Package: tidyOhdsiSolutions (via r-universe)

May 15, 2026

Title Tidy Utilities for Observational Medical Outcomes Partnership Common Data Model Workflows

Version 0.1.0

Depends R (>= 4.1.0)

Description Lightweight utilities for working with OMOP (Observational Medical Outcomes Partnership) Common Data Model (CDM) data in the Observational Health Data Sciences and Informatics ecosystem. Provides base-R re-implementations of common 'purrr' functional helpers, tools to convert plain data frames into 'CIRCE' concept set expressions, SQL generators for resolving concept sets against an OMOP vocabulary schema without requiring 'CirceR'.

License Apache License (>= 2)

Imports jsonlite, stats

Suggests CirceR, DatabaseConnector, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Alexander Alexeyuk [aut, cre]

Maintainer Alexander Alexeyuk <AlexanderAlexeyuk@gmail.com>

Repository https://a1exanderalexeyuk.r-universe.dev

Date/Publication 2026-04-15 14:08:57 UTC

RemoteUrl https://github.com/cran/tidyOhdsiSolutions

RemoteRef HEAD

RemoteSha 475378df05e78e2e77132ab8ffbcc9872d32b0e9

Contents

`as_mapper` *Convert Input to a Mapping Function*

Description

Converts a formula, existing function, or atomic vector into a function suitable for use in the `map`, `map2`, and `pmap` families.

Usage

```
as_mapper(.f)
```

Arguments

`.f`	A function, one-sided formula, or atomic vector. **Function** Returned unchanged. **Formula** A one-sided formula (e.g., `~ .x + 1`) is converted to a function. The right-hand side is evaluated in an environment where positional pronouns `.x`, `.y`, `.z` and numbered arguments `..1`, `..2`, ... are bound to the corresponding elements of `....` **Atomic vector** Returned unchanged (intended for use as an extractor with `pluck`).

Details

This helper mirrors the behaviour of `purrr::as_mapper` for the formula and function cases, providing a dependency-free alternative for internal use.

Value

A function. When `.f` is already a function or an atomic vector it is returned as-is. When `.f` is a formula the returned function accepts `...` and evaluates the formula's right-hand side with the positional bindings described above.

See Also

`map`, `map2`, `pmap`

Examples

```
f <- as_mapper(~ .x + 1)
f(10)

g <- as_mapper(~ .x + .y)
g(2, 3)
```

`buildConceptSetQueries` *Resolve Multiple Concept Sets to SQL*

Description

Applies `buildConceptSetQuery` to every element of a named list of concept set expressions and returns the results as a named list of SQL strings.

Usage

```
buildConceptSetQueries(conceptSetList, vocabularyDatabaseSchema = "vocabulary")
```

Arguments

`conceptSetList`	A named list of concept set expressions, each in CIRCE format (a list with an `$items` element).
`vocabularyDatabaseSchema`	Character. Schema containing the OMOP vocabulary tables. Defaults to `"vocabulary"`.

Value

A named list of SQL query strings, one per concept set.

See Also

`buildConceptSetQuery`

`buildConceptSetQuery` *Build SQL to resolve a concept set expression without Java/CirceR*

Description

Build SQL to resolve a concept set expression without Java/CirceR

Usage

```
buildConceptSetQuery(
  conceptSetExpression,
  vocabularyDatabaseSchema = "@vocabulary_database_schema"
)
```

Arguments

`conceptSetExpression`	An R list with structure matching CIRCE concept set format, or a JSON character string representing such a list. Must contain an `items` element, where each item has at minimum a `concept` list with a numeric `CONCEPT_ID`.
`vocabularyDatabaseSchema`	Character string. Schema where vocabulary tables live. Defaults to `"@vocabulary_database_schema"` for use with SqlRender.

Value

A SQL query string that resolves to a set of concept_ids

`cohortFromConceptSet` *Create a CirceR-compatible cohort definition from multiple concept set expressions*

Description

Builds a complete cohort definition from a named list of concept set expressions. Each concept set is assigned an auto-incremented ID (starting at 0) and all domains across all concept sets contribute to the primary criteria.

