Package: phiperio (via r-universe)

June 4, 2026

Title PhIP-Seq Data Import and Validation Tools

Version 0.5.0

Description Provides utilities to import, validate, and manage PhIP-Seq datasets, including standardized conversion pipelines, data checks, and access to cached peptide metadata.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports chk, cli, DBI, dbplyr, dplyr, duckdb, rlang, tibble, tools, withr, yaml

Suggests testthat, mockery, knitr, rmarkdown

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/Polymerase3/phiperio, https://polymerase3.github.io/phiperio/

BugReports https://github.com/Polymerase3/phiperio/issues

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev xz-utils

Repository https://polymerase3.r-universe.dev

Date/Publication 2026-02-04 13:42:22 UTC

RemoteUrl https://github.com/Polymerase3/phiperio

RemoteRef main

RemoteSha cd3f886b5dfed8be36e32ed628cb48c40a7993d0

Contents

`add_exist` *Ensure an existence flag (all ones) on* `data_long`

Description

Appends/overwrites a column (default: "exist") filled with 1L on the lazy `data_long` table. Preserves laziness; no collection is forced.

Usage

```
add_exist(phip_data, exist_col = "exist", overwrite = FALSE)
```

Arguments

`phip_data`	A <phip_data> object.
`exist_col`	Name of the existence column to append/overwrite.
`overwrite`	If FALSE and the column exists, abort with a phiperio-style error.

Value

Modified <phip_data> with updated `data_long`.

Examples

```
pd <- load_example_data()
pd <- add_exist(pd, overwrite = TRUE) # overwrites if present
```

close.phip_data *Close phip_data connections*

Description

Closes any open database connections held by a phip_data object. This includes the main data_long backend connection and any peptide-library connection stored in attributes or metadata. The method is idempotent and safe to call multiple times.

Usage

```
## S3 method for class 'phip_data'
close(con, ...)
```

Arguments

con	A valid phip_data object.
...	Unused (for S3 generic compatibility).

Value

The input phip_data object, invisibly.

Examples

```
pd <- load_example_data()
close(pd)
```

convert_legacy *Convert legacy Carlos-style input to a modern* **phip_data** *object*

Description

convert_legacy() ingests the original three-file PhIP-Seq input (binary *exist* matrix, *samples* metadata, optional *timepoints* map). Paths can be supplied directly or via a single YAML config; explicit arguments always override the YAML. The function normalises the chosen DuckDB storage, validates every file, and returns a ready-to-use phip_data object.

Usage

```
convert_legacy(
  exist_file = NULL,
  fold_change_file = NULL,
  samples_file = NULL,
  input_file = NULL,
  hit_file = NULL,
  timepoints_file = NULL,
  extra_cols = NULL,
  output_dir = NULL,
  peptide_library = TRUE,
  n_cores = 8,
  materialise_table = TRUE,
  config_yaml = NULL
)
```

Arguments

`exist_file`	Path to the **exist** CSV (peptide x sample binary matrix). *Required unless given in* `config_yaml`.
`fold_change_file`	Path to the **fold_change** CSV (peptide x sample numeric matrix). *Required unless given in* `config_yaml`.
`samples_file`	Path to the **samples** CSV (sample metadata). *Required unless given in* `config_yaml`.
`input_file`, `hit_file`	Paths to the **raw_counts** CSV (peptide x sample integer matrix). *Required unless given in* `config_yaml`.
`timepoints_file`	Path to the **timepoints** CSV (subject <-> sample mapping). Optional for cross-sectional data.
`extra_cols`	Character vector of extra metadata columns to retain.
`output_dir`	*Deprecated.* Ignored with a warning.
`peptide_library`	logical, defining if the `peptide_library` is to be downloaded from the official `phiperio` GitHub
`n_cores`	Integer >= 1. Number of CPU threads DuckDB may use while reading and writing files.
`materialise_table`	Logical. If `FALSE` the result is registered as a **view**; if `TRUE` the table is fully **materialised** and stored on disk, trading higher load time and storage for faster repeated queries.
`config_yaml`	Optional YAML file containing any of the above parameters (see example).

Details

Input files are validated in two stages:

- **Fast-fail** checks (paths, extensions, and required arguments) run during path resolution.
- **Data validation** (required columns, uniqueness, value ranges, etc.) is centralized in `validate_phip_data()`.

Value

A validated `phip_data` object whose `data_long` slot is backed by a DuckDB connection.

