

# Package ‘tidyOhdsiSolutions’

May 8, 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** CRAN

**Date/Publication** 2026-04-15 12:10:02 UTC

## Contents

as_mapper . . . . .	2
buildConceptSetQueries . . . . .	3
buildConceptSetQuery . . . . .	4
cohortFromConceptSet . . . . .	5

cohortToJson . . . . .	6
collectCsFromCohort . . . . .	6
createConceptSetCohort . . . . .	7
imap . . . . .	8
map . . . . .	9
map2 . . . . .	10
msg_abort . . . . .	11
msg_blank . . . . .	12
msg_bullet . . . . .	12
msg_danger . . . . .	13
msg_debug . . . . .	13
msg_header . . . . .	14
msg_info . . . . .	14
msg_kv . . . . .	15
msg_list . . . . .	15
msg_process . . . . .	16
msg_progress . . . . .	16
msg_rule . . . . .	17
msg_spinner . . . . .	17
msg_success . . . . .	18
msg_timed . . . . .	18
msg_todo . . . . .	19
msg_try . . . . .	19
msg_verbose . . . . .	20
msg_warn . . . . .	20
msg_warning . . . . .	21
pluck . . . . .	21
pmap . . . . .	22
toConceptSet . . . . .	24
toConceptSets . . . . .	24
walk . . . . .	25

<b>Index</b>	<b>27</b>
--------------	-----------

---

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	<i>Serialize a Cohort Expression to JSON</i>
--------------	--

---

**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	<i>Extract Concept Set Expressions from a Cohort Definition</i>
---------------------	---

---

**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::cohortExpressionFromJson()` 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

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

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

### 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

<code>.x</code>	A list or atomic vector.
<code>.f</code>	A function, one-sided formula, or atomic vector. Formulas are converted via <a href="#">as_mapper</a> .
<code>...</code>	Additional arguments passed to <code>.f</code> after the element of <code>.x</code> .

### 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

<code>map</code>	A list the same length as <code>.x</code> .
<code>map_chr</code>	A character vector the same length as <code>.x</code> .
<code>map_dbl</code>	A double (numeric) vector the same length as <code>.x</code> .
<code>map_int</code>	An integer vector the same length as <code>.x</code> .
<code>map_lgl</code>	A logical vector the same length as <code>.x</code> .
<code>map_dfr</code>	A <code>data.frame</code> formed by <a href="#">rbind</a> -ing the per-element results.
<code>map_dfc</code>	A <code>data.frame</code> formed by <a href="#">cbind</a> -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**

<code>.x, .y</code>	Two vectors or lists of the same length (or length one, which is recycled).
<code>.f</code>	A function or one-sided formula taking (at least) two arguments. Formulas are converted via <a href="#">as_mapper</a> ; <code>.x</code> is bound to the first argument and <code>.y</code> to the second.
<code>...</code>	Additional arguments passed to <code>.f</code> 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	<i>Stop with a styled error message</i>
-----------	---

---

**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	<i>Print a blank line</i>
-----------	---------------------------

---

**Description**

Print a blank line

**Usage**

msg\_blank()

**Value**

No return value, called for side effects.

---

msg_bullet	<i>Print a bullet point</i>
------------	-----------------------------

---

**Description**

Print a bullet point

**Usage**

msg\_bullet(...)

**Arguments**

...            Message parts (pasted together)

**Value**

No return value, called for side effects.

---

msg_danger	<i>Print a danger / error message (styled, non-stopping)</i>
------------	--

---

**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	<i>Print a debug message (only when option is set)</i>
-----------	--

---

**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	<i>Print a section header</i>
------------	-------------------------------

---

**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	<i>Print an informational message</i>
----------	---------------------------------------

---

**Description**

Print an informational message

**Usage**

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

**Arguments**

...	Message parts (pasted together)
-----	---------------------------------

**Value**

No return value, called for side effects.

---

msg_kv	<i>Print a two-column key-value table</i>
--------	---

---

**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	<i>Print a named list of items</i>
----------	------------------------------------

---

**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	<i>Print a process / action message</i>
-------------	---

---

**Description**

Print a process / action message

**Usage**

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

**Arguments**

...            Message parts (pasted together)

**Value**

No return value, called for side effects.

