Package 'DIZtools'

March 16, 2023

Title Lightweight Utilities for 'DIZ' R Package Development

Version 0.0.8

Date 2023-03-16

Description Lightweight utility functions used for the R package development infrastructure inside the data integration centers ('DIZ') to standardize and facilitate repetitive tasks such as setting up a database connection or issuing notification messages and to avoid redundancy.

License GPL-3

URL https://github.com/miracum/misc-diztools

BugReports https://github.com/miracum/misc-diztools/issues

Depends R (>= 3.1.0)

Imports cleaR, config, data.table, logger, magrittr, parsedate, R.utils

Suggests lintr, shiny, shinyjs, testthat

Encoding UTF-8

Language en-US

NeedsCompilation no

Author Jonathan M. Mang [aut, cre] (<https://orcid.org/0000-0003-0518-4710>), Lorenz A. Kapsner [aut] (<https://orcid.org/0000-0003-1866-860X>), MIRACUM - Medical Informatics in Research and Care in University Medicine [fnd], Universitätsklinikum Erlangen, Germany [cph]

Maintainer Jonathan M. Mang <jonathan.mang@uk-erlangen.de>

Repository CRAN

Date/Publication 2023-03-16 13:00:02 UTC

R topics documented:

assign_to_R_env *assign_to_R_env*

Description

Hack variable into global env (bypasses R CMD checks). This does create a new variable in the R environment but NOT a new variable in the system environment. To create a system environment variable being accessible via 'Sys.getenv(...)', use the function 'DIZtools::setenv2(key = "varname", val = 7)'. Old function name: 'global_env_hack()'

Usage

```
assign_to_R_env(key, val, pos = 1)
```

Arguments

key	A character (!) string. The name of the assigned variable
val	An object. The object that will be assigned to 'key'.
pos	An integer. The position of the environment (default: 1).

Value

No return value, called for side effects (see description).

See Also

http://adv-r.had.co.nz/Environments.html

Examples

```
utils_path <- tempdir()
assign_to_R_env(
  key = "utils_path",
  val = utils_path,
  pos = 1L
)
```

check_if_unique_rows *Takes a data.table dataset and checks if for each unique element in a specified column there is exaclty one row.*

Description

Takes a data.table dataset and checks if for each unique element in a specified column there is exaclty one row.

Usage

```
check_if_unique_rows(
  data,
  colname,
  findme = NULL,
  stop = FALSE,
  feedback = TRUE,
  print_invalid_rows = TRUE,
  return = TRUE
)
```

Arguments

data	A data.table
colname	The name of the column to check for uniqueness.
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
stop	(boolean, default = FALSE) Should the function call stop() if there are non-unique rows in the data?
feedback	(boolean, default = TRUE) Should the function print text to the console depending on the result?
print_invalid_rows	(boolean, default = TRUE) Should the function print invalid rows to the console?
return	(boolean, default = TRUE) Should the function return 'TRUE' or 'FALSE' depending on the result? If 'stop = TRUE' is set, the function will end with 'stop()' before returning anything.

Examples

```
## Not run:
  check_if_unique_rows(data)

## End(Not run)
```

cleanup_old_logfile *Archives the current logfile and creates a new blank one.*

Description

This function is called once at the beginning of the runtime of the tool. It checks whether there is an old logfile and renames it (if existing) to "logfile_20yy-mm-dd-HHMMSS.log". Then a new, empty, logfile "logfile.log" is created.

Usage

```
cleanup_old_logfile(logfile_dir)
```

Arguments

logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.

Value

No return value, called for side effects (see description)

Examples

```
cleanup_old_logfile("path/to/logfile/dir/")
```

clean_path_name *clean_path helper function*

Description

Function to clean paths to surely have a tailing slash or not.

Usage

```
clean_path_name(pathname, remove.slash = FALSE)
```

Arguments

pathname	A character string. A path name to be cleaned (to have a tailing slash or not).
remove.slash	(boolean) Default: FALSE. Should the result contain the tailing slash or remove it?

Value

The result is the input but with an tailing slash.

