

# Package: PhysioIO (via r-universe)

October 3, 2026

**Title** Input and Output Functions for 'PhysioExperiment' Objects

**Version** 0.2.10

**Description** Provides input and output capabilities for 'SummarizedExperiment'-derived physiological signal containers. Supported formats include 'EDF' and 'EDF+', 'BDF', 'GDF', 'BrainVision', 'HDF5', 'BIDS', comma-separated values, and 'MATLAB' files, together with 'DuckDB' database integration. Out-of-memory processing of large datasets is available through delayed 'HDF5' arrays.

**biocViews** Software, Infrastructure, DataImport, DataRepresentation

**Depends** R (>= 4.2), PhysioExperiment (>= 2.0.0)

**Imports** methods, SummarizedExperiment, S4Vectors, HDF5Array, rhdf5, jsonlite, DBI, stats, utils, digest

**Suggests** PhysioCore, duckdb, arrow, R.matlab, jsonvalidate, reticulate, knitr, rmarkdown, BiocStyle, testthat (>= 3.2.0)

**VignetteBuilder** knitr

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**Collate** 'PhysioIO-package.R' 'utils-provenance.R' 'io-readwrite.R' 'io-edf.R' 'io-hdf5.R' 'io-bids.R' 'io-bids-motion.R' 'io-prov.R' 'io-wfdb.R' 'io-repositories.R' 'io-bdf.R' 'io-fif.R' 'io-csv.R' 'io-clinical.R' 'io-matlab.R' 'io-brainvision.R' 'io-gdf.R' 'io-arrow.R' 'bridge-mne.R' 'db-interface.R' 'db-schema.R' 'zzz.R'

**RoxygenNote** 7.3.3

**URL** <https://github.com/x-biosignal/PhysioIO>,  
<https://x-biosignal.r-universe.dev/PhysioIO>,  
<https://x-biosignal.github.io/PhysioIO/>

**BugReports** <https://github.com/x-biosignal/PhysioIO/issues>

**Config/pak/sysreqs** libssl-dev zlib1g-dev  
**Repository** <https://x-biosignal.r-universe.dev>  
**Date/Publication** 2026-10-03 13:11:41 UTC  
**RemoteUrl** <https://github.com/x-biosignal/PhysioIO>  
**RemoteRef** HEAD  
**RemoteSha** b7e0e5b390ccbb58eb695aa7d3e4dbb38d8a4b73

## Contents

|                                      |    |
|--------------------------------------|----|
| attachBIDSPhysio . . . . .           | 3  |
| bdfTriggerEvents . . . . .           | 4  |
| connectDatabase . . . . .            | 5  |
| dbStats . . . . .                    | 6  |
| deleteExperiment . . . . .           | 7  |
| downloadFigshare . . . . .           | 8  |
| downloadFromDOI . . . . .            | 9  |
| downloadMendeley . . . . .           | 10 |
| downloadPhysioNet . . . . .          | 11 |
| downloadZenodo . . . . .             | 12 |
| figshareCollectionArticles . . . . . | 13 |
| fromMNE . . . . .                    | 14 |
| hasMNE . . . . .                     | 15 |
| initPhysioSchema . . . . .           | 15 |
| isHDF5Backed . . . . .               | 16 |
| listBIDSSessions . . . . .           | 17 |
| listBIDSSubjects . . . . .           | 18 |
| listWFDBRecords . . . . .            | 19 |
| loadExperiment . . . . .             | 19 |
| mapClinicalCodes . . . . .           | 20 |
| mendeleyDatasetFiles . . . . .       | 22 |
| pyMNEVersion . . . . .               | 23 |
| queryExperiments . . . . .           | 23 |
| readBDF . . . . .                    | 24 |
| readBDFStatus . . . . .              | 26 |
| readBIDS . . . . .                   | 26 |
| readBIDSDerivatives . . . . .        | 28 |
| readBIDSMotion . . . . .             | 28 |
| readBIDSPhysio . . . . .             | 29 |
| readBrainVision . . . . .            | 30 |
| readClinicalMetadataCSV . . . . .    | 31 |
| readCSV . . . . .                    | 32 |
| readEDF . . . . .                    | 34 |
| readElectrodePositionsCSV . . . . .  | 35 |
| readEventsCSV . . . . .              | 37 |
| readFIF . . . . .                    | 38 |
| readGDF . . . . .                    | 39 |

|                                      |           |
|--------------------------------------|-----------|
| readMAT . . . . .                    | 40        |
| readParquet . . . . .                | 41        |
| readPhysioHDF5 . . . . .             | 42        |
| readProv . . . . .                   | 43        |
| readWFDB . . . . .                   | 44        |
| readWFDBAnnotation . . . . .         | 45        |
| realizeHDF5 . . . . .                | 45        |
| registerExperiment . . . . .         | 46        |
| registerParquetAssay . . . . .       | 48        |
| toMNE . . . . .                      | 48        |
| validateBIDS . . . . .               | 49        |
| validateClinicalMetadata . . . . .   | 50        |
| writeAssayHDF5 . . . . .             | 51        |
| writeBDF . . . . .                   | 52        |
| writeBIDS . . . . .                  | 53        |
| writeBIDSDerivative . . . . .        | 54        |
| writeBIDSMotion . . . . .            | 55        |
| writeBIDSPhysio . . . . .            | 56        |
| writeBrainVision . . . . .           | 57        |
| writeCSV . . . . .                   | 58        |
| writeEDF . . . . .                   | 60        |
| writeElectrodePositionsCSV . . . . . | 61        |
| writeEventsCSV . . . . .             | 62        |
| writeGDF . . . . .                   | 63        |
| writeMAT . . . . .                   | 64        |
| writeParquet . . . . .               | 65        |
| writePhysio . . . . .                | 66        |
| writePhysioHDF5 . . . . .            | 67        |
| writeProv . . . . .                  | 68        |
| writeWFDB . . . . .                  | 69        |
| writeWFDBAnnotation . . . . .        | 70        |
| <b>Index</b>                         | <b>71</b> |

---

|                  |                                                                           |
|------------------|---------------------------------------------------------------------------|
| attachBIDSPhysio | <i>Attach a physio recording to a main recording as an aligned stream</i> |
|------------------|---------------------------------------------------------------------------|

---

## Description

Combines a main recording and a `_physio` recording into a `MultiPhysioExperiment`, using the `physio`'s `StartTime` as its clock offset. The two keep their own sampling rates.

## Usage

```
attachBIDSPhysio(main, physio)
```

**Arguments**

main            A PhysioExperiment (the primary recording).  
physio          A PhysioExperiment from [readBIDSPHYSIO](#).

**Value**

A [MultiPhysioExperiment](#).

**See Also**

[readBIDSPHYSIO](#)

---

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| bdfTriggerEvents | <i>Extract trigger events from a BioSemi Status channel</i> |
|------------------|-------------------------------------------------------------|

---

**Description**

Detects transitions of the low-16-bit trigger value to a new non-zero code and returns them as a PhysioEvents: each rising transition is an event whose onset is the transition time (seconds) and whose value is the trigger code.

**Usage**

```
bdfTriggerEvents(status, sampling_rate)
```

**Arguments**

status            Integer vector of raw Status-channel digital values (as from [readBDFStatus](#)).  
sampling\_rate    Sampling rate of the Status channel in Hz.

**Value**

A PhysioEvents object.

