

# Package: toscutil (via r-universe)

June 29, 2024

**Title** Utility Functions

**Version** 2.8.0

**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.

**URL** <https://github.com/toscm/toscutil/>,  
<https://toscm.github.io/toscutil/>

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**Imports** utils, rlang, tools

**Suggests** devtools, languageserver, 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** 2024-06-28 10:40:06 UTC

## Contents

caller	2
capture.output2	3
cat0	4
cat0n	5
cat2	6
catf	7
catfn	8
catn	9

catnn . . . . .	9
catsn . . . . .	10
check_pkg_docs . . . . .	11
config_dir . . . . .	11
config_file . . . . .	13
corn . . . . .	14
data_dir . . . . .	15
DOCSTRING_TEMPLATE . . . . .	16
dput2 . . . . .	16
fg . . . . .	17
find_description_file . . . . .	18
function_locals . . . . .	18
getfd . . . . .	19
getpd . . . . .	20
get_docstring . . . . .	21
get_formals . . . . .	21
get_pkg_docs . . . . .	22
help2 . . . . .	23
home . . . . .	24
ifthen . . . . .	24
is.none . . . . .	25
locals . . . . .	25
named . . . . .	26
norm_path . . . . .	27
now . . . . .	27
op-null-default . . . . .	28
predict.numeric . . . . .	29
read_description_file . . . . .	29
rm_all . . . . .	30
split_docstring . . . . .	30
stub . . . . .	31
sys.exit . . . . .	32
trace_package . . . . .	32
untrace_package . . . . .	34
update_docstring . . . . .	35
xdg_config_home . . . . .	35
xdg_data_home . . . . .	36

<b>Index</b>	<b>38</b>
--------------	-----------

---

caller	<i>Get Name of Calling Function</i>
--------	-------------------------------------

---

### 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 by `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 <code>capture.output()</code> .
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 python's `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 <code>cat</code>
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	<i>Concatenate and Print</i>
-------	------------------------------

---

**Description**

Same as `cat` but with an additional argument `end`, which gets printed after all other elements. Inspired by python's `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 be removed at any time after 31.12.2023.

**Usage**

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

**Arguments**

...	objects passed on to <code>cat</code>
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 `base::cat()` but with an additional argument `end`, which gets printed after all other elements. Inspired by python's `print` command.

### Usage

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

### Arguments

<code>...</code>	R objects (see 'Details' for the types of objects allowed).
<code>sep</code>	a character vector of strings to append after each element.
<code>end</code>	a string to print after all other elements.
<code>file</code>	A <a href="#">connection</a> , or a character string naming the file to print to. If "" (the default), <code>cat2</code> prints to the standard output connection, the console unless redirected by <a href="#">sink</a> .
<code>append</code>	logical. Only used if the argument <code>file</code> is the name of file (and not a connection or " <code> cmd</code> "). If TRUE output will be appended to <code>file</code> ; otherwise, it will overwrite the contents of <code>file</code> .
<code>fill</code>	a logical or (positive) numeric controlling how the output is broken into successive lines. If FALSE (default), only newlines created explicitly by " <code>\n</code> " are printed. Otherwise, the output is broken into lines with print width equal to the option width if <code>fill</code> is TRUE, or the value of <code>fill</code> if this is numeric. Linefeeds are only inserted <i>between</i> elements, strings wider than <code>fill</code> are not wrapped. Non-positive <code>fill</code> values are ignored, with a warning.
<code>labels</code>	character vector of labels for the lines printed. Ignored if <code>fill</code> is FALSE.