Examples

```
# Both function calls will return "home/test/"
clean_path_name("home/test")
clean_path_name("home/test/")
```

`clear` *Clean the console and environment-variables*

Description

Function to clean the local environment. The call of this function clears the console and the local environment variables.

Usage

```
clear(keep_environment = FALSE, keep_console = FALSE)
```

Arguments

`keep_environment`	(Optional, boolean) If true, the objects from the environment will not be deleted/emptied.
`keep_console`	(Optional, boolean) If true, the console will not be emptied.

Value

Nothing.

Examples

```
clear()
```

`close_all_connections` *Cleanup function to unset/close all open connections*

Description

This function is meant to be called at the end of a run of the app. It will close all open connections to files or databases. This closes ALL connections. Not just the ones opened by this package.

Usage

```
close_all_connections()
```

Value

No return value, called for side effects (see description)

Examples

```
close_all_connections()
```

`equals2` *Compare two elements and return true if both elements are the same.*

Description

The base-R function '==' is not working in an intended way for NAs and boolean. This function fixes this.

Usage

```
equals2(v1, v2)
```

Arguments

v1	First vector or element
v2	Second vector or element

Value

The equality between both vectors.

References

http://www.cookbook-r.com/Manipulating_data/ Comparing_vectors_or_factors_with_NA/>

Examples

```
## Not run:
  dt <-
    data.table::data.table(
      a = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, NA, NA, NA),
      b = c(TRUE, FALSE, NA, TRUE, FALSE, NA, TRUE, FALSE, NA)
    )
  dt[, "classic_result" := get("a") == get("b")]
  dt[, "result_expected" := equals2(get("a"), get("b"))]
  dt
  ## This is the result:
  #        a     b classic_result result_expected
  # 1:  TRUE  TRUE           TRUE            TRUE
  # 2:  TRUE FALSE          FALSE           FALSE
```

```
  # 3:  TRUE    NA             NA           FALSE
  # 4: FALSE  TRUE          FALSE           FALSE
  # 5: FALSE FALSE           TRUE            TRUE
  # 6: FALSE    NA             NA           FALSE
  # 7:    NA  TRUE             NA           FALSE
  # 8:    NA FALSE             NA           FALSE
  # 9:    NA    NA             NA            TRUE

## End(Not run)
```

`feedback` *Function to feedback messages either to the user and/or to the console and to the logfile.*

Description

This function provides the functionality to publish any kind of information to the user, the console and/or to the logfile. This might be a simple info, a warning or an error. The function can be used to select the output (console, ui, logfile). If no output is selected, the print_this string will be printed to the console and to logfile. One of these must be a string with length > 0: print_me, console, ui. Default parameters can be set using the function 'DIZtools::log_set_defaults'. This function uses 'logger' as package to log to the console. If you are new to this function, consider using 'logger' instead.

Usage

```
feedback(
  print_this = NULL,
  type = NULL,
  ui = NULL,
  console = NULL,
  logfile = NULL,
  logjs = NULL,
  prefix = NULL,
  suffix = NULL,
  findme = NULL,
  logfile_dir = NULL,
  headless = NULL
)
```

Arguments

`print_this`	(Optional, String, default: "")
`type`	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
`ui`	(Optional, Boolean/String, default: FALSE) If true, the message will also be printed to the user in form of a modal. Can also be a string.

console	(Optional, Boolean/String, default: TRUE) If true, the message will also be printed to the console as is. Can also be a string.
logfile	(Optional, Boolean, default: TRUE) If true (default) the print_this string will also be printed to the console.
logjs	(Optional, Boolean, default: FALSE) If true (default: false) the print_this string will also be printed to the javascript-console. This only makes sense, if the gui is active.
prefix	Prefix (Optional, String, default: "") This is useful if print_this is an array/list. Each entry will then be new row with this prefix.
suffix	Suffix (Optional, String, default: "") Same like prefix but at the end of each line.
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
headless	(Optional, Boolean, default: TRUE) Indicating, if the function is run only in the console (headless = TRUE) or on a GUI frontend (headless = FALSE).

