

Package ‘fuj’

October 22, 2023

Type Package

Title Functions and Utilities for Jordan

Version 0.1.4

Maintainer Jordan Mark Barbone <jmbarbone@gmail.com>

Description Provides core functions and utilities for packages and other code developed by Jordan Mark Barbone.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Depends R (>= 3.6)

Suggests covr, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

URL <https://jmbarbone.github.io/fuj/>, <https://github.com/jmbarbone/fuj>

BugReports <https://github.com/jmbarbone/fuj/issues>

NeedsCompilation no

Author Jordan Mark Barbone [aut, cph, cre]
(<<https://orcid.org/0000-0001-9788-3628>>)

Repository CRAN

Date/Publication 2023-10-22 09:50:10 UTC

R topics documented:

alias_arithmetic	2
alias_extract	2
collapse	3
colons	4
exattr	5
flip	5

if_null	6
list0	7
match_ext	8
muffle	9
names	10
new_condition	11
os	12
quick_df	12
require_namespace	13
struct	14
verbose	15

Index	17
--------------	-----------

alias_arithmetic	<i>Arithmetic wrappers</i>
------------------	----------------------------

Description

Arithmetic wrappers

Value

See [base::Arithmetic](#)

Examples

```

add(7, 2) # +
subtract(7, 2) # -
multiply(7, 2) # *
divide(7, 2) # /
raise_power(7, 2) # ^
remainder(7, 2) # %%
divide_int(7, 2) # %/%

```

alias_extract	<i>Extract and replace aliases</i>
---------------	------------------------------------

Description

Extract and replace aliases

Value

See [base::Extract](#)

Examples

```
df <- quick_df1(a = 1:5, b = 6:10)
# alias of `[`
subset1(df, 1)
subset1(df, 1, )
subset1(df, , 1)
subset1(df, , 1, drop = FALSE)

# alias of `[[`
subset2(df, 1)
subset2(df, 1, 2)

# alias of `$`
subset3(df, a)
subset3(df, "b")
subset3(df, "foo")

# alias of `[<-`
subassign1(df, "a", , 2)
```

collapse

Collapse

Description

Simple wrapper for concatenating strings

Usage

```
collapse(..., sep = "")
```

Arguments

... one or more R objects, to be converted to character vectors.
sep a character string to separate the terms. Not [NA_character_](#).

Value

A character vector of concatenated values. See [base::paste](#) for more details.

Examples

```
collapse(1:10)
collapse(list("a", b = 1:2))
collapse(quick_df1(a = 1:3, b = 4:6), sep = "-")
```

colons

Colons

Description

Get an object from a namespace

Usage

```
package %::% name
```

```
package %:::% name
```

```
package %colons% name
```

Arguments

package	Name of the package
---------	---------------------

name	Name to retrieve
------	------------------

Details

The functions mimic the use of `::` and `:::` for extracting values from namespaces. `%colons%` is an alias for `%::%`.

Value

The variable name from package `package`

WARNING

To reiterate from other documentation: it is not advised to use `:::` in your code as it will retrieve non-exported objects that may be more likely to change in their functionality than exported objects.

See Also

```
help("::")
```

Examples

```
identical("base" %::% "mean", base::mean)
"fuj" %:::% "colons_example" # unexported value
```

exattr	<i>Exact attributes</i>
--------	-------------------------

Description

Get the exact attributes of an object

Usage

```
exattr(x, which)
```

```
x %attr% which
```

Arguments

x an object whose attributes are to be accessed.

which a non-empty character string specifying which attribute is to be accessed.

Value

See [base::attr](#)

Examples

```
foo <- struct(list(), "foo", aa = TRUE)
attr(foo, "a") # TRUE : partial match successful
exattr(foo, "a") # NULL : partial match failed
exattr(foo, "aa") # TRUE : exact match
```

flip	<i>Flip</i>
------	-------------

Description

Flip an object.