**See Also**

[readBDFStatus](#), [readBDF](#)

---

|                 |                                       |
|-----------------|---------------------------------------|
| connectDatabase | <i>Lightweight database interface</i> |
|-----------------|---------------------------------------|

---

## Description

These functions establish a connection to a DuckDB database using the DBI interface. They form a minimal skeleton to be expanded with concrete schema management functions in future iterations.

## Usage

```
connectDatabase(path = ":memory:")
```

```
disconnectDatabase(con)
```

## Arguments

path                    Path to a DuckDB database file.

con                    A database connection produced by connectDatabase().

## Value

A database connection object.

## References

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

## See Also

[initPhysioSchema](#), [registerExperiment](#), [queryExperiments](#)

## Examples

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  # Always disconnect when done
  disconnectDatabase(con)
  unlink(tf)
}
```

---

|         |                                |
|---------|--------------------------------|
| dbStats | <i>Get database statistics</i> |
|---------|--------------------------------|

---

**Description**

Get database statistics

**Usage**

```
dbStats(con)
```

**Arguments**

|     |                      |
|-----|----------------------|
| con | A DuckDB connection. |
|-----|----------------------|

**Value**

A list with database statistics.

**References**

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

**See Also**

[connectDatabase](#), [initPhysioSchema](#), [queryExperiments](#)

**Examples**

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  stats <- dbStats(con)
  stats$n_experiments
  disconnectDatabase(con)
  unlink(tf)
}
```

---

|                  |                                        |
|------------------|----------------------------------------|
| deleteExperiment | <i>Delete experiment from database</i> |
|------------------|----------------------------------------|

---

**Description**

Delete experiment from database

**Usage**

```
deleteExperiment(con, experiment_id, confirm = TRUE)
```

**Arguments**

|               |                                 |
|---------------|---------------------------------|
| con           | A DuckDB connection.            |
| experiment_id | The experiment identifier.      |
| confirm       | If TRUE, requires confirmation. |

**Value**

Invisible NULL.

**References**

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

**See Also**

[registerExperiment](#), [loadExperiment](#), [queryExperiments](#)

**Examples**

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256
  )
  registerExperiment(con, pe, experiment_id = "demo1", subject_id = "sub01")
  # Remove the experiment and all related records
  deleteExperiment(con, "demo1")
  disconnectDatabase(con)
  unlink(tf)
}
```

---

downloadFigshare

*Download files from a public figshare article*


---

## Description

Downloads the files attached to a public figshare *article* into *dest* via the figshare public REST API, optionally verifying each file against the MD5 checksum figshare reports. No authentication is used, so only public (published) articles are reachable.

## Usage

```
downloadFigshare(
  article_id,
  dest = ".",
  files = NULL,
  version = NULL,
  verify_md5 = TRUE,
  overwrite = FALSE,
  quiet = TRUE,
  api_url = "https://api.figshare.com/v2"
)
```

## Arguments

|                         |                                                                                                                                               |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>article_id</code> | figshare article ID (numeric or character).                                                                                                   |
| <code>dest</code>       | Destination directory (created if needed). Default ".".                                                                                       |
| <code>files</code>      | Optional character vector of file names; restricts the download to those files (default NULL = every file in the article).                    |
| <code>version</code>    | Optional integer article version; NULL (default) fetches the latest public version.                                                           |
| <code>verify_md5</code> | Logical; if TRUE (default) each file is checked against the figshare MD5 and a mismatch is an error.                                          |
| <code>overwrite</code>  | Logical; re-download files already present in <i>dest</i> (default FALSE; an existing file with a known-bad checksum is still re-downloaded). |
| <code>quiet</code>      | Logical; suppress the per-file download progress (default TRUE).                                                                              |
| <code>api_url</code>    | figshare REST API base URL.                                                                                                                   |

## Details

The `article_id` is the numeric identifier of a figshare article – the trailing number of a figshare DOI or URL (e.g. 7636887 in 10.6084/m9.figshare.7636887). For a figshare *collection* (DOI ...figshare.c.<id>), list its member articles first with [figshareCollectionArticles\(\)](#) and call this function on each.

**Value**

A data frame (one row per file) with columns name, path, size, checksum, algo ("md5") and checksum\_ok (TRUE/FALSE, or NA when no checksum was available), invisibly.

**See Also**

[downloadZenodo\(\)](#), [downloadMendeley\(\)](#), [downloadFromDOI\(\)](#), [figshareCollectionArticles\(\)](#), [downloadPhysioNet\(\)](#)

**Examples**

```
## Not run:
# Download every file of a figshare article into a temporary directory
info <- downloadFigshare(7636887, dest = tempdir())
info$path

## End(Not run)
```

---

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| downloadFromDOI | <i>Download a dataset from its DOI (figshare, Zenodo or Mendeley Data)</i> |
|-----------------|----------------------------------------------------------------------------|

---

**Description**

Convenience dispatcher: routes a figshare, Zenodo or Mendeley Data DOI (or the bare doi.org URL) to [downloadFigshare\(\)](#), [downloadZenodo\(\)](#) or [downloadMendeley\(\)](#). A figshare *collection* DOI is rejected with a pointer to [figshareCollectionArticles\(\)](#).

**Usage**

```
downloadFromDOI(doi, dest = ".", ...)
```

**Arguments**

|      |                                                                                                                                                                               |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| doi  | A DOI such as 10.6084/m9.figshare.7636887 (figshare), 10.5281/zenodo.3374790 (Zenodo) or 10.17632/mzpz22x58x.3 (Mendeley Data), optionally as a full https://doi.org/... URL. |
| dest | Destination directory (created if needed). Default ".".                                                                                                                       |
| ...  | Passed to the underlying downloader (e.g. files, verify_md5, overwrite, quiet; access_token for Mendeley).                                                                    |

**Value**

The download-info data frame from the underlying function, invisibly.

**See Also**

[downloadFigshare\(\)](#), [downloadZenodo\(\)](#), [downloadMendeley\(\)](#), [figshareCollectionArticles\(\)](#)

**Examples**

```
## Not run:
downloadFromDOI("10.5281/zenodo.3374790", dest = tempdir())

## End(Not run)
```

---

|                  |                                                    |
|------------------|----------------------------------------------------|
| downloadMendeley | <i>Download files from a Mendeley Data dataset</i> |
|------------------|----------------------------------------------------|

---

**Description**

Downloads the files of a Mendeley Data dataset into dest, verifying each against the SHA-256 hash Mendeley reports.

**Usage**

```
downloadMendeley(
  dataset_id,
  dest = ".",
  files = NULL,
  version = NULL,
  access_token = Sys.getenv("MENDELEY_TOKEN", ""),
  verify_hash = TRUE,
  overwrite = FALSE,
  quiet = TRUE,
  folder_id = "root",
  api_url = "https://data.mendeley.com/public-api"
)
```

**Arguments**

|              |                                                                                                                           |
|--------------|---------------------------------------------------------------------------------------------------------------------------|
| dataset_id   | Mendeley dataset id (or a full DOI/URL).                                                                                  |
| dest         | Destination directory (created if needed). Default ".".                                                                   |
| files        | Optional character vector of file names; restricts the download to those files (default NULL = every file in the folder). |
| version      | Optional integer dataset version; NULL (default) resolves to the latest published version.                                |
| access_token | Mendeley OAuth bearer token; defaults to the MENDELEY_TOKEN environment variable. Required to download file bytes.        |
| verify_hash  | Logical; if TRUE (default) each file is checked against the Mendeley SHA-256 and a mismatch is an error.                  |
| overwrite    | Logical; re-download files already present in dest (default FALSE).                                                       |
| quiet        | Logical; suppress the per-file download progress (default TRUE).                                                          |
| folder_id    | Folder to download from (default "root").                                                                                 |
| api_url      | Mendeley Data public API base URL.                                                                                        |

## Details

Unlike figshare and Zenodo, Mendeley Data exposes dataset and file *metadata* publicly (see [mendeleyDatasetFiles\(\)](#)) but requires an **OAuth access token** to download the file *bytes*. Supply the token via `access_token` or the `MENDELEY_TOKEN` environment variable; obtain one through the Mendeley Data API (an OAuth2 application). Without a token this function errors early and points you at [mendeleyDatasetFiles\(\)](#) for the public manifest.

The `dataset_id` is the Mendeley Data identifier – the alphanumeric part of a Mendeley DOI `10.17632/<id>.<version>` or a `data.mendeley.com/datasets/<id>/<version>` URL.

## Value

A data frame (one row per file) with columns `name`, `path`, `size`, `checksum`, `algo` ("sha256") and `checksum_ok`, invisibly.

## See Also

[mendeleyDatasetFiles\(\)](#), [downloadFigshare\(\)](#), [downloadZenodo\(\)](#), [downloadFromDOI\(\)](#)

## Examples