Value

No return value, called for publishing a message.

See Also

https://daroczig.github.io/logger/

Examples

```
feedback(
  print_this = "This is an error message you can provide",
  type = "Error",
  findme = "215bb3695c",
  logfile_dir = tempdir(),
  headless = TRUE
)
```

---

feedback_get_formatted_string

Format the feedback string

Description

Helper function for the feedback function to combine the input parameters in proper manner to be a pretty and informative string which than can be added to the logfile and/or be displayed in the console. CAUTION: 'print_this' must be of length 1! For arrays loop through them by hand and call this function several times! Internal use. Use the robust 'feedback' function instead.

Usage

```
feedback_get_formatted_string(print_this, type, findme, prefix, suffix)
```

Arguments

print_this	(Optional, String, default: "")
type	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
prefix	Prefix (Optional, String, default: "") This is useful if print_this is an array/list. Each entry will then be new row with this prefix.
suffix	Suffix (Optional, String, default: "") Same like prefix but at the end of each line.

Value

Returns a properly an consistent formatted string containing the parameters handed over to this function.

---

feedback_to_console *Print to the console. Internal use only.*

Description

Helper function for the feedback function to print stuff to the console. Everything will also be added to the logfile. Internal use. Use the robust 'feedback' function instead.

Usage

```
feedback_to_console(
  print_this,
  type,
  findme,
  prefix,
  suffix,
  logjs,
  logfile_dir,
  headless = TRUE
)
```

Arguments

print_this	(Optional, String, default: "")
type	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
prefix	Prefix (Optional, String, default: "") This is useful if print_this is an array/list. Each entry will then be new row with this prefix.
suffix	Suffix (Optional, String, default: "") Same like prefix but at the end of each line.
logjs	(Optional, Boolean, default: FALSE) If true (default: false) the print_this string will also be printed to the javascript-console. This only makes sense, if the gui is active.
logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
headless	(Optional, Boolean, default: TRUE) Indicating, if the function is run only in the console (headless = TRUE) or on a GUI frontend (headless = FALSE).

Value

No return value, called for side effects (see description)

---

feedback_to_logfile *Add to the logfile. Internal use.*

Description

Helper function for the feedback function to add content to the logfile. Internal use. Use the robust 'feedback' function instead.

Usage

```
feedback_to_logfile(print_this, type, findme, prefix, suffix, logfile_dir)
```

Arguments

print_this	(Optional, String, default: "")
type	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
prefix	Prefix (Optional, String, default: "") This is useful if print_this is an array/list. Each entry will then be new row with this prefix.
suffix	Suffix (Optional, String, default: "") Same like prefix but at the end of each line.
logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.

Value

No return value, called for side effects (see description)

`feedback_to_logjs` *Feedback to the gui/browser-console with logjs. Internal use.*

Description

Helper function for the feedback function to also show the messages to the gui/user via the browser console. Internal use. Use the robust 'feedback' function instead.

Usage

```
feedback_to_logjs(print_this, logfile_dir, headless)
```

Arguments

`print_this`	(Optional, String, default: "")
`logfile_dir`	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
`headless`	(Optional, Boolean, default: TRUE) Indicating, if the function is run only in the console (headless = TRUE) or on a GUI frontend (headless = FALSE).

Value

No return value, called for side effects (see description)

`feedback_to_ui` *Feedback to the user with a modal. Internal use.*

Description

Helper function for the feedback function to show modals to the gui/user. Everything will also be added to the logfile. Internal use. Use the robust 'feedback' function instead.

Usage

```
feedback_to_ui(print_this, type, logfile_dir, headless = FALSE)
```

Arguments

`print_this`	(Optional, String, default: "")
`type`	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
`logfile_dir`	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
`headless`	(Optional, Boolean, default: TRUE) Indicating, if the function is run only in the console (headless = TRUE) or on a GUI frontend (headless = FALSE).

Value

No return value, called for side effects (see description)

`file_lines_to_list` *Read in lines from a file and store it in a list.*

Description

Read in lines from a file and store it in a list.

