Package 'coreval'

September 12, 2026

Title Check Clinical Trial Data Against 'CDISC' Open Rules

Version 0.1.0

Description Finds conformance problems in clinical trial data without leaving R, using the openly published 'CDISC' Open Rules ('CORE'). Check a single dataset while you are still writing the code that builds it, or a whole study folder once it exists, and get the findings back as a tidy data frame pointing at the exact row and variable. Reads transport ('XPT'), 'SAS' and comma-separated files, plus 'Define-XML' when present, and covers rules for the 'SDTM', 'SEND' and 'TIG' standards. The rules are bundled inside the package, so nothing is downloaded and your data never leaves your machine: no internet, no API key, no account. When a rule cannot be checked - because it needs a dataset you did not supply, for instance - it is reported as skipped with the reason, never counted as a pass. Meant as a quick first pass before a qualified validation system, never as a replacement for one. An independent project: not affiliated with or endorsed by 'CDISC', and not a 'CORE'-certified conformance engine.

Copyright Hrach Gevorgyan holds the copyright in this package's own code. The bundled rule definitions and CDISC standards metadata under inst/extdata are derived from cdisc-org/cdisc-open-rules, copyright CDISC, and are used under its MIT license. See inst/COPYRIGHTS for the full notice and for which files that covers.

License MIT + file LICENSE

URL https://github.com/hrach-gevorgyan/coreval

BugReports https://github.com/hrach-gevorgyan/coreval/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1)

Imports data.table, haven

Suggests knitr, rmarkdown, testthat (>= 3.0.0), writexl, xml2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Hrach Gevorgyan [aut, cre, cph]

Maintainer Hrach Gevorgyan <hrach.gevorgyan@yandex.com>

Repository CRAN

Date/Publication 2026-09-12 12:30:08 UTC

Contents

`check_dataset` *Check one dataset, without needing a study folder*

Description

For when you are writing the code that builds a domain and want to know what is wrong with it right now. Give it the data frame you already have open, or the path to a single file.

Usage

```
check_dataset(
  x,
  domain = NULL,
  standard = NULL,
  version = NULL,
  use_case = NULL,
  max_records = 1000,
  include_deprecated = FALSE
)
```

Arguments

x	A data frame (or data.table::data.table()), or the path to one .xpt, .sas7bdat or .csv file.
domain	Two-letter domain code, e.g. "AE". Left as NULL, coreval takes it from your DOMAIN column, or the file name if there isn't one - so ae1.xpt from a split dataset is still checked as AE. Set it yourself if that guess is wrong.
standard	The standard the data follows, e.g. "SDTMIG" or "SENDIG". Rules are scoped to it, which is usually what you want - a SENDIG rule has nothing to say about an SDTM study. It is not free, though, and CDISC's coverage is uneven. The general "dates must be valid ISO 8601" rule (CORE-000547) is published for SENDIG and TIG but **not for SDTMIG**, whose only equivalents are TSVAL-specific or deprecated. So standard = "SDTMIG" genuinely stops a malformed RFSTDTC being reported. The report says how many rules were set aside; leave standard unset to see everything.
version	The standard's version, e.g. "3-4".
use_case	Optional use case (e.g. "INDH"), as in list_rules().
max_records	Most records to keep per rule, default 1000. A rule that flags every row of a large dataset would otherwise produce more findings than anyone can read or Excel can hold. The true count is kept in truncated. Use Inf for every record.
include_deprecated	Also run rules CDISC has deprecated. FALSE by default: a deprecated rule has a published replacement, so running both reports the same defect twice.

Value

list(findings, skipped, truncated) - the same shape check_study() returns, so write_findings() works on it unchanged.

What it cannot check on its own

Plenty of CDISC rules compare one dataset against another - an adverse event date against the subject's reference dates in DM, a visit against the trial design. Hand over a single dataset and those questions cannot be answered.

coreval does not guess. Those rules are skipped, and $skipped names the dataset each one wanted. Running them anyway would compare your data against columns that are not there and report problems that do not exist.

Most rules still run - across AE, DM, LB and VS, 76-84% of the applicable ones work on a single dataset. But the ones that cannot are the cross-dataset checks, which are often the ones that matter.

So a short $findings table here does not mean the data is clean. It is a quick first pass, not a verdict. Run check_study() on the whole folder before drawing conclusions.