### Value

No return value, called for side effects

### See Also

[base::cat\(\)](#)

## 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 `cat(sprintf(fmt, ...))`

## Usage

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

## Arguments

fmt	A character vector of format strings, each of up to 8192 bytes.
...	Up to 100 values to be passed into <code>fmt</code> . Only logical, integer, real and character vectors are supported, but some coercion will be done: see the Details section of <a href="#">base::sprintf()</a> .
file	A <a href="#">connection</a> , or a character string naming the file to print to. If "" (the default), <code>cat2</code> prints to the standard output connection, the console unless redirected by <a href="#">sink</a> .
append	logical. Only used if the argument <code>file</code> is the name of file (and not a connection or " cmd"). If TRUE output will be appended to <code>file</code> ; otherwise, it will overwrite the contents of <code>file</code> .
fill	a logical or (positive) numeric controlling how the output is broken into successive lines. If FALSE (default), only newlines created explicitly by "\n" are printed. Otherwise, the output is broken into lines with print width equal to the option width if <code>fill</code> is TRUE, or the value of <code>fill</code> if this is numeric. Linefeeds are only inserted <i>between</i> elements, strings wider than <code>fill</code> are not wrapped. Non-positive <code>fill</code> values are ignored, with a warning.
labels	character vector of labels for the lines printed. Ignored if <code>fill</code> is FALSE.

## Value

No return value, called for side effects

## Examples

```
catf("%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 be removed at any time after 31.12.2023.

### Usage

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

### Arguments

<code>fmt</code>	passed on to <code>base::sprintf()</code>
<code>...</code>	passed on to <code>base::sprintf()</code>
<code>end</code>	passed on to <code>cat2()</code>
<code>file</code>	passed on to <code>cat2()</code> (which passes it on to <code>base::cat()</code> )
<code>sep</code>	passed on to <code>cat2()</code> (which passes it on to <code>base::cat()</code> )
<code>fill</code>	passed on to <code>cat2()</code> (which passes it on to <code>base::cat()</code> )
<code>labels</code>	passed on to <code>cat2()</code> (which passes it on to <code>base::cat()</code> )
<code>append</code>	passed on to <code>cat2()</code> (which passes it on to <code>base::cat()</code> )

### Value

No return value, called for side effects

### Examples

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



---

catn	<i>Concatenate and Print</i>
------	------------------------------

---

**Description**

Same as `cat` but with an additional argument `end`, which gets printed after all other elements. Inspired by python's `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 be removed at any time after 31.12.2023.

**Usage**

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

**Arguments**

<code>...</code>	objects passed on to <code>cat</code>
<code>sep</code>	a character vector of strings to append after each element
<code>end</code>	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	<i>Concatenate and Print</i>
-------	------------------------------

---

**Description**

Same as `cat` but with an additional argument `end`, which gets printed after all other elements. Inspired by python's `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 be 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 python's `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 be 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"
```

---

`check_pkg_docs`*Check Documented Functions in a Package*

---

**Description**

Lists all documented functions in a package and checks their documentation elements for potential issues. The following checks are performed:

1. Title is present and doesn't start with regex "Function".
2. Description is present and doesn't start with "This function".
3. Value is present.
4. Example is present.

**Usage**

```
check_pkg_docs(pkg = NULL)
```

**Arguments**

`pkg` The package name. If `NULL`, the package name is inferred from the `DESCRIPTION` file in the current directory or any parent directory. If no `DESCRIPTION` file is found, the function stops with an error message.

**Value**

Returns a dataframe with columns `title`, `description`, `value`, `examples` and rows corresponding to the documented functions in the package. Each cell contains a string describing the check result for the corresponding documentation element of that function.

**Examples**

```
df <- check_pkg_docs("tools")
try(df <- check_pkg_docs())
```

---

`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	<i>Get Normalized Configuration File Path of a Program</i>
-------------	--

---

### 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**

<code>x</code>	matrix like object
<code>n</code>	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	<i>Get Normalized Data Directory Path of a Program</i>
----------	--

---

### 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**

```
cat(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**

<code>...</code>	Arguments passed on to <code>dput()</code> .
<code>collapse</code>	Character to use for collapsing the lines.
<code>trim</code>	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)
```

---

fg

*Foreground Color Codes*

---

**Description**

Provides ANSI escape codes as a named list for changing terminal text colors and style resetting. These codes can modify the foreground color and reset styles to defaults (incl. background color and text formatting).

**Usage**

fg

**Format**

A named list of ANSI escape codes for text coloring and style resetting in the terminal. Includes colors: GREY, RED, GREEN, YELLOW, BLUE, PURPLE, CYAN, WHITE, and RESET for default style restoration.