Usage

```
file_lines_to_list(filepath)
```

Arguments

`filepath`	(string) Path to file to read in.

Value

A list with one element per row of the input file

`firstup` *Converts the first letter of the input string to uppercase*

Description

Converts the first letter of the input string to uppercase

Usage

```
firstup(x)
```

Arguments

`x`	A character string. E.g. "hello world" will become "Hello world".

Value

Returns the input string but with a capital first letter.

Examples

```
{
firstup("first letter of this string will be upper case as return")
}
```

format_posixct *Formats a given POSIXct timestamp without the need of manually specifing format parameters.*

Description

See title.

Usage

```
format_posixct(x, lang = "en", date = TRUE, time = TRUE)
```

Arguments

x	The POSIXct timestamp or a string to be automatically converted to a POSIXct timestamp.
lang	(Optional, String, Default = "en") The language of the result. Currently implemented: "en"/"de". If you supply another not yet implemented language here, "en" will be chosen automatically.
date	(Optional, Boolean, Default = TRUE) Should the date be part of the result string?
time	(Optional, Boolean, Default = TRUE) Should the time be part of the result string?

Value

(String) The formatted timestamp as a string.

Examples

```
## Not run:
format_POSIXct(x = "2021-12-31 12:34")
## Result: "2021-12-31, 12:34:00"
format_POSIXct(x = "2021-12-31 12:34", lang = "de")
## Result: "31.12.2021, 12:34:00"
format_posixct(Sys.time())
## Result: "2022-01-01, 09:10:50"
)
## End(Not run)
```

get_config *Reads a config yaml file and return the value for a given key.*

Description

Reads a config yaml file and return the value for a given key.

Usage

```
get_config(config_file, config_key)
```

Arguments

config_file	A character string. The path to the config.yml-file containing the database configuration.
config_key	A character string. The name of the corresponding database. This string must be conform with the corresponding config section in the config.yml-file.

Value

If successful it returns the value, Null otherwise.

Examples

```
utils_path <- tempdir()
config <- get_config(
  config_file = paste0(utils_path, "/MISC/email.yml"),
  config_key = "email"
)
```

get_current_timestamp *Quickly get the current time stamp*

Description

Function to quickly get the current time stamp without need to handle format-options etc.

Usage

```
get_current_timestamp(no_spaces = FALSE)
```

Arguments

no_spaces	Boolean. Default = 'FALSE'. Specifies whether the output can contain spaces or not. E.g. if the output is for human reading, 'no_spaces = FALSE' is a good option. As suffix for file names (e.g. logfiles), 'no_spaces = TRUE' might be a good option.

Value

The current timestamp in always the same format. #'

Examples

```
get_current_timestamp(no_spaces = TRUE)
# Result: "2020-12-03-134354"
get_current_timestamp()
# this is the same like
get_current_timestamp(no_spaces = FALSE)
# Result: "03.12.2020 - 13:43 UTC"
```

---

is.empty *Empty Value*

Description

Rails-inspired helper that checks if vector values are "empty", i.e. if it's: NULL, zero-length, NA, NaN, FALSE, an empty string or 0. Note that unlike its native R is.<something> sibling functions, is.empty is vectorised (hence the "values").

Usage

```
is.empty(x, trim = TRUE, all = FALSE, ...)
```

Arguments

x	an object to check its emptiness
trim	trim whitespace? (TRUE by default)
all	return overall result over list/vector instead of vector of results? is.empty(x, all = TRUE) is the same like all(unlist(is.empty(x)))
...	additional arguments for sapply

Source

Copied from 'rapportools::is.empty()'

Examples

```
## Not run:
is.empty(NULL)     # [1] TRUE
is.empty(c())      # [1] TRUE
is.empty(NA)       # [1] TRUE
is.empty(NaN)      # [1] TRUE
is.empty("")       # [1] TRUE
is.empty(0)        # [1] TRUE
is.empty(0.00)     # [1] TRUE
is.empty("    ")   # [1] TRUE
is.empty("foobar") # [1] FALSE
is.empty("    ", trim = FALSE)             # [1] FALSE
## is.empty is vectorised!
all(is.empty(rep("", 10)))                 # [1] TRUE
all(is.empty(matrix(NA, 10, 10)))          # [1] TRUE
is.empty(matrix(NA, 10, 10), all = TRUE))  # [1] TRUE

## End(Not run)
```

`is_date_format` *Checks if a string matches a given date format.*

Description

Checks if a string matches a given date format.

