Package 'ympes'

May 24, 2024

Type Package

Title Collection of Helper Functions

Version 1.2.0

Description Provides a collection of lightweight helper functions (imps) both for interactive use and for inclusion within other packages. These include functions for minimal input assertions, visualising colour palettes, quoting user input, searching rows of a data frame and capturing string tokens.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.1

Suggests clipr, tinytest

URL https://sr.ht/~tim-taylor/ympes/

BugReports https://lists.sr.ht/~tim-taylor/ympes

Imports methods, utils

NeedsCompilation no

Author Tim Taylor [aut, cre, cph] (<https://orcid.org/0000-0002-8587-7113>),
R Core Team [cph] (fstrcapture uses code from strcapture),
Toby Hocking [cph] (fstrcapture uses code from nc::capture_first_vec)

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

Repository CRAN

Date/Publication 2024-05-24 12:50:02 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_integer_not_na(
  x,
  arg = deparse(substitute(x)),
  call = sys.call(-1L)
)

.assert_scalar_int_not_na(
```

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

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

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

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

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

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

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

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

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

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

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

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

.assert_negative_or_na(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)
```

`fstrcapture` *Capture string tokens into a data frame*

Description

`fstrcapture()` is a more efficient alternative for `strcapture()` when using Perl-compatible regular expressions

Usage

```
fstrcapture(x, pattern, proto)
```

Arguments

`x`	A character vector in which to capture the tokens.
`pattern`	The regular expression with the capture expressions.
`proto`	A `data.frame` or S4 object that behaves like one. See details.

Value

A tabular data structure of the same type as proto, so typically a data.frame, containing a column for each capture expression. The column types are inherited from proto, as are the names unless the captures themselves are named (in which case these are prioritised). Cases in x that do not match the pattern have NA in every column.

See Also

`strcapture()`.

Examples

```
# from regexpr example ----------------------------------------------

# if named capture then pass names on irrespective of proto
notables <- c("  Ben Franklin and Jefferson Davis", "\tMillard Fillmore")
pattern <- "(?<first>[[:upper:]][[:lower:]]+) (?<last>[[:upper:]][[:lower:]]+)"
proto <- data.frame(a="", b="")
fstrcapture(notables, pattern, proto)
```

```
# from strcapture example -------------------------------------------
# if unnamed capture then proto names used
x <- "chr1:1-1000"
pattern <- "(.*?):([[:digit:]]+)-([[:digit:]]+)"
proto <- data.frame(chr=character(), start=integer(), end=integer())
fstrcapture(x, pattern, proto)

# if no proto supplied then all captures treated as character
str(fstrcapture(x, pattern))
str(fstrcapture(x, pattern, proto))
```

---

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
)
```

```
np(
  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