```
## Not run:
# metadata is public:
mendeleyDatasetFiles("mzp22x58x")
# bytes need a token:
downloadMendeley("mzp22x58x", dest = tempdir(),
                 access_token = Sys.getenv("MENDELEY_TOKEN"))

## End(Not run)
```

---

|                   |                                              |
|-------------------|----------------------------------------------|
| downloadPhysioNet | <i>Download a WFDB record from PhysioNet</i> |
|-------------------|----------------------------------------------|

---

## Description

Downloads the `.hea`, `.dat` and (optionally) annotation files of a record from a PhysioNet database into `dest`.

## Usage

```
downloadPhysioNet(
  record,
  database,
  dest = ".",
  annotator = "atr",
  base_url = "https://physionet.org/files"
)
```

**Arguments**

|           |                                                         |
|-----------|---------------------------------------------------------|
| record    | Record name (e.g. "100").                               |
| database  | PhysioNet database slug (e.g. "mitdb").                 |
| dest      | Destination directory (created if needed).              |
| annotator | Optional annotation extension(s) to fetch (e.g. "atr"). |
| base_url  | PhysioNet base URL.                                     |

**Value**

The paths of the downloaded files, invisibly.

---

|                |                                                   |
|----------------|---------------------------------------------------|
| downloadZenodo | <i>Download files from a public Zenodo record</i> |
|----------------|---------------------------------------------------|

---

**Description**

Downloads the files of a public Zenodo *record* into *dest* via the Zenodo public REST API, optionally verifying each file against the MD5 checksum Zenodo reports. No authentication is used, so only open/public records are reachable. Both the legacy file-array response and the newer InvenioRDM `files.entries` response are handled.

**Usage**

```
downloadZenodo(
  record_id,
  dest = ".",
  files = NULL,
  verify_md5 = TRUE,
  overwrite = FALSE,
  quiet = TRUE,
  api_url = "https://zenodo.org/api"
)
```

**Arguments**

|            |                                                                                                                                               |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| record_id  | Zenodo record ID (numeric or character).                                                                                                      |
| dest       | Destination directory (created if needed). Default ".".                                                                                       |
| files      | Optional character vector of file names; restricts the download to those files (default NULL = every file in the record).                     |
| verify_md5 | Logical; if TRUE (default) each file is checked against the Zenodo MD5 and a mismatch is an error.                                            |
| overwrite  | Logical; re-download files already present in <i>dest</i> (default FALSE; an existing file with a known-bad checksum is still re-downloaded). |
| quiet      | Logical; suppress the per-file download progress (default TRUE).                                                                              |
| api_url    | Zenodo REST API base URL.                                                                                                                     |

**Details**

The `record_id` is the numeric identifier of a Zenodo record – the trailing number of a Zenodo DOI or URL (e.g. 3374790 in 10.5281/zenodo.3374790). Note that a Zenodo *concept* DOI resolves to the latest version; pass a specific version's record ID to pin a version.

**Value**

A data frame (one row per file) with columns `name`, `path`, `size`, `checksum`, `algo` ("md5") and `checksum_ok` (TRUE/FALSE, or NA when no checksum was available), invisibly.

**See Also**

[downloadFigshare\(\)](#), [downloadMendeley\(\)](#), [downloadFromDOI\(\)](#), [downloadPhysioNet\(\)](#)

**Examples**

```
## Not run:
info <- downloadZenodo(3374790, dest = tempdir())
info$path

## End(Not run)
```

---

```
figshareCollectionArticles
```

*List the articles in a public figshare collection*

---

**Description**

Returns the articles that make up a public figshare *collection* (DOI 10.6084/m9.figshare.c.<id>), so each can be fetched with [downloadFigshare\(\)](#). Collections (e.g. GaitRec, c.4788012) bundle several articles; figshare has no single-call "download the whole collection", so the idiom is list-then-download-per-article.

**Usage**

```
figshareCollectionArticles(
  collection_id,
  api_url = "https://api.figshare.com/v2"
)
```

**Arguments**

|                            |                                                                                             |
|----------------------------|---------------------------------------------------------------------------------------------|
| <code>collection_id</code> | figshare collection ID (numeric or character); a full collection DOI/URL is also tolerated. |
| <code>api_url</code>       | figshare REST API base URL.                                                                 |

**Value**

A data frame with columns `article_id`, `title` and `doi`, one row per article (up to the API page-size cap of 1000).

**See Also**

[downloadFigshare\(\)](#)

**Examples**

```
## Not run:
arts <- figshareCollectionArticles(4788012) # GaitRec
downloadFigshare(arts$article_id[1], dest = tempdir())

## End(Not run)
```

---

fromMNE

*Convert an MNE-Python Raw object to a PhysioExperiment*

---

**Description**

Inverse of [toMNE\(\)](#): reads the signal matrix, channel names/types, sampling rate, electrode montage and annotations from an MNE Raw object into a PhysioExperiment. The import is recorded in the object's provenance.

**Usage**

```
fromMNE(raw)
```

**Arguments**

`raw` An MNE Raw object (e.g. from [toMNE\(\)](#) or `mne.io.read_raw_*`).

**Value**

A PhysioExperiment with the signal in the "raw" assay.

**See Also**

[toMNE\(\)](#), [hasMNE\(\)](#)

Examples

```
if (hasMNE()) {
  pe <- PhysioExperiment(
    assays = S4Vectors::SimpleList(raw = matrix(rnorm(300), 100, 3)),
    coldData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz"),
                                     type = rep("eeg", 3)),
    samplingRate = 100)
  pe2 <- fromMNE(toMNE(pe))
}
```

---

|        |                                                   |
|--------|---------------------------------------------------|
| hasMNE | <i>Is the Python <b>mne</b> module available?</i> |
|--------|---------------------------------------------------|

---

Description

Is the Python **mne** module available?

Usage

```
hasMNE()
```

Value

TRUE if **reticulate** and the Python mne module are both available, otherwise FALSE.

See Also

```
pyMNEVersion\(\), toMNE\(\), fromMNE\(\)
```

Examples

```
hasMNE()
```

---

|                  |                                                      |
|------------------|------------------------------------------------------|
| initPhysioSchema | <i>DuckDB Schema Management for PhysioExperiment</i> |
|------------------|------------------------------------------------------|

---

Description

Functions for managing experiment data in DuckDB database. Provides a relational schema for storing metadata, signals, and events. Initialize PhysioExperiment database schema

Usage

```
initPhysioSchema(con)
```

**Arguments**

con                      A DuckDB connection from connectDatabase().

**Details**

Creates the database tables for storing experiment metadata, signals, and events.

**Value**

Invisible NULL.

**References**

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

**See Also**

[connectDatabase](#), [registerExperiment](#), [dbStats](#)

**Examples**

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  # Freshly created schema has no experiments yet
  dbStats(con)$n_experiments
  disconnectDatabase(con)
  unlink(tf)
}
```

---

isHDF5Backed

---

*Check if assays are HDF5-backed*


---

**Description**

Tests whether the default assay of a PhysioExperiment is stored as an HDF5-backed DelayedArray.

