Package 'EDCimport'

June 24, 2025

Version 0.6.0

Title Import Data from EDC Software

Description A convenient toolbox to import data exported from Electronic Data Capture (EDC) software 'TrialMaster'.

License GPL-3

URL https://github.com/DanChaltiel/EDCimport, https://danchaltiel.github.io/EDCimport/

BugReports https://github.com/DanChaltiel/EDCimport/issues

Depends R (>= 3.6.0)

Imports cli, dplyr, forcats, fs, glue, ggplot2, haven, lubridate, purrr, readr, rlang, scales, stats, stringr, tibble, tidyr, tidyselect, utils, lifecycle

Suggests bslib, callr, crosstable, DT, gtools, htmlwidgets, janitor, knitr, openxlsx, patchwork, plotly, quarto, rmarkdown, rstudioapi, testthat (>= 3.1.8), shiny, usethis, vdiffr, withr

Encoding UTF-8

RoxygenNote 7.3.2

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first local, trialmaster, utils

VignetteBuilder quarto

NeedsCompilation no

Author Dan Chaltiel [aut, cre] (ORCID: <https://orcid.org/0000-0003-3488-779X>)

Maintainer Dan Chaltiel <dan.chaltiel@gmail.com>

Repository CRAN

Date/Publication 2025-06-24 12:30:02 UTC

Contents

`assert_no_duplicate` *Assert that a dataframe has one row per patient*

Description

Check that there is no duplicate on the column holding patient ID in a pipeable style.
Mostly useful after joining two datasets.

Usage

```
assert_no_duplicate(df, by = NULL, id_col = get_subjid_cols())
```

Arguments

`df`	a dataframe
`by`	*(optional)* grouping columns
`id_col`	the name of the columns holding patient ID

Value

the `df` dataset, unchanged

Examples

```
## Not run:
#without duplicate => no error, continue the pipeline
tibble(subjid=c(1:10)) %>% assert_no_duplicate() %>% nrow()

#with duplicate => throws an error
tibble(subjid=c(1:10, 1:2)) %>% assert_no_duplicate() %>% nrow()

#By groups
df = tibble(subjid=rep(1:10, 4), visit=rep(c("V1", "V2"), 2, each=10),
            group=rep(c("A", "B"), each=20))
df %>% assert_no_duplicate() #error
df %>% assert_no_duplicate(by=c(visit, group)) #no error

## End(Not run)
```

edc_clean_names *Clean up the names of all datasets*

Description

Clean the names of all the datasets in the database. By default, it converts names to lowercase letters, numbers, and underscores only.

Usage

```
edc_clean_names(database, clean_fun = NULL)
```

Arguments

database	an edc_database object, from `read_trialmaster()` or other EDCimport reading functions.
clean_fun	a cleaning function to be applied to column names.

Value

an edc_database object

Examples

```
#db = read_trialmaster("filename.zip", pw="xx")
db = edc_example() %>%
  edc_clean_names()
names(db$enrol)
```

edc_crf_plot *Show the current CRF status distribution*

Description

Generate a barplot showing the distribution of CRF status (Complete, Incomplete, ...) for each dataset of the database.

Usage

```
edc_crf_plot(
  crfstat_col = "CRFSTAT",
  ...,
  details = FALSE,
  pal = edc_pal_crf(),
  reverse = FALSE,
  x_label = "{dataset}",
```

```
  treat_as_worst = NULL,
  datasets = get_datasets(),
  lookup = edc_lookup()
)

edc_pal_crf()
```

Arguments

crfstat_col	the column name of the CRF status
...	unused
details	whether to show all the CRF status levels. When FALSE (default), recode the status into "Complete", "Incomplete", or "No Data".
pal	the palette, defaulting to the helper EDCimport:::edc_pal_crf(). The names give the CRF status levels, from "best" to "worst". The plot is ordered by the "worst" level.
reverse	whether to reverse the CRF status level order.
x_label	a glue pattern determining the tick label in the x axis. Available variables are the ones of edc_lookup(): c("dataset", "nrow", "ncol", "n_id", "rows_per_id", "crfname").
treat_as_worst	a regex for levels that should be treated as worst in the ordering.
datasets, lookup	internal

Value

a ggplot

Source

```
ggsci:::ggsci_db$lancet[["lanonc"]] %>% dput()
```

Examples

```
## Not run:
#import a TM database and use load_database(), then:
edc_crf_plot() + ggtitle(date_extraction)
edc_crf_plot(reverse=TRUE)
edc_crf_plot(details=TRUE, treat_as_worst="No Data")
edc_crf_plot(x_label="{crfname} (N={n_id}, n={nrow})")

p = edc_crf_plot(details=TRUE)
p$data$crfstat %>% unique()
#> [1] "Incomplete"        "No Data Locked"    "No Data"           "Signed"
#> [5] "Partial Monitored" "Monitored"         "Complete Locked"   "Complete"

## End(Not run)
```

edc_database *EDCimport Database*

Description

This class of object represents a database, as the result of an EDCimport reading function. It has its own `print()` method.

Functions returning `edc_database` objects

As per now, reading functions are: `read_trialmaster()`, `read_all_sas()`, `read_all_xpt()`, and `read_all_csv()`.

Structure

While it is not usually useful to query them, an `edc_database` object is a named list containing:

- all the datasets from the source files
- `datetime_extraction` and `date_extraction` the inferred date of data extraction
- `.lookup` a temporary copy of the lookup table

See Also

`read_trialmaster()`

edc_data_warn *Standardized warning system*

Description

When checking your data, filter your dataset to get only problematic rows.
Then, use either:

- `edc_data_warn()` to generate a standardized warning that can be forwarded to the datamanager.
- `edc_data_stop()` to abort the script if the problem is too serious.