Examples

```
## 1. Direct-path usage (package example files)
ext <- system.file("extdata", package = "phiperio")
pd <- convert_legacy(
  exist_file = file.path(ext, "exist.csv"),
  samples_file = file.path(ext, "samples_meta.csv"),
  timepoints_file = file.path(ext, "samples2ind_timepoints.csv"),
  peptide_library = FALSE
)

## 2. YAML-driven usage (explicit args override YAML)
pd <- convert_legacy(
  config_yaml = file.path(ext, "config.yaml"),
  peptide_library = FALSE
)
```

`convert_standard` *Convert raw PhIP-Seq output into a* `phip_data` *object*

Description

`convert_standard()` ingests a "long" table of PhIPsSeq read counts / enrichment statistics, optionally expands it to the full `sample_id x peptide_id` grid, and registers the result in DuckDB. The function returns a fully initialised `phip_data` object that can be queried with the tidy API used throughout the package.

Usage

```
convert_standard(
  data_long_path,
  sample_id = NULL,
  peptide_id = NULL,
  subject_id = NULL,
  timepoint = NULL,
  exist = NULL,
  fold_change = NULL,
  counts_input = NULL,
  counts_hit = NULL,
```

```
  sample_id_from_filenames = FALSE,
  n_cores = 8,
  materialise_table = TRUE,
  auto_expand = FALSE,
  peptide_library = TRUE
)
```

Arguments

`data_long_path` Character scalar. File or directory containing the *long-format* PhIP-Seq data. Allowed extensions are `.csv` and `.parquet`. Directories are treated as partitions of a parquet set.

`sample_id`, `peptide_id`, `subject_id`, `timepoint`, `exist`, `fold_change`, `counts_input`, `counts_hit`
Optional character strings. Supply these only if your column names differ from the defaults (`"sample_id"`, `"peptide_id"`, `"subject_id"`, `"timepoint"`, `"exist"`, `"fold_change"`, `"counts_input"`, `"counts_hit"`). Each argument should contain the *name* of the column in the incoming data; `NULL` lets the default stand.

`sample_id_from_filenames`
Logical. If `TRUE` and `data_long_path` is a **directory of files** (CSV or Parquet), automatically derive `sample_id` from each filename (basename without extension). Requires that no `sample_id` mapping is provided and that the input files do not already contain a `sample_id` column. Default: `FALSE`.

`n_cores` Integer >= 1. Number of CPU threads DuckDB may use while reading and writing files.

`materialise_table`
Logical. If `FALSE` the result is registered as a **view**; if `TRUE` the table is fully **materialised** and stored on disk, trading higher load time and storage for faster repeated queries.

`auto_expand` Logical. If `TRUE` and the incoming data are **not** a complete Cartesian product of `sample_id x peptide_id`, missing combinations are generated:

- Columns that are constant within each `sample_id` (metadata) are copied to the new rows.
- Non-recyclable measurement columns (`fold_change`, `exist`, `counts_input`, `counts_hit`, etc.) are initialised to 0. The expanded table replaces the original *in place*.

`peptide_library`
Logical. If `TRUE` (default) `convert_standard()` will attempt to locate and attach the matching peptide-library metadata for downstream annotation. Set to `FALSE` to skip this step.

Details

Paths are resolved to absolute form before any work begins, and explicit checks confirm existence as well as extension validity.

Value

An S3 object of class `phip_data` containing:

`data_long` The (possibly expanded) long-format table.

`peptide_library` Loaded peptide-library metadata (if `peptide_library = TRUE`).

`meta` List with DuckDB connection handles.

See Also

- `create_data()` for the object constructor.
- `dplyr::tbl()` to query DuckDB tables lazily.

Examples

```
# Basic import, auto-detecting default column names
phip_obj <- convert_standard(
  data_long_path = get_example_path("phip_mixture"),
  n_cores = 4,
  materialise_table = TRUE
)

# Import a CSV and rename columns
tmp_csv <- tempfile(fileext = ".csv")
utils::write.csv(
  data.frame(
    sample = c("s1", "s1"),
    pep = c("p1", "p2"),
    exist = c(1, 0),
    stringsAsFactors = FALSE
  ),
  tmp_csv,
  row.names = FALSE
)
phip_mem <- convert_standard(
  data_long_path = tmp_csv,
  sample_id      = "sample",
  peptide_id     = "pep",
  peptide_library = FALSE,
  materialise_table = FALSE
)
```

`create_data` *Construct a* **phip_data** *object*

Description

Creates a fully-validated S3 object that bundles the tidy PhIP-Seq counts (`data_long`), a peptide-library annotation table, and other metadata. The data itself is validated via `validate_phip_data()`.

