Package 'ympes'

April 14, 2025

Type Package

Title Collection of Helper Functions

Version 1.9.0

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 7.3.2

Suggests tinytest, litedown, clipr

URL https://timtaylor.github.io/ympes/

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

Depends R (>= 3.5.0)

Imports graphics, grDevices, methods, utils

VignetteBuilder litedown

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 2025-04-14 11:00:02 UTC

Contents

`assertions` *Argument assertions (Experimental)*

Description

Assertions for function arguments. Motivated by assertions from the vctrs package 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. They are somewhat experimental in nature and should be treated accordingly.

Usage

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

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

assert_integer_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_int_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_double(
```

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

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

assert_double_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_dbl_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_dbl_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)
```

```
assert_numeric_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_num_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_logical_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_lgl_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_character_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_chr_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_whole(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_integerish(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
```

```
)

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

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

assert_scalar_integer_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_scalar_int_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

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

```
  .subclass = NULL
)

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

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

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

assert_scalar_numeric_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_scalar_lgl(
  x,
  .arg = deparse(substitute(x)),
```

```
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_scalar_logical_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_scalar_lgl_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_scalar_whole(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

assert_scalar_integerish(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

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

assert_scalar_character(
  x,
```

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

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

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

assert_scalar_character_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

assert_string_not_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)

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

assert_non_positive_or_na(
```

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

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

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

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

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

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

assert_negative_or_na(
  x,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)
```

```
assert_between(
  x,
  lower = -Inf,
  upper = Inf,
  left_inclusive = TRUE,
  right_inclusive = TRUE,
  .arg = deparse(substitute(x)),
  .call = sys.call(-1L),
  .subclass = NULL
)
```

Arguments

x	Argument to check.
.arg	[character]
	Name of argument being checked (used in error message).
.call	[call]
	Call to use in error message.
.subclass	[character]
	The (optional) subclass of the returned error condition.
lower	[numeric]
	The lower bound to compare against.
upper	[numeric]
	The upper bound to compare against.
left_inclusive	[bool]
	Should the left (lower) bound be compared inclusively (<=) or exclusive (<).
right_inclusive	
	[bool]
	Should the right (upper) bound be compared inclusively (>=) or exclusive (>).

Value

If the assertion succeeds then the input is returned invisibly.

Otherwise and erro (with optional subclass if supplied when calling).

Examples

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

# Use in an internal function
internal_fun <- function(a) {
    assert_string(
```

```
        a,
        .arg = deparse(substitute(x)),
        .call = sys.call(-1L),
        .subclass = "example_error"
    )
    TRUE
}
external_fun <- function(b) {
    internal_fun(a=b)
}
external_fun(b="cat")
try(external_fun(b = letters))
tryCatch(external_fun(b = letters), error = class)
```

---

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

`...`	Either unquoted names (separated by commas) that you wish to quote or a length one character vector you wish to split by whitespace. Empty arguments (e.g. third item in `one,two,,four`) will be returned as `""`. Character vectors not of length one are returned as is.
`.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)
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. It is underpinned by the `regexpr()` function. Whilst `fstrcapture()` only returns the first occurrence of the captures in a string, `gstrcapture()`, built upon `gregexpr()`, will return all.

Usage

```
fstrcapture(x, pattern, proto)

gstrcapture(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. For `gstrcapture()` there is an additional column, `string_id`, which links the output to the relevant element of the input vector.

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)
gstrcapture(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.

grepvrows() is identical to greprows() except with the default value = TRUE.

greplrows() returns a logical vector (match or not for each row of dat).

Usage

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

greplrows(
  dat,
  pattern,
  cols = NULL,
  ignore.case = FALSE,
  perl = FALSE,
  fixed = FALSE,
  invert = FALSE
)

grepvrows(
  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 (defauly) 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_name *Generate column names for a data frame*

Description

new_name() generates unique names for additional data frame variables ensuring they are not already present.

Usage

```
new_name(x, n = 1L)
```

Arguments

x	A data frame.
n	Number of unique names to generate.

Value

A character vector of unique names not already found in x.

Examples

```
new_name(mtcars)
new_name(mtcars, 2)
```

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