Each time edc_data_warn is used, the warning is saved internally so that a summary of all your warnings can be retrieved using edc_data_warnings.
The result can be saved into an Excel file using `save_edc_data_warnings()`.

Usage

```
edc_data_warn(
  df,
  message,
  ...,
  issue_n = "xx",
  max_subjid = 5,
  csv_path = FALSE,
  envir = parent.frame(),
  col_subjid = get_subjid_cols()
)

edc_data_stop(df, message, ..., issue_n, max_subjid, csv_path, envir, col_subjid)

edc_data_warnings()
```

Arguments

df	the filtered dataframe
message	the message. Can use cli formats. df can be accessed using the .data special keyword (see example)
...	unused
issue_n	identifying row number
max_subjid	max number of subject ID to show in the message
csv_path	a path to save df in a csv file that can be shared with the DM for more details.
envir	the environment to evaluate message in.
col_subjid	column name for subject ID. Set to NULL to ignore.

Value

df invisibly

Examples

```
library(dplyr)
db = edc_example()
load_database(db)
enrol %>%
  filter(age>70) %>%
  edc_data_warn("Age should not be >70", issue_n=1)

enrol %>%
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", issue_n=2)

data1 %>%
  filter(n()>1, .by=subjid) %>%
 edc_data_warn("There are duplicated patients in `data1` ({nrow(.data)} rows)", issue_n=3)
```

```
enrol %>%
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", issue_n=NULL)

edc_data_warnings()

## Not run:
enrol %>%
  filter(age<25) %>%
  edc_data_warn("Age should not be <25", csv_path="check/check_age_25.csv")

enrol %>%
  filter(age<25) %>%
  edc_data_stop("Age should *never* be <25")

## End(Not run)
```

`edc_db_to_excel` *Save the database as an Excel file*

Description

Because RStudio is not very good at showing data, it can be more convenient to browse the database using MS Excel. This function turns the whole TM export (or any named list of datasets) into an Excel workbook, with one tab for each dataset.
Use `edc_db_to_excel()` to create the file and `edc_browse_excel()` to open it.

Usage

```
edc_db_to_excel(
  filename = tempfile(fileext = ".xlsx"),
  ...,
  datasets = get_datasets(),
  overwrite = FALSE,
  open = FALSE
)

edc_browse_excel()
```

Arguments

`filename`	the path to the Excel output file. Default to a temporary file. Use the special value `TRUE` to save in "data/database_{date_extraction}.xlsx".
`...`	unused
`datasets`	a named list of dataframes. Default to the TM export.
`overwrite`	whether to overwrite any existing file. Default to `FALSE`.
`open`	whether to open the Excel file afterward. Default to `FALSE`.

Value

nothing

Examples

```
## Not run:
  db = edc_example()
  load_database(db)
  edc_db_to_excel() #default arguments are usually OK
  edc_db_to_excel(filename=TRUE)

## End(Not run)
```

---

edc_example *Example database*

Description

A list of tables that simulates the extraction of a clinical database. Used in `EDCimport` examples and tests.

Usage

```
edc_example(N = 50, seed = 42, outdated = FALSE)
```

Arguments

`N`	the number of patients
`seed`	the random seed
`outdated`	whether to simulate times after the data extraction date

Value

A list of tables of class `edc_database`.

edc_find_value *Search the whole database*

Description

Find a keyword in columns or values, in all the datasets of the database.

Usage

```
edc_find_value(
  keyword,
  ignore_case = TRUE,
  data = get_datasets(),
  lookup = edc_lookup()
)

edc_find_column(keyword, ignore_case = TRUE, lookup = edc_lookup())
```

Arguments

keyword	The keyword to search for. Regular expressions are only supported in edc_find_column.
ignore_case	Logical. If TRUE (default), the search will ignore case differences.
data	A list of datasets.
lookup	A lookup table.

Value

a tibble

Examples

```
db = edc_example()
load_database(db)

edc_find_value("respi")
edc_find_value(2010)

edc_find_column("ad")
edc_find_column("date")
#with regex
edc_find_column("\\d")
edc_find_column("\\(") #you need to escape special characters
```

edc_inform_code *Shows how many code you wrote*

Description

Shows how many code you wrote

Usage

```
edc_inform_code(main = "main.R", Rdir = "R/")
```

Arguments

main	the main R file, which sources the other ones
Rdir	the R directory, where sourced R files are located

Value

Nothing

edc_left_join *Join within the EDCimport framework*

Description

Perform a join with default by to the Subject ID and default suffix to the name of the y dataset. See [dplyr::mutate-joins] for the description of the join logic.

Usage

```
edc_left_join(
  x,
  y,
  by = NULL,
  suffix = NULL,
  cols = everything(),
  remove_dups = FALSE
)
```

Arguments

x, y	Data frames to join
by	The key to join on, as character. Defaults to get_subjid_cols()
suffix	The disambiguation suffix. Defaults to the actual name of the y dataset.
cols	<tidy-select> The columns to select in y before joining.
remove_dups	Whether to remove columns in y that already exist in x.

Value

a dataframe

Examples

```
db = edc_example()
load_database(db)
data1$common = data2$common = "Common"
x = enrol %>%
  edc_left_join(data2) %>%
  edc_right_join(data1)

#crfname get a suffix, common
names(x)
```

---

edc_lookup *Retrieve the lookup table from options*

Description

Retrieve the lookup table from options

Usage

```
edc_lookup(..., check = TRUE)
```

Arguments

...	passed on to dplyr::arrange()
check	whether to check for internal consistency

Value

the lookup dataframe summarizing the database import

Examples

```
db = edc_example()
load_database(db)
edc_lookup()
edc_lookup(dataset)
```

edc_options *Set global options for* EDCimport

Description

Use this function to manage your EDCimport parameters globally while taking advantage of auto-completion.
Use edc_peek_options() to see which option is currently set and edc_reset_options() to set all options back to default.