**Usage**

```
isHDF5Backed(x)
```

**Arguments**

x                      A PhysioExperiment object.

**Value**

Logical scalar; TRUE if the default assay inherits from HDF5Array or DelayedArray, FALSE otherwise.

**References**

The HDF Group (1997-2024). "Hierarchical Data Format, version 5." <https://www.hdfgroup.org/HDF5/>

**See Also**

[readPhysioHDF5](#), [writePhysioHDF5](#), [realizeHDF5](#)

**Examples**

```
# Regular in-memory PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(1:100, nrow = 10)),
  samplingRate = 100
)
isHDF5Backed(pe) # FALSE
```

---

|                   |                                                    |
|-------------------|----------------------------------------------------|
| listBIDS Sessions | <i>List sessions for a subject in BIDS dataset</i> |
|-------------------|----------------------------------------------------|

---

**Description**

List sessions for a subject in BIDS dataset

**Usage**

```
listBIDS Sessions(bids_root, subject)
```

**Arguments**

|           |                                             |
|-----------|---------------------------------------------|
| bids_root | Path to the BIDS dataset root.              |
| subject   | Subject identifier (without 'sub-' prefix). |

**Value**

Character vector of session IDs (without 'ses-' prefix).

**References**

Gorgolewski KJ, et al. (2016). "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments." Scientific Data, 3, 160044. doi:10.1038/sdata.2016.44

**See Also**

[listBIDSSubjects](#), [readBIDS](#), [validateBIDS](#)

**Examples**

```
# Write a BIDS dataset with a session, then list sessions for the subject
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
bids_root <- file.path(tempdir(), "bids-session-demo")
writeBIDS(pe, bids_root, subject = "01", session = "baseline", task = "rest")

listBIDSSessions(bids_root, subject = "01")
unlink(bids_root, recursive = TRUE)
```

---

|                  |                                        |
|------------------|----------------------------------------|
| listBIDSSubjects | <i>List subjects in a BIDS dataset</i> |
|------------------|----------------------------------------|

---

**Description**

List subjects in a BIDS dataset

**Usage**

```
listBIDSSubjects(bids_root)
```

**Arguments**

bids\_root      Path to the BIDS dataset root.

**Value**

Character vector of subject IDs (without 'sub-' prefix).

**References**

Gorgolewski KJ, et al. (2016). "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments." Scientific Data, 3, 160044. doi:10.1038/sdata.2016.44

**See Also**

[listBIDSSessions](#), [readBIDS](#), [validateBIDS](#)

**Examples**

```
# Write a minimal BIDS dataset, then list its subjects
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
bids_root <- file.path(tempdir(), "bids-list-demo")
writeBIDS(pe, bids_root, subject = "01", task = "rest")

listBIDSSubjects(bids_root)
unlink(bids_root, recursive = TRUE)
```

---

|                 |                                                |
|-----------------|------------------------------------------------|
| listWFDBRecords | <i>List the records in a WFDB RECORDS file</i> |
|-----------------|------------------------------------------------|

---

**Description**

List the records in a WFDB RECORDS file

**Usage**

```
listWFDBRecords(path)
```

**Arguments**

path                      Path (or URL) to a PhysioNet RECORDS file: one record name per line.

**Value**

Character vector of record names.

---

|                |                                      |
|----------------|--------------------------------------|
| loadExperiment | <i>Load experiment from database</i> |
|----------------|--------------------------------------|

---

**Description**

Reconstructs a PhysioExperiment object from database records.

**Usage**

```
loadExperiment(con, experiment_id, load_signals = FALSE)
```

**Arguments**

con                    A DuckDB connection.  
 experiment\_id        The experiment identifier.  
 load\_signals         If TRUE, loads signal data from chunks.

**Value**

A PhysioExperiment object.

**References**

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

**See Also**

[registerExperiment](#), [queryExperiments](#), [deleteExperiment](#)

**Examples**

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256
  )
  registerExperiment(con, pe, experiment_id = "demo1",
    subject_id = "sub01", task = "rest", store_signals = TRUE)
  # Reconstruct the object, with signal data
  pe2 <- loadExperiment(con, "demo1", load_signals = TRUE)
  disconnectDatabase(con)
  unlink(tf)
}
```

**Description**

Harmonize clinical codes across sites

**Usage**

```
mapClinicalCodes(
  x,
  mapping,
  code_col = "scale_name",
  mapped_col = "scale_name_std",
  from_col = "from",
  to_col = "to",
  unmatched = c("keep", "na", "drop")
)
```

**Arguments**

|            |                                                                                                                                                                                                                       |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x          | Clinical metadata data frame.                                                                                                                                                                                         |
| mapping    | Code mapping, either: <ol style="list-style-type: none"> <li>1. named character vector (names = source code, values = target code), or</li> <li>2. data frame with columns defined by from_col and to_col.</li> </ol> |
| code_col   | Source code column in x.                                                                                                                                                                                              |
| mapped_col | Output mapped code column.                                                                                                                                                                                            |
| from_col   | Source column in mapping data.frame.                                                                                                                                                                                  |
| to_col     | Target column in mapping data.frame.                                                                                                                                                                                  |
| unmatched  | Behavior for unmatched codes: "keep", "na", or "drop".                                                                                                                                                                |

**Value**

Data frame with mapped\_col appended.

**References**

Goldberger AL, et al. (2000). "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals." *Circulation*, 101(23), e215-e220. doi:10.1161/01.CIR.101.23.e215

**See Also**

[readClinicalMetadataCSV](#), [validateClinicalMetadata](#)

**Examples**

```
df <- data.frame(scale_name = c("fim_total", "Berg"), stringsAsFactors = FALSE)
map <- c(fim_total = "FIM", Berg = "BBS")
mapClinicalCodes(df, map, code_col = "scale_name")
```

---

mendeleyDatasetFiles    *List the files of a public Mendeley Data dataset*


---

### Description

Returns the public file manifest of a Mendeley Data dataset – file names, sizes, SHA-256 hashes and content ids – via the Mendeley Data public REST API. This metadata is available **without authentication**; downloading the file *bytes*, however, requires an OAuth token (see [downloadMendeley\(\)](#)).

### Usage

```
mendeleyDatasetFiles(
  dataset_id,
  version = NULL,
  folder_id = "root",
  api_url = "https://data.mendeley.com/public-api"
)
```

### Arguments

|            |                                                                                                                      |
|------------|----------------------------------------------------------------------------------------------------------------------|
| dataset_id | Mendeley dataset id (or a full DOI/URL).                                                                             |
| version    | Optional integer dataset version; NULL (default) resolves to the latest published version.                           |
| folder_id  | Folder to list (default "root"; Mendeley organises files into folders and only one folder level is listed per call). |
| api_url    | Mendeley Data public API base URL.                                                                                   |

### Details

The dataset\_id is the Mendeley Data dataset identifier – the alphanumeric component of a Mendeley DOI 10.17632/<id>.<version> or a data.mendeley.com/datasets/<id>/<version> URL (both are tolerated).

### Value

A data frame with columns name, size, sha256, content\_id and version, one row per file.

### See Also

[downloadMendeley\(\)](#), [downloadFromDOI\(\)](#)

### Examples