Usage

```
cohortFromConceptSet(
  conceptSetList,
  limit = "earliest",
  requiredObservation = c(0L, 0L),
  end = "observation_period_end_date",
  endArgs = list(),
  addSourceCriteria = FALSE
)
```

Arguments

`conceptSetList`	A named list of concept set expressions. Each element must be a list with an `$items` element in CIRCE concept set format.
`limit`	Character. Event limit: `"earliest"`, `"all"`, or `"latest"`. Default `"earliest"`.
`requiredObservation`	Integer vector of length 2: days of continuous observation required `c(prior, post)`. Default `c(0, 0)`.
`end`	Character. End strategy: `"observation_period_end_date"`, `"drug_exit"`, or `"fixed_exit"`. Default `"observation_period_end_date"`.
`endArgs`	A list of parameters for the chosen end strategy. See `buildEndStrategy` for details.
`addSourceCriteria`	Logical. If `TRUE`, duplicate each primary criteria entry with a source concept variant. Default `FALSE`.

Value

A nested R list representing a complete CirceR cohort expression. Can be converted to valid JSON via `cohortToJson`.

cohortToJson *Serialize a Cohort Expression to JSON*

Description

Converts the nested R list produced by `createConceptSetCohort` or `cohortFromConceptSet` to a JSON character string that is compatible with `CirceR::cohortExpressionFromJson()` and `CirceR::buildCohortQuery()`.

Usage

```
cohortToJson(cohort)
```

Arguments

`cohort`	A list produced by `createConceptSetCohort` or `cohortFromConceptSet`.

Value

A single character string of pretty-printed JSON.

See Also

`createConceptSetCohort`, `cohortFromConceptSet`

collectCsFromCohort *Extract Concept Set Expressions from a Cohort Definition*

Description

Iterates over the `ConceptSets` element of a CirceR-style cohort definition list and returns the concept set expressions as a named list. Names are derived from the concept set names, cleaned to lower camel case.

Usage

```
collectCsFromCohort(cohortDonor)
```

Arguments

`cohortDonor`	A named list with the structure of a CirceR cohort definition. Must contain a `ConceptSets` element where each entry has a `name` and an `expression` field.

Value

A named list where each name is a lower-camel-case concept set name and each value is the corresponding CIRCE concept set expression.

createConceptSetCohort *Create a CirceR-compatible cohort definition from a concept set expression*

Description

Builds a complete cohort definition as a nested R list that matches the structure expected by `CirceR::cohortExpressionFro` and `CirceR::buildCohortQuery()`. No Java or Capr dependency required.

Usage

```
createConceptSetCohort(
  conceptSetExpression,
  name = "Concept Set Cohort",
  limit = "first",
  requiredObservation = c(0L, 0L),
  end = "observation_period_end_date",
  endArgs = list(),
  addSourceCriteria = FALSE,
  codesetId = 0L
)
```

Arguments

`conceptSetExpression`	An R list with `$items` in CIRCE concept set format. Each item must have a `$concept` list with at minimum `CONCEPT_ID` and `DOMAIN_ID`.
`name`	Character. Cohort / concept set name. Default `"Concept Set Cohort"`.
`limit`	Character. Event limit: `"first"`, `"all"`, or `"last"`. Default `"first"`.
`requiredObservation`	Integer vector of length 2: days of continuous observation required `c(prior, post)`. Default `c(0, 0)`.
`end`	Character. End strategy: `"observation_period_end_date"`, `"drug_exit"`, or `"fixed_exit"`. Default `"observation_period_end_date"`.
`endArgs`	A list of parameters for the chosen end strategy. See `buildEndStrategy` for details.
`addSourceCriteria`	Logical. If `TRUE`, duplicate each primary criteria entry with a source concept variant. Default `FALSE`.
`codesetId`	Integer. The concept set ID to use internally. Default `0L`.