Usage

```
edc_options(
  ...,
  trialmaster_pw,
  path_7zip,
  edc_lookup,
  edc_subjid_ref,
  edc_plotly,
  edc_fct_yesno,
  edc_cols_subjid,
  edc_cols_meta,
  edc_cols_id,
  edc_cols_crfname,
  edc_meta_cols_pct,
  edc_warn_max_subjid,
  edc_read_verbose,
  edc_correction_verbose,
  edc_get_key_cols_verbose,
  edc_lookup_overwrite_warn,
  .local = FALSE
)
```

Arguments

...	unused
trialmaster_pw	the password of the trialmaster zip archive. For instance, you can use edc_options(trialmaster_pw="my in the console once per session, so that you don't have to write the password in clear in your R code
path_7zip	the path to the 7zip executable. Default to "C:/Program Files/7-Zip/".
edc_lookup	**(Internal)** a reference to the lookup table (usually .lookup). Should usually not be changed manually.
edc_subjid_ref	**used in edc_warn_patient_diffs** the vector of the reference subject IDs. You should usually write edc_options(edc_subjid_ref=enrolres$subjid).
edc_plotly	**used in edc_swimmerplot** whether to use plotly to visualize the plot.

edc_fct_yesno **used in fct_yesno** list of values to be considered as Yes/No values. Defaults to `get_yesno_lvl()`.

edc_cols_subjid, edc_cols_meta
: the name of the columns holding the subject id (default to `c("ptno", "subjid")`) and the CRF form name (default to `c("crfname")`). It is case-insensitive.

edc_cols_id, edc_cols_crfname
: deprecated

edc_meta_cols_pct
: The minimal proportion of datasets a column has to reach to be considered "meta"

edc_warn_max_subjid
: The max number of subject IDs to show in edc_data_warn

edc_read_verbose, edc_correction_verbose, edc_get_key_cols_verbose
: the verbosity of the output of functions read_trialmaster and read_all_xpt, and manual_correction. For example, set `edc_options(edc_read_verbose=0)` to silence the first 2.

edc_lookup_overwrite_warn
: default to TRUE. Whether there should be warning when overwriting `.lookup` (like when reading 2 databases successively)

.local
: if TRUE, the effect will only apply to the local frame (internally using `rlang::local_options()`)

Value

Nothing, called for its side effects

`edc_patient_gridplot` *Patient gridplot*

Description

Draw a gridplot giving, for each patient and each dataset, whether the patient is present in the dataset. Data are drawn from get_datasets.

Usage

```
edc_patient_gridplot(
  sort_rows = TRUE,
  sort_cols = TRUE,
  gradient = FALSE,
  axes_flip = FALSE,
  show_grid = TRUE,
  preprocess = NULL,
  palette = c(Yes = "#00468BFF", No = "#ED0000FF"),
  datasets = get_datasets(),
  lookup = edc_lookup()
)
```

Arguments

sort_rows	whether to sort patients from "present in most datasets" to "present in least datasets"
sort_cols	whether to sort datasets from "containing the most patients" to "containing the least patients"
gradient	whether to add a color gradient for repeating measures
axes_flip	whether to flip the axes, so that patients are on the Y axis and datasets on the X axis
show_grid	whether to show the grid
preprocess	a function to preprocess the patient ID, e.g. `as.numeric`, or a custom function with string replacement
palette	the colors to use
datasets, lookup	internal

Value

a `ggplot` object

Examples

```
## Not run:
  tm = read_trialmaster("path/to/archive.zip")
  load_database(db)
  edc_patient_gridplot(sort_rows=FALSE, sort_cols=FALSE)
  edc_patient_gridplot(axes_flip=TRUE, show_grid=TRUE,
                       preprocess=~str_remove(.x, "\\D*")) #remove all non-digits

## End(Not run)
```

---

edc_peek_options *See which* `EDCimport` *option is currently set*

Description

See which `EDCimport` option is currently set

Usage

```
edc_peek_options(keep_null = FALSE)
```

Arguments

keep_null	set to TRUE to get a list

Value

A named list of EDCimport options

edc_population_plot *Plot the populations*

Description

In a RCT, you usually have several populations of analysis, and this function allow to show which patient is in which population graphically.

Usage

```
edc_population_plot(x, id_per_row = 50, ref = "first")
```

Arguments

x	a named list of subject ID, as numeric or factor.
id_per_row	number of patients per rows.
ref	the whole population. Default to the first member of x.

Value

a ggplot

Examples

```
#in real word code, use filter and pull to get these vectors
pop_total = c(1:180) %>% setdiff(55) #screen failure, no patient 55
pop_itt = pop_total %>% setdiff(10) #patient 10 has had the wrong treatment
pop_safety = pop_total %>% setdiff(c(40,160)) #patients 40 and 160 didn't receive any treatment
pop_m_itt = pop_total %>% setdiff(c(40,160,80)) #patient 80 had a wrong inclusion criterion
pop_evaluable = pop_total %>% setdiff(c(40,160,101,147,186)) #patients with no recist evaluation

l = list(
  "Total population"=pop_total,
  "ITT population"=pop_itt,
  "Safety population"=pop_safety,
  "mITT population"=pop_m_itt,
  "Evaluable population"=pop_evaluable
)
edc_population_plot(l)
edc_population_plot(l[-1], ref=pop_total)
edc_population_plot(l, ref=1:200)
edc_population_plot(l, id_per_row=60)
```

edc_reset_options *Reset all* EDCimport *options*

Description

Reset all EDCimport options

Usage

```
edc_reset_options(
  except = c("edc_lookup", "trialmaster_pw", "path_7zip"),
  quiet = FALSE
)
```

Arguments

except	options that are not reset by default
quiet	set to TRUE to remove the message.