```
## Not run:
files <- mendeleyDatasetFiles("mzp22x58x") # latest version
files[, c("name", "size", "sha256")]

## End(Not run)
```

---

|              |                                                          |
|--------------|----------------------------------------------------------|
| pyMNEVersion | <i>Version of the available Python <b>mne</b> module</i> |
|--------------|----------------------------------------------------------|

---

**Description**

Version of the available Python **mne** module

**Usage**

```
pyMNEVersion()
```

**Value**

The `mne.__version__` string, or `NA_character_` when **mne** is not available.

**See Also**

[hasMNE\(\)](#)

**Examples**

```
pyMNEVersion()
```

---

|                  |                                        |
|------------------|----------------------------------------|
| queryExperiments | <i>Query experiments from database</i> |
|------------------|----------------------------------------|

---

**Description**

Query experiments from database

**Usage**

```
queryExperiments(con, subject_id = NULL, task = NULL, date_range = NULL)
```

**Arguments**

|                         |                                          |
|-------------------------|------------------------------------------|
| <code>con</code>        | A DuckDB connection.                     |
| <code>subject_id</code> | Optional filter by subject.              |
| <code>task</code>       | Optional filter by task.                 |
| <code>date_range</code> | Optional date range (vector of 2 dates). |

**Value**

A `data.frame` of matching experiments.

## References

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

## See Also

[registerExperiment](#), [loadExperiment](#), [dbStats](#)

## Examples

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256
  )
  registerExperiment(con, pe, experiment_id = "demo1",
    subject_id = "sub01", task = "rest")
  # Query all experiments, or filter by subject / task
  all_exps <- queryExperiments(con)
  sub01_exps <- queryExperiments(con, subject_id = "sub01")
  disconnectDatabase(con)
  unlink(tf)
}
```

---

readBDF

*Read BDF (BioSemi Data Format) file*

---

## Description

Reads a BDF file and returns a PhysioExperiment object. BDF is a 24-bit extension of the EDF format used by BioSemi systems.

## Usage

```
readBDF(
  path,
  channels = NULL,
  start_time = NULL,
  end_time = NULL,
  resample = FALSE,
  status = TRUE
)
```

**Arguments**

|                         |                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>path</code>       | Path to the BDF file.                                                                                                                                                                                                                                                                                                                                                   |
| <code>channels</code>   | Optional character vector of channel names to load. If <code>NULL</code> , all channels are loaded.                                                                                                                                                                                                                                                                     |
| <code>start_time</code> | Optional start time in seconds for reading a subset. Note that BDF subsetting is record-granular: <code>start_time/end_time</code> are rounded outward to whole data records (typically 1 s), so the returned signal and any extracted trigger onsets share that record-aligned window (onsets are relative to it and may extend slightly past <code>end_time</code> ). |
| <code>end_time</code>   | Optional end time in seconds for reading a subset.                                                                                                                                                                                                                                                                                                                      |
| <code>resample</code>   | Logical. If <code>FALSE</code> (default) and channels have differing native rates, a <code>MultiPhysioExperiment</code> is returned; if <code>TRUE</code> , all channels are resampled to the highest rate (legacy).                                                                                                                                                    |
| <code>status</code>     | Logical. If <code>TRUE</code> (default) and a BioSemi "Status" channel is present, stimulus triggers are extracted from it (see <a href="#">bdfTriggerEvents</a> ) and attached to the result as events.                                                                                                                                                                |

**Details**

BDF (BioSemi Data Format) is a 24-bit variant of EDF used by BioSemi acquisition systems. The main differences from EDF are:

- 24-bit data resolution (vs 16-bit in EDF)
- Header starts with 0xFF followed by "BIOSEMI"
- Digital range is -8388608 to 8388607

**Value**

A `PhysioExperiment`, or a `MultiPhysioExperiment` when native rates differ and `resample = FALSE`.

**Examples**

```
# Round-trip a small recording through a temporary BDF file
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("A1", "A2", "B1", "B2")),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".bdf")
writeBDF(pe, tmp)

# Read the file back
pe_in <- readBDF(tmp)

# Read only specific channels
pe_sub <- readBDF(tmp, channels = c("A1", "B1"))

unlink(tmp)
```

---

readBDFStatus

*Read the BioSemi Status channel of a BDF file*


---

### Description

Returns the raw digital values of the "Status" channel. The low 16 bits of each value are the stimulus trigger inputs.

### Usage

```
readBDFStatus(path)
```

### Arguments

path                      Path to a BDF file.

### Value

A list with values (integer status words) and fs (the Status channel sampling rate), or NULL if the file has no Status channel.

### See Also

[bdfTriggerEvents](#), [readBDF](#)

---

readBIDS

*BIDS Format Support for PhysioExperiment*


---

### Description

Functions for reading and writing data in BIDS (Brain Imaging Data Structure) format. Supports BIDS-EEG and BIDS-iEEG specifications. Read PhysioExperiment from BIDS dataset

### Usage

```
readBIDS(
  bids_root,
  subject,
  session = NULL,
  task,
  run = NULL,
  modality = c("eeg", "ieeg"),
  load_events = TRUE
)
```

**Arguments**

|                          |                                                       |
|--------------------------|-------------------------------------------------------|
| <code>bids_root</code>   | Path to the BIDS dataset root directory.              |
| <code>subject</code>     | Subject identifier (without 'sub-' prefix).           |
| <code>session</code>     | Session identifier (without 'ses-' prefix). Optional. |
| <code>task</code>        | Task name.                                            |
| <code>run</code>         | Run number. Optional.                                 |
| <code>modality</code>    | Data modality: "eeg" or "ieeg".                       |
| <code>load_events</code> | If TRUE, loads events from the events.tsv file.       |

**Details**

Reads EEG/IEEG data from a BIDS-compliant directory structure.

**Value**

A `PhysioExperiment` object.

**References**

Gorgolewski KJ, et al. (2016). "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments." *Scientific Data*, 3, 160044. doi:10.1038/sdata.2016.44

**See Also**

[writeBIDS](#), [validateBIDS](#), [readEDF](#), [listBIDSSubjects](#)

**Examples**

```
# Write a minimal BIDS dataset to a temporary directory, then read it back
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
bids_root <- file.path(tempdir(), "bids-demo")
writeBIDS(pe, bids_root, subject = "01", task = "rest")

# Read EEG data back from the BIDS dataset
pe_in <- readBIDS(bids_root, subject = "01", task = "rest")
dim(SummarizedExperiment::assay(pe_in, "raw"))

unlink(bids_root, recursive = TRUE)
```

---

|                     |                                                    |
|---------------------|----------------------------------------------------|
| readBIDSDerivatives | <i>Read the derivative recordings of a subject</i> |
|---------------------|----------------------------------------------------|

---

**Description**

Read the derivative recordings of a subject

**Usage**

```
readBIDSDerivatives(deriv_root, subject, session = NULL)
```

**Arguments**

|            |                          |
|------------|--------------------------|
| deriv_root | Derivative dataset root. |
| subject    | Subject label.           |
| session    | Optional session label.  |

**Value**

A named list of PhysioExperiment objects, one per desc-<label> derivative found for the subject.

