Package 'sdtm.oak'

May 22, 2025

Type Package

Title SDTM Data Transformation Engine

Version 0.2.0

Maintainer Rammprasad Ganapathy <ganapathy.rammprasad@gene.com>

Description An Electronic Data Capture system (EDC) and Data Standard agnostic solution that enables the pharmaceutical programming community to develop Clinical Data Interchange Standards Consortium (CDISC) Study Data Tabulation Model (SDTM) datasets in R. The reusable algorithms concept in 'sdtm.oak' provides a framework for modular programming and can potentially automate the conversion of raw clinical data to SDTM through standardized SDTM specifications. SDTM is one of the required standards for data submission to the Food and Drug Administration (FDA) in the United States and Pharmaceuticals and Medical Devices Agency (PMDA) in Japan. SDTM standards are implemented following the SDTM Implementation Guide as defined by CDISC <https://www.cdisc.org/standards/foundational/sdtmig>.

Language en-US

License Apache License (>= 2)

BugReports https://github.com/pharmaverse/sdtm.oak/issues

URL https://pharmaverse.github.io/sdtm.oak/, https://github.com/pharmaverse/sdtm.oak

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 4.2)

Imports admiraldev (>= 1.1.0), dplyr (>= 1.0.0), purrr (>= 1.0.1), tidyr (>= 1.2.0), rlang (>= 1.0.2), tibble (>= 3.2.0), vctrs (>= 0.5.0), stringr (>= 1.4.0), assertthat, pillar, cli

Suggests knitr, htmltools, lifecycle, magrittr, rmarkdown, spelling, testthat (>= 3.1.7), DT, readr

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Rammprasad Ganapathy [aut, cre],
Adam Forys [aut],
Edgar Manukyan [aut],
Rosemary Li [aut],
Preetesh Parikh [aut],
Lisa Houterloot [aut],
Yogesh Gupta [aut],
Omar Garcia [aut],
Ramiro Magno [aut] (ORCID: <https://orcid.org/0000-0001-5226-3441>),
Kamil Sijko [aut] (ORCID: <https://orcid.org/0000-0002-2203-1065>),
Shiyu Chen [aut],
Pattern Institute [cph, fnd],
F. Hoffmann-La Roche AG [cph, fnd],
Pfizer Inc [cph, fnd],
Mohsin Uzzama [aut],
Transition Technologies Science [cph, fnd]

Repository CRAN

Date/Publication 2025-05-22 16:50:07 UTC

Contents

`assign_datetime` *Derive an ISO8601 date-time variable*

Description

`assign_datetime()` maps one or more variables with date/time components in a raw dataset to a target SDTM variable following the ISO8601 format.

Usage

```
assign_datetime(
  tgt_dat = NULL,
  tgt_var,
  raw_dat,
  raw_var,
  raw_fmt,
  raw_unk = c("UN", "UNK"),
  id_vars = oak_id_vars(),
  .warn = TRUE
)
```

Arguments

`tgt_dat`	Target dataset: a data frame to be merged against `raw_dat` by the variables indicated in `id_vars`. This parameter is optional, see section Value for how the output changes depending on this argument value.
`tgt_var`	The target SDTM variable: a single string indicating the name of variable to be derived.
`raw_dat`	The raw dataset (dataframe); must include the variables passed in `id_vars` and `raw_var`.
`raw_var`	The raw variable(s): a character vector indicating the name(s) of the raw variable(s) in `raw_dat` with date or time components to be parsed into a ISO8601 format variable in `tgt_var`.
`raw_fmt`	A date/time parsing format. Either a character vector or a list of character vectors. If a character vector is passed then each element is taken as parsing format for each variable indicated in `raw_var`. If a list is provided, then each element must be a character vector of formats. The first vector of formats is used for parsing the first variable in `raw_var`, and so on.

raw_unk	A character vector of string literals to be regarded as missing values during parsing.
id_vars	Key variables to be used in the join between the raw dataset (raw_dat) and the target data set (tgt_dat).
.warn	Whether to warn about parsing failures.

Value

The returned data set depends on the value of tgt_dat:

- If no target dataset is supplied, meaning that tgt_dat defaults to NULL, then the returned data set is raw_dat, selected for the variables indicated in id_vars, and a new extra column: the derived variable, as indicated in tgt_var.
- If the target dataset is provided, then it is merged with the raw data set raw_dat by the variables indicated in id_vars, with a new column: the derived variable, as indicated in tgt_var.

Examples

```
# `md1`: an example raw data set.
md1 <-
  tibble::tribble(
    ~oak_id, ~raw_source, ~patient_number, ~MDBDR,        ~MDEDR,        ~MDETM,
    1L,      "MD1",       375,             NA,            NA,            NA,
    2L,      "MD1",       375,             "15-Sep-20",   NA,            NA,
    3L,      "MD1",       376,             "17-Feb-21",   "17-Feb-21",   NA,
    4L,      "MD1",       377,             "4-Oct-20",    NA,            NA,
    5L,      "MD1",       377,             "20-Jan-20",   "20-Jan-20",   "10:00:00",
    6L,      "MD1",       377,             "UN-UNK-2019", "UN-UNK-2019", NA,
    7L,      "MD1",       377,             "20-UNK-2019", "20-UNK-2019", NA,
    8L,      "MD1",       378,             "UN-UNK-2020", "UN-UNK-2020", NA,
    9L,      "MD1",       378,             "26-Jan-20",   "26-Jan-20",   "07:00:00",
    10L,     "MD1",       378,             "28-Jan-20",   "1-Feb-20",    NA,
    11L,     "MD1",       378,             "12-Feb-20",   "18-Feb-20",   NA,
    12L,     "MD1",       379,             "10-UNK-2020", "20-UNK-2020", NA,
    13L,     "MD1",       379,             NA,            NA,            NA,
    14L,     "MD1",       379,             NA,            "17-Feb-20",   NA
  )

# Using the raw data set `md1`, derive the variable CMSTDTC from MDBDR using
# the parsing format (`raw_fmt`) `"d-m-y"` (day-month-year), while allowing
# for the presence of special date component values (e.g. `"UN"` or `"UNK"`),
# indicating that these values are missing/unknown (unk).
cm1 <-
  assign_datetime(
    tgt_var = "CMSTDTC",
    raw_dat = md1,
    raw_var = "MDBDR",
    raw_fmt = "d-m-y",
    raw_unk = c("UN", "UNK")
  )
```

```
cm1

# Inspect parsing failures associated with derivation of CMSTDTC.
problems(cm1$CMSTDTC)

# `cm_inter`: an example target data set.
cm_inter <-
  tibble::tibble(
    oak_id = 1L:14L,
    raw_source = "MD1",
    patient_number = c(
      375, 375, 376, 377, 377, 377, 377, 378,
      378, 378, 378, 379, 379, 379
    ),
    CMTRT = c(
      "BABY ASPIRIN",
      "CORTISPORIN",
      "ASPIRIN",
      "DIPHENHYDRAMINE HCL",
      "PARCETEMOL",
      "VOMIKIND",
      "ZENFLOX OZ",
      "AMITRYPTYLINE",
      "BENADRYL",
      "DIPHENHYDRAMINE HYDROCHLORIDE",
      "TETRACYCLINE",
      "BENADRYL",
      "SOMINEX",
      "ZQUILL"
    ),
    CMINDC = c(
      "NA",
      "NAUSEA",
      "ANEMIA",
      "NAUSEA",
      "PYREXIA",
      "VOMITINGS",
      "DIARHHEA",
      "COLD",
      "FEVER",
      "LEG PAIN",
      "FEVER",
      "COLD",
      "COLD",
      "PAIN"
    )
  )

# Same derivation as above but now involving the merging with the target
# data set `cm_inter`.
cm2 <-
  assign_datetime(
    tgt_dat = cm_inter,
```

```
    tgt_var = "CMSTDTC",
    raw_dat = md1,
    raw_var = "MDBDR",
    raw_fmt = "d-m-y"
  )

cm2

# Inspect parsing failures associated with derivation of CMSTDTC.
problems(cm2$CMSTDTC)

# Derive CMSTDTC using both MDEDR and MDETM variables.
# Note that the format `"d-m-y"` is used for parsing MDEDR and `"H:M:S"` for
# MDETM (correspondence is by positional matching).
cm3 <-
  assign_datetime(
    tgt_var = "CMSTDTC",
    raw_dat = md1,
    raw_var = c("MDEDR", "MDETM"),
    raw_fmt = c("d-m-y", "H:M:S"),
    raw_unk = c("UN", "UNK")
  )

cm3

# Inspect parsing failures associated with derivation of CMSTDTC.
problems(cm3$CMSTDTC)
```

