name, "_config_dir()"))),
  create = FALSE,
  sep = "/"
)
```

Arguments

`app_name`	Name of the program/package/app
`cl_arg`	Value of app specific commandline parameter
`env_var`	Value of app specific environment variable
`create`	whether to create returned path, if it doesn't exists yet
`sep`	Path separator to be used on Windows

Details

The following algorithm is used to determine the location of the configuration directory for application `<app_name>`:

1. If parameter `<cl_arg>` is a non-empty string, return `cl_arg`
2. Else, if parameter `<env_var>` is a non-empty string, return `<env_var>`
3. Else, if environment variable (EV) `$XDG_CONFIG_HOME` exists, return `$XDG_CONFIG_HOME/<app_name>`
4. Else, if EV `$HOME` exists, return `$HOME/.config/<app_name>`
5. Else, if EV `$USERPROFILE` exists, return `$USERPROFILE/.config/<app_name>`
6. Else, return `<current-working-directory>/.config/<app-name>`

Value

Normalized path to the configuration directory of `<app_name>`.

See Also

`data_dir()`, `config_file()`, `xdg_config_home()`

Examples

```
config_dir("myApp")
```

---

config_file *Return Normalized Configuration File Path of a Program*

Description

`config_file` returns the absolute, normalized path to the configuration file of a program/package/app based on an optional app-specific commandline argument, an optional app-specific environment variable and the XDG Base Directory Specification

Usage

```
config_file(
  app_name,
  file_name,
  cl_arg = {
     commandArgs()[grep("--config-file", commandArgs()) + 1]
 },
  env_var = "",
  sep = "/",
  copy_dir = norm_path(xdg_config_home(), app_name),
  fallback_path = NULL
)
```

Arguments

app_name	Name of the program/package/app
file_name	Name of the configuration file
cl_arg	Value of app specific commandline parameter
env_var	Value of app specific environment variable
sep	Path separator to be used on Windows
copy_dir	Path to directory where <fallback_path> should be copied to in case it gets used.
fallback_path	Value to return as fallback (see details)

Details

The following algorithm is used to determine the location of <file_name>:

1. If <cl_arg> is a non-empty string, return it
2. Else, if <env_var> is a non-empty string, return it
3. Else, if ${PWD}/.config/<app-name> exists, return it
4. Else, if $XDG_CONFIG_HOME/<app_name>/<file_name> exists, return it
5. Else, if $HOME/.config/<app_name>/<file_name> exists, return it
6. Else, if $USERPROFILE/.config/<app_name>/<file_name> exists, return it
7. Else, if <copy_dir> is non-empty string and <fallback_path> is a path to an existing file, then try to copy <fallback_path> to copy_dir/<file_name> and return copy_dir/<file_name> (Note, that in case <copy_dir> is a non-valid path, the function will throw an error.)
8. Else, return <fallback_path>

Value

Normalized path to the configuration file of <app_name>.

See Also

config_dir(), xdg_config_home()

Examples

```
config_dir("myApp")
```

corn *Return Corners of Matrix like Objects*

Description

Like head and tail, but returns n rows/cols from each side of x (i.e. the corners of x)

Usage

```
corn(x, n = 2L)
```

Arguments

x	matrix like object
n	number of cols/rows from each corner to return

Value

x[c(1:n, N-n:N), c(1:n, N-n:N)]

Examples

```
corn(matrix(1:10000, 100))
```

data_dir *Return Normalized Data Directory Path of a Program*

Description

data_dir returns the absolute, normalized path to the data directory of a program/package/app based on an optional app-specific commandline argument, an optional app-specific environment variable and the XDG Base Directory Specification

Usage

```
data_dir(
  app_name,
  cl_arg = commandArgs()[grep("--data-dir", commandArgs()) + 1],
  env_var = Sys.getenv(toupper(paste0(app_name, "_DATA_DIR"))),
  create = FALSE,
  sep = "/"
)
```

Arguments

`app_name`	Name of the program/package/app
`cl_arg`	Value of app specific commandline parameter
`env_var`	Value of app specific environment variable
`create`	whether to create returned path, if it doesn't exists yet
`sep`	Path separator to be used on Windows

Details

The following algorithm is used to determine the location of the data directory for application `<app_name>`:

1. If parameter `<cl_arg>` is a non-empty string, return `cl_arg`
2. Else, if parameter `<env_var>` is a non-empty string, return `<env_var>`
3. Else, if environment variable (EV) `$XDG_DATA_HOME` exists, return `$XDG_DATA_HOME/<app_name>`
4. Else, if EV `$HOME` exists, return `$HOME/.local/share/<app_name>`
5. Else, if EV `$USERPROFILE` exists, return `$USERPROFILE/.local/share/<app_name>`
6. Else, return `<current-working-directory>/.local/share`

Value

Normalized path to the data directory of `<app_name>`.

See Also

`config_dir()`, `xdg_data_home()`

Examples

```
data_dir("myApp")
```

`function_locals` *Return function environment as list*

Description

Return current env without function arguments as list. Raises an error when called outside a function.

Usage

