Package 'ympes'

March 9, 2023

Type Package

Title Collection of Helper Functions

Version 0.4.0

Description Provides a collection of lightweight helper functions (imps) both for interactive use and for inclusion within other packages. These include minimal assertion functions with a focus on informative error messaging for both missing and incorrect function arguments as well as other functions for visualising colour palettes, quoting user input and searching rows of a data frame.

License GPL-2

Encoding UTF-8

RoxygenNote 7.2.3

Suggests clipr, tinytest

URL https://github.com/TimTaylor/ympes

BugReports https://github.com/TimTaylor/ympes/issues

Imports utils

NeedsCompilation no

Author Tim Taylor [aut, cre, cph] (<https://orcid.org/0000-0002-8587-7113>)

Maintainer Tim Taylor <tim.taylor@hiddenelephants.co.uk>

Repository CRAN

Date/Publication 2023-03-09 17:30:03 UTC

R topics documented:

assertions *Argument assertions*

Description

Assertions for function arguments. Motivated by `vctrs::vec_assert()` but with lower overhead at a cost of less informative error messages. Designed to make it easy to identify the top level calling function whether used within a user facing function or internally.

Usage

```
assert_integer(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_int(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_double(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_dbl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_numeric(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_num(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_logical(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_lgl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_character(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_chr(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_data_frame(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_list(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_integer(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_int(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_double(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_dbl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_numeric(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_num(x, arg = deparse(substitute(x)), call = sys.call(-1L))
```

```
assert_scalar_logical(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_lgl(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_bool(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_boolean(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_character(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_scalar_chr(x, arg = deparse(substitute(x)), call = sys.call(-1L))

assert_string(x, arg = deparse(substitute(x)), call = sys.call(-1L))
```

Arguments

x	Argument to check.
arg	[character]
	Name of argument being checked (used in error message).
call	[call]
	Call to use in error message.

Value

NULL if the assertion succeeds (error otherwise).

Examples

```
# Use in a user facing function
fun <- function(i, d, l, chr, b) {
    assert_scalar_int(i)
    TRUE
}
fun(i=1L)
try(fun())
try(fun(i="cat"))

# Use in an internal function
internal_fun <- function(a) {
    assert_string(a, arg = deparse(substitute(a)), call = sys.call(-1L))
    TRUE
}
external_fun <- function(b) {
    internal_fun(a=b)
}
external_fun(b="cat")
try(external_fun())
try(external_fun(b = letters))
```

cc *Quote names*

Description

`cc()` quotes comma separated names whilst trimming outer whitespace. It is intended for interactive use only.

Usage

```
cc(..., .clip = getOption("imp.clipboard", FALSE))
```

Arguments

`...`	Unquoted names (separated by commas) that you wish to quote. Empty arguments (e.g. third item in `one,two,,four`) will be returned as `""`.
`.clip`	`[bool]` Should the code to generate the constructed character vector be copied to your system clipboard. Defaults to FALSE unless the option "imp.clipboard" is set to TRUE. Note that copying to clipboard requires the availability of package clipr.

Value

A character vector of the quoted input.

Examples

```
cc(dale, audrey, laura, hawk)
```

greprows *Pattern matching on data frame rows*

Description

`greprows()` searches for pattern matches within a data frames columns and returns the related rows or row indices.

Usage

```
greprows(
  dat,
  pattern,
  cols = NULL,
  value = TRUE,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  invert = FALSE
)
```

Arguments

dat	Data frame
pattern	character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced by as.character to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are allowed except for regexpr, gregexpr and regexec.
cols	[character] Character vector of columns to search. If NULL (default) all character and factor columns will be searched.
value	[logical] Should a data frame of rows be returned. If FALSE row indices will be returned instead of the rows themselves.
ignore.case	if FALSE, the pattern matching is *case sensitive* and if TRUE, case is ignored during matching.
perl	logical. Should Perl-compatible regexps be used?
fixed	logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.
invert	logical. If TRUE return indices or values for elements that do *not* match.

Value

A data frame of the corresponding rows or, if value = FALSE, the corresponding row numbers.

See Also

grep

Examples

```
dat <- data.frame(
    first = letters,
    second = factor(rev(LETTERS)),
```

```
    third = "Q"
)
greprows(dat, "A|b")
greprows(dat, "A|b", ignore.case = TRUE)
greprows(dat, "c", value = FALSE)
```

---

new_package *Create a package skeleton*

Description

new_package() create a package skeleton based on my preferred folder structure.

Usage

```
new_package(
  name = "mypackage",
  dir = ".",
  firstname = getOption("ympes.firstname", "Joe"),
  surname = getOption("ympes.surname", "Bloggs"),
  email = getOption("ympes.email", "Joe.Bloggs@missing.com"),
  orcid = getOption("ympes.orcid", default = NULL),
  enter = TRUE
)
```

Arguments

name	[character]
	Package name
dir	[character]
	Directory to start in.
firstname	[character]
	Maintainer's firstname.
surname	[character]
	Maintainer's surname.
email	[character]
	Maintainer's email address.
orcid	[character]
	Maintainer's ORCID.
enter	[bool]
	Should you move in to the package directory after creation.
	Only applicable in interactive sessions.

Value

Created directory (invisibly)

Examples

```
# usage without entering directory
p <- new_package("my_package_1", dir = tempdir(), enter = FALSE)

# clean up
unlink(p, recursive = TRUE)
```

`plot_palette` *Plot a colour palette*

Description

`plot_palette()` plots a palette from a vector of colour values (name or hex).

Usage

```
plot_palette(values, label = TRUE, square = FALSE)
```

Arguments

`values`	`[character]`
	Vector of named or hex colours.
`label`	`[bool]`
	Do you want to label the plot or not?
	If `values` is a named vector the names are used for labels, otherwise, the values.
`square`	`[bool]`
	Display palette as square?

Value

The input (invisibly).

Examples

```
plot_palette(c("#5FE756", "red", "black"))
plot_palette(c("#5FE756", "red", "black"), square=TRUE)
```

Index