`assign_no_ct` *Derive an SDTM variable*

Description

- `assign_no_ct()` maps a variable in a raw dataset to a target SDTM variable that has no terminology restrictions.
- `assign_ct()` maps a variable in a raw dataset to a target SDTM variable following controlled terminology recoding.

Usage

```
assign_no_ct(
  tgt_dat = NULL,
  tgt_var,
  raw_dat,
  raw_var,
  id_vars = oak_id_vars()
)
```

```
assign_ct(
  tgt_dat = NULL,
  tgt_var,
  raw_dat,
  raw_var,
  ct_spec,
  ct_clst,
  id_vars = oak_id_vars()
)
```

Arguments

tgt_dat	Target dataset: a data frame to be merged against raw_dat by the variables indicated in id_vars. This parameter is optional, see section Value for how the output changes depending on this argument value.
tgt_var	The target SDTM variable: a single string indicating the name of variable to be derived.
raw_dat	The raw dataset (dataframe); must include the variables passed in id_vars and raw_var.
raw_var	The raw variable: a single string indicating the name of the raw variable in raw_dat.
id_vars	Key variables to be used in the join between the raw dataset (raw_dat) and the target data set (raw_dat).
ct_spec	Study controlled terminology specification: a dataframe with a minimal set of columns, see ct_spec_vars() for details.
ct_clst	A codelist code indicating which subset of the controlled terminology to apply in the derivation.

Value

The returned data set depends on the value of tgt_dat:

- If no target dataset is supplied, meaning that tgt_dat defaults to NULL, then the returned data set is raw_dat, selected for the variables indicated in id_vars, and a new extra column: the derived variable, as indicated in tgt_var.
- If the target dataset is provided, then it is merged with the raw data set raw_dat by the variables indicated in id_vars, with a new column: the derived variable, as indicated in tgt_var.

Examples

```
md1 <-
  tibble::tibble(
    oak_id = 1:14,
    raw_source = "MD1",
    patient_number = 101:114,
    MDIND = c(
      "NAUSEA", "NAUSEA", "ANEMIA", "NAUSEA", "PYREXIA",
      "VOMITINGS", "DIARHHEA", "COLD",
```

```
      "FEVER", "LEG PAIN", "FEVER", "COLD", "COLD", "PAIN"
    )
  )

assign_no_ct(
  tgt_var = "CMINDC",
  raw_dat = md1,
  raw_var = "MDIND"
)

cm_inter <-
  tibble::tibble(
    oak_id = 1:14,
    raw_source = "MD1",
    patient_number = 101:114,
    CMTRT = c(
      "BABY ASPIRIN",
      "CORTISPORIN",
      "ASPIRIN",
      "DIPHENHYDRAMINE HCL",
      "PARCETEMOL",
      "VOMIKIND",
      "ZENFLOX OZ",
      "AMITRYPTYLINE",
      "BENADRYL",
      "DIPHENHYDRAMINE HYDROCHLORIDE",
      "TETRACYCLINE",
      "BENADRYL",
      "SOMINEX",
      "ZQUILL"
    ),
    CMROUTE = c(
      "ORAL",
      "ORAL",
      NA,
      "ORAL",
      "ORAL",
      "ORAL",
      "INTRAMUSCULAR",
      "INTRA-ARTERIAL",
      NA,
      "NON-STANDARD",
      "RANDOM_VALUE",
      "INTRA-ARTICULAR",
      "TRANSDERMAL",
      "OPHTHALMIC"
    )
  )

# Controlled terminology specification
(ct_spec <- read_ct_spec_example("ct-01-cm"))

assign_ct(
```

```
  tgt_dat = cm_inter,
  tgt_var = "CMINDC",
  raw_dat = md1,
  raw_var = "MDIND",
  ct_spec = ct_spec,
  ct_clst = "C66729"
)

# Variables are derived in sequence from multiple input sources.
# For each target variable, only missing (`NA`) values are filled
# during each step—previously assigned (non-missing) values are retained.

cm_raw <-
  tibble::tibble(
    oak_id = 1:4,
    raw_source = "cm_raw",
    patient_number = 370L + oak_id,
    PATNUM = patient_number,
    IT.CMTRT = c("BABY ASPIRIN", "CORTISPORIN", NA, NA),
    IT.CMTRTOTH = c("Other Treatment - ", NA, "Other Treatment - Baby Aspirin", NA)
  )

cm_raw

# Derivation of `CMTRT` first from `IT.CMTRT` and then from `IT.CMTRTOTH`.
assign_no_ct(
  raw_dat = cm_raw,
  raw_var = "IT.CMTRT",
  tgt_var = "CMTRT"
) |>
  assign_no_ct(
    raw_dat = cm_raw,
    raw_var = "IT.CMTRTOTH",
    tgt_var = "CMTRT"
  )

# Derivation of `CMTRT` first from `IT.CMTRTOTH` and then from `IT.CMTRT`.
assign_no_ct(
  raw_dat = cm_raw,
  raw_var = "IT.CMTRTOTH",
  tgt_var = "CMTRT"
) |>
  assign_no_ct(
    raw_dat = cm_raw,
    raw_var = "IT.CMTRT",
    tgt_var = "CMTRT"
  )

# Another example of variables derived in sequence from multiple input
# sources but now with controlled terminology remapping, in this case,
# CDISC Dose Unit (C71620) recoding.

cm_raw2 <- tibble::tibble(
```

```
  oak_id = c(1:3, 6, 8:10, 12:14),
  raw_source = "cm_raw",
  patient_number = c(rep(375L, 2), 376:377, rep(378L, 3), rep(379L, 3)),
  PATNUM = patient_number,
  `IT.DOSUO` = c(NA, NA, NA, NA, NA, "Other Dose Unit", "cap", NA, NA, NA),
  `IT.CMDOSU` = c("mg", "Gram", NA, "Tablet", "g", "mg", NA, "IU", "mL", "%")
)

assign_ct(
  raw_dat = cm_raw2,
  raw_var = "IT.DOSUO",
  tgt_var = "CMDOSU",
  ct_spec = ct_spec,
  ct_clst = "C71620",
  # Dose Unit
  id_vars = oak_id_vars()
) |>
  assign_ct(
    raw_dat = cm_raw2,
    raw_var = "IT.CMDOSU",
    tgt_var = "CMDOSU",
    ct_spec = ct_spec,
    ct_clst = "C71620",
    id_vars = oak_id_vars()
  )
```