Usage

```
is_date_format(date, format)
```

Arguments

`date`	The list applied from rv$restricting_date
`format`	The format parameters. See `?strptime` for parameter info.

Value

TRUE/FALSE

log_get_current_options *Get the current settings for the logging function as list.*

Description

Get the current settings for the logging function as list

Usage

```
log_get_current_options()
```

Value

The list with the current parameters.

Examples

```
log_get_current_options()
```

log_get_default_options *Get the default settings for the logging function as list.*

Description

Get the default settings for the logging function as list

Usage

```
log_get_default_options()
```

Value

The list with the default parameters.

Examples

```
log_get_default_options()
```

log_internal_test *Internal function for debugging only.*

Description

Internal function for debugging only.

Usage

```
log_internal_test()
```

Value

Nothing.

log_map_type_to_loggertype *Get the logger type from the type string (the argument of the 'feedback()' function)*

Description

Mapping the log-types from string to logger::<type>. E.g. the string "Info" will be mapped to 'logger::INFO'.

Usage

```
log_map_type_to_loggertype(type)
```

Arguments

type	(String) The type of the message. E.g. "error", "Info".

Value

The 'logger'type. If no corresponding logger-type is found, the result will be 'NULL'.

log_remove_options *Remove all log-related options from 'options()'.*

Description

Remove all log-related options from 'options()'.

Usage

```
log_remove_options()
```

Value

Nothing.

Examples

```
log_remove_options()
```

log_set_defaults *Set default options for all log-functions*

Description

This function sets the default log options. Parameters not supplied to this function will be set with the default value. If you want to reset all parameters to the default ones, run `log_set_defaults(reset = TRUE)`. This can also be combined with a new custom default value: `log_set_defaults(reset = TRUE, prefix = "Prefix")` which will reset all parameters to default and afterwards assign "Prefix" as new global prefix.

Usage

```
log_set_defaults(
  print_this = NULL,
  type = NULL,
  ui = NULL,
  console = NULL,
  logfile = NULL,
  logjs = NULL,
  prefix = NULL,
  suffix = NULL,
  findme = NULL,
  logfile_dir = NULL,
  headless = NULL,
  reset = FALSE
)
```

Arguments

print_this	(Optional, String, default: "")
type	(Optional, String, default: "Info") E.g. "Warning", "Error". Default: "Info"
ui	(Optional, Boolean/String, default: FALSE) If true, the message will also be printed to the user in form of a modal. Can also be a string.
console	(Optional, Boolean/String, default: TRUE) If true, the message will also be printed to the console as is. Can also be a string.
logfile	(Optional, Boolean, default: TRUE) If true (default) the print_this string will also be printed to the console.
logjs	(Optional, Boolean, default: FALSE) If true (default: false) the print_this string will also be printed to the javascript-console. This only makes sense, if the gui is active.
prefix	Prefix (Optional, String, default: "") This is useful if print_this is an array/list. Each entry will then be new row with this prefix.
suffix	Suffix (Optional, String, default: "") Same like prefix but at the end of each line.
findme	(Optional, String, default: "") Recommended with length 10. String to find the message in the code. E.g. 10-digit random hex from https://onlinetools.com/random/generate-random-hexadecimal-numbers
logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
headless	(Optional, Boolean, default: TRUE) Indicating, if the function is run only in the console (headless = TRUE) or on a GUI frontend (headless = FALSE).
reset	(boolean, default = FALSE) Should all parameters be reset to their default values?

Value

No return value, called for side effects (see description).