Usage

```
create_data(
  data_long,
  peptide_library = TRUE,
  auto_expand = TRUE,
  materialise_table = TRUE,
  meta = list()
)
```

Arguments

data_long	A tidy data frame (or tbl_lazy) with one row per peptide_id x sample_id combination. **Required.**
peptide_library	A data frame with one row per peptide_id and its annotations. If NULL, the package's current default library is used.
auto_expand	Logical. If TRUE and the input is **not** already the full Cartesian product of sample_id x peptide_id, the function fills in the missing combinations. • Columns that are constant within a sample_id (metadata) are duplicated to the newly created rows. • Measurement columns such as fold_change, exist, raw counts, or any other non-recyclable fields are initialised to 0. The expanded table replaces data_long in place.
materialise_table	Logical. If FALSE (default) the result is registered as a **view**. If TRUE the result is fully **materialised** and stored as a physical table, which speeds up repeated queries at the cost of extra memory/disk.
meta	Optional named list of metadata flags to pre-populate the meta slot (rarely needed by users).

Value

An object of class "phip_data".

Examples

```
## minimal constructor call
tidy_counts <- data.frame(
  sample_id = c("s1", "s1"),
  peptide_id = c("p1", "p2"),
  exist = c(1, 0),
  stringsAsFactors = FALSE
)
pd <- create_data(
  data_long = tidy_counts,
  peptide_library = FALSE,
  materialise_table = FALSE
)
```

expand_data *Expand to a full* sample_id * peptide_id *grid*

Description

Create the full Cartesian product of samples and peptides and join back per-sample metadata. For rows introduced by the expansion, numeric/integer columns are filled with 0 and logical columns with FALSE, unless overridden via fill_override.

Usage

```
expand_data(
  x,
  key_col = "sample_id",
  id_col = "peptide_id",
  fill_override = NULL,
  add_exist = FALSE,
  exist_col = "exist",
  validate = TRUE,
  ...
)
```

Arguments

x	A <phip_data> object.
key_col	Name(s) of the sample identifier column(s). Character scalar or vector, e.g. "sample_id" or c("subject_id", "timepoint_factor").
id_col	Name of the peptide identifier column. Default "peptide_id".
fill_override	Optional named list of fill values for **introduced** rows, e.g. list(present = 0L, fold_change = NA_real_). User-provided entries take precedence over the defaults.
add_exist	If TRUE, add an integer existence flag (0/1) marking whether a row was present before the expansion.
exist_col	Name for the existence flag. If this column already exists, it will be **overwritten**.
validate	Logical; if TRUE, perform input checks for required columns and uniqueness. Set to FALSE when these checks were already performed upstream (e.g., inside validate_phip_data()).
...	Reserved for future extensions; currently unused.

Details

Updates x$data_long in place (preserving laziness unless you later compute() / collect()).

Value

The updated <phip_data> object.

Examples

```
pd <- load_example_data()
pd <- expand_data(pd, fill_override = list(fold_change = NA_real_))
```

`export_parquet` *Export a phip_data Table to Parquet*

Description

Exports the `data_long` table from a **phip_data** object to disk in Apache Parquet format.

Usage

```
export_parquet(x, path)
```

Arguments

`x`	A <phip_data> object or a data frame.
`path`	File path (character) to save the output `.parquet` file.

Value

NULL (invisibly).

Note

The export is performed directly and efficiently from the database/lazy table without reading all data into memory.

Examples

```
pd <- load_example_data()
out_path <- tempfile(fileext = ".parquet")
export_parquet(pd, out_path)
unlink(out_path)
```

get_counts *Retrieve the main PhIP-Seq counts table*

Description

Quick accessor for the `data_long` slot of a **phip_data** object.

Usage

```
get_counts(x)
```

Arguments

`x`	A valid `phip_data` object.

Value

A tibble or lazy table with one row per peptide * sample pair.

Examples

```
pd <- load_example_data()
tbl <- get_counts(pd)
```

get_example_path *Path to example PhIP-Seq datasets shipped with phiperio*

Description

Path to example PhIP-Seq datasets shipped with phiperio

Usage

```
get_example_path(name = c("phip_mixture"))
```

Arguments

`name`	Character scalar. Name of the example dataset. Currently supported: `"phip_mixture"`.

Value

A character scalar with an absolute path to the file.