`cal_min_max_date` *Calculate minimum and maximum date and time in the data frame*

Description

This function derives the earliest/latest date as ISO8601 datetime

Usage

```
cal_min_max_date(
  raw_dataset,
  date_variable,
  time_variable,
  val_type = "min",
  date_format,
  time_format
)
```

Arguments

`raw_dataset`	Raw source data frame
`date_variable`	Single character string. Name of the date variable

time_variable	Single character string. Name of the time variable
val_type	Single character string determining whether to look for the earliest or the latest datetime combination. Permitted values: "min", "max". Default to "min".
date_format	Format of source date variable
time_format	Format of source time variable

Value

Data frame with 2 columns: unique patient_number and datetime variable column storing the earliest/latest datetime.

Examples

```
ex_raw <- tibble::tribble(
  ~patient_number,    ~EX_ST_DT, ~EX_ST_TM,
  "001",           "25-04-2022",   "10:20",
  "001",           "25-04-2022",   "10:15",
  "001",           "25-04-2022",   "10:19",
  "002",           "26-05-2022", "UNK:UNK",
  "002",           "26-05-2022",   "05:59"
)

min <- cal_min_max_date(ex_raw,
  date_variable = "EX_ST_DT",
  time_variable = "EX_ST_TM",
  val_type = "min",
  date_format = "dd-mmm-yyyy",
  time_format = "H:M"
)

max <- cal_min_max_date(ex_raw,
  date_variable = "EX_ST_DT",
  time_variable = "EX_ST_TM",
  val_type = "max",
  date_format = "dd-mmm-yyyy",
  time_format = "H:M"
)
```

---

condition_add *Add filtering tags to a data set*

Description

`condition_add()` tags records in a data set, indicating which rows match the specified conditions, resulting in a conditioned data frame. Learn how to integrate conditioned data frames in your SDTM domain derivation in `vignette("cnd_df")`.

Usage

```
condition_add(dat, ..., .na = NA, .dat2 = rlang::env())
```

Arguments

dat	A data frame.
...	Conditions to filter the data frame.
.na	Return value to be used when the conditions evaluate to NA.
.dat2	An optional environment to look for variables involved in logical expression passed in .... A data frame or a list can also be passed that will be coerced to an environment internally.

Value

A conditioned data frame, meaning a tibble with an additional class cnd_df and a logical vector attribute indicating matching rows.

Examples

```
(df <- tibble::tibble(x = 1L:3L, y = letters[x]))

# Mark rows for which `x` greater than `1`
(cnd_df <- condition_add(dat = df, x > 1L))
```

---

create_iso8601 *Convert date or time collected values to ISO 8601*

Description

create_iso8601() converts vectors of dates, times or date-times to ISO 8601 format. Learn more in vignette("iso_8601").

Usage

```
create_iso8601(
  ...,
  .format,
  .fmt_c = fmt_cmp(),
  .na = NULL,
  .cutoff_2000 = 68L,
  .check_format = FALSE,
  .warn = TRUE
)
```

Arguments

...	Character vectors of dates, times or date-times' components.
.format	Parsing format(s). Either a character vector or a list of character vectors. If a character vector is passed then each element is taken as parsing format for each vector passed in .... If a list is provided, then each element must be a character vector of formats. The first vector of formats is used for parsing the first vector passed in ..., and so on.
.fmt_c	A list of regexps to use when parsing .format. Use fmt_cmp() to create such an object to pass as argument to this parameter.
.na	A character vector of string literals to be regarded as missing values during parsing.
.cutoff_2000	An integer value. Two-digit years smaller or equal to .cutoff_2000 are parsed as though starting with 20, otherwise parsed as though starting with 19.
.check_format	Whether to check the formats passed in .format, meaning to check against a selection of validated formats in dtc_formats; or to have a more permissible interpretation of the formats.
.warn	Whether to warn about parsing failures.

Value

A vector of dates, times or date-times in ISO 8601 format

Examples

```
# Converting dates
create_iso8601(c("2020-01-01", "20200102"), .format = "y-m-d")
create_iso8601(c("2020-01-01", "20200102"), .format = "ymd")
create_iso8601(c("2020-01-01", "20200102"), .format = list(c("y-m-d", "ymd")))

# Two-digit years are supported
create_iso8601(c("20-01-01", "200101"), .format = list(c("y-m-d", "ymd")))

# `.cutoff_2000` sets the cutoff for two-digit to four-digit year conversion
# Default is at 68.
create_iso8601(c("67-01-01", "68-01-01", "69-01-01"), .format = "y-m-d")

# Change it to 80.
create_iso8601(c("79-01-01", "80-01-01", "81-01-01"), .format = "y-m-d", .cutoff_2000 = 80)

# Converting times
create_iso8601("15:10", .format = "HH:MM")
create_iso8601("2:10", .format = "HH:MM")
create_iso8601("2:1", .format = "HH:MM")
create_iso8601("02:01:56", .format = "HH:MM:SS")
create_iso8601("020156.5", .format = "HHMMSS")

# Converting date-times
create_iso8601("12 NOV 202015:15", .format = "dd mmm yyyyHH:MM")
```

```
# Indicate allowed missing values to make the parsing pass
create_iso8601("U DEC 201914:00", .format = "dd mmm yyyyHH:MM")
create_iso8601("U DEC 201914:00", .format = "dd mmm yyyyHH:MM", .na = "U")

create_iso8601("NOV 2020", .format = "m y")
create_iso8601(c("MAR 2019", "MaR 2020", "mar 2021"), .format = "m y")

create_iso8601("2019-04-041045-", .format = "yyyy-mm-ddHHMM-")

create_iso8601("20200507null", .format = "ymd(HH:MM:SS)")
create_iso8601("20200507null", .format = "ymd((HH:MM:SS)|null)")

# Fractional seconds
create_iso8601("2019-120602:20:13.1230001", .format = "y-mdH:M:S")

# Use different reserved characters in the format specification
# Here we change "H" to "x" and "M" to "w", for hour and minute, respectively.
create_iso8601("14H00M", .format = "HHMM")
create_iso8601("14H00M", .format = "xHwM", .fmt_c = fmt_cmp(hour = "x", min = "w"))

# Alternative formats with unknown values
datetimes <- c("UN UNK 201914:00", "UN JAN 2021")
format <- list(c("dd mmm yyyy", "dd mmm yyyyHH:MM"))
create_iso8601(datetimes, .format = format, .na = c("UN", "UNK"))

# Dates and times may come in many format variations
fmt <- "dd MMM yyyy HH nn ss"
fmt_cmp <- fmt_cmp(mon = "MMM", min = "nn", sec = "ss")
create_iso8601("05 feb 1985 12 55 02", .format = fmt, .fmt_c = fmt_cmp)
```

`ctl_new_rowid_pillar.cnd_df`

Conditioned tibble pillar print method

Description

Conditioned tibble pillar print method

Usage

```
## S3 method for class 'cnd_df'
ctl_new_rowid_pillar(controller, x, width, ...)
```

Arguments

`controller`	The object of class `"tbl"` currently printed.
`x`	A simple (one-dimensional) vector.
`width`	The available width, can be a vector for multiple tiers.
`...`	These dots are for future extensions and must be empty.