Examples

```
DIZtools::log_set_defaults(logfile_dir = tempdir())
```

---

number_to_position *Converts an integer number to its "verbal position". 1 –> "1st", 2 –> "2nd", 3 –> "3rd", 4 –> "4th", ...*

Description

Converts an integer number to its "verbal position". 1 –> "1st", 2 –> "2nd", 3 –> "3rd", 4 –> "4th", ...

Usage

```
number_to_position(x)
```

Arguments

x	A number.

Value

Returns the input number as string with a new suffix depending on the numbers position in the numbers bar.

Examples

```
{

}
```

`paste2` *Normal 'paste' function with additional 'collapse_last' argument.*

Description

The base 'paste' function but with the add on to also supply a 'collapse_last' value to change the 'collapse' argument at the last position. To get from "cat", "mouse", "dog" to a string "cat, mouse and dog", one simply needs to call 'paste(c("cat","mouse","dog"), collapse = ", ", collapse_last = " and ")'

Usage

```
paste2(..., collapse = NULL, collapse_last = NULL, sep = " ", recycle0 = FALSE)
```

Arguments

...	one or more R objects, to be converted to character vectors.
collapse	an optional character string to separate the results. Not NA_character_.
collapse_last	(string, optional) The string to use for the last instance while collapsing. All other elements will be pasted using the normal 'collapse' argument. If 'collapse' is not set, 'collapse_last' will be ignored.
sep	a character string to separate the terms. Not NA_character_.
recycle0	logical indicating if zero-length character arguments should lead to the zero-length character(0) after the sep-phase (which turns into "" in the collapse-phase, i.e., when collapse is not NULL).

Value

String. See'?paste' for details.

References

https://stackoverflow.com/a/38276239

Examples

```
{
  paste2(c("cat", "mouse", "dog"),
         collapse = ", ",
         collapse_last = " and ")
  #> [1] "cat, mouse and dog"
}
```

`pretty_timestamp` *Quickly get a pretty timestamp without need to handle format-options etc.*

Description

Function to quickly get a pretty timestamp without need to handle format-options etc.

Usage

```
pretty_timestamp(timestamp, no_spaces = FALSE)
```

Arguments

`timestamp`	A POSIXct timestamp or a string which ca be converted to a POSIXct timestamp.
`no_spaces`	Boolean. Default = 'FALSE'. Specifies whether the output can contain spaces or not. E.g. if the output is for human reading, 'no_spaces = FALSE' is a good option. As suffix for file names (e.g. logfiles), 'no_spaces = TRUE' might be a good option.

Value

The timestamp in always the same format. #'

Examples

```
pretty_timestamp("2023-10-30 12:34:56", no_spaces = TRUE)
# Result: "2023-10-30T123456"
pretty_timestamp("2023-10-30 12:34:56")
# this is the same like
pretty_timestamp("2023-10-30 12:34:56", no_spaces = FALSE)
# Result: "30. Oct 2023 - 12:34 UTC"
```

rep2 *Repeat something with the ability to also collapse the output.*

Description

Repeat something with the ability to also collapse the output. The base ‘rep("ha", 3)‘ function does not support arguments like ‘collapse‘ or ‘sep‘ like ‘paste(...)‘. ‘rep2‘ closes this gap.

Usage

```
rep2(x, n, ...)
```

Arguments

x	The object to repeat
n	The amount how often the object should be repeated
...	Further arguments passed to ‘paste‘ (see ‘help("paste")‘ for more information).

Value

The result from ‘paste(rep(x, n), sep = sep, collapse = collapse)‘

Examples

```
## rep2 is the same like rep:
rep(x = "ha", 3)
#> "ha" "ha" "ha"
rep2(x = "ha", 3)
#> "ha" "ha" "ha"

## ... but you can also use the arguments from `paste`:
rep2(x = "ha", n = 3, collapse = "")
#> "hahaha"
```

robust_round *Round numbers without problems.*

Description

Round numbers without problems.