Examples

```
sim_path <- get_example_path("phip_mixture")
# phip_obj <- convert_standard(sim_path)
```

get_meta *Retrieve the metadata list*

Description

Accesses the meta slot, which holds flags such as whether the table is a full peptide * sample grid, the available outcome columns, etc.

Usage

```
get_meta(x)
```

Arguments

x	A valid phip_data object.

Value

A named list.

Examples

```
pd <- load_example_data()
meta <- get_meta(pd)
```

get_peptide_library *Retrieve the peptide metadata table into DuckDB, forcing atomic types*

Description

This function uses the phiperio logging utilities for consistent, ASCII-only progress messages and timing. Long-running steps are bracketed with .ph_with_timing(), and informational/warning/error messages are emitted via .ph_log_info(), .ph_log_ok(), .ph_warn(), and .ph_abort().

- Downloads the RDS once, sanitizes types (logical, character, numeric), and writes into a DuckDB cache on disk.
- Subsequent calls return a lazy tbl_dbi without loading into R memory.

Usage

```
get_peptide_library(force_refresh = FALSE)
```

Arguments

force_refresh	Logical. If TRUE, re-downloads and rebuilds the cache.

Details

Caching: A persistent DuckDB database is created under the user cache directory (via `tools::R_user_dir("phiperio", "cache")`). You can override this location with `options(phiperio.cache_dir = \"...\")`. The `force_refresh` argument bypasses the fast path and rebuilds the cache.

Sanitization: Columns are stripped of attributes, list-columns are flattened, textual `"NaN"` and numeric `NaN` are coerced to `NA`. Binary 0/1 fields are converted to `logical`, `"TRUE"/"FALSE"` (case-insensitive) are converted to `logical`, and numeric-looking character columns (beyond trivial 0/1) are converted to `numeric`. All other atomic types are preserved.

Integrity check: If a SHA-256 checksum is provided, a warning is logged when the downloaded file's checksum does not match the expected value.

Value

A `dplyr::tbl_dbi` pointing to the `peptide_meta` table. The returned object carries an attribute `"duckdb_con"` with the open `DBI` connection.

See Also

`dplyr::tbl()`, `DBI::dbConnect()`, `duckdb::duckdb()`

Examples

```
lib <- get_peptide_library()
```

---

load_example_data *Load Example PhIP-Seq Dataset as <phip_data>*

Description

Convenience helper to quickly load a shipped example dataset ("phip_mixture") into a `<phip_data>` object, suitable for downstream analysis and visualization. This function wraps `convert_standard`, automatically supplying the correct parameters for the included example data.

Usage

```
load_example_data(name = c("phip_mixture", "small_mixture"))
```

Arguments

`name`	Character scalar. Name of the shipped example dataset. Currently supported: `"phip_mixture"`, `"small_mixture"`.

Value

A `<phip_data>` object created from the chosen example dataset.

Examples

```
# Load the example data shipped with the package:
ex <- load_example_data()
# ex is now a <phip_data> object ready for analysis

# Specify the dataset name explicitly
ex2 <- load_example_data("small_mixture")

# Use with downstream analysis/plotting functions as needed
```

`merge.phip_data` *Merge or join a* `phip_data` *object*

Description

Merge or join a `phip_data` object

Usage

```
## S3 method for class 'phip_data'
merge(x, y, ...)
```

Arguments

`x`	A `phip_data` object.
`y`	A data-frame-like object *or* another `phip_data`.
`...`	Arguments forwarded to either `base::merge()` or the chosen **dplyr** join (e.g. `by =`, `suffix =`, etc.).

Value

A new `phip_data` whose `data_long` contains the merged / joined tibble.

Examples

```
pd <- load_example_data()
merged <- merge(pd, pd, by = c("sample_id", "peptide_id"))
```

phip_data_join *dplyr joins for* phip_data

Description

dplyr joins for phip_data

Usage

```
## S3 method for class 'phip_data'
left_join(x, y, ...)

## S3 method for class 'phip_data'
right_join(x, y, ...)

## S3 method for class 'phip_data'
inner_join(x, y, ...)

## S3 method for class 'phip_data'
full_join(x, y, ...)

## S3 method for class 'phip_data'
semi_join(x, y, ...)

## S3 method for class 'phip_data'
anti_join(x, y, ...)
```

Arguments

x	A phip_data object.
y	A phip_data or a data frame / tbl.
...	Passed to the corresponding dplyr::<join> function.

Value

A phip_data object with updated data_long.

Examples

```
pd <- load_example_data()
joined <- dplyr::left_join(pd, pd, by = c("sample_id", "peptide_id"))
```

Index