Value

Nothing, called for its side effects

edc_split_mixed *Split mixed datasets*

Description

Split mixed tables, i.e. tables that hold both long data (N values per patient) and short data (one value per patient, duplicated on N lines), into one long table and one short table.

Usage

```
edc_split_mixed(
  database,
  datasets = everything(),
  ...,
  ignore_cols = NULL,
  verbose = FALSE
)
```

Arguments

database	an edc_database object, from read_trialmaster() or other EDCimport reading functions.
datasets	<tidy-select> datasets to split in the database
...	not used, ensure arguments are named
ignore_cols	columns to ignore in long tables. Default to getOption("edc_cols_crfname", "CRFNAME"). Case-insensitive. Avoid splitting tables for useless columns.
verbose	whether to print informations about the process.

Value

an edc_database object

Examples

```
#db = read_trialmaster("filename.zip", pw="xx")
db = edc_example() %>%
  edc_split_mixed(c(ae, starts_with("long")),
                  ignore_cols="crfstat")

names(db)
edc_lookup()

db$ae #`aesoc`, `aegr`, and `sae` are long, but `n_ae` is short

db$ae_short
db$ae_long
```

---

edc_swimmerplot *Swimmer plot of all dates columns*

Description

Join all tables on id with only date columns to build a ggplot (or a plotly if plotly=TRUE) showing all dates for each subject.
This allows outliers to be easily identified.

Usage

```
edc_swimmerplot(
  ...,
  group = NULL,
  origin = NULL,
  include = NULL,
  exclude = NULL,
  id_subset = "all",
  id_sort = FALSE,
```

```
  id_cols = get_subjid_cols(),
  time_unit = c("days", "weeks", "months", "years"),
  aes_color = c("variable", "label"),
  plotly = getOption("edc_plotly", FALSE),
  id = "deprecated",
  id_lim = "deprecated",
  .lookup = "deprecated"
)
```

Arguments

`...`	not used
`group`	a grouping variable, given as "dataset$column".
`origin`	a variable to consider as time 0, given as "dataset$column".
`include, exclude`	a character vector of variables to exclude/include, in the form `dataset$column`. Can be a regex (apart from $ symbols that will be automatically escaped). Case-insensitive.
`id_subset`	the subjects to include in the plot.
`id_sort`	whether to sort subjects by date (or time).
`id_cols`	the subject identifiers columns. Identifiers be coerced as numeric if possible. See get_subjid_cols if needed.
`time_unit`	if `origin!=NULL`, the unit to measure time. One of `c("days", "weeks", "months", "years")`.
`aes_color`	either `variable` ("{dataset} - {column}") or `label` (the column label).
`plotly`	whether to use `{plotly}` to get an interactive plot.
`id`	deprecated
`id_lim`	deprecated
`.lookup`	deprecated

Value

either a `plotly` or a `ggplot`

Examples

```
#db = read_trialmaster("filename.zip", pw="xx")
db = edc_example()
load_database(db)
edc_swimmerplot(id_lim=c(5,45))

edc_swimmerplot(origin="enrol$enrol_date", time_unit="months",
                include=c("data1", "data3"),
                exclude=c("DATA1$DATE2", "data3$date\\d\\d"),
                id_sort=TRUE)
```

```
edc_swimmerplot(group="enrol$arm", id_subset=1:10, aes_color="label")

## Not run:
p = edc_swimmerplot(plotly=TRUE)
save_plotly(p, "edc_swimmerplot.html")

## End(Not run)
```

`edc_unify_subjid` *Harmonize the subject ID of the database*

Description

Turns the subject ID columns of all datasets into a factor containing levels for all the subjects of the database. Avoid problems when joining tables, and some checks can be performed on the levels. See `vignette("postprocessing")` for a real-life case.

Usage

```
edc_unify_subjid(
  database,
  preprocess = NULL,
  mode = c("factor", "numeric"),
  col_subjid = NULL
)
```

Arguments

`database`	an edc_database object, from `read_trialmaster()` or other EDCimport reading functions.
`preprocess`	an optional function to modify the subject ID column (at the character level). Default behavior is only to remove trailing zeros if numeric.
`mode`	the output type of the subject ID columns
`col_subjid`	names of the subject ID columns (as character)

Value

database, with subject id modified

Examples

```
db = edc_example()
db$enrol$subjid %>% head()  #double vector

db2 = edc_unify_subjid(db)
db2$enrol$subjid %>% head() #factor with 50 levels

db3 = edc_unify_subjid(db, preprocess=function(x) paste0("#", x))
```

```
db3$enrol$subjid %>% head()

#use numeric mode to get a numeric output
db4 = edc_unify_subjid(db, preprocess=function(x) as.numeric(x)+1, mode="numeric")
db4$enrol$subjid %>% head()
```

---

edc_viewer *Shiny data explorer*

Description

Run a Shiny application that allows to browse the datasets.

Usage

```
edc_viewer(data = NULL, background = TRUE, port = 1209)
```

Arguments

data	A list of dataframes to view. If NULL, defaults to the last datasets loaded using EDCimport functions.
background	Whether the app should run in a background process.
port	The TCP port that the application should listen on.

---

edc_warn_extraction_date
Warn if extraction is too old

Description

Warn if extraction is too old

Usage

```
edc_warn_extraction_date(max_days = 30)
```

Arguments

max_days	the max acceptable age of the data

Value

nothing

Examples

```
db = edc_example()
load_database(db)
edc_warn_extraction_date()
```

`edc_warn_patient_diffs` *Check the validity of the subject ID column*

Description

Compare a subject ID vector to the study's reference subject ID (usually something like `enrolres$subjid`), and warn if any patient is missing or extra.
`check_subjid()` is the old, deprecated name.