Value

A nested R list representing a complete CirceR cohort expression. Can be serialized to JSON via `jsonlite::toJSON()` and passed to `CirceR::buildCohortQuery()` or `CirceR::cohortExpressionFromJson()`.

`imap` *Apply a Function to Each Element with Its Index or Name*

Description

A family of indexed mapping functions modelled after `purrr::imap`. `.f` receives two arguments: the element value and its name (if `.x` is named) or its integer position (if `.x` is unnamed).

Usage

```
imap(.x, .f, ...)

imap_chr(.x, .f, ...)

imap_dbl(.x, .f, ...)

imap_dfr(.x, .f, ...)
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function or one-sided formula taking two arguments: the element value (`.x` / `..1`) and the index or name (`.y` / `..2`). Formulas are converted via `as_mapper`.
`...`	Additional arguments passed to `.f`.

Details

When `.x` has names, the second argument to `.f` is the element name (a character string). When `.x` is unnamed, the second argument is the positional index (an integer).

Value

`imap` A list the same length as `.x`.

`imap_chr` A character vector.

`imap_dbl` A double (numeric) vector.

`imap_dfr` A `data.frame` formed by `rbind`-ing per-element results.

See Also

`map`, `as_mapper`

Examples

```
imap_chr(c(a = 1, b = 2), ~ paste(.y, "=", .x))

imap_dbl(10:12, ~ .x + .y)
```

map *Apply a Function to Each Element of a List or Vector*

Description

A family of lightweight mapping functions modelled after `purrr::map`. `map()` always returns a list; the typed variants (`map_chr`, `map_dbl`, `map_int`, `map_lgl`) return atomic vectors of the indicated type; and the data-frame variants (`map_dfr`, `map_dfc`) row-bind or column-bind the results into a single `data.frame`.

Usage

```
map(.x, .f, ...)

map_chr(.x, .f, ...)

map_dbl(.x, .f, ...)

map_int(.x, .f, ...)

map_lgl(.x, .f, ...)

map_dfr(.x, .f, ...)

map_dfc(.x, .f, ...)
```

Arguments

`.x`	A list or atomic vector.
`.f`	A function, one-sided formula, or atomic vector. Formulas are converted via `as_mapper`.
`...`	Additional arguments passed to `.f` after the element of `.x`.

Details

These functions provide a dependency-free subset of the **purrr** mapping interface. `map()` is a thin wrapper around `lapply`; the typed variants use `vapply` with an appropriate `FUN.VALUE` template.

Value

`map` A list the same length as `.x`.

`map_chr` A character vector the same length as `.x`.

`map_dbl` A double (numeric) vector the same length as `.x`.

`map_int` An integer vector the same length as `.x`.

`map_lgl` A logical vector the same length as `.x`.

`map_dfr` A `data.frame` formed by `rbind`-ing the per-element results.

`map_dfc` A `data.frame` formed by `cbind`-ing the per-element results.

See Also

as_mapper, map2, pmap, imap, walk

Examples

```
map(1:3, ~ .x * 2)

map_dbl(1:3, ~ .x^2)

map_chr(letters[1:3], toupper)
```

---

map2 *Apply a Function to Pairs of Elements from Two Lists*

Description

A family of mapping functions that iterate over two inputs in parallel, modelled after purrr::map2. map2() returns a list; the typed variants return atomic vectors; and the data-frame variants row-bind or column-bind the results.

Usage

```
map2(.x, .y, .f, ...)

map2_chr(.x, .y, .f, ...)

map2_dbl(.x, .y, .f, ...)

map2_int(.x, .y, .f, ...)

map2_lgl(.x, .y, .f, ...)

map2_dfr(.x, .y, .f, ...)

