

Package: PhysioLake (via r-universe)

October 3, 2026

Title Versioned, Lineage-Tracked Data Management for the Physio Ecosystem

Version 0.1.3

Description Integrates the PhysioExperiment ecosystem with 'OmicsLake', a versioned, on-disk data-management engine built on 'DuckDB', 'Apache Arrow' and 'Parquet' with automatic dataset-level lineage tracking. Rather than reimplementing a data lake, PhysioLake inherits the domain-agnostic OmicsLake core (storage backend, snapshots and time-travel, cross-dataset lineage, query layer) and registers physiology-specific adapters and utilities: 'PhysioExperiment' objects are stored and retrieved with full fidelity, preserving the sampling-rate slot and the W3C-PROV provenance held in object metadata. This connects the ecosystem's per-object (micro) provenance with the lake's cross-dataset (macro) lineage.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.3.0), PhysioExperiment (>= 2.0.0)

Imports R6, methods, S4Vectors, SummarizedExperiment

Suggests PhysioCore, OmicsLake (>= 0.99.4), PhysioPreprocess, testthat (>= 3.2.0), knitr, rmarkdown

VignetteBuilder knitr

Remotes matsui-lab/OmicsLake

Additional_repositories <https://matsui-lab.r-universe.dev>

Config/testthat/edition 3

Config/pak/sysreqs zlib1g-dev

Repository <https://x-biosignal.r-universe.dev>

Date/Publication 2026-10-03 13:11:47 UTC

RemoteUrl <https://github.com/x-biosignal/PhysioLake>
RemoteRef HEAD
RemoteSha 106fdbe2ba7fd51a0a3b0db15836cbfea54a29c3

Contents

PhysioExperimentAdapter	2
physioProvenance	4
physioPut	5
physioPutBundle	6
registerPhysioAdapters	7
Index	8

PhysioExperimentAdapter
<i>PhysioExperiment adapter for OmicsLake</i>

Description

PhysioExperiment adapter for OmicsLake
PhysioExperiment adapter for OmicsLake

Details

An [OmicsLake::LakeAdapter](#) that stores and retrieves [PhysioExperiment](#) objects in an OmicsLake with full fidelity.

A [PhysioExperiment](#) is a [SummarizedExperiment](#) subclass with one extra slot (`samplingRate`); its provenance, events, and derived estimates live in `metadata()`. [OmicsLake](#)’s generic [SEAdapter](#) already round-trips assays, `colData`, `rowData`, and `metadata`, but it reconstructs a *plain* [SummarizedExperiment](#) – losing the class identity and the sampling rate. This adapter inherits [SEAdapter](#) and adds only that missing piece: it stashes the sampling rate in `metadata` on `put()` and rebuilds a [PhysioExperiment](#) on `get()`. Because provenance is carried in `metadata()`, it survives the round trip unchanged (verified: `provenanceHash()` is identical before and after), so the ecosystem’s per-object (micro) W3C-PROV provenance lands intact next to [OmicsLake](#)’s cross-dataset (macro) lineage. Assays that are not 2-D matrices (e.g. epoched *time x channel x trial* arrays), which the generic [SEAdapter](#) would collapse when storing them as flat tables, are likewise stashed in `metadata` and spliced back in their original order on `get()`.

A higher `priority()` than the generic [SEAdapter](#) ensures this adapter is selected for [PhysioExperiment](#) objects (which also satisfy [SEAdapter](#)).

Value

An R6 generator for the [PhysioExperimentAdapter](#).

Super classes

OmicsLake::LakeAdapter -> OmicsLake::SEAdapter -> PhysioExperimentAdapter

Methods**Public methods:**

- PhysioExperimentAdapter\$name()
- PhysioExperimentAdapter\$can_handle()
- PhysioExperimentAdapter\$priority()
- PhysioExperimentAdapter\$put()
- PhysioExperimentAdapter\$get()
- PhysioExperimentAdapter\$clone()

Method name(): Adapter type name.

Usage:

PhysioExperimentAdapter\$name()

Method can_handle(): Whether this adapter handles data.

Usage:

PhysioExperimentAdapter\$can_handle(data)

Arguments:

data An object to test.

Method priority(): Selection priority (above the generic SummarizedExperiment adapter, which is 100).

Usage:

PhysioExperimentAdapter\$priority()

Method put(): Store a PhysioExperiment in the lake.

Usage:

PhysioExperimentAdapter\$put(lake, name, data)

Arguments:

lake An OmicsLake Lake.

name Object name.

data A PhysioExperiment.

Method get(): Retrieve a PhysioExperiment from the lake.

Usage:

PhysioExperimentAdapter\$get(lake, name, ref = "@latest")

Arguments:

lake An OmicsLake Lake.

name Object name.

ref Version reference (default "@latest").

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PhysioExperimentAdapter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

[registerPhysioAdapters\(\)](#), [OmicsLake::register_adapter\(\)](#)

Examples

```
if (requireNamespace("OmicsLake", quietly = TRUE)) {
  adapter <- PhysioExperimentAdapter$new()
  adapter$name()      # "PhysioExperiment"
  adapter$priority()  # 200, above the generic SummarizedExperiment adapter
}
```

physioProvenance	<i>Retrieve the provenance op-DAG stored for a lake object</i>
------------------	--

Description

Returns the companion provenance table written by [physioPut\(\)](#).

Usage

```
physioProvenance(lake, name, provenance_suffix = "__prov")
```

Arguments

<code>lake</code>	An OmicsLake Lake.
<code>name</code>	Object name.
<code>provenance_suffix</code>	Suffix used when storing (default "__prov").