Value

A character vector to print the tibble which is a conditioned dataframe.

See Also

`tbl_sum.cnd_df()`.

`ct_map` *Recode according to controlled terminology*

Description

`ct_map()` recodes a vector following a controlled terminology.

Usage

```
ct_map(
  x,
  ct_spec = NULL,
  ct_clst = NULL,
  from = ct_spec_vars("from"),
  to = ct_spec_vars("to")
)
```

Arguments

`x`	A character vector of terms to be recoded following a controlled terminology.
`ct_spec`	A tibble providing a controlled terminology specification.
`ct_clst`	A character vector indicating a set of possible controlled terminology codelists codes to be used for recoding. By default (`NULL`) all codelists available in `ct_spec` are used.
`from`	A character vector of column names indicating the variables containing values to be matched against for terminology recoding.
`to`	A single string indicating the column whose values are to be recoded into.

Value

A character vector of terminology recoded values from `x`. If no match is found in the controlled terminology spec provided in `ct_spec`, then `x` values are returned in uppercase. If `ct_spec` is not provided `x` is returned unchanged.

Examples

```
# A few example terms.
terms <-
  c(
    "/day",
    "Yes",
    "Unknown",
    "Prior",
    "Every 2 hours",
    "Percentage",
    "International Unit"
  )

# Load a controlled terminology example
(ct_spec <- read_ct_spec_example("ct-01-cm"))

# Use all possible matching terms in the controlled terminology.
ct_map(x = terms, ct_spec = ct_spec)

# Note that if the controlled terminology mapping is restricted to a codelist
# code, e.g. C71113, then only `"/day"` and `"Every 2 hours"` get mapped to
# `"QD"` and `"Q2H"`, respectively; remaining terms won't match given the
# codelist code restriction, and will be mapped to an uppercase version of
# the original terms.
ct_map(x = terms, ct_spec = ct_spec, ct_clst = "C71113")
```

---

ct_spec_example *Find the path to an example controlled terminology file*

Description

`ct_spec_example()` resolves the local path to an example controlled terminology file.

Usage

```
ct_spec_example(example)
```

Arguments

example	A string with either the basename, file name, or relative path to a controlled terminology file bundled with {stdm.oak}, see examples.

Value

The local path to an example file if `example` is supplied, or a character vector of example file names.

Examples

```
# Get the local path to controlled terminology example file 01
# Using the basename only:
ct_spec_example("ct-01-cm")

# Using the file name:
ct_spec_example("ct-01-cm.csv")

# Using the relative path:
ct_spec_example("ct/ct-01-cm.csv")

# If no example is provided it returns a vector of possible choices.
ct_spec_example()
```

`derive_blfl` *Derive Baseline Flag or Last Observation Before Exposure Flag*

Description

Derive the baseline flag variable (`--BLFL`) or the last observation before exposure flag (`--LOBXFL`), from the observation date/time (`--DTC`), and a DM domain reference date/time.

Usage

```
derive_blfl(
  sdtm_in,
  dm_domain,
  tgt_var,
  ref_var,
  baseline_visits = character(),
  baseline_timepoints = character()
)
```

Arguments

`sdtm_in`	Input SDTM domain.
`dm_domain`	DM domain with the reference variable `ref_var`
`tgt_var`	Name of variable to be derived (`--BLFL` or `--LOBXFL` where `--` is domain).
`ref_var`	vector of a date/time from the Demographics (DM) dataset, which serves as a point of comparison for other observations in the study. Common choices for this reference variable include "RFSTDTC" (the date/time of the first study treatment) or "RFXSTDTC" (the date/time of the first exposure to the study drug).

`baseline_visits`
: A character vector specifying the baseline visits within the study. These visits are identified as critical points for data collection at the start of the study, before any intervention is applied. This allows the function to assign the baseline flag if the –DTC matches to the reference date.

`baseline_timepoints`
: A character vector of timepoints values in –TPT that specifies the specific timepoints during the baseline visits when key assessments or measurements were taken. This allows the function to assign the baseline flag if the –DTC matches to the reference date.

Details

The derivation is as follows:

- Remove records where the result (`--ORRES`) is missing. Also, exclude records with results labeled as "ND" (No Data) or "NOT DONE" in the `--ORRES` column, which indicate that the measurement or observation was not completed.
- Remove records where the status (`--STAT`) indicates the observation or test was not performed, marked as "NOT DONE".
- Divide the date and time column (`--DTC`) and the reference date/time variable (`ref_var`) into separate date and time components. Ignore any seconds recorded in the time component, focusing only on hours and minutes for further calculations.
- Set partial or missing dates to `NA`.
- Set partial or missing times to `NA`.
- Filter on rows that have domain and reference dates not equal to `NA`. (Ref to as **X**)
- Filter **X** on rows with domain date (–DTC) prior to (less than) reference date. (Ref to as **A**)
- Filter **X** on rows with domain date (–DTC) equal to reference date but domain and reference times not equal to `NA` and domain time prior to (less than) reference time. (Ref to as **B**)
- Filter **X** on rows with domain date (–DTC) equal to reference date but domain and/or reference time equal to NA and:
  - VISIT is in baseline visits list (if it exists) and
  - xxTPT is in baseline timepoints list (if it exists). (Ref to as **C**)
- Combine the rows from **A**, **B**, and **C** to get a data frame of pre-reference date observations. Sort the rows by `USUBJID`, `--STAT`, and `--ORRES`.
- Group by `USUBJID` and `--TESTCD` and filter on the rows that have maximum value from `--DTC`. Keep only the oak id variables and `--TESTCD` (because these are the unique values). Remove any duplicate rows. Assign the baseline flag variable, `--BLFL`, the last observation before exposure flag (`--LOBXFL`) variable to these rows.
- Join the baseline flag onto the input dataset based on oak id vars

Value

Modified input data frame with baseline flag variable `--BLFL` or last observation before exposure flag `--LOBXFL` added.