**See Also**

[writeBIDSDerivative](#)

---

|                |                                     |
|----------------|-------------------------------------|
| readBIDSMotion | <i>Read a Motion-BIDS recording</i> |
|----------------|-------------------------------------|

---

**Description**

Read a Motion-BIDS recording

**Usage**

```
readBIDSMotion(  
  bids_root,  
  subject,  
  task,  
  session = NULL,  
  run = NULL,  
  tracking_system = NULL  
)
```

**Arguments**

|                 |                                                 |
|-----------------|-------------------------------------------------|
| bids_root       | BIDS dataset root.                              |
| subject, task   | Subject and task labels.                        |
| session, run    | Optional session label and run number.          |
| tracking_system | Optional tracking-system label to disambiguate. |

**Value**

A `PhysioExperiment` with the motion data, `colData` carrying `tracked_point/component/units`, and the sampling rate.

**See Also**

[writeBIDSMotion](#)

---

|                |                                                       |
|----------------|-------------------------------------------------------|
| readBIDSPhysio | <i>Read a BIDS continuous physiological recording</i> |
|----------------|-------------------------------------------------------|

---

**Description**

Read a BIDS continuous physiological recording

**Usage**

```
readBIDSPhysio(path)
```

**Arguments**

|      |                                                                                                         |
|------|---------------------------------------------------------------------------------------------------------|
| path | Path to the <code>_physio.tsv.gz</code> file (its <code>_physio.json</code> sidecar is read alongside). |
|------|---------------------------------------------------------------------------------------------------------|

**Value**

A `PhysioExperiment` with the physio signals; `StartTime` is stored in `metadata()`\$bids\_start\_time.

**See Also**

[writeBIDSPhysio](#), [attachBIDSPhysio](#)

---

readBrainVision

*BrainVision File Format I/O for PhysioExperiment*


---

## Description

Functions for reading and writing BrainVision format files (.vhdr/.vmrk/.eeg). BrainVision is one of the official BIDS-EEG formats and is widely supported by EEG analysis software (MNE-Python, EEGLAB, FieldTrip, etc.). Read PhysioExperiment from BrainVision files

## Usage

```
readBrainVision(path, channels = NULL)
```

## Arguments

|          |                                                                                  |
|----------|----------------------------------------------------------------------------------|
| path     | Path to the .vhdr header file.                                                   |
| channels | Optional character vector of channel names to load. If NULL, loads all channels. |

## Details

Reads EEG data from BrainVision format files. The format consists of three files: a header file (.vhdr), a marker file (.vmrk), and a binary data file (.eeg).

## Value

A PhysioExperiment object.

## Examples

```
# Round-trip a small recording through temporary BrainVision files
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
base <- tempfile()
writeBrainVision(pe, base)

pe_in <- readBrainVision(paste0(base, ".vhdr"))
dim(SummarizedExperiment::assay(pe_in, "raw"))

unlink(paste0(base, c(".vhdr", ".vmrk", ".eeg")))
```

---

readClinicalMetadataCSV

*Clinical metadata CSV I/O and validation*


---

## Description

Utilities for loading and validating clinical assessment metadata so physiological sessions can be linked with EDC/EHR variables using `subject_id` and `visit_id`. Read clinical metadata from CSV

## Usage

```
readClinicalMetadataCSV(
  path,
  col_map = NULL,
  required_cols = c("subject_id", "visit_id", "scale_name", "scale_score"),
  date_cols = c("assessment_date", "visit_date"),
  validate = TRUE,
  sep = ",",
  header = TRUE,
  ...
)
```

## Arguments

|                            |                                                                                                                    |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>path</code>          | Path to the CSV/TSV file.                                                                                          |
| <code>col_map</code>       | Optional named character vector for renaming columns. Names are source columns and values are target column names. |
| <code>required_cols</code> | Required columns for validation.                                                                                   |
| <code>date_cols</code>     | Columns to parse as Date (YYYY-MM-DD) when present.                                                                |
| <code>validate</code>      | Logical; run <code>validateClinicalMetadata()</code> when TRUE.                                                    |
| <code>sep</code>           | Field separator, default <code>" , "</code> .                                                                      |
| <code>header</code>        | Logical, default TRUE.                                                                                             |
| <code>...</code>           | Additional arguments passed to <code>utils::read.csv()</code> .                                                    |

## Value

Data frame containing standardized clinical metadata.

## References

Goldberger AL, et al. (2000). "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals." *Circulation*, 101(23), e215-e220. doi:10.1161/01.CIR.101.23.e215

**See Also**

[validateClinicalMetadata](#), [mapClinicalCodes](#), [readCSV](#)

**Examples**

```
tmp <- tempfile(fileext = ".csv")
write.csv(data.frame(
  sid = "S01",
  vid = "V01",
  scale_name = "FIM",
  scale_score = 90,
  assessment_date = "2026-01-10"
), tmp, row.names = FALSE)

df <- readClinicalMetadataCSV(
  tmp,
  col_map = c(sid = "subject_id", vid = "visit_id")
)
unlink(tmp)
```

---

readCSV

---

*CSV/TSV I/O for PhysioExperiment*


---

**Description**

Functions for reading and writing signal data in CSV/TSV format. Supports both wide format (time x channels) and long format. Read PhysioExperiment from CSV file

**Usage**

```
readCSV(
  path,
  format = c("wide", "long"),
  time_col = NULL,
  channel_cols = NULL,
  sampling_rate = NULL,
  sep = ",",
  header = TRUE,
  ...
)
```

**Arguments**

|          |                                                                                              |
|----------|----------------------------------------------------------------------------------------------|
| path     | Path to the CSV file.                                                                        |
| format   | Data format: "wide" (time x channels) or "long" (stacked).                                   |
| time_col | Name of the time column. If NULL, assumes first column or generates time from sampling rate. |

|               |                                                                                               |
|---------------|-----------------------------------------------------------------------------------------------|
| channel_cols  | For wide format, column names/indices to use as channels. If NULL, uses all non-time columns. |
| sampling_rate | Sampling rate in Hz. Required if time column is not present.                                  |
| sep           | Column separator. Default is ",".                                                             |
| header        | Logical. If TRUE, first row contains column names.                                            |
| ...           | Additional arguments passed to read.csv/read.table.                                           |

## Details

Reads physiological signal data from a CSV file.

## Value

A `PhysioExperiment` object.

## References

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, 59(10), 1-23. doi:10.18637/jss.v059.i10

## See Also

[writeCSV](#), [readEventsCSV](#), [readEDF](#), [readMAT](#)

## Examples

```
# Write a wide-format CSV, then read it back
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".csv")
writeCSV(pe, tmp)

# Read using the written time column
pe_in <- readCSV(tmp, time_col = "time")

# Read without a time column (generate time from sampling rate)
pe_sr <- readCSV(tmp, channel_cols = c("Fz", "Cz", "Pz", "Oz"),
  sampling_rate = 100)

unlink(tmp)
```

readEDF

*EDF/EDF+ file I/O***Description**

Functions for reading European Data Format (EDF/EDF+) files commonly used for EEG, PSG, and other physiological recordings. Read EDF/EDF+ file

**Usage**

```
readEDF(
  path,
  channels = NULL,
  start_time = NULL,
  end_time = NULL,
  resample = FALSE
)
```

**Arguments**

|            |                                                                                                                                                                                                                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path       | Path to the EDF file.                                                                                                                                                                                                                                                                           |
| channels   | Optional character vector of channel names to load. If NULL, all channels are loaded.                                                                                                                                                                                                           |
| start_time | Optional start time in seconds for reading a subset.                                                                                                                                                                                                                                            |
| end_time   | Optional end time in seconds for reading a subset.                                                                                                                                                                                                                                              |
| resample   | Logical. If FALSE (default) and channels have differing native sampling rates, the native rates are preserved by returning a MultiPhysioExperiment (one stream per rate). If TRUE, all channels are resampled to the highest rate and a single PhysioExperiment is returned (legacy behaviour). |

**Details**

Reads an EDF or EDF+ file and returns a PhysioExperiment object.