See Also

check_study() to check a whole study folder.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30") # 30 February is not a date
)
result <- check_dataset(ae)
result$findings[result$findings$Value == "2024-02-30", ]

# Always look at what could not run:
nrow(result$skipped)
```

---

check_study *Check a whole study against CDISC Open Rules*

Description

Runs every rule that applies to every dataset in the study, including the ones that compare datasets against each other. Use this once the datasets exist as files; to check a single dataset while you are still writing the code that builds it, see `check_dataset()`.

Usage

```
check_study(
  study,
  use_case = NULL,
  max_records = 1000,
  include_deprecated = FALSE
)
```

Arguments

`study`	A study folder path, or a study object from `read_study()`. Passing the path is the usual way; reading first is only worth it when you want to check the same large study more than once without re-reading it, or to look at what was parsed.
`use_case`	Optional use case (e.g. `"INDH"`) to further filter which rules apply, as in `list_rules()`.
`max_records`	Most records to keep per rule, default 1000. A rule can flag every row - a missing `EPOCH` on a 200 000-row `LB` is 200 000 identical findings, more than Excel can hold. The true count is kept in `truncated` and the report shows it, so nothing is under-reported. Use `Inf` for every record.
`include_deprecated`	Also run rules CDISC has deprecated. `FALSE` by default: a deprecated rule has a published replacement, so running both reports the same defect twice.

Details

Findings come back one row per (dataset, record, variable), pointing at the exact spot. Some rules ask about a dataset as a whole rather than a particular row - those leave `Record` blank. A few ask about the study as a whole, such as "is DM present at all?"; those are answered once and reported under `Dataset = "STUDY"` rather than repeated for every domain.

Rules comparing against a define.xml do run, as long as the study has one and the `xml2` package is installed. Without both, they are skipped with a reason instead of being run against columns that are not there, which would report problems that do not exist. The same goes for any rule needing an operator or join coreval does not implement yet.

Value

Three tables:

- `findings` - what is wrong. One row per problem, with `Dataset`, `Record`, `Variable`, `Value`, the `issue` in words, and its `triage`. `Not in dataset` under `Value` means the rule wanted a variable you do not have, which is usually the finding itself.
- `skipped` - what could not be checked, with a `reason` for each. Read this one: an empty `findings` table can mean clean data *or* rules that never ran, and they look identical otherwise.
- `truncated` - rules that flagged more records than `max_records` kept, with how many they really found.

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))

result <- check_study(dir)
result$findings
unlink(dir, recursive = TRUE)
```

`filter_findings` *Narrow a result to the findings you care about*

Description

Saves writing `subset()` over `$findings` by hand, and - because it returns a result rather than a plain table - what comes back still prints as a readable report.

Usage

```
filter_findings(
  result,
  triage = NULL,
  dataset = NULL,
  rule = NULL,
  variable = NULL
)
```

Arguments

result	A result from check_dataset() or check_study().
triage	Keep only these triage levels, e.g. "wrong value". See print.coreval_result() for what the levels mean.
dataset	Keep only these datasets, e.g. "AE".
rule	Keep only these rule ids, e.g. "CORE-000547".
variable	Keep only findings naming these variables.

Value

A coreval_result holding the matching findings. $skipped is left alone: what could not be checked does not become less true because you narrowed what you are looking at.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
result <- check_dataset(ae)

# Just the things that are definitely wrong:
filter_findings(result, triage = "wrong value")
```

---

list_rules *Look up CORE rules*

Description

One way in for every question about the rule set: what rules exist, what a particular one checks, and which of them apply to a domain.

Usage

```
list_rules(
  id = NULL,
  domain = NULL,
  standard = NULL,
  version = NULL,
  use_case = NULL,
  include_deprecated = TRUE
)
```

Arguments

id	Return only these rules, e.g. "CORE-000547". This is how you look up a rule id the report gave you.
domain	Return only rules that apply to this domain, e.g. "AE", resolving each rule's Scope > Classes and Scope > Domains.
standard	Return only rules for this standard, e.g. "SDTMIG". Matched exactly, so "SENDIG" does not pick up "SENDIG-DART".
version	The standard's version, e.g. "3.4" (or "3-4"). Only narrows alongside standard, since a bare version is ambiguous across standards.
use_case	Optional use case (e.g. "INDH"). Rules with no Use Case constraint always pass.
include_deprecated	Include superseded rules. TRUE here, unlike check_dataset() and check_study(), which default to FALSE. The difference is deliberate: **listing is not running**. This function is the catalog of what is bundled, so hiding a fifth of it would make nrow(list_rules()) stop meaning "how many rules are there" - and source is right here to filter on. Checking is a different question, and there a superseded rule would report the same defect twice. Pass FALSE to see exactly what would run.

Details

The columns are the same whatever you ask for, so the result is safe to filter, join and script against.

Value

A data.table::data.table(), one row per rule: id, the one-line issue it reports, its fuller description, the guidance sentence from the Implementation Guide it enforces, the legacy_ids Pinnacle 21 uses for it, standard and standard_version, authority, rule_type, sensitivity, executability, source and status.

The commit the bundled rules came from is on the result as attr(x, "rules_version"); write_findings() records it in every exported file.

What source tells you

Not every bundled rule carries the same weight:

- "published" - Published/ upstream, fully tested. The trusted core.
- "deprecated_dir" - superseded by a published replacement. Not returned unless you ask for it, since running both reports the same defect twice.
- "fda_business_rules_draft" - FDA drafts that already ship test data.

Examples

```
# Everything
nrow(list_rules())