**Examples**

```
cat(fg$RED, "This text will be red.", fg$RESET, "\n")
cat(fg$GREEN, "This text will be green.", fg$RESET, "\n")
cat(fg$RESET, "Text back to default.", "\n")
```

---

find\_description\_file *Find DESCRIPTION File*

---

**Description**

Searches for a DESCRIPTION file starting from the current or specified directory and moving upwards through the directory hierarchy until the file is found or the root directory is reached.

**Usage**

```
find_description_file(start_dir = getwd())
```

**Arguments**

start\_dir        The starting directory for the search. Defaults to the current working directory.

**Value**

The path to the DESCRIPTION file if found. If not found, the function stops with an error message.

**Examples**

```
# Start search from a specific directory
graphics_pkg_dir <- system.file(package = "graphics")
find_description_file(graphics_pkg_dir)

## Not run:
# Below example will only work if executed from a package directory
find_description_file()

## End(Not run)
```

---

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

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

---

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).

## Examples

```
local({  
  base_pkg_root_dir <- system.file(package = "base")  
  base_pkg_R_dir <- file.path(base_pkg_root_dir, "R")  
  owd <- setwd(base_pkg_R_dir); on.exit(setwd(owd))  
  getpd()  
})
```

---

get_docstring	<i>Get docstring for a Function</i>
---------------	-------------------------------------

---

**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	<i>Get formals of a Function</i>
-------------	----------------------------------

---

**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 occurrence 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"
if (requireNamespace("languageserver", quietly = TRUE)) {
  get_formals(uri, content, func)
}
```

---

get\_pkg\_docs

*Get Documented Functions in a Package*


---

**Description**

Lists all documented functions in a package together with some of their documentation elements as raw text. Only works for installed packages.

**Usage**

```
get_pkg_docs(pkg = NULL, unload = TRUE, reload = TRUE)
```

**Arguments**

pkg	The package name. If NULL, the package name is inferred from the DESCRIPTION file in the current directory or any parent directory. If no DESCRIPTION file is found, the function stops with an error message.
unload	Whether to unload a potential currently developed package using <code>devtools::unload()</code> before checking the documentation. Required when the package was loaded with <code>devtools::load_all()</code> as the documentation database only exists for installed packages.
reload	Whether to reload the package using <code>devtools::load_all()</code> after checking the documentation.

**Value**

Returns a dataframe with columns `title`, `description`, `value`, `examples` and rows corresponding to the documented functions in the package.

**Examples**

```
df <- get_pkg_docs("tools")
nchars <- as.data.frame(apply(df, 2, function(col) sapply(col, nchar)))
head(nchars)
```

---

help2

*Return help for topic*

---

**Description**

Returns the help text for the specified topic formatted either as plain text, html or latex.

**Usage**

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

**Arguments**

topic	character, the topic for which to return the help text. See argument topic of function <a href="#">help()</a> 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 <code>library(package)</code> .

**Value**

The help text for the specified topic in the specified format as string.

**Examples**

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

---

home	<i>Get USERPROFILE or HOME</i>
------	--------------------------------

---

**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	<i>Shortcut for multiple if else statements</i>
--------	---

---

**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, NA and 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).

**Examples**

```
f <- function() {  
  x <- 1  
  y <- 2  
  z <- 3  
  locals()  
}  
ret <- f()  
stopifnot(identical(ret, list(z = 3, y = 2, x = 1)))
```

---

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)  
stopifnot(identical(l1, l2))  
l3 <- list(z = a, b = b)  
l4 <- named(z = a, b)  
stopifnot(identical(l3, l4))
```

---

norm_path	<i>Return Normalized Path</i>
-----------	-------------------------------

---

**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	<i>Get Current Date and Time as String</i>
-----	--

---

**Description**

Returns the current system time as a string in the format `YYYY-MM-DD hh:mm:ss[.XX][ TZ]`. Square brackets indicate optional parts of the string, 'XX' stands for milliseconds and 'TZ' for 'Timezone'.