Usage

```
edc_warn_patient_diffs(
  x,
  ref = getOption("edc_subjid_ref"),
  issue_n = "xx",
  data_name = NULL,
  col_subjid = get_subjid_cols()
)
```

Arguments

`x`	the subject ID vector to check, or a dataframe which ID column will be guessed
`ref`	the reference for subject ID. Should usually be set through `edc_options(edc_subjid_ref=xxx)`. See example.
`issue_n`	identifying row number
`data_name`	the name of the data (for the warning message)
`col_subjid`	name of the subject ID column if `x` is a dataframe.

Value

nothing, called for errors/warnings

Examples

```
db = edc_example()
load_database(db)
options(edc_subjid_ref=enrol$subjid)
#usually, you set something like:
#options(edc_subjid_ref=enrolres$subjid)
edc_warn_patient_diffs(data1)
data1 %>% dplyr::filter(subjid>1) %>% edc_warn_patient_diffs(issue_n=NULL)
edc_warn_patient_diffs(c(data1$subjid, 99, 999))
```

`fct_yesno` *Format factor levels as Yes/No*

Description

Format factor levels as arbitrary values of Yes/No (with Yes always first) while **leaving untouched** all vectors that contain other information.

Usage

```
fct_yesno(
  x,
  input = list(yes = c("Yes", "Oui"), no = c("No", "Non"), na = c("NA", "")),
  output = c("Yes", "No"),
  strict = FALSE,
  mutate_character = TRUE,
  fail = TRUE
)
```

Arguments

`x`	a vector of any type/class.
`input`	list of values to be considered as "yes", "no", and `NA`.
`output`	the output factor levels.
`strict`	whether to match the input strictly or use stringr::str_detect to find them. Can also be "ignore_case" to just ignore the case.
`mutate_character`	whether to turn characters into factor.
`fail`	whether to fail if some levels cannot be recoded to yes/no.

Value

a factor, or `x` untouched.

Examples

```
fct_yesno(c("No", "Yes")) #levels are in order

set.seed(42)
N=6
x = tibble(
  a=sample(c("Yes", "No"), size=N, replace=TRUE),
  b=sample(c("Oui", "Non"), size=N, replace=TRUE),
  c=sample(0:1, size=N, replace=TRUE),
  d=sample(c(TRUE, FALSE), size=N, replace=TRUE),
  e=sample(c("1-Yes", "0-No", "2-NA"), size=N, replace=TRUE),
```

```
  y=sample(c("aaa", "bbb", "ccc"), size=N, replace=TRUE),
  z=1:N,
)

x
#y and z are left untouched (or throw an error if fail=TRUE)
sapply(x, fct_yesno, fail=FALSE, simplify=FALSE)

# as "1-Yes" is not in `input`, x$e is untouched/fails if strict=TRUE
fct_yesno(x$e)
fct_yesno(x$e, strict=TRUE, fail=FALSE)
fct_yesno(x$e, output=c("Ja", "Nein"))
```

`get_common_cols` *Get columns that are common to multiple datasets*

Description

[Experimental] Attempt to list all columns in the database and group the ones that are common to some datasets. Useful to find keys to pivot or summarise data.

Usage

```
get_common_cols(lookup = edc_lookup(), min_datasets = 3)

## S3 method for class 'common_cols'
summary(object, ...)
```

Arguments

`lookup`	the lookup table, default to `edc_lookup()`
`min_datasets`	the minimal number of datasets to be considered
`object`	an object of class "common_cols"
`...`	unused

Value

a tibble of class "common_cols"

Examples

```
db = edc_example()
load_database(db)
x = get_common_cols(min_datasets=1)
x
summary(x)
```

get_datasets *Retrieve the datasets as a list of data.frames*

Description

Get the datasets from the lookup table as a list of data.frames.

Usage

```
get_datasets(lookup = edc_lookup(), envir = parent.frame())
```

Arguments

lookup	the lookup table
envir	(internal use)

Value

a list of all datasets

lastnews_table *Get a table with the latest date for each patient*

Description

This function search for date columns in every tables and returns the latest date for each patient with the variable it comes from. Useful in survival analysis to get the right censoring time.

Usage

```
lastnews_table(
  except = NULL,
  with_ties = FALSE,
  show_delta = FALSE,
  numeric_id = TRUE,
  prefer = NULL,
  regex = FALSE,
  warn_if_future = TRUE
)
```

Arguments

except	the datasets/columns that should not be searched. Example: a scheduled visit for which the patient may have died before attending should not be considered.
with_ties	in case of tie, whether to return the first origin (FALSE) or all the origins that share this tie (TRUE).
show_delta	whether to compute the difference between the last prefer date and the actual last date
numeric_id	set to FALSE if the patient ID column is not numeric
prefer	preferred origins in the event of a tie. Usually the followup table.
regex	whether to consider except and prefer as regex.
warn_if_future	whether to show a warning about dates that are after the extraction date. Can also be a csv file path to save the warning as csv (see csv_path argument in edc_data_warn).