map2_dfc(.x, .y, .f, ...)
```

Arguments

.x, .y	Two vectors or lists of the same length (or length one, which is recycled).
.f	A function or one-sided formula taking (at least) two arguments. Formulas are converted via as_mapper; .x is bound to the first argument and .y to the second.
...	Additional arguments passed to .f after the pair of elements.

Details

All variants are thin wrappers around mapply with SIMPLIFY = FALSE. The typed variants coerce the unlisted result to the target type.

Value

`map2` A list the same length as `.x`.

`map2_chr` A character vector.

`map2_dbl` A double (numeric) vector.

`map2_int` An integer vector.

`map2_lgl` A logical vector.

`map2_dfr` A `data.frame` formed by `rbind`-ing per-pair results.

`map2_dfc` A `data.frame` formed by `cbind`-ing per-pair results.

See Also

`as_mapper`, `map`, `pmap`, `walk2`

Examples

```
map2(1:3, 4:6, ~ .x + .y)

map2_dbl(1:3, 4:6, `+`)
```

`msg_abort` *Stop with a styled error message*

Description

Stop with a styled error message

Usage

```
msg_abort(..., call. = FALSE, class = NULL)
```

Arguments

`...`	Message parts
`call.`	Include call in error? Default FALSE
`class`	Optional condition class(es)

Value

Does not return; stops execution by throwing an error condition.

msg_blank *Print a blank line*

Description

Print a blank line

Usage

```
msg_blank()
```

Value

No return value, called for side effects.

msg_bullet *Print a bullet point*

Description

Print a bullet point

Usage

```
msg_bullet(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_danger *Print a danger / error message (styled, non-stopping)*

Description

Print a danger / error message (styled, non-stopping)

Usage

```
msg_danger(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_debug *Print a debug message (only when option is set)*

Description

Print a debug message (only when option is set)

Usage

```
msg_debug(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_header *Print a section header*

Description

Print a section header

Usage

```
msg_header(title, char = .sym("line"), width = getOption("width", 80L))
```

Arguments

title	Header text
char	Line character
width	Console width

Value

No return value, called for side effects.

msg_info *Print an informational message*

Description

Print an informational message

Usage

```
msg_info(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_kv *Print a two-column key-value table*

Description

Print a two-column key-value table

Usage

```
msg_kv(x, pad = 2L)
```

Arguments

x	Named vector / list
pad	Padding between columns

Value

No return value, called for side effects.

msg_list *Print a named list of items*

Description

Print a named list of items

Usage

```
msg_list(items, header = NULL)
```

Arguments

items	Named character vector or list
header	Optional header line

Value

No return value, called for side effects.

msg_process *Print a process / action message*

Description

Print a process / action message

Usage

```
msg_process(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_progress *Simple inline progress counter*

Description

Simple inline progress counter

Usage

```
msg_progress(total, prefix = "Progress", suffix = "")
```

Arguments

total	Total number of items
prefix	Text before counter
suffix	Text after counter

Value

A list with `$tick()`, `$reset()`, `$done()` methods

Examples

```
pb <- msg_progress(10, prefix = "Processing")
for (i in 1:10) { Sys.sleep(0.05); pb$tick() }
pb$done()
```

msg_rule *Print a simple rule / divider*

Description

Print a simple rule / divider

Usage

```
msg_rule(char = .sym("line"), width = getOption("width", 80L))
```

Arguments

char	Line character
width	Console width

Value

No return value, called for side effects.

msg_spinner *Animated spinner for long-running tasks*

Description

Animated spinner for long-running tasks

Usage

```
msg_spinner(msg = "Working", frames = c("|", "/", "-", "\\"))
```

Arguments

msg	Message to display
frames	Spinner frames (character vector)

Value

A list with `$spin()`, `$done()`, `$fail()` methods

Examples

```
sp <- msg_spinner("Loading data")
for (i in 1:20) { Sys.sleep(0.05); sp$spin() }
sp$done()
```

msg_success *Print a success message*

Description

Print a success message

Usage

```
msg_success(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_timed *Time a block and report duration*

Description

Time a block and report duration

Usage

```
msg_timed(expr, msg = "Elapsed", digits = 2L)
```

Arguments

expr	Expression to time
msg	Label for the timing message
digits	Decimal places for seconds

Value

The value of `expr`, returned invisibly.

Examples