Value

A data.frame of the operation DAG (one row per recorded activity), or NULL if none was stored.

See Also

[physioPut\(\)](#)

Examples

```
if (requireNamespace("OmicsLake", quietly = TRUE)) {
  lake <- OmicsLake::Lake$new(basename(tempfile("lake")), root = tempdir())
  pe <- PhysioExperiment::PhysioExperiment(
    assays = list(raw = matrix(as.numeric(1:20), nrow = 10, ncol = 2)),
    samplingRate = 100
  )
  physioPut(lake, "subj01", pe)
  # NULL here because this object carries no recorded op-DAG.
  physioProvenance(lake, "subj01")
}
```

physioPut

Store a Physio object in a lake with its provenance linked in the lineage

Description

Wraps [OmicsLake::Lake](#)'s `put()` so that the object's W3C-PROV operation DAG (from [PhysioExperiment::provenance\(\)](#)) is stored as a queryable companion table and recorded as a **lineage dependency** of the object (via `put(..., depends_on=)`). The ecosystem's per-object (micro) provenance thus becomes a first-class node in OmicsLake's cross-dataset (macro) lineage, visible to `lake$tree()`, `lake$deps()`, and `lake$impact()`.

Usage

```
physioPut(lake, name, x, tags = "physio", provenance_suffix = "__prov")
```

Arguments

lake	An OmicsLake Lake.
name	Object name.
x	A Physio object, e.g. a PhysioExperiment .
tags	Character tags for the object (default "physio").
provenance_suffix	Suffix for the companion provenance table (default "__prov").

Details

Objects that carry no provenance are stored normally (no companion table).

Value

Invisibly, the provenance table's name, or `NA_character_` if the object carried no provenance.

See Also

[physioProvenance\(\)](#)

Examples

```
if (requireNamespace("OmicsLake", quietly = TRUE)) {
  lake <- OmicsLake::Lake$new(basename(tempfile("lake")), root = tempdir())
  pe <- PhysioExperiment::PhysioExperiment(
    assays = list(raw = matrix(as.numeric(1:20), nrow = 10, ncol = 2)),
    samplingRate = 100
  )
  physioPut(lake, "subj01", pe)
  restored <- lake$get("subj01")      # a PhysioExperiment again
  PhysioExperiment::samplingRate(restored)
}
```

physioPutBundle	<i>Store a reproducibility-substrate / PhysioAgent run bundle in a lake</i>
-----------------	---

Description

Stores the parts of a run bundle (e.g. the frozen pre-registration, the run manifest, the operation DAG, the terminal artifact, the claims registry, the verification report) as individual OmicsLake entries under a common prefix, wired together with lineage edges so the run's internal structure is queryable via `lake$tree()` / `lake$deps()`. Combined with `lake$snap()`, a whole run becomes a versioned, lineage-tracked lake object — giving the substrate the persistent, queryable store it otherwise lacks (it content-addresses runs but does not persist them).

Usage

```
physioPutBundle(lake, name, components, edges = list(), tags = "run-bundle")
```

Arguments

lake	An OmicsLake Lake.
name	Bundle name; used as the entry prefix (" <code><name>__<key></code> ").
components	Named list of bundle parts.
edges	Named list: for each component key, a character vector of the component keys it depends on (a within-bundle lineage DAG).
tags	Character tags applied to every entry (default "run-bundle").

Details

data.frame components are stored as queryable tables; other objects are serialised. Components are stored parents-first (topological order over edges) so each `depends_on` target already exists.

Value

Invisibly, a named character vector mapping each component key to its stored lake entry name.

See Also[physioPut\(\)](#)**Examples**

```
if (requireNamespace("OmicsLake", quietly = TRUE)) {  
  lake <- OmicsLake::Lake$new(basename(tempfile("lake")), root = tempdir())  
  components <- list(  
    prereg = data.frame(key = "alpha", value = 1),  
    result = data.frame(metric = "hr", value = 72)  
  )  
  # `result` depends on `prereg`; the bundle stores that lineage edge.  
  physioPutBundle(lake, "run001", components,  
    edges = list(result = "prereg"))  
  lake$deps("run001__result")  
}
```

`registerPhysioAdapters`*Register the Physio adapters with OmicsLake*

Description

Registers PhysioLake's adapters (currently [PhysioExperimentAdapter](#)) with the OmicsLake adapter registry so that `lake$put()` / `lake$get()` handle Physio objects automatically. Called on package load; exported so it can be re-run after `OmicsLake::clear_adapters()`.

Usage

```
registerPhysioAdapters()
```

Value

Invisibly TRUE.

Examples

```
if (requireNamespace("OmicsLake", quietly = TRUE)) {  
  # Re-run after OmicsLake::clear_adapters(); also called on package load.  
  registerPhysioAdapters()  
}
```

Index

OmicsLake::Lake, [5](#)
OmicsLake::LakeAdapter, [2](#), [3](#)
OmicsLake::register_adapter(), [4](#)
OmicsLake::SEAdapter, [3](#)

PhysioExperiment, [2](#), [5](#)
PhysioExperiment::provenance(), [5](#)
PhysioExperimentAdapter, [2](#), [7](#)
physioProvenance, [4](#)
physioProvenance(), [5](#)
physioPut, [5](#)
physioPut(), [4](#), [7](#)
physioPutBundle, [6](#)

registerPhysioAdapters, [7](#)
registerPhysioAdapters(), [4](#)