```
function_locals(without = c(), strip_function_args = TRUE)
```

Arguments

`without`	character vector of symbols to exclude
`strip_function_args`	Whether to exclude symbols with the same name as the function arguments

Details

The order of the symbols in the returned list is arbitrary.

Value

The function environment as list

Examples

```
f <- function(a = 1, b = 2) {
  x <- 3
  y <- 4
  return(function_locals())
}
all.equal(setdiff(f(), list(x = 3, y = 4)), list())
```

`getfd` *Get File Directory*

Description

Return full path to current file directory

Usage

```
getfd(
  on.error = stop("No file sourced. Maybe you're in an interactive shell?", call. =
     FALSE),
   winslash = "/"
)
```

Arguments

`on.error`	Expression to use if the current file directory cannot be determined. In that case, `normalizePath(<on.error>, winslash)` is returned. Can also be an expression like `stop("message")` to stop execution (default).
`winslash`	Path separator to use for windows

Value

Current file directory as string

Examples

```
## Not run: getfd()
getfd(on.error=getwd())
```

---

getpd *Get Project Directory*

Description

Find the project root directory by traversing the current working directory filepath upwards until a given set of files is found.

Usage

```
getpd(root.files = c(".git", "DESCRIPTION", "NAMESPACE"))
```

Arguments

`root.files`	if any of these files is found in a parent folder, the path to that folder is returned

Value

`getpd` returns the project root directory as string

---

home *Get USERPROFILE or HOME*

Description

Returns normalized value of environment variable USERPROFILE, if defined, else value of HOME.

Usage

```
home(winslash = "/")
```

Arguments

`winslash`	path separator to be used on Windows (passed on to `normalizePath`)

Value

normalized value of environment variable USERPROFILE, if defined, else value of HOME.

Examples

```
home()
```

`ifthen` *Shortcut for multiple else if statements*

Description

`ifthen(a, b, c, d, e, f, ...) == if (a) b else if (c) d else if (e) f`

Usage

```
ifthen(...)
```

Arguments

`...`	pairs of checks and corresponding return values

Value

`ifelse` returns the first value for which the corresponding statement evaluates to TRUE

Examples

```
x <- 2; y <- 2; z <- 1
ifthen(x==0, "foo", y==0, "bar", z==1, "this string gets returned")
```

`is.none` *Like* R*hrefhttps://docs.python.org/3/library/functions.html#boolpythons* `bool`

Description

TRUE for FALSE, 0, NULL, NA, empty lists and empty string

Usage

```
is.none(x)
```

Arguments

`x`	object to test

Value

TRUE if `x` is FALSE, 0, NULL, NA, an empty list or an empty string. Else FALSE.

Examples

```
is.none(FALSE) # TRUE
is.none(0) # TRUE
is.none(NA) # TRUE
is.none(list()) # TRUE
is.none("") # TRUE
is.none(1) # FALSE
```

`locals` *Return specified Environment as List*

Description

Return symbols in given environment as list.

Usage

```
locals(without = c(), env = parent.frame())
```

Arguments

`without`	Character vector. Symbols from current env to exclude.
`env`	Environment to use. Defaults to the environment from which `locals` is called.

Value

Specified environment as list (without the mentioned symbols).

`named` *Automatically named List*

Description

Like normal `list()`, except that unnamed elements are automatically named according to their symbol

Usage

```
named(...)
```

Arguments

`...`	List elements

Value

Object of type `list` with names attribute set

See Also

`list()`

Examples

```
a <- 1:10
b <- "helloworld"
l1 <- list(a, b)
names(l1) <- c("a", "b")
l2 <- named(a, b)
identical(l1, l2)
l3 <- list(z=a, b=b)
l4 <- named(z=a, b)
identical(l3, l4)
```

`norm_path` *Return Normalized Path*

Description

Shortcut for `normalizePath(file.path(...), winslash=sep, mustWork=FALSE)`

Usage

```
norm_path(..., sep = "/")
```

Arguments

`...`	Parts used to construct the path
`sep`	Path separator to be used on Windows

Value

Normalized path constructed from ...

Examples

```
norm_path("C:/Users/max", "a\\b", "c") # returns C:/Users/max/a/b/c
norm_path("a\\b", "c") # return <current-working-dir>/a/b/c
```

`now` *Get Current Date and Time as string*

Description

`now` returns current system time as string of the form "YYYY-MM-DD hh:mm:ss TZ", where TZ means "timezone".

Usage

```
now()
```

Value

`now` returns current system time as string of the form "YYYY-MM-DD hh:mm:ss TZ", where TZ means "timezone" (strictly speaking, the format as given to `format()` is `%Y-%m-%d %H:%M:%S`, for details see `[format.POSIXct()]`).

See Also

`now_ms()`, `Sys.time()`, `format.POSIXct()`

Examples

```
now() # "2021-11-27 19:19:31 CEST"
```

`now_ms` *Get Current Date and Time as string*

Description

`now_ms` returns current system time as string of the form "YYYY-MM-DD hh:mm:ss.XX TZ", where XX means "milliseconds" and TZ means "timezone".

Usage