Examples

```
dm <- tibble::tribble(
  ~USUBJID, ~RFSTDTC, ~RFXSTDTC,
  "test_study-375", "2020-09-28T10:10", "2020-09-28T10:10",
  "test_study-376", "2020-09-21T11:00", "2020-09-21T11:00",
  "test_study-377", NA, NA,
  "test_study-378", "2020-01-20T10:00", "2020-01-20T10:00",
  "test_study-379", NA, NA,
)

dm

sdtm_in <-
  tibble::tribble(
    ~DOMAIN,
    ~oak_id,
    ~raw_source,
    ~patient_number,
    ~USUBJID,
    ~VSDTC,
    ~VSTESTCD,
    ~VSORRES,
    ~VSSTAT,
    ~VISIT,
    "VS",
    1L,
    "VTLS1",
    375L,
    "test_study-375",
    "2020-09-01T13:31",
    "DIABP",
    "90",
    NA,
    "SCREENING",
    "VS",
    2L,
    "VTLS1",
    375L,
    "test_study-375",
    "2020-10-01T11:20",
    "DIABP",
    "90",
    NA,
    "SCREENING",
    "VS",
    1L,
    "VTLS1",
    375L,
    "test_study-375",
    "2020-09-28T10:10",
    "PULSE",
    "ND",
```

```
NA,
"SCREENING",
"VS",
2L,
"VTLS1",
375L,
"test_study-375",
"2020-10-01T13:31",
"PULSE",
"85",
NA,
"SCREENING",
"VS",
1L,
"VTLS2",
375L,
"test_study-375",
"2020-09-28T10:10",
"SYSBP",
"120",
NA,
"SCREENING",
"VS",
2L,
"VTLS2",
375L,
"test_study-375",
"2020-09-28T10:05",
"SYSBP",
"120",
NA,
"SCREENING",
"VS",
1L,
"VTLS1",
376L,
"test_study-376",
"2020-09-20",
"DIABP",
"75",
NA,
"SCREENING",
"VS",
1L,
"VTLS1",
376L,
"test_study-376",
"2020-09-20",
"PULSE",
NA,
"NOT DONE",
"SCREENING",
"VS",
```

```
    2L,
    "VTLS1",
    376L,
    "test_study-376",
    "2020-09-20",
    "PULSE",
    "110",
    NA,
    "SCREENING",
    "VS",
    2L,
    "VTLS1",
    378L,
    "test_study-378",
    "2020-01-20T10:00",
    "PULSE",
    "110",
    NA,
    "SCREENING",
    "VS",
    3L,
    "VTLS1",
    378L,
    "test_study-378",
    "2020-01-21T11:00",
    "PULSE",
    "105",
    NA,
    "SCREENING"
  )

sdtm_in

# Example 1:
observed_output <- derive_blfl(
  sdtm_in = sdtm_in,
  dm_domain = dm,
  tgt_var = "VSLOBXFL",
  ref_var = "RFXSTDTC",
  baseline_visits = c("SCREENING")
)
observed_output

# Example 2:
observed_output2 <- derive_blfl(
  sdtm_in = sdtm_in,
  dm_domain = dm,
  tgt_var = "VSLOBXFL",
  ref_var = "RFXSTDTC",
  baseline_timepoints = c("PRE-DOSE")
)
observed_output2
```

```
# Example 3: Output is the same as Example 2
observed_output3 <- derive_blfl(
  sdtm_in = sdtm_in,
  dm_domain = dm,
  tgt_var = "VSLOBXFL",
  ref_var = "RFXSTDTC",
  baseline_visits = c("SCREENING"),
  baseline_timepoints = c("PRE-DOSE")
)
observed_output3
```

`derive_seq` *Derive the sequence number (`--SEQ`) variable*

Description

`derive_seq()` creates a new identifier variable: the sequence number (`--SEQ`).

This function adds a newly derived variable to `tgt_dat`, namely the sequence number (`--SEQ`) whose name is the one provided in `tgt_var`. An integer sequence is generated that uniquely identifies each record within the domain.

Prior to the derivation of `tgt_var`, the data frame `tgt_dat` is sorted according to grouping variables indicated in `rec_vars`.

Usage

```
derive_seq(
  tgt_dat,
  tgt_var,
  rec_vars,
  sbj_vars = sdtm.oak::sbj_vars(),
  start_at = 1L
)
```

Arguments

`tgt_dat`	The target dataset, a data frame.
`tgt_var`	The target SDTM variable: a single string indicating the name of the sequence number (`--SEQ`) variable, e.g. `"DSSEQ"`. Note that supplying a name not ending in `"SEQ"` will raise a warning.
`rec_vars`	A character vector of record-level identifier variables.
`sbj_vars`	A character vector of subject-level identifier variables.
`start_at`	The sequence numbering starts at this value (default is 1).

Value

Returns the data frame supplied in `tgt_dat` with the newly derived variable, i.e. the sequence number (`--SEQ`), whose name is that passed in `tgt_var`. This variable is of type integer.

Examples

```
# A VS raw data set example
(vs <- read_domain_example("vs"))

# Derivation of VSSEQ
rec_vars <- c("STUDYID", "USUBJID", "VSTESTCD", "VSDTC", "VSTPTNUM")
derive_seq(tgt_dat = vs, tgt_var = "VSSEQ", rec_vars = rec_vars)

# An APSC raw data set example
(apsc <- read_domain_example("apsc"))

# Derivation of APSEQ
derive_seq(
  tgt_dat = apsc,
  tgt_var = "APSEQ",
  rec_vars = c("STUDYID", "RSUBJID", "SCTESTCD"),
  sbj_vars = c("STUDYID", "RSUBJID")
)
```

`derive_study_day` `derive_study_day` *performs study day calculation*

Description

This function takes the an input data frame and a reference data frame (which is DM domain in most cases), and calculate the study day from reference date and target date. In case of unexpected conditions like reference date is not unique for each patient, or reference and input dates are not actual dates, NA will be returned for those records.

Usage

```
derive_study_day(
  sdtm_in,
  dm_domain,
  tgdt,
  refdt,
  study_day_var,
  merge_key = "USUBJID"
)
```

Arguments

`sdtm_in`	Input data frame that contains the target date.
`dm_domain`	Reference date frame that contains the reference date.
`tgdt`	Target date from `sdtm_in` that will be used to calculate the study day.
`refdt`	Reference date from `dm_domain` that will be used as reference to calculate the study day.

study_day_var	New study day variable name in the output. For example, AESTDY for AE domain and CMSTDY for CM domain.
merge_key	Character to represent the merging key between sdtm_in and dm_domain.

Value

Data frame that takes all columns from sdtm_in and a new variable to represent the calculated study day.

Examples

```
ae <- data.frame(
  USUBJID = c("study123-123", "study123-124", "study123-125"),
  AESTDTC = c("2012-01-01", "2012-04-14", "2012-04-14")
)
dm <- data.frame(
  USUBJID = c("study123-123", "study123-124", "study123-125"),
  RFSTDTC = c("2012-02-01", "2012-04-14", NA)
)
ae$AESTDTC <- as.Date(ae$AESTDTC)
dm$RFSTDTC <- as.Date(dm$RFSTDTC)
derive_study_day(ae, dm, "AESTDTC", "RFSTDTC", "AESTDY")
```

---

domain_example *Find the path to an example SDTM domain file*

Description

domain_example() resolves the local path to a SDTM domain example file. The domain examples files were imported from pharmaversesdtm. See Details section for available datasets.

Usage

```
domain_example(example)
```

Arguments

example	A string with either the basename, file name, or relative path to a SDTM domain example file bundled with {stdm.oak}, e.g. "cm" (Concomitant Medication) or "ae" (Adverse Events).

Details

Datasets were obtained from pharmaversesdtm but are originally sourced from the CDISC pilot project or have been constructed ad-hoc by the admiral team. These datasets are bundled with {sdtm.oak}, thus obviating a dependence on {pharmaversesdtm}.

Example SDTM domains:

- "ae": Adverse Events (AE) data set.
- "apsc": Associated Persons Subject Characteristics (APSC) data set.
- "cm": Concomitant Medications (CM) data set.
- "vs": Vital Signs (VS) data set.

Value

The local path to an example file if example is supplied, or a character vector of example file names.

Source

See https://cran.r-project.org/package=pharmaversesdtm.