**Usage**

```
now(usetz = TRUE, color = NULL, digits.sec = 0)
```

```
now_ms(usetz = TRUE, color = NULL, digits.sec = 2)
```

**Arguments**

usetz	Logical, indicating whether to include the timezone in the output.
color	Optional color to use for the timestamp. This parameter is only effective if the output is directed to a terminal that supports color, which is checked via <code>isatty(stdout())</code> .
digits.sec	Integer, the number of digits to include for seconds. Default is 0.

**Value**

For `now`, the current system time as a string in the format `YYYY-MM-DD hh:mm:ss TZ`. For `now_ms`, the format is `YYYY-MM-DD hh:mm:ss.XX TZ`, where `XX` represents milliseconds.

**See Also**

[Sys.time\(\)](#), [format.POSIXct\(\)](#)

**Examples**

```
now()                # "2021-11-27 19:19:31 CEST"
now_ms()            # "2022-06-30 07:14:26.82 CEST"
now(usetz = FALSE) # "2022-06-30 07:14:26.82"
now(color = fg$GREY) # "\033[1;30m2024-06-27 14:41:20 CEST\033[0m"
```

---

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	<i>Predict Method for Numeric Vectors</i>
-----------------	---

---

**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)
```

---

read_description_file	<i>Read DESCRIPTION File into a List</i>
-----------------------	--

---

**Description**

Reads the DESCRIPTION file of an R package and converts it into a list where each element corresponds to a field in the DESCRIPTION file.

**Usage**

```
read_description_file(p = NULL)
```

**Arguments**

p	The path to the DESCRIPTION file. If NULL, the function attempts to find the DESCRIPTION file by searching upwards from the current directory.
---	--

**Value**

A list where each element is a field from the DESCRIPTION file.

**Examples**

```
# Read DESCRIPTION file from a specific path
graphics_pkg_dir <- system.file(package = "graphics")
graphics_pkg_descfile <- find_description_file(graphics_pkg_dir)
desc_list <- read_description_file(graphics_pkg_descfile)
str(desc_list)

## Not run:
# Below example will only work if executed from a package directory
read_description_file()

## End(Not run)
```

---

rm_all	<i>Remove all objects from global environment</i>
--------	---

---

**Description**

Same as `rm(list=ls())`

**Usage**

```
rm_all()
```

**Value**

No return value, called for side effects

**Examples**

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

---

split_docstring	<i>Split a docstring into a Head, Param and Tail Part</i>
-----------------	---

---

**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	<i>Terminate a non-interactive R Session</i>
----------	--

---

### 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)
```

---

trace_package	<i>Traces function calls from a package</i>
---------------	---

---

### Description

Traces all function calls from a package and writes them to file with timestamps and callstack depth. Should always be used in combination with `untrace_package()` to untrace the package after use. For example `trace_package("graphics"); on.exit(untrace_package("graphics"))`. See examples for more details.



## Usage

```
trace_package(  
  pkg,  
  file = stdout(),  
  max = Inf,  
  funign = character(),  
  opsign = TRUE,  
  dotign = TRUE,  
  silent = TRUE,  
  exitmsg = "exit"  
)
```

## Arguments

pkg	Package name to trace.
file	File to write to.
max	Maximum depth of callstack to print.
funign	Functions to ignore.
opsign	Whether to ignore operators.
dotign	Whether to ignore functions starting with a dot.
silent	Whether to suppress messages.
exitmsg	Message to print on function exits.

## Details

Some function define their own `on.exit()` handlers with option `add = FALSE`. For those functions, exit tracing is impossible (as described in `trace()`). For now those functions have to be detected and ignored manually by the user using argument `funign`.

## Value

No return value, called for side effects

## See Also

[untrace\\_package\(\)](#)

## Examples

```
## Not run:  
# Trace all function from the graphics package, except for `plot.default`  
# as it defines its own on.exit() handler, i.e. exit tracing is impossible.  
local({  
  trace_package("graphics", funign = "plot.default")  
  on.exit(untrace_package("graphics"), add = TRUE)  
  plot(1:10)  
})  
  
## End(Not run)
```

---

untrace_package	<i>Untraces function calls from a package</i>
-----------------	---

---

### Description

Removes tracing from all function calls in a package that were previously traced by [trace\\_package\(\)](#).

### Usage

```
untrace_package(pkg)
```

### Arguments

pkg                    Package name to untrace.

