Package: n2khelper (via r-universe)

August 13, 2024

Title Auxiliary Functions for the Analysis and Reporting of the Natura 2000 Monitoring

Version 0.5.0

Description Auxiliary functions for analysing Natura 2000 monitoring data.

License GPL-3

URL https://doi.org/10.5281/zenodo.835732

BugReports https://github.com/inbo/n2khelper/issues

Depends R (>= 3.2.0)

Imports DBI, RODBC, RPostgreSQL, assertthat, dplyr, git2r, lazyeval, lubridate, methods, odbc, plyr, rlang, tidyr

Suggests testthat, tibble

Remotes tidyverse/lubridate

Encoding UTF-8

Language eng

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Collate 'check_character.R' 'check_dataframe_covariate.R' 'check_dataframe_variable.R' 'check_dbtable.R' 'check_dbtable_variable.R' 'check_id.R' 'check_path.R' 'check_single_posix.R' 'check_single_probability.R' 'connect_result.R' 'cut_date.R' 'get_nbn_key.R' 'get_nbn_key_multi.R' 'get_nbn_name.R' 'git_connect.R' 'gitconnection_class.R' 'git_connection.R' 'is_chartor.R' 'match_nbn_key.R' 'odbc_connect.R' 'odbc_get_id.R' 'odbc_get_multi_id.R' 'odbc_insert.R' 'read_object_environment.R'

Repository https://inbo.r-universe.dev

RemoteUrl https://github.com/inbo/n2khelper

RemoteRef HEAD

RemoteSha 3f02dbac8397a6d9a825620ef7851805fd4760e4

Contents

`check_character` *Check if the object is a character*

Description

Factors are converted to character.

Usage

```
check_character(x, name = "x", na_action = na.fail)
```

Arguments

`x`	the object to check
`name`	the name of the object to use in the error message
`na_action`	`stats::na.fail()` throws an error in case of `NA` (default). `stats::na.omit()` will return `x` without the `NA` values. `stats::na.pass()` will return `x` with the `NA` values.

Value

The function gives the character back. It throws an error when the input is not a character.

Examples

```
check_character(c("20", "b"))
```

`check_dataframe_covariate`

Check if the covariates are available in a dataframe

Description

Check if the covariates are available in a dataframe

Usage

```
check_dataframe_covariate(df, covariate, response = "Count", error = TRUE)
```

Arguments

`df`	the `data.frame` to check
`covariate`	The right hand side of the model as a character
`response`	The left hand side of the model as a character
`error`	When TRUE (default), the function returns an error when a variable is missing. Otherwise it returns a warning.

`check_dataframe_variable`

Check if a data.frame contains variables

Description

Check if a data.frame contains variables

Usage

```
check_dataframe_variable(
  df,
  variable,
  name = "df",
  force_na = FALSE,
  error = TRUE
)
```

Arguments

`df`	the `data.frame` to check
`variable`	either a character vector with the names of the variable to check or a named list. The names of the list must match the names of the required variables in the data.frame. The elements of the list contain the accepted classes for each varaible.
`name`	the name of the `data.frame` to use in the error message
`force_na`	check the class of variables with all NA
`error`	When TRUE (default), the function returns an error when a variable is missing. Otherwise it returns a warning.

Value

The function returns TRUE when all variables are present. If returns FALSE when a variable is missing and `error = FALSE`.

Examples

```
check_dataframe_variable(
 df = data.frame(a = integer(0)),
 variable = "a"
)
check_dataframe_variable(
 df = data.frame(a = integer(0)),
 variable = list(a = c("integer", "numeric"))
)
```

`check_dbtable` *Check if a table is available in a given ODBC connection*

Description

Check if a table is available in a given ODBC connection

Usage

```
check_dbtable(table, schema = "public", channel, error = TRUE)
```

Arguments

`table`	The name of the table
`schema`	The schema. Defaults to 'public'
`channel`	the open dplyr connection to the database.
`error`	Indicates the behaviour when a table is missing. Gives an error when error = TRUE (default). Return FALSE otherwise.

Value

TRUE when all tables are present in the ODBC connection.

---

check_dbtable_variable

Check if a variable is available in a given table

Description

Check if a variable is available in a given table

Usage

```
check_dbtable_variable(
  table,
  variable,
  schema = "public",
  channel,
  error = TRUE
)
```

Arguments

`table`	The name of the table
`variable`	A vector with the names of the columns
`schema`	The schema of the table. Defaults to public
`channel`	the open dplyr connection to the database.
`error`	Indicates the behaviour when a variable is missing. Gives an error when error = `TRUE` (default). Return FALSE otherwise.