Usage

```
flip(x, ...)
```

```
## Default S3 method:
```

```
flip(x, ...)
```

```
## S3 method for class 'matrix'
```

```
flip(x, by = c("rows", "columns"), keep_rownames = NULL, ...)
```

```
## S3 method for class 'data.frame'
```

```
flip(x, by = c("rows", "columns"), keep_rownames = NULL, ...)
```

Arguments

x	An object
...	Additional arguments passed to methods
by	Flip by "rows" or "columns" (partial matches accepted)
keep_rownames	Logical, if TRUE will not reset row names; NULL

Value

A vector of values, equal length of x that is reversed or a data frame with flipped rows/columns

Examples

```
flip(letters[1:3])
flip(seq.int(9, -9, by = -3))
flip(head(iris))
flip(head(iris), keep_rownames = TRUE)
flip(head(iris), by = "col")
```

if_null

Default value for NULL or no length

Description

Replace if NULL or not length

Usage

```
x %||% y
```

```
x %|||% y
```

```
x %len% y
```

Arguments

x, y	If x is NULL returns y; otherwise x
------	-------------------------------------

Details

A mostly copy of rlang's %||% except does not use rlang::is_null(), which, currently, calls the same primitive [base::is.null](#) function.

Value

x if it is not NULL or has length, depending on check

Examples

```
# replace NULL
NULL %||% 1L
2L %||% 1L

# replace empty
"" %|||% 1L
NA %|||% 1L
double() %|||% 1L
NULL %|||% 1L

# replace no length
logical() %len% TRUE
```

list0

Listing for dots

Description

Tries to not complain about empty arguments

Usage

```
list0(...)
```

Arguments

... Arguments to collect in a list

Value

A list of ...

Examples

```
try(list(1, ))
list0(1, )
list0(a = 1, )
```

`match_ext`*Value matching - Extensions*

Description

Non matching alternatives and supplementary functions.

Usage

```
is_in(x, table)
```

```
is_out(x, table)
```

```
x %out% table
```

```
is_within(x, table)
```

```
x %wi% table
```

```
is_without(x, table)
```

```
x %wo% table
```

```
no_match(x, table)
```

```
any_match(x, table)
```

Arguments

`x` vector or NULL: the values to be matched. [Long vectors](#) are supported.

`table` vector or NULL: the values to be matched against. [Long vectors](#) are not supported.

Details

Contrast with [base::match\(\)](#), [base::intersect\(\)](#), and [base::%in%\(\)](#). The functions of [%wi%](#) and [%wo%](#) can be used in lieu of [base::intersect\(\)](#) and [base::setdiff\(\)](#). The primary difference is that the base functions return only unique values, which may not be a desired behavior.

Value

- [%out%](#): A logical vector of equal length of `x`, `table`
- [%wo%](#), [%wi%](#): A vector of values of `x`
- [any_match\(\)](#), [no_match\(\)](#): TRUE or FALSE
- [is_in\(\)](#): see [base::%in%\(\)](#)

Examples

```

1:10 %in% c(1, 3, 5, 9)
1:10 %out% c(1, 3, 5, 9)
letters[1:5] %wo% letters[3:7]
letters[1:5] %wi% letters[3:7]

# base functions only return unique values

      c(1:6, 7:2) %wo% c(3, 7, 12) # -> keeps duplicates
setdiff(c(1:6, 7:2),      c(3, 7, 12)) # -> unique values

      c(1:6, 7:2) %wi% c(3, 7, 12) # -> keeps duplicates
intersect(c(1:6, 7:2),      c(3, 7, 12)) # -> unique values

```

muffle

Muffle messages

Description

Aliases for `base::suppressMessages()` and `base::suppressWarnings()`

Usage

```
muffle(expr, fun, classes = "message")
```

```
wuffle(expr, fun, classes = "warning")
```

Arguments

expr	An expression to evaluate
fun	A function to <i>muffle</i> (or <i>wuffle</i>)
classes	A character vector of classes to suppress

Value

The result of `expr` or a function wrapping `fun`

Examples

```

# load function
foo <- function(...) {
  message("You entered :", paste0(...))
  c(...)
}

# wrap around function or muffle the function ti's

```

```
muffle(foo(1, 2))
muffle(fun = foo)(1, 2)
sapply(1:3, muffle(fun = foo))

# silence warnings
wuffle(as.integer("a"))
sapply(list(1, "a", "0", ".2"), wuffle(fun = as.integer))
```

names	<i>Set names</i>
-------	------------------

Description

Sets or removes names

Usage

```
set_names(x, nm = x)

remove_names(x)

x %names% nm

is_named(x, zero_ok = TRUE)
```

Arguments

x	A vector of values
nm	A vector of names
zero_ok	If TRUE allows use of "" as a <i>special</i> name

Value

x with nm values assigned to names (if x is NULL, NULL is returned)

Examples

```
set_names(1:5)
set_names(1:5, c("a", "b", "c", "d", "e"))

x <- c(a = 1, b = 2)
remove_names(x)
x %names% c("c", "d")
is_named(x)
```

new_condition	<i>New condition</i>
---------------	----------------------

Description

Template for a new condition. See more at [base::conditions](#)