Value

a dataframe

Examples

```
db = edc_example()
load_database(db)
lastnews_table()
lastnews_table(except="data3")
lastnews_table(except="data3$date9")
lastnews_table(prefer="date10", show_delta=TRUE)
lastnews_table() %>%
  dplyr::count(origin = glue::glue("{origin_data}${origin_col}"),
  sort=TRUE)

csv_file = tempfile(fileext=".csv")
lastnews_table(prefer="date9", warn_if_future=csv_file)
```

---

load_database *Load a list in an environment*

Description

Load a list in an environment

Usage

```
load_database(db, env = parent.frame(), remove = TRUE)
```

Arguments

db	an edc_database object (to be fair, any list would do)
env	the environment onto which the list should be loaded
remove	if TRUE, db will be removed from the environment afterward

Value

nothing, called for its side-effect

Examples

```
db = edc_example()
load_database(db, remove=FALSE)
print(db)
print(lengths(db))
```

---

manual_correction *Manual correction*

Description

[Experimental]

When finding wrong or unexpected values in an exported dataset, it can be useful to temporarily correct them by hard-coding a value. However, this manual correction should be undone as soon as the central database is updated with the correction.

- `manual_correction()` applies a correction in a specific dataset column location and throws an error if the correction is already in place. This check applies only once per R session so you can source your script without errors.
- `reset_manual_correction()` resets all checks. For instance, it is called by `read_trialmaster()`.

Usage

```
manual_correction(
  data,
  col,
  rows,
  wrong,
  correct,
  verbose = getOption("edc_correction_verbose", TRUE)
)

reset_manual_correction()
```

Arguments

data, col, rows	the rows of a column of a dataframe where the error lies
wrong	the actual wrong value
correct	the temporary correction value
verbose	whether to print informations (once)

Value

Nothing, used for side effects

Examples

```
library(dplyr)
x = iris %>% mutate(id=row_number(), .before=1) %>% as_tibble()
x$Sepal.Length[c(1,3,5)]

#1st correction is silent
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))
x$Sepal.Length[c(1,3,5)]

#further correction is silent
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))

#if the database is corrected, an error is thrown
## Not run:
reset_manual_correction()
x$Sepal.Length[c(1,3,5)] = c(5, 4, 3) #mimics db correction
manual_correction(x, Sepal.Length, rows=c(1,3,5),
                  wrong=c(5.1, 4.7, 5.0), correct=c(5, 4, 3))

## End(Not run)
```

---

read_all_csv *Read all* .csv *files in a directory*

Description

Read all .csv files in a directory, with labels if specified.

Usage

```
read_all_csv(
  path,
  ...,
  labels_from = NULL,
  format_file = NULL,
```

```
  subdirectories = FALSE,
  read_fun = "guess",
  datetime_extraction = "guess",
  verbose = getOption("edc_read_verbose", 1),
  clean_names_fun = NULL
)
```

Arguments

path	[character(1)] path to the directory containing .csv files.
...	unused
labels_from	[character(1)] path to the file containing the labels. See section "Labels file" below.
format_file	[character(1)] the path to the file that should be used to apply formats. See section "Format file" below. Use NULL to not apply formats.
subdirectories	[logical(1)] whether to read subdirectories
read_fun	[function] if "guess" doesn't work properly, a function to read the files in path, e.g. read.csv, read.csv2,...
datetime_extraction	[POSIXt(1)] the datetime of the data extraction. Default to the most common date of last modification in path.
verbose	[numeric(1)] one of c(0, 1, 2). The higher, the more information will be printed.
clean_names_fun	**[Deprecated]** use edc_clean_names() instead.

Value

a list containing one dataframe for each .csv file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Labels file

labels_from should contain the information about column labels. It should be a data file (.csv) containing 2 columns: one for the column name and the other for its associated label. Use options(edc_col_name="xxx", edc_col_label="xxx") to specify the names of the columns.

Format file

format_file should contain the information about SAS formats. It can be either:

- a procformat.sas file, containing the whole PROC FORMAT

- or a data file (`.csv` or `.sas7bdat`) containing 3 columns:
  - `FMTNAME` the SAS format name (repeated)
  - `START` the variable level
  - `LABEL` the label associated to the level

  You can get this datafile from SAS using `PROC FORMAT` with option `CNTLOUT`. Otherwise, you can use `options(edc_var_format_name="xxx", edc_var_level="xxx", edc_var_label="xxx")` to specify different column names.

See Also

Other EDCimport reading functions: `read_all_sas()`, `read_all_xpt()`, `read_trialmaster()`

Examples

```
# Create a directory with multiple csv files and a label lookup.
path = paste0(tempdir(), "/read_all_csv")
dir.create(paste0(path, "/subdir"), recursive=TRUE)
write.csv(iris, paste0(path, "/iris.csv"))
write.csv(mtcars, paste0(path, "/mtcars.csv"))
write.csv(mtcars, paste0(path, "/subdir/mtcars.csv"))
write.csv(airquality, paste0(path, "/airquality.csv"))
labs = c(iris, mtcars, airquality) %>% names()
write.csv(data.frame(name=labs, label=toupper(labs)), paste0(path, "/labels.csv"))


db = read_all_csv(path, labels_from="labels.csv", subdirectories=TRUE) %>%
  set_project_name("My great project")
db
edc_lookup()
```

---

read_all_sas *Read all* `.sas7bdat` *files in a directory*

Description

Read all `.sas7bdat` files in a directory. Formats (factors levels) can be applied from a `procformat.sas` SAS file, or from a format dictionary. See the "Format file" section below. Column labels are read directly from the `.sas7bdat` files.

Usage

```
read_all_sas(
  path,
  ...,
  format_file = "procformat.sas",
  subdirectories = FALSE,
  datetime_extraction = "guess",
  verbose = getOption("edc_read_verbose", 1),
  clean_names_fun = NULL
)
```

Arguments

path	[character(1)] the path to the directory containing all .sas7bdat files.
...	unused
format_file	[character(1)] the path to the file that should be used to apply formats. See section "Format file" below. Use NULL to not apply formats.
subdirectories	[logical(1)] whether to read subdirectories
datetime_extraction	[POSIXt(1)] the datetime of the data extraction. Default to the most common date of last modification in path.
verbose	[numeric(1)] one of c(0, 1, 2). The higher, the more information will be printed.
clean_names_fun	**[Deprecated]** use edc_clean_names() instead.