Value

TRUE when all variables are present in the table.

check_id *Test if an id exists in a given field of the table*

Description

Test if an id exists in a given field of the table

Usage

```
check_id(value, variable, table, channel)
```

Arguments

`value`	the id value
`variable`	A vector with the names of the columns
`table`	The name of the table
`channel`	the open dplyr connection to the database.

check_path *check if a path is an exisiting file or directory*

Description

check if a path is an exisiting file or directory

Usage

```
check_path(path, type = c("file", "directory"), error = TRUE)
```

Arguments

`path`	the path of the directory or file name
`type`	either "file" or "directory"
`error`	When TRUE (default), the function returns an error when a variable is missing. Otherwise it returns a warning.

`check_single_posix` *Check if the object is a single POSIX*

Description

Check if the object is a single POSIX

Usage

```
check_single_posix(x, name = "x", past = FALSE)
```

Arguments

`x`	the object to check
`name`	the name of the object to use in the error message
`past`	Should the function throw an error when x is in the future? Default is `FALSE`.

Value

The function gives the single POSIX back. It throws an error when the input is not a single character.

Examples

```
check_single_posix(Sys.time())
```

`check_single_probability` *Check if the object is a single probability*

Description

Check if the object is a single probability

Usage

```
check_single_probability(x, name = "x")
```

Arguments

`x`	the object to check
`name`	the name of the object to use in the error message

Value

The function gives the single probability back. It throws an error when the input is not a single probability.

Examples

```
check_single_probability(0.5)
```

`connect_nbn` *Open a trusted connection to the NBN database*

Description

Open a trusted connection to the NBN database

Usage

```
connect_nbn()
```

`connect_result` *Opens an ODBC connection to the 'results' database*

Description

Opens an ODBC connection to the 'results' database

Usage

```
connect_result(username, password, develop = TRUE)
```

Arguments

`username`	the username to connect to the database.
`password`	the password for the username.
`develop`	Logical value. Indicates the location of the results database

connect_ut_db — *connect to the unit test database*

Description

connect to the unit test database

Usage

```
connect_ut_db(
  host = "localhost",
  dbname = "n2kunittest",
  user = "unittest_analysis",
  password = "unittest",
  port = 5432,
  ...
)
```

Arguments

`host`	Host name and port number of PostgreSQL database.
`dbname`	Database name.
`user`	User name and password.
`password`	User name and password.
`port`	Port number of database. Defaults to 5432
`...`	arguments past to `DBI::dbConnect()`.

cut_date — *Split dates into periods within each year*

Description

The periods are defined by a day and month. The same day from different years with be in the same period.

Usage

```
cut_date(x, dm, include_last = TRUE)
```

Arguments

`x`	the dates in POSIXt or Date format.
`dm`	the breakpoints of the periods in ’day-month’ format.
`include_last`	Should the last period include the last day? Defaults to `TRUE`.

Examples

```
x <- as.POSIXct(
    c(
    "2015-01-01", "2014-01-02", "2013-01-03", "2012-01-31", "2011-02-01",
    "2012-12-31"
   )
)
cut_date(x, dm = c("1-1", "1-2", "1-3"))
```

`get_nbn_key` *Get the NBN key of a species*

Description

Get the NBN key of a species

Usage

```
get_nbn_key(name, language = "la", channel, authority = FALSE)
```

Arguments

`name`	a vector of species names to check
`language`	The language to use. Defaults to "la" 'scientific name"
`channel`	An open RODBC channel to the NBN database
`authority`	Do the species names include authority?

`get_nbn_key_multi` *Try multiple languages to get a matching NBN key*

Description

Try multiple languages to get a matching NBN key

Usage

```
get_nbn_key_multi(species, orders = c("la", "nl", "en"), channel)
```

Arguments

`species`	A data.frame with the name of species in one or more languages
`orders`	the order in which the languages are tried to get a matching NBN key.
`channel`	An open RODBC channel to the NBN database

`get_nbn_name` *Get the name associated with an NBN key*

Description

Get the name associated with an NBN key

Usage

```
get_nbn_name(nbn_key, channel)
```

Arguments

`nbn_key`	A vector with NBN keys
`channel`	An open RODBC channel to the NBN database

`gitConnection-class` *The gitConnection class*

Description

The gitConnection class

Open a git connection

Usage

```
git_connection(repo_path, key, username, password, commit_user, commit_email)
```

Arguments

`repo_path`	The path of the root of the repository
`key`	Optional: the path to a private ssh key. The public key is assumed to have the same path with a '.pub' extension. Using in case of ssh authentication.
`username`	The optional username used in case of https authentication. Ignored when `key` is provided.
`password`	The password required for the ssh key or the username. Should be missing when the ssh-key doesn't require a password.
`commit_user`	the name of the user how will commit
`commit_email`	the email of the user how will commit

Slots

`Repository` a git repository

`Credentials` the credentials for the repository

`CommitUser` the name of the user how will commit

`CommitEmail` the email of the user how will commit

`git_connect`	*Returns the path of the datasource within the git repository*

Description

The details are stored in the results database.