# What does the rule the report just named actually check?
```

```
list_rules(id = "CORE-000547")$issue

# What applies to AE under SDTMIG 3.4?
nrow(list_rules(domain = "AE", standard = "SDTMIG", version = "3.4"))

# Which snapshot of CDISC's rules is this?
attr(list_rules(), "rules_version")
```

`print.coreval_result` *Print a coreval check result as a readable report*

Description

Describes each problem in the rule's own words, worst first, with the rows and values that caused it. A whole-study result is grouped by dataset, with a summary first, so you can see where the trouble is before reading detail.

Usage

```
## S3 method for class 'coreval_result'
print(x, n = 10, rows = 3, guidance = FALSE, ...)
```

Arguments

`x`	A result from `check_dataset()` or `check_study()`.
`n`	Maximum problems to describe - per dataset, for a study result. The rest are counted, not listed. Default 10.
`rows`	Maximum example records to show per problem. Default 3.
`guidance`	Also print the sentence from the Implementation Guide that each rule enforces - the "why" behind it. Off by default: it roughly doubles the length of the report.
`...`	Ignored.

Value

`x`, invisibly.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
check_dataset(ae)
```

read_study *Read a study into coreval's internal representation*

Description

Detects whether path is a directory of XPT datasets (a real study) or a CORE test-case data/ directory (_variables.csv + one CSV per dataset, usually also .env and _datasets.csv), and reads either into the same internal representation, so the evaluator never has to know which one it got.

Usage

```
read_study(path)
```

Arguments

path	Directory path.

Details

Character columns use "" for blank/missing (never NA) to match how SAS XPT round-trips blanks; numeric columns use NA. Column types are taken from the source (XPT's own types, or _variables.csv's declared Char/Num) rather than guessed from the data, so numeric-looking identifiers (e.g. "007") are never silently coerced.

Value

A study object: list(datasets = <named list of domain -> list(data, meta, label)>, define, ct = NULL, standa define is the parsed Define-XML if one was found in path and the xml2 package is installed, and NULL otherwise; ct is always NULL (controlled terminology is not bundled). Each data is a data.table::data.table(); each meta is a data.table with columns variable, label, type; label is the dataset's own label (e.g. "Adverse Events"), or NA if unavailable. standard is the study's declared standard/version (e.g. list(product = "SDTMIG", version = "3-4")), read from a CORE test case's .env file - both NA for a real XPT-based study (no .env).

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))
study <- read_study(dir)
study$datasets$DM$data
unlink(dir, recursive = TRUE)
```

summary.coreval_result

Summarize a check result in a few lines

Description

The short form of `print.coreval_result()`, for when you have run a check inside a script, or just want to know whether the last fix helped.

Usage

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

Arguments

`object`	A result from `check_dataset()` or `check_study()`.
`...`	Ignored.

Value

A one-row `data.table::data.table()`, invisibly, with the counts it printed - so it can be logged or compared.

Examples

```
ae <- data.frame(
  STUDYID = "S1", DOMAIN = "AE", USUBJID = c("01", "01"),
  AESEQ = c(1, 2), AETERM = c("Headache", "Rash"),
  AESTDTC = c("2024-01-10", "2024-02-30")
)
summary(check_dataset(ae))
```

write_findings

Write conformance findings to a file

Description

Saves the result of `check_study()` to CSV or Excel, so findings can be shared with people who don't use R, tracked in a spreadsheet, or attached to a data-review document.

Usage

```
write_findings(result, path, tracking = TRUE)
```

Arguments

result	A result from check_dataset() or check_study().
path	Output file path. The extension decides the format: .xlsx for Excel, .csv (or anything else) for CSV.
tracking	Add the empty Status/Owner/Notes columns. TRUE by default; set FALSE for a file you intend to read back into R.

Details

Both tables are always written, never just the findings. A short findings table can mean clean data, or it can mean many rules were skipped, and those two situations look identical if the skipped table is dropped:

- **Excel** (.xlsx) — one workbook, two sheets: findings and skipped.
- **CSV** (.csv) — two files, since CSV has no notion of sheets. Findings go to path; skipped rules go to a sibling file with _skipped before the extension (issues.csv gives issues_skipped.csv).

Excel output needs the writexl package. It is a Suggests, so if it isn't installed you get a clear message telling you to install it or use .csv instead, rather than a failure part-way through writing.

Value

The paths actually written, invisibly. One element for Excel (a single workbook). For CSV, one path per file written: the findings, plus siblings for skipped and about, plus one for truncated when any rule flagged more records than were kept.

Columns for tracking

Three empty columns are added to the findings — Status, Owner and Notes — for you to fill in by hand once the file is open. They exist so a finding you have looked at and decided not to act on ("expected, see protocol deviation log") can be recorded next to the finding itself, rather than in a separate document nobody reads.

Examples

```
dir <- tempfile("coreval_study_")
dir.create(dir)
haven::write_xpt(data.frame(USUBJID = c("1", "2"), AGE = c(30, 65)), file.path(dir, "dm.xpt"))
study <- read_study(dir)
result <- check_study(study)

out <- file.path(dir, "findings.csv")
written <- write_findings(result, out)
basename(written)

unlink(dir, recursive = TRUE)
```

Index