---

msg_progress	<i>Simple inline progress counter</i>
--------------	---------------------------------------

---

**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	<i>Print a success message</i>
-------------	--------------------------------

---

**Description**

Print a success message

**Usage**

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

**Arguments**

...            Message parts (pasted together)

**Value**

No return value, called for side effects.

---

msg_timed	<i>Time a block and report duration</i>
-----------	---

---

**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	<i>Print a todo item</i>
----------	--------------------------

---

**Description**

Print a todo item

**Usage**

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

**Arguments**

...            Message parts (pasted together)

**Value**

No return value, called for side effects.

---

msg_try	<i>Run expr and catch + style any errors/warnings</i>
---------	---

---

**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	<i>Message only when verbose option is TRUE</i>
-------------	---

---

**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	<i>Print a warning message (styled, non-stopping)</i>
----------	---

---

**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	<i>Warn with a styled warning message</i>
-------------	---

---

**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	<i>Extract an Element from a Nested Structure</i>
-------	---

---

**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: <ul style="list-style-type: none"> <li><b>Integer or double</b> Used as a positional index via <code>[[</code>. Out-of-range indices return <code>.default</code>.</li> <li><b>Character</b> Used as a name-based index via <code>[[</code>. Missing names return <code>.default</code>.</li> <li><b>Function</b> Called with the current intermediate value as its sole argument. The return value becomes the new intermediate value.</li> </ul>
.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	<i>Convert a single data.frame to a concept set expression list</i>
--------------	---

---

### 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  $i$  items.

---

toConceptSets	<i>Convert Concept Set Expression Tibble(s) to Concept Set Expression Lists</i>
---------------	---

---

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

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

- \* **imap**
  - imap, 8
- \* **map2**
  - map2, 10
- \* **map**
  - map, 9
- \* **pmap**
  - pmap, 22
- \* **walk**
  - walk, 25

as\_mapper, 2, 3, 8–11, 22, 23, 26

buildConceptSetQueries, 3  
buildConceptSetQuery, 3, 4, 4  
buildEndStrategy, 5, 7

cbind, 9, 11, 23  
cohortFromConceptSet, 5, 6  
cohortToJson, 5, 6  
collectCsFromCohort, 6  
createConceptSetCohort, 6, 7

do.call, 23

imap, 8, 8, 10  
imap\_chr (imap), 8  
imap\_dbl (imap), 8  
imap\_dfr (imap), 8

lapply, 9

map, 3, 8, 9, 9, 11, 22, 23, 26  
map2, 3, 10, 10, 23, 26  
map2\_chr (map2), 10  
map2\_dbl (map2), 10  
map2\_dfc (map2), 10  
map2\_dfr (map2), 10  
map2\_int (map2), 10  
map2\_lgl (map2), 10  
map\_chr (map), 9  
map\_dbl (map), 9  
map\_dfc (map), 9  
map\_dfr (map), 9  
map\_int (map), 9  
map\_lgl (map), 9  
mapply, 10, 23  
msg\_abort, 11  
msg\_blank, 12  
msg\_bullet, 12  
msg\_danger, 13  
msg\_debug, 13  
msg\_header, 14  
msg\_info, 14  
msg\_kv, 15  
msg\_list, 15  
msg\_process, 16  
msg\_progress, 16  
msg\_rule, 17  
msg\_spinner, 17  
msg\_success, 18  
msg\_timed, 18  
msg\_todo, 19  
msg\_try, 19  
msg\_verbos, 20  
msg\_warn, 20  
msg\_warning, 21

pluck, 3, 21, 22  
pmap, 3, 10, 11, 22, 22, 26  
pmap\_chr (pmap), 22  
pmap\_dbl (pmap), 22  
pmap\_dfc (pmap), 22  
pmap\_dfr (pmap), 22  
pmap\_int (pmap), 22  
pmap\_lgl (pmap), 22  
pwalk, 23  
pwalk (walk), 25

rbind, 8, 9, 11, 23

toConceptSet, [24](#)  
toConceptSets, [24](#)

vapply, [9](#)

walk, [10](#), [25](#)  
walk2, [11](#)  
walk2 (walk), [25](#)