Usage

```
new_condition(  
  msg = "",  
  class = NULL,  
  call = NULL,  
  type = c("error", "warning", NA_character_),  
  message = msg,  
  pkg = TRUE  
)
```

Arguments

msg, message	A message to print
class	Character string of a single condition class
call	A call expression
type	The type (additional class) of condition: either "error", "warning" or NA, which is treated as NULL
pkg	Control or adding package name to condition. If TRUE will try to get the current package name (via <code>.packageName</code>) from, presumably, the developmental package. If FALSE, no package name is prepended to the condition class as a new class. Otherwise, a package can be explicitly set with a single length character.

Details

The use of `.packageName` when `pkg = TRUE` may not be valid during active development. When the attempt to retrieve the `.packageName` object is unsuccessful, the error is quietly ignored. However, this should be successful once the package is build and functions can then utilize this created object.

Value

A condition with the classes specified from `class` and `type`

Examples

```
# empty condition  
x <- new_condition("informative error message", class = "foo")  
try(stop(x))  
  
# with pkg
```

```
x <- new_condition("msg", class = "foo", pkg = "bar")
# class contains multiple identifiers, including a "bar:fooError"
class(x)
# message contains package information at the end
try(stop(x))
```

os

Determine operating systems

Description

Determine operating systems

Usage

```
is_windows()
```

```
is_macos()
```

```
is_linux()
```

Value

TRUE or FALSE

Examples

```
is_windows()
is_macos()
is_linux()
```

quick_df

Quick DF

Description

This is a speedier implementation of `as.data.frame()` but does not provide the same sort of checks. It should be used with caution.

Usage

```
quick_df(x = NULL)
```

```
empty_df()
```

```
quick_dfl(...)
```

Arguments

x A list or NULL (see return)
 ... Columns as tag = value (passed to list())

Value

A data.frame; if x is NULL a data.frame with 0 rows and 0 columns is returned (similar to calling data.frame() but faster). empty_df() returns a data.frame with 0 rows and 0 columns.

Examples

```
# unnamed will use make.names()
x <- list(1:10, letters[1:10])
quick_df(x)

# named is preferred
names(x) <- c("numbers", "letters")
quick_df(x)

# empty data.frame
empty_df() # or quick_df(NULL)
```

require_namespace	<i>Require namespace</i>
-------------------	--------------------------

Description

Require namespace

Usage

```
require_namespace(package, ...)
```

Arguments

package, ... Package names

Value

TRUE (invisibly) if found; otherwise errors

Examples

```
isTRUE(require_namespace("base")) # returns invisibly
try(require_namespace("1package")) # (using a purposefully bad name)
```

struct *Simple structures*

Description

Create simple structures

Usage

```
struct(x, class, ..., .keep_attr = FALSE)
```

Arguments

x	An object; if NULL, coerced to <code>list()</code>
class	A vector of classes; can also be NULL
...	Named attributes to set to x; overwrites any attributes in x even if defined in <code>.keep_attr</code>
.keep_attr	Control for keeping attributes from x: TRUE will retain all attributes from x; a character vector will pick out specifically defined attributes to retain; otherwise only attributes defined in ... will be used

Details

Unlike `base::structure()` this does not provide additional checks for special names, performs no `base::storage.mode()` conversions for factors (x therefor has to be an integer), attributes from x are not retained, and class is specified outside of other attributes and assigned after `base::attributes()` is called.

Essentially, this is just a wrapper for calling `base::attributes()` then `base::class()`.

Note that `base::structure()` provides a warning when the first argument is NULL. `struct()` does not. The coercion from NULL to `list()` is done, and documented, in `base::attributes()`.

Value

An object with class defined as class and attributes ...