```
now_ms()
```

Value

Current system time as string of the form "YYYY-MM-DD hh:mm:ss.XX TZ", where XX means "milliseconds" and TZ means "timezone".

See Also

`now()`, `Sys.time()`, `format.POSIXct()`

Examples

```
now() # something like "2022-06-30, 07:14:26.82 CEST"
```

---

op-null-default *Default operator*

Description

Like rlang's `%||%` but also checks for empty lists and empty strings (for details see *https://rdrr.io/cran/rlang/man/op-null-default.html*).

Usage

```
x %none% y
```

Arguments

x	object to test
y	object to return if `is.none(x)`

Value

Returns `y` if `is.none(x)` else `x`

See Also

`is.none()`

Examples

```
FALSE %none% 2 # returns 2
0 %none% 2 # returns 2
NA %none% 2 # returns 2
list() %none% 2 # returns 2
"" %none% 2 # returns 2
1 %none% 2 # returns 1
```

predict.numeric *Predict Method for Numeric Vectors*

Description

Interprets the provided numeric vector as linear model and uses it to generate prediction.

Usage

```
## S3 method for class 'numeric'
predict(object, newdata, ...)
```

Arguments

`object`	Named numeric vector of beta values. If an element is named "Intercept", that element is interpreted as model intercept.
`newdata`	Matrix with samples as rows and features as columns.
`...`	further arguments passed to or from other methods

Value

Named numeric vector of predicted scores

Examples

```
X <- matrix(1:4, 2, 2, dimnames=list(c("s1", "s2"), c("a", "b")))
b <- c(Intercept=3, a=2, b=1)
predict(b, X)
```

rm_all *Remove all objects from global environment*

Description

Same as rm(list=ls())

Usage

```
rm_all()
```

Value

No return value, called for side effects

Examples

```
## Not run: rm_all()
```

`stub` *Stub Function Arguments*

Description

`stub()` assigns all arguments of a given function as symbols to the specified environment (usually the current environemnt)

Usage

```
stub(func, ..., envir = parent.frame())
```

Arguments

`func`	function for which the arguments should be stubbed
`...`	non-default arguments of `func`
`envir`	environment to which symbols should be assigned

Details

Stub is thought to be used for interactive testing and unit testing. It does not work for primitiv functions.

Value

list of symbols that are assigned to `envir`

Examples

```
f <- function(x, y = 2, z = 3) x + y + z
args <- stub(f, x = 1) # assigns x = 1, y = 2 and z = 3 to current env
```

`sys.exit` *Terminate a non-interactive R Session*

Description

Similar to Python's sys.exit. If used interactively, code execution is stopped with an error message, giving the provided status code. If used non-interactively (e.g. through Rscript), code execution is stopped silently and the process exits with the provided status code.

Usage

```
sys.exit(status = 0)
```

Arguments

`status`	exitcode for R process

Value

No return value, called for side effects

Examples

```
## Not run:
if (!file.exists("some.file")) {
  cat("Error: some.file does not exist.\n", file=stderr())
  sys.exit(1)
} else if (Sys.getenv("IMPORTANT_ENV")=="") {
  cat("Error: IMPORTANT_ENV not set.\n", file=stderr())
  sys.exit(2)
} else {
  cat("Everything good. Starting calculations...")
  # ...
  cat("Finished with success!")
  sys.exit(0)
}

## End(Not run)
```

`xdg_config_home` *Return $XDG_CONFIG_HOME*

Description

Return value for $XDG_CONFIG_HOME as defined by the XDG Base Directory Specification

Usage

```
xdg_config_home(sep = "/", fallback = normalizePath(getwd(), winslash = sep))
```

Arguments

`sep`	Path separator to be used on Windows
`fallback`	Value to return as fallback (see details)

Value

The following algorithm is used to determine the returned path:

1. If environment variable (EV) `$XDG_CONFIG_HOME` exists, return its value
2. Else, if EV `$HOME` exists, return `$HOME/.config`
3. Else, if EV `$USERPROFILE` exists, return `$USERPROFILE/.config`
4. Else, return `<fallback>`

See Also

`xdg_data_home()`

Examples

```
xdg_config_home()
```

xdg_data_home *Return $XDG_DATA_HOME*

Description

Return value for $XDG_DATA_HOME as defined by the XDG Base Directory Specification

Usage

```
xdg_data_home(sep = "/", fallback = normalizePath(getwd(), winslash = sep))
```

Arguments

`sep`	Path separator to be used on Windows
`fallback`	Value to return as fallback (see details)

Value

The following algorithm is used to determine the returned path:

1. If environment variable (EV) `$XDG_DATA_HOME` exists, return its value
2. Else, if EV `$HOME` exists, return `$HOME/.local/share`
3. Else, if EV `$USERPROFILE` exists, return `$USERPROFILE/.local/share`
4. Else, return `<fallback>`

See Also

`xdg_config_home()`

Examples

```
xdg_data_home()
```

Index