EDF (European Data Format) is a standard file format for storing multichannel physiological signals. EDF+ extends this with annotations and discontinuous recordings.

The function parses the EDF header to extract:

- Channel labels and types
- Sampling rates (may differ per channel)
- Physical dimensions (units)
- Recording start date/time

If channels have different native sampling rates, the rates are preserved by returning a MultiPhysioExperiment (one stream per rate); pass `resample = TRUE` for the legacy single-rate behaviour.

**Value**

A [PhysioExperiment](#) with the EDF signal data in the "raw" assay, or - when channels have differing native rates and `resample = FALSE` - a [MultiPhysioExperiment](#). Channel metadata (label, transducer, physical dimensions, digital/physical min/max) are stored in `colData`, and recording metadata in `metadata`.

**References**

Kemp, B., et al. (1992). "A simple format for exchange of digitized polygraphic recordings." *Electroencephalography and Clinical Neurophysiology*, 82(5), 391-393. doi:10.1016/0013-4694(92)90009-7

**See Also**

[writeEDF](#), [readBIDS](#), [readPhysioHDF5](#), [readCSV](#)

**Examples**

```
# Round-trip a small recording through a temporary EDF file
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "C3", "C4")),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".edf")
writeEDF(pe, tmp)

# Read the whole file back
pe_in <- readEDF(tmp)
dim(SummarizedExperiment::assay(pe_in, "raw"))

# Read only specific channels
pe_sub <- readEDF(tmp, channels = c("Fp1", "C3"))

unlink(tmp)
```

---

readElectrodePositionsCSV

*Read electrode positions from CSV*

---

**Description**

Reads electrode positions from a CSV file with x, y, z coordinates.

**Usage**

```
readElectrodePositionsCSV(
  path,
  name_col = "name",
  x_col = "x",
  y_col = "y",
  z_col = "z",
  sep = ",",
  ...
)
```

**Arguments**

|          |                                          |
|----------|------------------------------------------|
| path     | Path to the CSV file.                    |
| name_col | Name of the electrode name column.       |
| x_col    | Name of the x coordinate column.         |
| y_col    | Name of the y coordinate column.         |
| z_col    | Name of the z coordinate column.         |
| sep      | Column separator.                        |
| ...      | Additional arguments passed to read.csv. |

**Value**

A data.frame with electrode positions.

**References**

Wickham H (2014). "Tidy Data." Journal of Statistical Software, 59(10), 1-23. doi:10.18637/jss.v059.i10

**See Also**

[writeElectrodePositionsCSV](#), [readCSV](#), [readBIDS](#)

**Examples**

```
# Write electrode positions to a temporary CSV, then read them back
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
pe <- applyMontage(pe, "10-20")
tmp <- tempfile(fileext = ".csv")
writeElectrodePositionsCSV(pe, tmp)

positions <- readElectrodePositionsCSV(tmp)
unlink(tmp)
```

---

|               |                                      |
|---------------|--------------------------------------|
| readEventsCSV | <i>Read events from CSV/TSV file</i> |
|---------------|--------------------------------------|

---

## Description

Reads event markers from a CSV file with onset, duration, and type columns.

## Usage

```
readEventsCSV(  
  path,  
  onset_col = "onset",  
  duration_col = "duration",  
  type_col = "type",  
  value_col = "value",  
  sep = ",",  
  ...  
)
```

## Arguments

|              |                                          |
|--------------|------------------------------------------|
| path         | Path to the CSV file.                    |
| onset_col    | Name of the onset column (in seconds).   |
| duration_col | Name of the duration column.             |
| type_col     | Name of the event type column.           |
| value_col    | Name of the event value column.          |
| sep          | Column separator.                        |
| ...          | Additional arguments passed to read.csv. |

## Value

A PhysioEvents object.

## References

Wickham H (2014). "Tidy Data." Journal of Statistical Software, 59(10), 1-23. doi:10.18637/jss.v059.i10

## See Also

[writeEventsCSV](#), [readCSV](#), [readBIDS](#)

## Examples

```
# Write events to a temporary CSV, then read them back
events <- PhysioEvents(
  onset = c(1.0, 2.5, 4.0),
  duration = c(0.5, 0.5, 0.5),
  type = c("stimulus", "stimulus", "response"),
  value = c("A", "B", "correct")
)
tmp <- tempfile(fileext = ".csv")
writeEventsCSV(events, tmp)

events_in <- readEventsCSV(tmp)
unlink(tmp)
```

---

|         |                                                    |
|---------|----------------------------------------------------|
| readFIF | <i>Read an Elekta/Neuromag/MNE FIF (FIFF) file</i> |
|---------|----------------------------------------------------|

---

## Description

Reads a raw FIFF file into a PhysioExperiment. The native parser reads the FIFF tag stream directly (channel info, sampling rate and data buffers). For files the native path cannot handle, and when the Python **mne** module is available through **reticulate**, it can fall back to `mne.io.read_raw_fif()`.

## Usage

```
readFIF(path, backend = c("auto", "native", "mne"))
```

## Arguments

|         |                                                                                                                |
|---------|----------------------------------------------------------------------------------------------------------------|
| path    | Path to a .fif file.                                                                                           |
| backend | One of "auto" (native, then mne on failure), "native" (native only), or "mne" (force the reticulate/mne path). |

## Value

A PhysioExperiment with calibrated signal values in the "raw" assay, channel labels/types/units in colData, and the sampling rate set.

## References

FIFF file format (Elekta Neuromag); MNE-Python `read_raw_fif`.

## See Also

[readEDF](#), [readBDF](#)

readGDF

*GDF (General Data Format) File I/O for PhysioExperiment***Description**

Functions for reading and writing GDF format files. GDF is designed to overcome limitations of EDF and is commonly used in BCI research. Supports GDF version 1.x and 2.x. Read PhysioExperiment from GDF file

**Usage**

```
readGDF(path, channels = NULL, start_time = NULL, end_time = NULL)
```

**Arguments**

|            |                                                                                                                       |
|------------|-----------------------------------------------------------------------------------------------------------------------|
| path       | Path to the GDF file.                                                                                                 |
| channels   | Optional integer vector of channel indices or character vector of channel names to load. If NULL, loads all channels. |
| start_time | Optional start time in seconds for selective loading.                                                                 |
| end_time   | Optional end time in seconds for selective loading.                                                                   |

**Details**

Reads physiological signal data from a GDF (General Data Format) file. GDF is an extension of EDF with improved features including better event support, more data types, and subject information.

**Value**

A PhysioExperiment object.

**Examples**

```
# Round-trip a small recording through a temporary GDF file
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(300), nrow = 100, ncol = 3)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz")),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".gdf")
writeGDF(pe, tmp)

pe_in <- readGDF(tmp)
pe_sub <- readGDF(tmp, channels = c("Fz", "Cz"))
unlink(tmp)
```

---

|         |                                                    |
|---------|----------------------------------------------------|
| readMAT | <i>Read PhysioExperiment from MATLAB .mat file</i> |
|---------|----------------------------------------------------|

---

### Description

Reads physiological signal data from a MATLAB .mat file. Supports both standard .mat files and EEGLAB .set structures.

