Package ‘toscutil’

September 5, 2023

Title Utility Functions

Version 2.7.4

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 7.2.3

Imports utils, rlang, tools, languageserver

Suggests testthat (>= 3.0.0),

Config/testthat/edition 3

URL `https://toscm.github.io/toscutil/`

NeedsCompilation no

Author Tobias Schmidt [aut, cre]

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

Repository CRAN

Date/Publication 2023-09-05 09:00:02 UTC

R topics documented:

`caller` *Get 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 evaluate the function first, 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 the 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`.
}
f() # returns either list(x = 3, y = 4) or list(y = 4, x = 3)

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

`capture.output2` *Capture output from a command*

Description

Like classic `capture.output()`, but with additional arguments `collapse` and `trim`.

Usage

```
capture.output2(..., collapse = "\n", trim = FALSE)
```

Arguments

`...`	Arguments passed on to `capture.output()`.
`collapse`	If `TRUE`, lines are collapsed into a single string. If `FALSE`, lines are returned as is. If any character, lines are collapsed using that character.
`trim`	If `TRUE`, leading and trailing whitespace from each line is removed before the lines are collapsed and/or returned.

Value

If `collapse` is `TRUE` or `"\n"`, a character vector of length 1. Else, a character vector of length `n`, where `n` corresponds to the number of lines outputted by the expression passed to `capture.output()`.

See Also

`capture.output()`

Examples

```
x <- capture.output2(str(list(a=1, b=2, c=1:3)))
cat2(x)
```

`cat0` *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.

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

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

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)
```

---

cat0n *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.

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

```
cat0n(..., sep = "", 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

```
cat0n("hello", "world") # prints "helloworld\n"
```

---

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.

Usage

```
cat2(
  ...,
  sep = " ",
  end = "\n",
  file = "",
  append = FALSE,
  fill = FALSE,
  labels = NULL
)
```

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
`file`	passed on to `base::cat()`
`append`	passed on to `base::cat()`
`fill`	passed on to `base::cat()`
`labels`	passed on to `base::cat()`

Value

No return value, called for side effects

Examples

```
x <- 1
cat("x:", x, "\n") # prints 'Number: 1 \n' (with a space between 1 and \n)
cat2("x:", x) # prints 'Number: 1\n'  (without space)
```

`catf` *Format and Print*

Description

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

Usage

```
catf(
  fmt,
  ...,
  end = "",
  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` *Format and Print*

Description

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

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

```
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

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

`catn` *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.

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

```
catn(..., sep = " ", 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

```
catn("hello", "world") # prints "hello world\n"
```

`catnn` *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.

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

```
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

```
catnn("hello", "world") # prints "hello\nworld\n"
```

`catsn` *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.

Warning: this function is deprecated and should no longer be used. The function is guaranteed to be available as part of the package until the end of 2023 but might removed at any time after 31.12.2023.

Usage

```
catsn(..., sep = " ", 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

```
catsn("hello", "world") # prints "hello world\n"
```

config_dir *Get 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 it
2. Else, if parameter `env_var` is a non-empty string, return it
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 `$WD/.config/{app-name}`

where `$WD` equals the current working directory and the notation `$VAR` is used to specify the value of a parameter or environment variable VAR.

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` *Get 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

Similar to `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 — *Get 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 `$WD/.local/share`

Value

Normalized path to the data directory of `$app_name`.

See Also

`config_dir()`, `xdg_data_home()`

Examples

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

`DOCSTRING_TEMPLATE` *Docstring Template*

Description

A minimal docstring template

Usage

```
DOCSTRING_TEMPLATE
```

Format

A single string, i.e. a character vector of length 1.

Examples

```
DOCSTRING_TEMPLATE
```

---

dput2 *Return ASCII representation of an R object*

Description

Like classic `dput()`, but instead of writing to stdout, the text representation is returned as string.

Usage

```
dput2(..., collapse = " ", trim = TRUE)
```

Arguments

`...`	Arguments passed on to `dput()`.
`collapse`	Character to use for collapsing the lines.
`trim`	If `TRUE`, leading and trailing whitespace from each line is cleared before the lines are collapsed and/or returned.

Value

If `collapse == '\n'`, a character vector of length 1. Else, a character vector of length `n`, where `n` corresponds to the number of lines outputted by classic `dput()`.

See Also

`dput()`

Examples

```
# Classic dput prints directly to stdout
x <- iris[1,]
dput(x)