Usage

```
robust_round(x, digits = 2, thousands_separator = "", decimal_separator = ".")
```

Arguments

x	(numeric) The numeric input vector to round.
digits	(int, optional) The number of digits after the decimal separator to round to.
thousands_separator	(string, optional) Used as mark between every 3 decimals before the decimal point.
decimal_separator	(string, optional) The character to be used to indicate the numeric decimal point.

Value

Rounded numbers as string.

Examples

```
{
  robust_round(c(1.234567, 987123.987654321))
  #> [1] "1.23"   "987.99"
}
```

---

setdiff_all *Get the difference of two vectors in both directions.*

Description

The base-R function ‘setdiff‘ is asymmetric meaning ‘setdiff(vec1, vec2)‘ is not the same as ‘setdiff(vec2, vec1)‘. Only the first vector will be compared to the second vector and all elements not contained in the second are in the resulting vector. So if you also want in include all elements being in the second vector but not in the first, you can use this function. In this case you are searching for elements being in the union of both vectors but not in the intersect of both vectors. This function is a symmetric function. It doesn't matter in which order you input the vectors, the content will be the same. Only the order of the elements inside the output differs.

Usage

```
setdiff_all(vec1, vec2)
```

Arguments

vec1	First vector
vec2	Second vector

Value

The difference between both vectors.

Examples

```
## Not run:
vec1 <- c(1,2,3,4)
vec2 <- c(3,4,5,6)
# setdiff(vec1, vec2) = c(1,2)
# setdiff(vec2, vec1) = c(5,6)
# setdiff_all(vec1, vec2) = c(1,2,5,6)
# setdiff_all(vec2, vec1) = c(5,6,1,2)

## End(Not run)
```

`setenv2` *Assign variables to the system environment.*

Description

Create a system environment variable with the use of variables. While ‘var.name = "testname"; var.value = 7‘ and ‘Sys.setenv(var.name = var.value)‘ will create ‘var.name = 7‘ in the system environment, ‘DIZtools::setenv2(key = var.name, val = var.value)‘ will create ‘testname = 7‘ in the system environment.

Usage

```
setenv2(key, val)
```

Arguments

`key`	A character (!) string. The name of the assigned variable
`val`	An object. The object that will be assigned to ’key’.

Value

No return value, called for side effects (see description).

See Also

https://stackoverflow.com/a/12533155

Examples

```
var.name = "testname"
var.value = 7

Sys.setenv(var.name = var.value)

Sys.getenv("testname")
#> [1] ""
```

```
Sys.getenv("var.name")
#> [1] "7"

Sys.unsetenv("var.name")
Sys.unsetenv("testname")

setenv2(key = var.name, val = var.value)
Sys.getenv("testname")
#> [1] "7"
Sys.getenv("var.name")
#> [1] ""
```

`setenv_file` *Set all variables of a '.env' file to the system environment.*

Description

Internal function to set environment variables that are necessary for the database connections with `db_connection`. Old function name: 'set_env_vars()'.

Usage

```
setenv_file(env_file)
```

Arguments

`env_file`	A character. The full path including the file name to the file containing the environment variable definitions to be loaded.

Value

No return value, called for side effects (see description)

See Also

`Sys.setenv`

Examples

```
## Not run: set_env_vars("./.env")
```

`string_replacements` *Clean string with a given set of replacements*

Description

This function provides the functionality to clean a string with a given set of replacements. This is e.g. useful to create filenames or paths that are not allowed to contain spaces.

Usage

```
string_replacements(
  input,
  replace_mapping = "default",
  tolower = FALSE,
  toupper = FALSE
)
```

Arguments

`input`	(string) The character string to be processed.
`replace_mapping`	(Optional, list, default = "default") The mapping containing what should be replaced with what: ‘replace_mapping <- list("replace_this" = "with_this")‘
`tolower`	(boolean, default = FALSE) Should the result be lowercase?
`toupper`	(boolean, default = FALSE) Should the result be uppercase?

Value

(String) All elements (names) of the input ‘replace_mapping‘ or the default mapping are replaced by its values of the mapping.