Usage

```
git_connect(
  data_source_name,
  channel,
  type = c("ssh", "https"),
  username = character(0),
  password = character(0),
  commit_user,
  commit_email
)
```

Arguments

`data_source_name`	The name of the data source
`channel`	the ODBC channel to the database with the connection strings
`type`	Use 'ssh' or 'https' for authentication
`username`	the username in case the ConnectMethod is `"Credentials supplied by the user running the report"`. Ignored in all other cases.
`password`	the password to be used in combination with the username.
`commit_user`	the name of the user how will commit
`commit_email`	the email of the user how will commit

`is_chartor`	*Test if the argument is either character or factor*

Description

Test if the argument is either character or factor

Usage

```
is_chartor(x)
```

Arguments

`x`	the object to check

`match_nbn_key` *Merge NBN keys into a species dataframe*

Description

Merge NBN keys into a species dataframe

Usage

```
match_nbn_key(species, nbn_key, variable)
```

Arguments

`species`	a data.frame with the species names
`nbn_key`	a data.frame with the NBN keys
`variable`	the name of the variable of species to match with InputName from nbn_key

`odbc_connect` *connect to a data source through ODBC*

Description

The connection string is stored in the results database.

Usage

```
odbc_connect(data_source_name, username, password, channel)
```

Arguments

`data_source_name`	The name of the data source
`username`	the username in case the ConnectMethod is `"Credentials supplied by the user running the report"`. Ignored in all other cases.
`password`	the password to be used in combination with the username.
`channel`	the ODBC channel to the database with the connection strings

`odbc_get_id` *Get the id of the matching records*

Description

Get the id of the matching records

Usage

```
odbc_get_id(table, ..., schema = "public", channel, id_variable = "id")
```

Arguments

`table`	The name of the table
`...`	arguments passed to `filter()`.
`schema`	The schema of the table. Defaults to public
`channel`	the open dplyr connection to the database.
`id_variable`	name of the id variable

`odbc_get_multi_id` *Get the corresponding id's*

Description

Get the corresponding id's

Usage

```
odbc_get_multi_id(
  data,
  id_field,
  merge_field,
  table,
  channel,
  create = FALSE,
  select = TRUE,
  rows_at_time = 1000
)
```

Arguments

`data`	the data.frame
`id_field`	the id fields
`merge_field`	the merge fields
`table`	The name of the table
`channel`	the open dplyr connection to the database.
`create`	When `TRUE`, the function creates unmatching records AND updates attributes. Defaults to `FALSE`.
`select`	Return the matching ID's when `TRUE`. Returns invisible `NULL` when `FALSE`. select = `FALSE` is only relevant in combination with create = `TRUE`.
`rows_at_time`	Number of rows to insert in one SQL statement

Value

a data.frame with data and the id's

`odbc_insert` *Append a data.frame to a table through an ODBC connection*

Description

Append a data.frame to a table through an ODBC connection

Usage

```
odbc_insert(
  data,
  table,
  channel,
  schema = "dbo",
  append = TRUE,
  rows_at_time = 1000
)
```

Arguments

`data`	the data.frame
`table`	The name of the table
`channel`	the open dplyr connection to the database.
`schema`	The schema of the table. Defaults to public
`append`	Append the data or overwrite existing rows?
`rows_at_time`	Number of rows to insert in one SQL statement

Value

The status of the SQL INSERT for each row in returned but invisible.

`read_object_environment` *Read an object from an environment*

Description

Read an object from an environment

Usage

```
read_object_environment(object, env, warn = TRUE)
```

Arguments

`object`	the name of the object
`env`	the environment
`warn`	Issue a warning if the object is not found in the environment. Defaults to TRUE

Value

the object or `NULL` is the object doesn't exists in the environment

Examples

```
object <- "test"
value <- TRUE
env <- new.env()
assign(x = object, value = value, envir = env)
read_object_environment(object, env)
```

Index