

Package: ympes (via r-universe)

October 31, 2024

Type Package

Title Collection of Helper Functions

Version 1.6.0

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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests tinytest, litedown

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

Config/Needs/check yihui/litedown

Repository <https://timtaylor.r-universe.dev>

RemoteUrl <https://github.com/TimTaylor/ympes>

RemoteRef HEAD

RemoteSha b025f59c6c2191e8a128ca549a95cba4632a28d2

Contents

assertions	2
cc	12
fstrcapture	13

greprows	14
new_name	16
new_package	16
plot_palette	18

Index	19
--------------	-----------

assertions	<i>Argument assertions (Experimental)</i>
------------	---

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. 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  
)
```

```
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

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

Value

NULL if the assertion succeeds.

Otherwise an error of class "ympes-error" (with optional subclass if supplied when calling the assertion).

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 = 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] Not currently used.

Value

A character vector of the quoted input.

Examples

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

fstrcapture	<i>Capture string tokens into a data frame</i>
-------------	--

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.

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

<code>dat</code>	Data frame
<code>pattern</code>	character string containing a regular expression (or character string for <code>fixed = TRUE</code>) 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 <code>regexpr</code> , <code>gregexpr</code> and <code>regexec</code> .
<code>cols</code>	[character] Character vector of columns to search. If NULL (default) all character and factor columns will be searched.
<code>value</code>	[logical] Should a data frame of rows be returned. If FALSE (default) row indices will be returned instead of the rows themselves.
<code>ignore.case</code>	if FALSE, the pattern matching is <i>case sensitive</i> and if TRUE, case is ignored during matching.
<code>perl</code>	logical. Should Perl-compatible regexps be used?
<code>fixed</code>	logical. If TRUE, <code>pattern</code> is a string to be matched as is. Overrides all conflicting arguments.
<code>invert</code>	logical. If TRUE return indices or values for elements that do <i>not</i> 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	<i>Generate column names for a data frame</i>
----------	---

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)
```

new_package	<i>Create a package skeleton (Experimental)</i>
-------------	---

Description

new_package() create a package skeleton based on my preferred folder structure. It is somewhat experimental in nature and should be treated accordingly.

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	<i>Plot a colour palette</i>
--------------	------------------------------

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

as.character, [15](#)
assert_between (assertions), [2](#)
assert_bool (assertions), [2](#)
assert_boolean (assertions), [2](#)
assert_character (assertions), [2](#)
assert_character_not_na (assertions), [2](#)
assert_chr (assertions), [2](#)
assert_chr_not_na (assertions), [2](#)
assert_data_frame (assertions), [2](#)
assert_dbl (assertions), [2](#)
assert_dbl_not_na (assertions), [2](#)
assert_double (assertions), [2](#)
assert_double_not_na (assertions), [2](#)
assert_int (assertions), [2](#)
assert_int_not_na (assertions), [2](#)
assert_integer (assertions), [2](#)
assert_integer_not_na (assertions), [2](#)
assert_integerish (assertions), [2](#)
assert_lgl (assertions), [2](#)
assert_lgl_not_na (assertions), [2](#)
assert_list (assertions), [2](#)
assert_logical (assertions), [2](#)
assert_logical_not_na (assertions), [2](#)
assert_negative (assertions), [2](#)
assert_negative_or_na (assertions), [2](#)
assert_non_negative (assertions), [2](#)
assert_non_negative_or_na (assertions),
[2](#)
assert_non_positive (assertions), [2](#)
assert_non_positive_or_na (assertions),
[2](#)
assert_num (assertions), [2](#)
assert_num_not_na (assertions), [2](#)
assert_numeric (assertions), [2](#)
assert_numeric_not_na (assertions), [2](#)
assert_positive (assertions), [2](#)
assert_positive_or_na (assertions), [2](#)
assert_scalar_character (assertions), [2](#)
assert_scalar_character_not_na
(assertions), [2](#)
assert_scalar_chr (assertions), [2](#)
assert_scalar_chr_not_na (assertions), [2](#)
assert_scalar_dbl (assertions), [2](#)
assert_scalar_dbl_not_na (assertions), [2](#)
assert_scalar_double (assertions), [2](#)
assert_scalar_double_not_na
(assertions), [2](#)
assert_scalar_int (assertions), [2](#)
assert_scalar_int_not_na (assertions), [2](#)
assert_scalar_integer (assertions), [2](#)
assert_scalar_integer_not_na
(assertions), [2](#)
assert_scalar_integerish (assertions), [2](#)
assert_scalar_lgl (assertions), [2](#)
assert_scalar_lgl_not_na (assertions), [2](#)
assert_scalar_logical (assertions), [2](#)
assert_scalar_logical_not_na
(assertions), [2](#)
assert_scalar_num (assertions), [2](#)
assert_scalar_num_not_na (assertions), [2](#)
assert_scalar_numeric (assertions), [2](#)
assert_scalar_numeric_not_na
(assertions), [2](#)
assert_scalar_whole (assertions), [2](#)
assert_string (assertions), [2](#)
assert_string_not_na (assertions), [2](#)
assert_whole (assertions), [2](#)
assertions, [2](#)
cc, [12](#)
fstrcapture, [13](#)
grep(), [15](#)
greplrows (greprows), [14](#)
greprows, [14](#)
grepvrows (greprows), [14](#)
new_name, [16](#)

`new_package`, [16](#)
`np (new_package)`, [16](#)
`plot_palette`, [18](#)
regular expression, [15](#)
`strcapture()`, [13](#)