### Details

This function reverses the effects of `trace_package` by removing all tracing from the specified package's functions.

### Value

No return value, called for side effects

### See Also

[trace\\_package\(\)](#)

### Examples

```
## Not run:
local({
  trace_package("graphics", funign = "plot.default")
  on.exit(untrace_package("graphics"), add = TRUE)
  plot(1:10)
})

## End(Not run)
```

---

update_docstring	<i>Update docstring for a Function</i>
------------------	--

---

### 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 occurrence 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	<i>Get XDG_CONFIG_HOME</i>
-----------------	----------------------------

---

### 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

*xdg\_data\_home*

37

**See Also**

[xdg\\_config\\_home\(\)](#)

**Examples**

`xdg_data_home()`

# Index

- \* **S3**
  - predict.numeric, 29
- \* **base**
  - capture.output2, 3
  - cat2, 6
  - catf, 7
  - dput2, 16
  - fg, 17
  - help2, 23
  - named, 26
  - norm\_path, 27
- \* **check**
  - ifthen, 24
  - is.none, 25
  - op-null-default, 28
- \* **depr**
  - cat0, 4
  - cat0n, 5
  - catfn, 8
  - catn, 9
  - catnn, 9
  - catsn, 10
- \* **doc**
  - check\_pkg\_docs, 11
  - DOCSTRING\_TEMPLATE, 16
  - find\_description\_file, 18
  - get\_docstring, 21
  - get\_formals, 21
  - get\_pkg\_docs, 22
  - read\_description\_file, 29
  - split\_docstring, 30
  - update\_docstring, 35
- \* **func**
  - caller, 2
  - function\_locals, 18
  - locals, 25
  - sys.exit, 32
- \* **live**
  - corn, 14
  - rm\_all, 30
  - stub, 31
  - trace\_package, 32
  - untrace\_package, 34
- \* **path**
  - config\_dir, 11
  - config\_file, 13
  - data\_dir, 15
  - getfd, 19
  - getpd, 20
  - home, 24
  - xdg\_config\_home, 35
  - xdg\_data\_home, 36
- \* **time**
  - now, 27
  - %none%(op-null-default), 28
- base::cat(), 6, 8
- base::sprintf(), 7, 8
- caller, 2
- capture.output(), 3, 4
- capture.output2, 3
- cat, 4, 5, 9, 10
- cat0, 4
- cat0n, 5
- cat2, 6
- cat2(), 8
- catf, 7
- catfn, 8
- catn, 9
- catnn, 9
- catsn, 10
- check\_pkg\_docs, 11
- config\_dir, 11
- config\_dir(), 14, 15
- config\_file, 13
- config\_file(), 12
- connection, 6, 7
- corn, 14

data\_dir, 15  
data\_dir(), 12  
devtools::load\_all(), 22  
devtools::unload(), 22  
DOCSTRING\_TEMPLATE, 16  
dput(), 16, 17  
dput2, 16  
  
fg, 17  
find\_description\_file, 18  
formals(), 22  
format.POSIXct(), 28  
function\_locals, 18  
  
get\_docstring, 21  
get\_formals, 21  
get\_pkg\_docs, 22  
getfd, 19  
getpd, 20  
  
head(), 14  
help(), 23  
help2, 23  
home, 24  
  
ifthen, 24  
is.none, 25  
is.none(), 28  
  
list(), 26  
locals, 25  
  
named, 26  
norm\_path, 27  
now, 27  
now\_ms(now), 27  
  
on.exit(), 33  
op-null-default, 28  
  
predict.numeric, 29  
  
read\_description\_file, 29  
rlang::%, 28  
rm\_all, 30  
  
sink, 6, 7  
split\_docstring, 30  
stub, 31  
sys.exit, 32  
  
Sys.time(), 28  
  
tail(), 14  
trace(), 33  
trace\_package, 32  
trace\_package(), 34  
  
untrace\_package, 34  
untrace\_package(), 32, 33  
update\_docstring, 35  
  
xdg\_config\_home, 35  
xdg\_config\_home(), 12, 14, 37  
xdg\_data\_home, 36  
xdg\_data\_home(), 15, 36