```
msg_timed(Sys.sleep(0.1), "Sleep")
```

msg_todo *Print a todo item*

Description

Print a todo item

Usage

```
msg_todo(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_try *Run expr and catch + style any errors/warnings*

Description

Run expr and catch + style any errors/warnings

Usage

```
msg_try(expr, on_error = "abort")
```

Arguments

expr	Expression to evaluate
on_error	What to do: "abort" (re-stop), "warn", "message", "ignore"

Value

The value of `expr` if evaluation succeeds, or `NULL` invisibly if an error is caught and `on_error` is not `"abort"`.

msg_verbose *Message only when verbose option is TRUE*

Description

Message only when verbose option is TRUE

Usage

```
msg_verbose(..., verbose = NULL)
```

Arguments

...	Passed to `msg_info()`
verbose	Override; uses `getOption("pkg.verbose")` if NULL

Value

No return value, called for side effects.

msg_warn *Print a warning message (styled, non-stopping)*

Description

Print a warning message (styled, non-stopping)

Usage

```
msg_warn(...)
```

Arguments

...	Message parts (pasted together)

Value

No return value, called for side effects.

msg_warning *Warn with a styled warning message*

Description

Warn with a styled warning message

Usage

```
msg_warning(..., call. = FALSE, class = NULL)
```

Arguments

`...`	Message parts
`call.`	Include call in warning? Default FALSE
`class`	Optional condition class(es)

Value

No return value, called for side effects.

pluck *Extract an Element from a Nested Structure*

Description

Safely navigates into a nested list or vector using a sequence of accessors (names, integer positions, or functions), returning a default value when any accessor fails.

Usage

```
pluck(.x, ..., .default = NULL)
```

Arguments

`.x`	A list, vector, or other sub-settable object.
`...`	A sequence of accessors applied left-to-right to progressively drill into `.x`. Each accessor may be: **Integer or double** Used as a positional index via `[[`. Out-of-range indices return `.default`. **Character** Used as a name-based index via `[[`. Missing names return `.default`. **Function** Called with the current intermediate value as its sole argument. The return value becomes the new intermediate value.
`.default`	The value to return if any accessor fails (index out of bounds, name not found, intermediate or final value is `NULL`). Defaults to `NULL`.

Details

This function provides a dependency-free equivalent of `purrr::pluck`. It is intentionally strict: accessor types other than integer, character, or function cause an error.

When no accessors are supplied (`...` is empty), `.x` itself is returned.

Value

The value found at the end of the accessor chain, or `.default` if any step along the way fails.

See Also

`map`, `as_mapper`

Examples

```
x <- list(a = list(b = list(c = 42)))

pluck(x, "a", "b", "c")

pluck(x, "a", "z", .default = NA)

# Positional access
pluck(list(10, 20, 30), 2)

# Out-of-range returns .default
pluck(list(10, 20), 5, .default = -1)
```

---

pmap *Apply a Function to Multiple Lists in Parallel*

Description

A family of mapping functions that iterate over an arbitrary number of inputs in parallel, modelled after `purrr::pmap`. `pmap()` returns a list; the typed variants return atomic vectors; and the data-frame variants row-bind or column-bind the results.

Usage

```
pmap(.l, .f, ...)

pmap_chr(.l, .f, ...)

pmap_dbl(.l, .f, ...)

pmap_int(.l, .f, ...)

pmap_lgl(.l, .f, ...)
```

```
pmap_dfr(.l, .f, ...)

pmap_dfc(.l, .f, ...)
```

Arguments

`.l`	A list of vectors or lists, all of the same length. Each element of `.l` supplies one argument to `.f` at each position. If `.l` is named, the names are matched to the formal arguments of `.f`.
`.f`	A function or one-sided formula. The function should accept as many arguments as there are elements in `.l`. Formulas are converted via `as_mapper`.
`...`	Additional arguments passed to `.f` after the elements drawn from `.l`.