See Also

read_domain_example()

Examples

```
# If no example is provided it returns a vector of possible choices.
domain_example()

# Get the local path to the Concomitant Medication dataset file.
domain_example("cm")

# Local path to the Adverse Events dataset file.
domain_example("ae")
```

---

dtc_formats *Date/time collection formats*

Description

Date/time collection formats

Usage

```
dtc_formats
```

Format

A tibble of 20 formats with three variables:

fmt Format string.

type Whether a date, time or date-time.

description Description of which date-time components are parsed.

Examples

```
dtc_formats
```

`fmt_cmp` *Regexps for date/time format components*

Description

`fmt_cmp()` creates a character vector of patterns to match individual format date/time components.

Usage

```
fmt_cmp(
  sec = "S+",
  min = "M+",
  hour = "H+",
  mday = "d+",
  mon = "m+",
  year = "y+"
)
```

Arguments

`sec`	A string pattern for matching the second format component.
`min`	A string pattern for matching the minute format component.
`hour`	A string pattern for matching the hour format component.
`mday`	A string pattern for matching the month day format component.
`mon`	A string pattern for matching the month format component.
`year`	A string pattern for matching the year format component.

Value

A named character vector of date/time format patterns. This a vector of six elements, one for each date/time component.

Examples

```
# Regexps to parse format components
fmt_cmp()

fmt_cmp(year = "yyyy")
```

`generate_oak_id_vars` *A function to generate oak_id_vars*

Description

A function to generate oak_id_vars

Usage

```
generate_oak_id_vars(raw_dat, pat_var, raw_src)
```

Arguments

`raw_dat`	The raw dataset (dataframe)
`pat_var`	Variable that holds the patient number
`raw_src`	Name of the raw source

Value

dataframe

Examples

```
raw_dataset <-
  tibble::tribble(
    ~patnum, ~MDRAW,
    101L, "BABY ASPIRIN",
    102L, "CORTISPORIN",
    103L, NA_character_,
    104L, "DIPHENHYDRAMINE HCL"
  )

# Generate oak_id_vars
generate_oak_id_vars(
  raw_dat = raw_dataset,
  pat_var = "patnum",
  raw_src = "Concomitant Medication"
)
```

`generate_sdtm_supp` *Function to generate final SDTM domain and supplemental domain SUPP–*

Description

Function to generate final SDTM domain and supplemental domain SUPP–

Usage

```
generate_sdtm_supp(
  sdtm_dataset,
  idvar = NULL,
  supp_qual_info,
  qnam_var,
  label_var,
  orig_var
)
```

Arguments

`sdtm_dataset`	SDTM output used to split supplemental domains.
`idvar`	Variable name for IDVAR variable.
`supp_qual_info`	User-defined data frame of specifications for suppquals which contains `qnam_var`, `label_var` and `orig_var`.
`qnam_var`	Variable name in user-defined `supp_qual_info` for QNAM variable.
`label_var`	Variable name in user-defined `supp_qual_info` for QLABEL variable.
`orig_var`	Variable name in user-defined `supp_qual_info` for QORIG variable.

Value

List of SDTM domain with suppquals dropped and corresponding supplemental domain.

Examples

```
dm <- read_domain_example("dm")
supp_qual_info <- read.csv(system.file("spec/suppqual_spec.csv", package = "sdtm.oak"))

dm_suppdm <-
  generate_sdtm_supp(
    dm,
    idvar = NULL,
    supp_qual_info = supp_qual_info,
    qnam_var = "Variable",
    label_var = "Label",
    orig_var = "Origin"
  )
```

harcode *Derive an SDTM variable with a hardcoded value*

Description

- hardcode_no_ct() maps a hardcoded value to a target SDTM variable that has no terminology restrictions.
- hardcode_ct() maps a hardcoded value to a target SDTM variable with controlled terminology recoding.

Usage

```
hardcode_no_ct(
  tgt_dat = NULL,
  tgt_val,
  raw_dat,
  raw_var,
  tgt_var,
  id_vars = oak_id_vars()
)

hardcode_ct(
  tgt_dat = NULL,
  tgt_val,
  raw_dat,
  raw_var,
  tgt_var,
  ct_spec,
  ct_clst,
  id_vars = oak_id_vars()
)
```

Arguments

tgt_dat	Target dataset: a data frame to be merged against raw_dat by the variables indicated in id_vars. This parameter is optional, see section Value for how the output changes depending on this argument value.
tgt_val	The target SDTM value to be hardcoded into the variable indicated in tgt_var.
raw_dat	The raw dataset (dataframe); must include the variables passed in id_vars and raw_var.
raw_var	The raw variable: a single string indicating the name of the raw variable in raw_dat.
tgt_var	The target SDTM variable: a single string indicating the name of variable to be derived.
id_vars	Key variables to be used in the join between the raw dataset (raw_dat) and the target data set (raw_dat).

ct_spec — Study controlled terminology specification: a dataframe with a minimal set of columns, see ct_spec_vars() for details. This parameter is optional, if left as NULL no controlled terminology recoding is applied.

ct_clst — A codelist code indicating which subset of the controlled terminology to apply in the derivation. This parameter is optional, if left as NULL, all possible recodings in ct_spec are attempted.

Value

The returned data set depends on the value of tgt_dat:

- If no target dataset is supplied, meaning that tgt_dat defaults to NULL, then the returned data set is raw_dat, selected for the variables indicated in id_vars, and a new extra column: the derived variable, as indicated in tgt_var.
- If the target dataset is provided, then it is merged with the raw data set raw_dat by the variables indicated in id_vars, with a new column: the derived variable, as indicated in tgt_var.

