Package 'DIZutils'

January 24, 2022

Title Utilities for 'DIZ' R Package Development

Version 0.0.10

Date 2022-01-24

Description 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-dizutils

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

Depends R (>= 3.1.0)

Imports config, data.table, DBI (>= 1.1.0), Hmisc, parsedate, psych, rapportools, RJDBC, RJSONIO, RPostgres, shiny, shinyjs, xml2

Suggests lintr, testthat

Encoding UTF-8

Language en-US

SystemRequirements libpq >= 9.0: libpq-dev (deb) or postgresql-devel (rpm)

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 2022-01-24 13:22:42 UTC

R topics documented:

`check_if_table_exists` *Check if a database table exists.*

Description

Check if a database table exists.

Usage

```
check_if_table_exists(db_con, table_name)
```

Arguments

`db_con`	A DBI database connection. See ‘db_connection()‘ for details.
`table_name`	(String) The name of the table or view to be checked.

Value

True, if the table exists, false otherwise.

Examples

```
## Not run:
res <- DIZutils::check_if_table_exists(
  db_con = DIZutils::db_connection(...),
  table_name = "my_table"
)
## 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_20xx-xx-xx-xxxxxx.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_name 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.

`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(logfile_dir = tempdir(), headless = TRUE)
```

Arguments

`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)

Examples

```
close_all_connections(
  logfile_dir = tempdir(),
  headless = TRUE
)
```

`combine_stats` *Combine aggregated statistics.*

Description

This function provides the functionality to combine multiple statistics to a single statistical overview. This is e.g. useful if you are only allowed to export statistical characteristics from a site but not the data itself. So in this case you have e.g. mean, median and N from each site but want to say something about the mean, median and N over all sites like you had the data of all sites in one big pool and would do the statisitcs there.

Usage

```
combine_stats(summaries, demo = FALSE)
```

Arguments

summaries	(data.table) Data table containing all stats you want to combine as rows. This data.table must contain the columns ‘Min‘, ‘Q10‘, ‘Q25‘, ‘Median‘, ‘Mean‘, ‘SD‘, ‘Q75‘, ‘Q90‘, ‘Max‘, ‘N‘. Each row in this data table represents a site as of the example described above.
demo	(boolean, default = FALSE) Do you want to see how the function works? Then call ‘combine_stats(summaries = NULL, demo = TRUE)‘.

Value

A one-row data.table containing the calculated, aggregates statistics of the input.

---

db_connection *db_connection helper function*

Description

Internal function to test and get the database connection of the target data system.

Usage

```
db_connection(
  system_name = NULL,
  db_type,
  headless = FALSE,
  from_env = TRUE,
  settings = NULL,
  timeout = 30,
  logfile_dir = NULL,
  lib_path = NULL
)
```

Arguments

system_name	(Default = NULL) A character. Name of the database system. Used to find the correct settings from the env. If you don’t want to load the settings from the environment, use the ‘settings‘ parameter. Otherwise this funcion will search for all settings beginning with ‘system_name‘ in the environment. If ‘system_name = "i2b2"‘ settings like ‘I2B2_HOST‘ or ‘I2B2_PORT‘ (notice the uppercase) will be loaded from the environment. You can load such an env file e.g. by using ‘DIZutils::set_env_vars(path_to_file)‘.
db_type	A character. Type of the database system. Currently implemented systems are: ’postgres’, ’oracle’.

headless	A boolean (default: 'FALSE'). Indicating, if the function is run only in the console ('headless = TRUE') or on a GUI frontend ('headless = FALSE').
from_env	A boolean (default: 'TRUE'). Should database connection be read from the environment or from a settings file. All necessary parameters must be uppercase and have the prefix of the db_name. E.g.: 'I2B2_HOST' or 'I2B2_PORT'. See the 'settings' parameter for all necessary variables.
settings	A list. Required if 'from_env == FALSE'. A list containing settings for the database connection. Required fields are 'host', 'db_name', 'port', 'user' and 'password'. Additionally for Oracle DB's: 'sid' (instead of 'db_name'). If 'settings' is set, 'from_env' will be set to 'FALSE' automatically.
timeout	A timeout in sec. for the db-connection establishment. Values below 2 seconds are not recommended. Default is 30 seconds.
logfile_dir	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
lib_path	A character string. The path to the ojdbc*.jar file. If you run one of the R-containers from the UK-Erlangen DIZ, there might be a lib for oracle here: 'lib_path = "/opt/libs/ojdbc8.jar"'

Value

If successful, the result will be the established connection. Otherwise the result will be null.

See Also

dbConnect, RPostgres

Examples

```
## Not run:
db_con <- DIZutils::db_connection(
  db_name = "i2b2",
  db_type = "postgres",
  headless = TRUE,
  logfile_dir = tempdir()
)
## End(Not run)
```

---

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

Usage

```
feedback(
  print_this = "",
  type = "Info",
  ui = FALSE,
  console = TRUE,
  logfile = TRUE,
  logjs = FALSE,
  prefix = "",
  suffix = "",
  findme = "",
  logfile_dir = tempdir(),
  headless = TRUE
)
```

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://www.browserling.com/tools/random-hex>) or (<https://onlinerandomtools.com/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.