Value

a list containing one dataframe for each .xpt file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Format file

format_file should contain the information about SAS formats. It can be either:

- a procformat.sas file, containing the whole PROC FORMAT
- or a data file (.csv or .sas7bdat) containing 3 columns:
  - FMTNAME the SAS format name (repeated)
  - START the variable level
  - LABEL the label associated to the level

  You can get this datafile from SAS using PROC FORMAT with option CNTLOUT. Otherwise, you can use options(edc_var_format_name="xxx", edc_var_level="xxx", edc_var_label="xxx") to specify different column names.

See Also

Other EDCimport reading functions: read_all_csv(), read_all_xpt(), read_trialmaster()

Examples

```
# Create a directory with multiple sas files.
path = paste0(tempdir(), "/read_all_sas")
dir.create(paste0(path, "/subdir"), recursive=TRUE)
haven::write_sas(attenu, paste0(path, "/attenu.sas7bdat"))
haven::write_sas(mtcars, paste0(path, "/mtcars.sas7bdat"))
```

```
haven::write_sas(mtcars, paste0(path, "/subdir/mtcars.sas7bdat"))
haven::write_sas(esoph, paste0(path, "/esoph.sas7bdat"))

db = read_all_sas(path, format_file=NULL, subdirectories=TRUE) %>%
  set_project_name("My great project")
db
edc_lookup()
```

`read_all_xpt` *Read all* `.xpt` *files in a directory*

Description

Read all `.xpt` files in a directory (unzipped TrialMaster archive).
If `7zip` is installed, you should probably rather use `read_trialmaster()` instead.
Formats (factors levels) can be applied from a `procformat.sas` SAS file, or from a format dictionary. See the "Format file" section below. Column labels are read directly from the `.xpt` files.

Usage

```
read_all_xpt(
  path,
  ...,
  format_file = "procformat.sas",
  datetime_extraction = "guess",
  subdirectories = FALSE,
  verbose = getOption("edc_read_verbose", 1),
  clean_names_fun = NULL,
  directory = "deprecated",
  key_columns = "deprecated"
)
```

Arguments

`path`	`[character(1)]` the path to the directory containing all `.xpt` files.
`...`	unused
`format_file`	`[character(1)]` the path to the file that should be used to apply formats. See section "Format file" below. Use `NULL` to not apply formats.
`datetime_extraction`	`[POSIXt(1)]` the datetime of the data extraction. Default to the most common date of last modification in `path`.
`subdirectories`	`[logical(1)]` whether to read subdirectories

verbose [numeric(1)]
one of c(0, 1, 2). The higher, the more information will be printed.

clean_names_fun
[Deprecated] use edc_clean_names() instead.

directory deprecated in favor for path

key_columns deprecated

Value

a list containing one dataframe for each .xpt file in the folder, the extraction date (datetime_extraction), and a summary of all imported tables (.lookup).

Format file

format_file should contain the information about SAS formats. It can be either:

- a procformat.sas file, containing the whole PROC FORMAT
- or a data file (.csv or .sas7bdat) containing 3 columns:
  - FMTNAME the SAS format name (repeated)
  - START the variable level
  - LABEL the label associated to the level

  You can get this datafile from SAS using PROC FORMAT with option CNTLOUT. Otherwise, you can use options(edc_var_format_name="xxx", edc_var_level="xxx", edc_var_label="xxx") to specify different column names.

See Also

Other EDCimport reading functions: read_all_csv(), read_all_sas(), read_trialmaster()

Examples

```
# Create a directory with multiple .xpt files.
path = paste0(tempdir(), "/read_all_xpt")
dir.create(paste0(path, "/subdir"), recursive=TRUE)
haven::write_xpt(attenu, paste0(path, "/attenu.xpt"))
haven::write_xpt(mtcars, paste0(path, "/mtcars.xpt"))
haven::write_xpt(mtcars, paste0(path, "/subdir/mtcars.xpt"))
haven::write_xpt(esoph, paste0(path, "/esoph.xpt"))

db = read_all_xpt(path, format_file=NULL, subdirectories=TRUE) %>%
  set_project_name("My great project")
db
edc_lookup()
```

read_trialmaster *Read the* .zip *archive of a TrialMaster export*

Description

Import the .zip archive of a TrialMaster trial export as a list of dataframes. The archive filename should be leaved untouched as it contains the project name and the date of extraction.
Generate a .rds cache file for future reads.
If 7zip is not installed or available, use read_all_xpt() instead.

The TM export should be of type SAS Xport, with the checkbox "Include Codelists" ticked.

Usage

```
read_trialmaster(
  archive,
  ...,
  use_cache = "write",
  clean_names_fun = NULL,
  subdirectories = FALSE,
  pw = getOption("trialmaster_pw"),
  verbose = getOption("edc_read_verbose", 1),
  key_columns = "deprecated"
)
```

Arguments

archive	[character(1)] the path to the archive
...	unused
use_cache	[mixed(1): "write"] controls the .rds cache. If TRUE, read the cache if any or extract the archive and create a cache. If FALSE extract the archive without creating a cache file. Can also be "read" or "write".
clean_names_fun	**[Deprecated]** use edc_clean_names() instead.
subdirectories	[logical(1)] whether to read subdirectories
pw	[character(1)] The password if the archive is protected. To avoid writing passwords in plain text, it is probably better to use options(trialmaster_pw="xxx") instead though.
verbose	[numeric(1)] one of c(0, 1, 2). The higher, the more information will be printed.
key_columns	deprecated

Value

a list containing one dataframe for each `.xpt` file in the folder, the extraction date (`datetime_extraction`), and a summary of all imported tables (`.lookup`).