Examples

```
md1 <-
  tibble::tribble(
    ~oak_id, ~raw_source, ~patient_number, ~MDRAW,
    1L,      "MD1",       101L,            "BABY ASPIRIN",
    2L,      "MD1",       102L,            "CORTISPORIN",
    3L,      "MD1",       103L,            NA_character_,
    4L,      "MD1",       104L,            "DIPHENHYDRAMINE HCL"
  )

# Derive a new variable `CMCAT` by overwriting `MDRAW` with the
# hardcoded value "GENERAL CONCOMITANT MEDICATIONS".
hardcode_no_ct(
  tgt_val = "GENERAL CONCOMITANT MEDICATIONS",
  raw_dat = md1,
  raw_var = "MDRAW",
  tgt_var = "CMCAT"
)

cm_inter <-
  tibble::tribble(
    ~oak_id, ~raw_source, ~patient_number, ~CMTRT,                ~CMINDC,
    1L,      "MD1",       101L,            "BABY ASPIRIN",        NA,
    2L,      "MD1",       102L,            "CORTISPORIN",         "NAUSEA",
    3L,      "MD1",       103L,            "ASPIRIN",             "ANEMIA",
    4L,      "MD1",       104L,            "DIPHENHYDRAMINE HCL", "NAUSEA",
    5L,      "MD1",       105L,            "PARACETAMOL",         "PYREXIA"
  )

# Derive a new variable `CMCAT` by overwriting `MDRAW` with the
# hardcoded value "GENERAL CONCOMITANT MEDICATIONS" with a prior join to
# `target_dataset`.
hardcode_no_ct(
```

```
  tgt_dat = cm_inter,
  tgt_val = "GENERAL CONCOMITANT MEDICATIONS",
  raw_dat = md1,
  raw_var = "MDRAW",
  tgt_var = "CMCAT"
)

# Controlled terminology specification
(ct_spec <- read_ct_spec_example("ct-01-cm"))

# Hardcoding of `CMCAT` with the value `"GENERAL CONCOMITANT MEDICATIONS"`
# involving terminology recoding. `NA` values in `MDRAW` are preserved in
# `CMCAT`.
hardcode_ct(
  tgt_dat = cm_inter,
  tgt_var = "CMCAT",
  raw_dat = md1,
  raw_var = "MDRAW",
  tgt_val = "GENERAL CONCOMITANT MEDICATIONS",
  ct_spec = ct_spec,
  ct_clst = "C66729"
)


# Variables are derived in sequence from multiple input sources.
# For each target variable, only missing (`NA`) values are filled
# during each step—previously assigned (non-missing) values are retained.

cm_raw <-
  tibble::tibble(
    oak_id = 1:4,
    raw_source = "cm_raw",
    patient_number = 370 + oak_id,
    PATNUM = patient_number,
    IT.CMTRT = c("BABY ASPIRIN", "CORTISPORIN", NA, NA),
    IT.CMTRTOTH = c("Other Treatment - ", NA, "Other Treatment - Baby Aspirin", NA)
  )

cm_raw

# Hardcoding of values of `CMCAT` is based firstly on the presence of missing
# values (`NA`) in `IT.CMTRT` and only secondly on `IT.CMTRTOTH`.
hardcode_no_ct(
  tgt_val = "General Concomitant Medications",
  raw_dat = cm_raw,
  raw_var = "IT.CMTRT",
  tgt_var = "CMCAT"
) |>
  hardcode_no_ct(
    tgt_val = "Other General Concomitant Medications",
    raw_dat = cm_raw,
    raw_var = "IT.CMTRTOTH",
    tgt_var = "CMCAT"
```

```
  )

# Note that hardcoding application is reversed in this example, this impacts
# the result.
hardcode_no_ct(
  tgt_val = "Other General Concomitant Medications",
  raw_dat = cm_raw,
  raw_var = "IT.CMTRTOTH",
  tgt_var = "CMCAT"
) |>
  hardcode_no_ct(
    tgt_val = "General Concomitant Medications",
    raw_dat = cm_raw,
    raw_var = "IT.CMTRT",
    tgt_var = "CMCAT"
  )
```

`mutate.cnd_df` *Mutate method for conditioned data frames*

Description

`mutate.cnd_df()` is an S3 method to be dispatched by mutate generic on conditioned data frames. This function implements a conditional mutate by only changing rows for which the condition stored in the conditioned data frame is `TRUE`.

Usage

```
## S3 method for class 'cnd_df'
mutate(
  .data,
  ...,
  .by = NULL,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)
```

Arguments

`.data`	A conditioned data frame.
`...`	<data-masking> Name-value pairs. The name gives the name of the column in the output. The value can be: • A vector of length 1, which will be recycled to the correct length. • A vector the same length as the current group (or the whole data frame if ungrouped).

- NULL, to remove the column.
- A data frame or tibble, to create multiple columns in the output.

.by	Not used when .data is a conditioned data frame.
.keep	Control which columns from .data are retained in the output. Grouping columns and columns created by ... are always kept. • "all" retains all columns from .data. This is the default. • "used" retains only the columns used in ... to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side. • "unused" retains only the columns *not* used in ... to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them. • "none" doesn't retain any extra columns from .data. Only the grouping variables and columns created by ... are kept.
.before	Not used, use .after instead.
.after	Control where new columns should appear, i.e. after which columns.

Value

A conditioned data frame, meaning a tibble with mutated values.

---

oak_cal_ref_dates *Calculate Reference dates in ISO8601 character format.*

Description

Derive RFSTDTC, RFENDTC, RFXENDTC, RFXSTDTC, etc. based on the input dates and time.

Usage

```
oak_cal_ref_dates(
  ds_in = dm,
  der_var,
  min_max = "min",
  ref_date_config_df,
  raw_source
)
```

Arguments

ds_in	Data frame. DM domain.
der_var	Character string. The reference date to be derived.
min_max	Minimum or Maximum date to be calculated based on the input. Default set to Minimum. Values should be min or max.

`ref_date_config_df`
: Data frame which has the details of the variables to be used for the calculation of reference dates. Should have columns listed below: raw_dataset_name : Name of the raw dataset. date_var : Date variable name from the raw dataset. time_var : Time variable name from the raw dataset. dformat : Format of the date collected in raw data. tformat: Format of the time collected in raw data. sdtm_var_name : Reference variable name.

`raw_source`
: List contains all the raw datasets.

Details

Populate Reference date variables in demographic domain in ISO8601 character format.

Value

DM data frame with the reference dates populated.

Examples

```
dm <- tibble::tribble(
  ~patient_number,   ~USUBJID, ~SUBJID, ~SEX,
  "001",           "XXXX-001",   "001",  "F",
  "002",           "XXXX-002",   "002",  "M",
  "003",           "XXXX-003",   "003",  "M"
)

ref_date_config_df <- tibble::tribble(
  ~raw_dataset_name,   ~date_var,   ~time_var,      ~dformat, ~tformat, ~sdtm_var_name,
  "ex1_raw",         "EX_ST_DT1", "EX_ST_TM1",  "dd-mm-yyyy",    "H:M",      "RFSTDTC",
  "ex2_raw",         "EX_ST_DT2",          NA, "dd-mmm-yyyy",       NA,      "RFSTDTC",
  "ex1_raw",         "EX_EN_DT1", "EX_EN_TM1",  "dd-mm-yyyy",    "H:M",      "RFENDTC",
  "ex2_raw",         "EX_ST_DT2",          NA, "dd-mmm-yyyy",       NA,      "RFENDTC"
)

ex1_raw <- tibble::tribble(
  ~patient_number, ~EX_ST_DT1,   ~EX_EN_DT1, ~EX_ST_TM1, ~EX_EN_TM1,
  "001",         "15-05-2023", "15-05-2023",    "10:20",    "11:00",
  "001",         "15-05-2023", "15-05-2023",     "9:15",    "10:00",
  "001",         "15-05-2023", "15-05-2023",     "8:19",    "09:00",
  "002",         "02-10-2023", "02-10-2023",  "UNK:UNK",         NA,
  "002",         "03-11-2023", "03-11-2023",    "11:19",         NA
)

ex2_raw <- tibble::tribble(
  ~patient_number,     ~EX_ST_DT2,
  "001",            "11-JUN-2023",
  "002",            "24-OCT-2023",
  "002",            "25-JUL-2023",
  "002",            "30-OCT-2023",
  "002",           "UNK-OCT-2023"
)
```

```
raw_source <- list(ex1_raw = ex1_raw, ex2_raw = ex2_raw)

dm_df <- oak_cal_ref_dates(dm,
  der_var = "RFSTDTC",
  min_max = "min",
  ref_date_config_df = ref_date_config_df,
  raw_source
)
```

`oak_id_vars` *Raw dataset keys*

Description

`oak_id_vars()` is a helper function providing the variable (column) names to be regarded as keys in tibbles representing raw datasets. By default, the set of names is oak_id, raw_source, and patient_number. Extra variable names may be indicated and passed in `extra_vars` which are appended to the default names.