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 ge 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://www.browserling.com/tools/random-hex>) or (<https://onlinerandomtools.com/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://www.browserling.com/tools/random-hex>) or (<https://onlinerandomtools.com/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://www.browserling.com/tools/random-hex>) or (<https://onlinerandomtools.com/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)

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"
)
## End(Not run)
```

---

get_config *get_config helper function*

Description

Internal function to read config files

Usage

```
get_config(config_file, config_key, logfile_dir, headless)
```

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.
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

If successful it returns the config, Null otherwise.

Examples

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

```
  logfile_dir = tempdir(),
  headless = TRUE
)
```

`get_config_env` *get_config_env helper function*

Description

Internal function to read settings for a certain system from the environment. IMPORTANT: If yout want to get any result with your input as prefix, use 'ignore_presets = TRUE'! See param-definition for more details. This function will look at uppercase system_names at default.

Usage

```
get_config_env(
  system_name,
  logfile_dir = tempdir(),
  headless = TRUE,
  ignore_presets = FALSE,
  uppercase_system = TRUE
)
```

Arguments

`system_name`	The name of the system (This is also the prefix used to get the environment variables with 'SYSTEM_KEY', e.g. 'I2B2_DBNAME'). This function also works if there are multiple instances like 'I2B2_1_DBNAME' and 'I2B2_2_DBNAME'. Then the result will contain nested lists for each occurence.
`logfile_dir`	(Optional, String, default: "tempdir()") The absolute path to folder where the logfile will be stored.
`headless`	A boolean (default: 'FALSE'). Indicating, if the function is run only in the console ('headless = TRUE') or on a GUI frontend ('headless = FALSE').
`ignore_presets`	(boolean, default = FALSE) Only return something if all elements from the presets are found? These are currently 'host', 'port', 'user', 'password', 'sid', 'path'. If you have another suffix after 'system_name' in your config file, you won't see it here. To see everything with prefix 'system_name' simply set 'ignore_presets = TRUE'.
`uppercase_system`	(boolean) Default: True. Otherwise: case-sensitive.

Value

If successful it returns the config, null otherwise.

Examples

```
get_config_env(
  system_name = "i2b2",
  logfile_dir = tempdir(),
  headless = FALSE
)
```

`get_current_timestamp` *Quickly get the current timestamp*

Description

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

Usage

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

Arguments

`no_spaces`	Boolean. Default = 'FALSE'. Specifies wether 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 filenames (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"
```

get_db_systems *Quickly get all currently implemented database systems*

Description

Function to quickly get the currently implemented database systems

Usage

```
get_db_systems()
```

Value

The currently implemented database systems as string array. ‘E.g. c("postgres", "oracle")‘ #’

Examples

```
get_db_systems()
# Result: c("postgres", "oracle")
```

global_env_hack *global_env_hack*

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 beeing accessible via ‘Sys.getenv(...)‘, use the function ‘DIZutils::setenv2(key = "varname", val = 7)‘.

Usage

```
global_env_hack(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()
global_env_hack(
  key = "utils_path",
  val = utils_path,
  pos = 1L
)
```

`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 paramters. See '?strptime' for parameter infos.

Value

TRUE/FALSE

`query_database` *query_database helper function*

Description

Internal function to query the database. The function sends a sql statement to the database and returns a data.table.

Usage

```
query_database(
  db_con,
  sql_statement,
  no_result = FALSE,
  close_connection = FALSE
)
```

Arguments

db_con	A DBI database connection.
sql_statement	A character string containing a valid SQL statement. Caution: Everything after the first ';' will be cut off.
no_result	(boolean, default: FALSE) Is the sql meant to return nothing? E.g. if you just insert or update a table. Then supply 'TRUE' here. If you supply 'FALSE' here, the function expects to receive a result table and tries to convert it to a data.table.
close_connection	(boolean, default = FALSE). If TRUE, the connection will be closed after the query was sent and the result received.

Value

Returns the result of the db-query. If 'no_result' is 'TRUE', the return value will be 'TRUE' if the query was successfully sent. Otherwise (if 'no_result' is 'FALSE' which is the default), the result will be the result of the sql query as data.table.

Examples

```
## Not run:
db_con <- DIZutils::db_connection(
  db_name = "i2b2",
  db_type = "postgres"
)

query_database(
  db_con = db_con,
  sql_statement = "SELECT * FROM table_name;"
)

query_database(
  db_con = db_con,
  sql_statement = "INSERT INTO table_name DEFAULT VALUES;",
  no_result = TRUE
)


## End(Not run)
```

`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 beeing 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, ‘DIZutils::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")

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

set_env_vars *set_env_vars helper function*

Description

Internal function to set environment variables that are necessary for the database connections with db_connection.

Usage

```
set_env_vars(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"
```

---

xml_2_json *Quickly transform a xml objet into a json object.*

Description

See title.

Usage

```
xml_2_json(xml)
```

Arguments

xml	An xml object.

Value

The json-representation of the xml object.

`%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