# Traditional formatting using dput2
y <- dput2(x, collapse = "\n", trim = FALSE)
cat2(y)

# Single line formatting
z <- dput2(x)
cat2(z)
```

function_locals *Get Function Environment as List*

Description

Returns the function environment as list. Raises an error when called outside a function. By default, objects specified as arguments are removed from the environment.

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()

## End(Not run)
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 absolute, normalized project root directory as string. The forward slash is used as path separator (independent of the OS).

get_docstring *Get docstring for a Function*

Description

The roxygen2 package makes it possible to write documentation for R functions directly above the corresponding function. This function can be used to retrieve the full documentation string (docstring).

Usage

```
get_docstring(content, func, collapse = TRUE, template = DOCSTRING_TEMPLATE)
```

Arguments

`content`	R code as string.
`func`	Name of function to get docstring for.
`collapse`	Whether to collapse all docstring into a single string.
`template`	String to return in case no docstring could be found.

Value

A character vector of length 1 containing either the docstring or the empty string (in case no documentation could be detected).

Examples

```
uri <- system.file("testfiles/funcs.R", package = "toscutil")
content <- readLines(uri)
func <- "f2"
get_docstring(content, func)
get_docstring(content, func, collapse = TRUE)
```

get_formals *Get formals of a Function*

Description

Returns the arguments of a function from a valid R file.

Usage

```
get_formals(uri, content, func)
```

Arguments

`uri`	Path to R file.
`content`	R code as string.
`func`	Function name. If a function is defined multiple times inside the provided file, only the last occurence will be considered.

Value

A named character vector as returned by `formals()`.

Examples

```
uri <- system.file("testfiles/funcs.R", package = "toscutil")
content <- readLines(uri)
func <- "f2"
get_formals(uri, content, func)
```

`help2` *Return help for topic*

Description

This function returns the help text for the specified topic formatted either as plain text, html or latex.

Usage

```
help2(topic, format = c("text", "html", "latex"), package = NULL)
```

Arguments

`topic`	character, the topic for which to return the help text. See argument `topic` of function `help()` for details.
`format`	character, either `"text"`, `"html"` or `"latex"`
`package`	character, the package for which to return the help text. This argument will be ignored if topic is specified. Package must be attached to the search list first, e.g. by calling `library(package)`.

Details

This function was copied in part from: `https://www.r-bloggers.com/2013/06/printing-r-help-files-in-the-cons`

Examples

```
htm <- help2("sum", "html")
txt <- help2(topic = "sum", format = "text")
cat2(txt)
```

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 if else 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 *Truth checking as in Python*

Description

Returns TRUE if x is either FALSE, 0, NULL, NAand empty lists or an empty string. Inspired by python's bool.

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(1) # FALSE
is.none(NA) # TRUE
is.none(list()) # TRUE
is.none("") # TRUE
is.none(character()) # TRUE
is.none(numeric()) # TRUE
is.none(logical()) # TRUE
```

locals *Get specified Environment as List*

Description

Return all symbols in the specified 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` *Create 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_ms() # something like "2022-06-30, 07:14:26.82 CEST"
```

---

op-null-default *Return Default if None*

Description

Like `rlang::%||%()` but also checks for empty lists and empty strings.

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 predictions. If an element of the numeric vector has the name `"Intercept"` this element is treated as the intercept of the linear model.

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()
```

`split_docstring` *Split a docstring into a Head, Param and Tail Part*

Description

Split a docstring into a head, param and tail part.

Usage

```
split_docstring(docstring)
```

Arguments

`docstring`	Docstring of a R function as string, i.e. as character vector of length 1.

Value

List with 3 elements: head, param and tail.

Examples

```
uri <- system.file("testfiles/funcs.R", package = "toscutil")
func <- "f4"
content <- readLines(uri)
docstring <- get_docstring(content, func)
split_docstring(docstring)
```

`stub` *Stub Function Arguments*

Description

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

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 primitive 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)
```

`update_docstring` *Update docstring for a Function*

Description

The roxygen2 package makes it possible to write documentation for R functions directly above the corresponding function. This function can be used to update the parameter list of a documentation string (docstring) of a valid function of a valid R file. The update is done by comparing the currently listed parameters with the actual function parameters. Outdated parameters are removed and missing parameters are added to the docstring.

Usage

```
update_docstring(uri, func, content = NULL)
```

Arguments

`uri`	Path to R file.
`func`	Function name. If a function is defined multiple times inside the provided file, only the last occurence will be considered.
`content`	R code as string. If provided, uri is ignored.

Value

A character vector of length 1 containing the updated docstring.

Examples

```
uri <- system.file("testfiles/funcs.R", package = "toscutil")
func <- "f4"
update_docstring(uri, func)
```

`xdg_config_home` *Get 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 *Get 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