Usage

```
oak_id_vars(extra_vars = NULL)
```

Arguments

`extra_vars`	A character vector of extra column names to be appended to the default names: oak_id, raw_source, and patient_number.

Value

A character vector of column names to be regarded as keys in raw datasets.

`problems` *Retrieve date/time parsing problems*

Description

`problems()` is a companion helper function to `create_iso8601()`. It retrieves ISO 8601 parsing problems from an object of class iso8601, which is `create_iso8601()`'s return value and that might contain a `problems` attribute in case of parsing failures. `problems()` is a helper function that provides easy access to these parsing problems.

Usage

```
problems(x = .Last.value)
```

Arguments

x	An object of class iso8601, as typically obtained from a call to `create_iso8601()`. The argument can also be left empty, in that case `problems()` will use the last returned value, making it convenient to use immediately after `create_iso8601()`.

Value

If there are no parsing problems in `x`, then the returned value is `NULL`; otherwise, a tibble of parsing failures is returned. Each row corresponds to a parsing problem. There will be a first column named `..i` indicating the position(s) in the inputs to the `create_iso8601()` call that resulted in failures; remaining columns correspond to the original input values passed on to `create_iso8601()`, with columns being automatically named `..var1`, `..var2`, and so on, if the inputs to `create_iso8601()` were unnamed, otherwise, the original variable names are used instead.

Examples

```
dates <-
  c(
    "2020-01-01",
    "2020-02-11",
    "2020-01-06",
    "2020-0921",
    "2020/10/30",
    "2020-12-05",
    "20231225"
  )

# By inspecting the problematic dates it can be understood that
# the `.format` parameter needs to updated to include other variations.
iso8601_dttm <- create_iso8601(dates, .format = "y-m-d")
problems(iso8601_dttm)

# Including more parsing formats addresses the previous problems
formats <- c("y-m-d", "y-md", "y/m/d", "ymd")
iso8601_dttm2 <- create_iso8601(dates, .format = list(formats))

# So now `problems()` returns `NULL` because there are no more parsing issues.
problems(iso8601_dttm2)

# If you pass named arguments when calling `create_iso8601()` then they will
# be used to create the problems object.
iso8601_dttm3 <- create_iso8601(date = dates, .format = "y-m-d")
problems(iso8601_dttm3)
```

`read_ct_spec` *Read in a controlled terminology*

Description

read_ct_spec() imports a controlled terminology specification data set as a tibble.

Usage

```
read_ct_spec(file = cli::cli_abort("`file` must be specified"))
```

Arguments

file	A path to a file containing a controlled terminology specification data set. The following are expected of this file: • The file is expected to be a CSV file; • The file is expected to contain a first row of column names; • This minimal set of variables is expected: codelist_code, collected_value, term_synonyms, and term_value.

Value

A tibble with a controlled terminology specification.

Examples

```
# Get the local path to one of the controlled terminology example files.
path <- ct_spec_example("ct-01-cm")

# Import it to R.
read_ct_spec(file = path)
```

---

read_ct_spec_example *Read an example controlled terminology specification*

Description

read_ct_spec_example() imports one of the bundled controlled terminology specification data sets as a tibble into R.

Usage

```
read_ct_spec_example(example)
```

Arguments

example	The file name of a controlled terminology data set bundled with {stdm.oak}, run read_ct_spec_example() for available example files.

Value

A tibble with a controlled terminology specification data set, or a character vector of example file names.

Examples

```
# Leave the `example` parameter as missing for available example files.
read_ct_spec_example()

# Read an example controlled terminology spec file.
read_ct_spec_example("ct-01-cm.csv")

# You may omit the file extension.
read_ct_spec_example("ct-01-cm")
```

---

read_domain_example *Read an example SDTM domain*

Description

`read_domain_example()` imports one of the bundled SDTM domain examples as a tibble into R. See `domain_example()` for possible choices.

Usage

```
read_domain_example(example)
```

Arguments

`example`	The name of SDTM domain example, e.g. `"cm"` (Concomitant Medication) or `"ae"` (Adverse Events). Run `read_domain_example()` for available example files.

Value

A tibble with an SDTM domain dataset, or a character vector of example file names.

See Also

`domain_example()`

Examples

```
# Leave the `example` parameter as missing for available example files.
read_domain_example()

# Read the example Concomitant Medication domain.
read_domain_example("cm")

# Read the example Adverse Events domain.
read_domain_example("ae")
```

`sbj_vars` *Subject-level key variables*

Description

`sbj_vars()` returns the set of variable names that uniquely define a subject.

Usage

```
sbj_vars()
```

Value

A character vector of variable names.

Examples

```
sbj_vars()
```

`tbl_sum.cnd_df` *Conditioned tibble header print method*

Description

Conditioned tibble header print method. This S3 method adds an extra line in the header of a tibble that indicates the tibble is a conditioned tibble (`# Cond. tbl:`) followed by the tally of the conditioning vector: number of TRUE, FALSE and NA values: e.g., `1/1/1`.

Usage

```
## S3 method for class 'cnd_df'
tbl_sum(x, ...)
```

Arguments

x	A conditioned tibble of class cnd_df.
...	Additional arguments passed to the default print method.

Value

A character vector with header values of the conditioned data frame.

See Also

ctl_new_rowid_pillar.cnd_df().

Examples

```
df <- data.frame(x = c(1L, NA_integer_, 3L))
(cnd_df <- condition_add(dat = df, x >= 2L))
pillar::tbl_sum(cnd_df)
```

%.>% *Explicit Dot Pipe*

Description

[Experimental]

This operator pipes an object forward into a function or call expression using an explicit placement of the dot (.) placeholder. Unlike magrittr's %>% operator, %.>% does not automatically place the left-hand side (lhs) as the first argument in the right-hand side (rhs) call. This operator provides a simpler alternative to the use of braces with magrittr, while achieving similar behavior.

Usage

```
lhs %.>% rhs
```

Arguments

lhs	A value to be piped forward.
rhs	A function call that utilizes the dot (.) placeholder to specify where lhs should be placed.

Details

The `%.>%` operator is used to pipe the `lhs` value into the `rhs` function call. Within the `rhs` expression, the placeholder `.` represents the position where `lhs` will be inserted. This provides more control over where the `lhs` value appears in the `rhs` function call, compared to the magrittr pipe operator which always places `lhs` as the first argument of `rhs`.

Unlike magrittr's pipe, which may require the use of braces to fully control the placement of `lhs` in nested function calls, `%.>%` simplifies this by directly allowing multiple usages of the dot placeholder without requiring braces. For example, the following expression using magrittr's pipe and braces:

```
library(magrittr)

1:10 %>% { c(min(.), max(.)) }
```

can be written as:

```
1:10 %.>% c(min(.), max(.))
```

without needing additional braces.

Downside:
The disadvantage of `%.>%` is that you always need to use the dot placeholder, even when piping to the first argument of the right-hand side (`rhs`).

Value

No Return Value.

Examples

```
# Equivalent to `subset(head(iris), 1:nrow(head(iris)) %% 2 == 0)`
head(iris) %.>% subset(., 1:nrow(.) %% 2 == 0)

# Equivalent to `c(min(1:10), max(1:10))`
1:10 %.>% c(min(.), max(.))
```

Index