Examples

```
string_replacements(input = "Ab 20. April 2020 (((___((N = 1.234)")
# Result: "Ab_20_April_2020_N_1234"
```

`time_diff_print` *Calculate time difference of two timestamps, round the value and return a string with suiting unit.*

Description

Create string with time difference in suitable unit. Additional automatically add the remaining time depending on the position in an iteration process, or the estimated time of arrival by just providing the current and total iteration step(s).

Usage

```
time_diff_print(
  older_timestamp,
  newer_timestamp = NULL,
  iteration = NULL,
  iterations = NULL,
  remaining_time = TRUE,
  eta = TRUE,
  prefix_iteration = "Iteration ",
  prefix_time_elapsed = "Time elapsed: ",
  prefix_time_remaining = "Remaining: ",
  prefix_eta = "ETA: ",
  digits = 2,
  thousands_separator = "",
  decimal_separator = "."
)
```

Arguments

`older_timestamp`	(POSIXct) Start time.
`newer_timestamp`	(POSIXct, Optional) End time. If not set, the current time will be used.
`iteration`	(Numeric, Optional) The current iteration if also the time process within all iterations additional to the elapsed time is of interest.
`iterations`	(Numeric, Optional) The total number of iterations.
`remaining_time`	(Boolean, Optional, Default = TRUE) Should the estimated time needed to finish all iterations be displayed?
`eta`	(Boolean, Optional, Default = TRUE) Should the estimated time of arrival needed to finish all iterations be displayed?
`prefix_iteration, prefix_time_elapsed, prefix_time_remaining, prefix_eta`	Prefixes for the output string.
`digits`	(int, optional) The number of digits after the decimal separator to round to.

thousands_separator
: (string, optional) Used as mark between every 3 decimals before the decimal point.

decimal_separator
: (string, optional) The character to be used to indicate the numeric decimal point.

Value

A list with one element per row of the input file

Examples

```
## Since no second timestamp is provided, the current time
## (currently 2023-03-08) will be assumed:
DIZtools::time_diff_print("2023-01-01 10:00")
#> [1] "Time elapsed: 66.20 days"
DIZtools::time_diff_print("2023-01-01 10:00", iteration = 7, iterations = 10)
#> [1] "Iteration 7 of 10 (70.00 %), Time elapsed: 66.20 days
#>     (Remaining: ~ 28.37 days, ETA: ~ 05. Apr 2023 - 23:42 UTC)"
```

---

trim.space *Trim Spaces*

Description

Removes leading and/or trailing space(s) from a character vector. By default, it removes both leading and trailing spaces.

Usage

```
trim.space(
  x,
  what = c("both", "leading", "trailing", "none"),
  space.regex = "[:space:]",
  ...
)
```

Arguments

x	a character vector which values need whitespace trimming
what	which part of the string should be trimmed. Defaults to both which removes trailing and leading spaces. If none, no trimming will be performed.
space.regex	a character value containing a regex that defines a space character
...	additional arguments for gsub function

Value

a character vector with (hopefully) trimmed spaces

Source

Copied from ‘rapportools::is.empty()‘

---

vgsub *Vectorised String Replacement*

Description

A simple wrapper for gsub that replaces all patterns from `pattern` argument with ones in `replacement` over vector provided in argument `x`.

Usage

```
vgsub(pattern, replacement, x, ...)
```

Arguments

`pattern`	see eponymous argument for gsub function
`replacement`	see eponymous argument for gsub function
`x`	see eponymous argument for gsub function
`...`	additional arguments for gsub function

Value

a character vector with string replacements

Source

Copied from package ‘rapportools‘

References

See original thread for more details `https://stackoverflow.com/a/6954308/457898`. Special thanks to user Jean-Robert for this one!

`%notin%` *notin helper function*

Description

Function to return elements of x that are not in y.

Usage

```
x %notin% y
```

Arguments

x	Object 1.
y	Object 2.

Value

Returns the result of !

Examples

```
tmp1 <- c("a","b","c")
tmp2 <- c("b", "c", "d")
tmp1 %notin% tmp2
```

Index