Details

All variants delegate to `mapply` via `do.call`, passing the elements of `.l` as parallel arguments.

Value

`pmap` A list whose length equals the common length of the elements of `.l`.

`pmap_chr` A character vector.

`pmap_dbl` A double (numeric) vector.

`pmap_int` An integer vector.

`pmap_lgl` A logical vector.

`pmap_dfr` A `data.frame` formed by `rbind`-ing per-index results.

`pmap_dfc` A `data.frame` formed by `cbind`-ing per-index results.

See Also

`as_mapper`, `map`, `map2`, `pwalk`

Examples

```
pmap(list(1:3, 4:6, 7:9), ~ ..1 + ..2 + ..3)

pmap_dbl(list(a = 1:3, b = 4:6), function(a, b) a * b)
```

toConceptSet — *Convert a single data.frame to a concept set expression list*

Description

Convenience wrapper around `toConceptSets` for a single concept set.

Usage

```
toConceptSet(
  df,
  name = "Concept Set",
  connection = NULL,
  vocabularyDatabaseSchema = NULL
)
```

Arguments

`df`	A data.frame/tibble with at minimum a `concept_id` column.
`name`	Character. Name for the concept set.
`connection`	Optional DatabaseConnector connection.
`vocabularyDatabaseSchema`	Optional vocabulary schema.

Value

A single concept set expression list with `$items`.

toConceptSets — *Convert Concept Set Expression Tibble(s) to Concept Set Expression Lists*

Description

Takes a named list of tibbles (each with concept details and flags) and converts them to the full R list structure matching the CIRCE concept set format. Optionally resolves concept metadata from a CDM database connection.

Usage

```
toConceptSets(x, connection = NULL, vocabularyDatabaseSchema = NULL)
```

Arguments

x	A named list of data.frames/tibbles. Each must contain at minimum: **concept_id** Integer. The OMOP concept ID. Optional columns (defaults applied if missing): **excluded** Logical. Whether the concept is excluded. Default FALSE. **descendants** Logical. Whether to include descendants. Default FALSE. **mapped** Logical. Whether to include mapped concepts. Default FALSE. **concept_name** Character. Concept name. **domain_id** Character. Domain ID (e.g., "Condition", "Drug"). **vocabulary_id** Character. Vocabulary ID (e.g., "SNOMED", "RxNorm"). **concept_class_id** Character. Concept class ID. **standard_concept** Character. Standard concept flag ("S", "C", etc.). **concept_code** Character. Source concept code. **invalid_reason** Character. Validity flag ("V" = valid).
connection	Optional. A DatabaseConnector connection object. If provided, concept metadata will be resolved from the vocabulary tables.
vocabularyDatabaseSchema	Character. Required if connection is provided. The schema containing the OMOP vocabulary tables.

Value

A named list of concept set expressions, each with an $items element in CIRCE format.

---

walk *Apply a Function for Side Effects*

Description

Execute .f on each element (or pair / tuple of elements) for its side effects, returning the input invisibly. walk() iterates over a single input, walk2() over two inputs in parallel, and pwalk() over an arbitrary number of inputs stored in a list.

Usage

```
walk(.x, .f, ...)

walk2(.x, .y, .f, ...)

pwalk(.l, .f, ...)
```

Arguments

.x	A list or atomic vector.
.f	A function, one-sided formula, or atomic vector. Formulas are converted via as_mapper.
...	Additional arguments passed to .f.
.y	A vector or list the same length as .x (used by walk2 only).
.l	A list of vectors or lists of equal length (used by pwalk only).

Details

These functions are the side-effect counterparts of map, map2, and pmap, respectively.

Value

walk Invisibly returns .x.

walk2 Invisibly returns .x.

pwalk Invisibly returns .l.

See Also

map, map2, pmap

Examples

```
walk(1:3, print)

walk2(letters[1:3], 1:3, function(l, n) cat(l, "=", n, "\n"))
```

Index