Examples

```
x <- list(a = 1, b = 2)
# structure() retains the $names attribute of x but struct() does not
structure(x, class = "data.frame", row.names = 1L)
struct(x, "data.frame", row.names = 1L)
struct(x, "data.frame", row.names = 1L, names = names(x))

# structure() corrects entries for "factor" class
# but struct() demands the data to be an integer
structure(1, class = "factor", levels = "a")
try(struct(1, "factor", levels = "a"))
```

```

struct(1L, "factor", levels = "a")

# When first argument is NULL -- attributes() coerces
try(structure(NULL)) # NULL, no call to attributes()
struct(NULL, NULL) # list(), without warning
x <- NULL
attributes(x) <- NULL
x # NULL
attributes(x) <- list() # struct() always grabs ... into a list
x # list()

# Due to the use of class() to assign class, you may experience some
# other differences between structure() and struct()
x <- structure(1, class = "integer")
y <- struct(1, "integer")
str(x)
str(y)

all.equal(x, y)

# Be careful about carrying over attributes
x <- quick_df(list(a = 1:2, b = 3:4))
# returns empty data.frame
struct(x, "data.frame", new = 1)

# safely changing names without breaking rownames
struct(x, "data.frame", names = c("c", "d")) # breaks
struct(x, "data.frame", names = c("c", "d"), .keep_attr = TRUE)
struct(x, "data.frame", names = c("c", "d"), .keep_attr = "row.names")

# safely adds comments
struct(x, "data.frame", comment = "hi", .keep_attr = TRUE)
struct(x, "data.frame", comment = "hi", .keep_attr = c("names", "row.names"))

# assignment in ... overwrites attributes
struct(x, "data.frame", names = c("var1", "var2"), .keep_attr = TRUE)

```

 verbose

Verbose

Description

Simple verbose condition handling

Usage

```
verbose(...)
```

Arguments

... A message to display. When ... is NULL (and only NULL), no message will display.

Details

`verbose()` can be safely placed in scripts to signal additional message conditions. `verbose()` can be controlled with `options(verbose)` (the default) and an override, `options(fuj.verbose)`. The latter can be set to a function whose result will be used for conditional evaluation.

Value

None, called for its side-effects. When conditions are met, will signal a `verboseMessage` condition.

Examples

```
op <- options(verbose = FALSE)
verbose("will not show")

options(verbose = TRUE)
verbose("message printed")
verbose("multiple lines ", "will be ", "combined")
options(op)

op <- options(fuj.verbose = function() TRUE)
verbose("function will evaluate")
verbose(NULL) # nothing
verbose(NULL, "something")
verbose(if (FALSE) {
  "`if` returns `NULL` when not `TRUE`, which makes for additional control"
})
options(op)
```

Index

`%::%` (colons), 4
`%::%` (colons), 4
`%attr%` (exattr), 5
`%colons%` (colons), 4
`%len%` (if_null), 6
`%names%` (names), 10
`%out%` (match_ext), 8
`%wi%` (match_ext), 8
`%wo%` (match_ext), 8

`add` (alias_arithmetic), 2
`alias_arithmetic`, 2
`alias_extract`, 2
`any_match` (match_ext), 8

`base::%in%`(), 8
`base::Arithmetic`, 2
`base::attr`, 5
`base::attributes()`, 14
`base::class()`, 14
`base::conditions`, 11
`base::Extract`, 2
`base::intersect()`, 8
`base::is.null`, 6
`base::match()`, 8
`base::paste`, 3
`base::setdiff()`, 8
`base::storage.mode()`, 14
`base::structure()`, 14
`base::suppressMessages()`, 9
`base::suppressWarnings()`, 9

`collapse`, 3
`colons`, 4

`divide` (alias_arithmetic), 2
`divide_int` (alias_arithmetic), 2

`empty_df` (quick_df), 12
`exattr`, 5

`flip`, 5

`if_null`, 6
`is_in` (match_ext), 8
`is_linux` (os), 12
`is_macos` (os), 12
`is_named` (names), 10
`is_out` (match_ext), 8
`is_windows` (os), 12
`is_within` (match_ext), 8
`is_without` (match_ext), 8

`list0`, 7
Long vectors, 8

`match_ext`, 8
`muffle`, 9
`multiply` (alias_arithmetic), 2

`NA_character_`, 3
`names`, 10
`new_condition`, 11
`no_match` (match_ext), 8

`os`, 12

`quick_df`, 12
`quick_df1` (quick_df), 12

`raise_power` (alias_arithmetic), 2
`remainder` (alias_arithmetic), 2
`remove_names` (names), 10
`require_namespace`, 13

`set_names` (names), 10
`struct`, 14
`subassign1` (alias_extract), 2
`subassign2` (alias_extract), 2
`subassign3` (alias_extract), 2
`subset1` (alias_extract), 2
`subset2` (alias_extract), 2

`subset3(alias_extract)`, [2](#)
`subtract(alias_arithmetic)`, [2](#)

`verbose`, [15](#)
`verbose()`, [16](#)

`wuffle(muffle)`, [9](#)