See Also

Other EDCimport reading functions: `read_all_csv()`, `read_all_sas()`, `read_all_xpt()`

---

save_edc_data_warnings

Save EDCimport warning to Excel

Description

Each time edc_data_warn is used, the warning is saved internally so that a summary can be retrieved using edc_data_warnings. This summary can then be saved into a `.xlsx` file using `save_edc_data_warnings()`.

Usage

```
save_edc_data_warnings(
  edc_warnings = edc_data_warnings(),
  path = "edc_data_warnings.xlsx",
  overwrite = TRUE,
  open = FALSE
)
```

Arguments

`edc_warnings`	the result of edc_data_warnings
`path`	a `.xlsx` file path
`overwrite`	If `TRUE`, overwrite any existing file.
`open`	If `TRUE`, overwrite any existing file.

Value

a logical(1), whether the file could be written, invisibly

save_plotly *Save a plotly to an HTML file*

Description

Save a plotly to an HTML file

Usage

```
save_plotly(p, file, ...)
```

Arguments

p	a plot object (plotly or ggplot)
file	a file path to save the HTML file
...	passed on to htmlwidgets::saveWidget

Value

nothing, used for side effect

Examples

```
## Not run:
db = edc_example()
load_database(db)
p = edc_swimmerplot(id_lim=c(5,45))
save_plotly(p, "graph/swimplots/edc_swimmerplot.html", title="My Swimmerplot")

## End(Not run)
```

save_sessioninfo *Save* sessionInfo() *output*

Description

Save sessionInfo() output into a text file.

Usage

```
save_sessioninfo(path = "check/session_info.txt", with_date = TRUE)
```

Arguments

path	target path to write the file
with_date	whether to insert the date before the file extension

Value

nothing

Examples

```
## Not run:
   save_sessioninfo()

## End(Not run)
```

`search_for_newer_data` *Search for newer data*

Description

Search in some folders if a TrialMaster database more recent than the current extraction is present. By default, it will search the "data" folder and the OS usual "Downloads" folder. If a newer database is found, user will be asked if they want to move it to the "data" folder.

Usage

```
search_for_newer_data(
  archive,
  ...,
  source = path_home("Downloads"),
  target = "data",
  ask = TRUE,
  advice = TRUE
)
```

Arguments

`archive`	TM archive path, giving the project name and date
`...`	unused
`source`	the path vector to be searched, default to both "data" and the usual "Downloads" folder
`target`	the path where files should be copied
`ask`	whether to ask the user to move the file to "data"
`advice`	whether to advice how to move it instead, if `ask==FALSE`

Value

the path to the newer file, invisibly.

Examples

```
## Not run:
  archive = "data/MYPROJECT_ExportTemplate_xxx_SAS_XPORT_2024_06_01_12_00.zip"
  #tm = read_trialmaster(archive)
  search_for_newer_data(archive)

## End(Not run)
```

`select_distinct` *Select only distinct columns*

Description

Select all columns that has only one level for a given grouping scope. Useful when dealing with mixed datasets containing both long data and repeated short data.

Usage

```
select_distinct(df, .by)
```

Arguments

`df`	a dataframe
`.by`	optional grouping columns

Value

`df` with less columns

Examples

```
db = edc_example()
db$ae %>% colnames()
#`crfname` has one level for the whole dataset
db$ae %>% select_distinct() %>% colnames()
#`n_ae` has one level per patient
db$ae %>% select_distinct(.by=subjid) %>% colnames()
```

set_project_name *Set the project name*

Description

Set or override the project name

Usage

```
set_project_name(db, name)
```

Arguments

db	the edc_database
name	the project name

Value

nothing

Examples

```
db = edc_example() %>%
 set_project_name("My great project")
edc_lookup()
```

table_format *Identify if a dataframe has a long or a wide format*

Description

A dataset is either in the wide format or in the long format. This function identifies the format of a dataframe with respect to a subject ID. If a dataframe has some wide and long columns, it is considered "mixed".

Usage

```
table_format(
  df,
  id = get_subjid_cols(),
  ...,
  ignore_cols = get_meta_cols(0.95),
  na_rm = FALSE,
  warn = TRUE
)
```

Arguments

df	a dataframe
id	the identifying subject ID
...	not used
ignore_cols	columns to ignore.
na_rm	whether to consider missing values
warn	whether to warn if ID is not found

Value

a string value in `c("wide", "long", "mixed)`

See Also

https://tidyr.tidyverse.org/articles/pivot.html

Examples

```
db = edc_example()
sapply(db, table_format, warn=FALSE)
```

---

unify *Unify a vector*

Description

Turn a vector of length N to a vector of length 1 after checking that there is only one unique value. Useful to safely flatten a duplicated table. Preserves the `label` attribute if set.

Usage

```
unify(x, collapse_chr = FALSE, warn = TRUE)
```

Arguments

x	a vector
collapse_chr	whether to collapse non-unique character values
warn	whether to warn if non-unique values were found

Value

a vector of length 1

Examples

```
unify(c(1,1,1,1))
#unify(c(1,1,2,1)) #warning

library(dplyr)
set.seed(42)
x=tibble(id=rep(letters[1:5],10), value=rep(1:5,10),
         value2=sample(letters[6:10], 50, replace=TRUE))
x %>% summarise(value=unify(value), .by=id) #safer than `value=value[1]`
x %>% summarise(value2=unify(value2, collapse_chr=TRUE, warn=FALSE), .by=id)
x$value[2]=1
x %>% summarise(value2=unify(value2), .by=id) #warning about that non-unique value
```

Index