### Usage

```
readMAT(
    path,
    data_var = NULL,
    sr_var = NULL,
    channel_var = NULL,
    event_var = NULL,
    transpose = FALSE
)
```

### Arguments

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| path        | Path to the .mat file.                                                         |
| data_var    | Name of the variable containing signal data. If NULL, attempts to auto-detect. |
| sr_var      | Name of the variable containing sampling rate.                                 |
| channel_var | Name of the variable containing channel labels.                                |
| event_var   | Name of the variable containing events.                                        |
| transpose   | Logical. If TRUE, transposes the data matrix.                                  |

### Details

The function attempts to auto-detect the data structure if variable names are not specified. It looks for common variable names used in EEG toolboxes:

- data, EEG.data, signal, X for signal data
- srate, fs, Fs, samplingRate for sampling rate
- chanlocs, channels, labels for channel information
- event, events, EEG.event for events

### Value

A PhysioExperiment object.

### References

MathWorks (2024). "MAT-File Format." Technical documentation. [https://www.mathworks.com/help/matlab/import\\_export/mat-file-versions.html](https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html)

**See Also**

[writeMAT](#), [readEDF](#), [readCSV](#), [readPhysioHDF5](#)

**Examples**

```
# 'R.matlab' is an optional dependency; guard the round-trip example so it
# only runs when the package is installed.
if (requireNamespace("R.matlab", quietly = TRUE)) {
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256
  )
  tf <- tempfile(fileext = ".mat")
  writeMAT(pe, tf)
  pe2 <- readMAT(tf)
  unlink(tf)
}
```

---

readParquet

---

*Read a PhysioExperiment from a Parquet dataset directory*


---

**Description**

Inverse of [writeParquet\(\)](#): reconstructs a PhysioExperiment (or MultiPhysioExperiment) from a directory written by [writeParquet\(\)](#).

**Usage**

```
readParquet(dir)
```

**Arguments**

dir                    A directory written by [writeParquet\(\)](#).

**Value**

A PhysioExperiment or MultiPhysioExperiment.

**See Also**

[writeParquet\(\)](#)

## Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
  pe <- PhysioExperiment(
    assays = S4Vectors::SimpleList(raw = matrix(rnorm(30), 10, 3)),
    samplingRate = 256)
  d <- tempfile("pe-parquet")
  writeParquet(pe, d)
  readParquet(d)
}
```

---

|                |                                        |
|----------------|----------------------------------------|
| readPhysioHDF5 | <i>Read PhysioExperiment from HDF5</i> |
|----------------|----------------------------------------|

---

## Description

Reads a `PhysioExperiment` object from HDF5 format. By default, returns `DelayedArray`-backed assays for out-of-memory processing.

## Usage

```
readPhysioHDF5(path, as_delayed = TRUE)
```

## Arguments

|                         |                                                                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>path</code>       | Path to the HDF5 file.                                                                                                                     |
| <code>as_delayed</code> | Logical. If <code>TRUE</code> (default), returns <code>DelayedArray</code> -backed assays. If <code>FALSE</code> , loads data into memory. |

## Value

A `PhysioExperiment` object. When `as_delayed = TRUE`, assays are `HDF5Array` objects enabling out-of-memory access; when `FALSE`, assays are standard in-memory arrays.

## References

The HDF Group (1997-2024). "Hierarchical Data Format, version 5." <https://www.hdfgroup.org/HDF5/>

## See Also

`writePhysioHDF5`, `isHDF5Backed`, `realizeHDF5`, `writeAssayHDF5`

## Examples

```
# Write a small PhysioExperiment to a temporary HDF5 file
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".h5")
writePhysioHDF5(pe, tmp)

# Read it back with the DelayedArray backend (out-of-memory)
pe_delayed <- readPhysioHDF5(tmp)
isHDF5Backed(pe_delayed) # TRUE

# Read it back into memory
pe_mem <- readPhysioHDF5(tmp, as_delayed = FALSE)
isHDF5Backed(pe_mem) # FALSE

unlink(tmp)
```

---

readProv

Read a W3C PROV interchange file back into a provenance table

---

## Description

Inverse of [writeProv](#). The format is auto-detected from the file contents (PROV-N text, PROV JSON-LD, or PROV-JSON).

## Usage

```
readProv(path)
```

## Arguments

path                      Path to a PROV document written by [writeProv](#).

## Value

A provenance data.frame with the same columns as [provenance](#).

## See Also

[writeProv](#)

## Examples

```
pe <- PhysioExperiment(
  S4Vectors::SimpleList(raw = matrix(rnorm(20), 10, 2)), samplingRate = 100)
pe <- logStep(pe, "import")
tf <- tempfile(fileext = ".json")
writeProv(pe, tf)
readProv(tf)
```

---

readWFDB

*Read a WFDB record*


---

## Description

Reads a WFDB record's header (.hea) and interleaved signal file (.dat) into a PhysioExperiment. Storage formats 16, 61, 80, 212, 24 and 32 are supported.

## Usage

```
readWFDB(record)
```

## Arguments

record                      Path to the record with or without extension (e.g. "path/to/100" or "path/to/100.hea").

## Value

A PhysioExperiment with physical signal values in the "raw" assay, channel labels/units/gain in colData, and the sampling rate set.

## References

Moody, G. B. & Mark, R. G. (2001). The impact of the MIT-BIH Arrhythmia Database. *IEEE Eng. Med. Biol.*, 20(3), 45-50.

## See Also

[writeWFDB](#), [readWFDBAnnotation](#)

---

|                    |                                    |
|--------------------|------------------------------------|
| readWFDBAnnotation | <i>Read a WFDB annotation file</i> |
|--------------------|------------------------------------|

---

**Description**

Parses a WFDB binary annotation file (MIT format) into a `PhysioEvents`: each annotation's sample number becomes the event onset (in samples), its beat symbol the event type, and any auxiliary string the event value.

**Usage**

```
readWFDBAnnotation(path)
```

**Arguments**

|      |                                               |
|------|-----------------------------------------------|
| path | Path to the annotation file (e.g. "100.atr"). |
|------|-----------------------------------------------|

**Value**

A `PhysioEvents` object (onset in samples).

**See Also**

[writeWFDBAnnotation](#)

---

|             |                                           |
|-------------|-------------------------------------------|
| realizeHDF5 | <i>Realize HDF5-backed data to memory</i> |
|-------------|-------------------------------------------|

---

**Description**

Loads HDF5-backed assays into memory as regular arrays.

**Usage**

```
realizeHDF5(x, assays = NULL)
```

**Arguments**

|        |                                                                                                  |
|--------|--------------------------------------------------------------------------------------------------|
| x      | A <code>PhysioExperiment</code> object.                                                          |
| assays | Optional character vector of assay names to realize. If <code>NULL</code> , realizes all assays. |

**Value**

A `PhysioExperiment` object with the specified assays realized as in-memory arrays.

## References

The HDF Group (1997-2024). "Hierarchical Data Format, version 5." <https://www.hdfgroup.org/HDF5/>

## See Also

[readPhysioHDF5](#), [writePhysioHDF5](#), [isHDF5Backed](#)

## Examples

```
# Write and read back HDF5-backed data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".h5")
writePhysioHDF5(pe, tmp)
pe_delayed <- readPhysioHDF5(tmp)

# Realize all assays to memory
pe_mem <- realizeHDF5(pe_delayed)
isHDF5Backed(pe_mem) # FALSE

unlink(tmp)
```

---

|                    |                                                    |
|--------------------|----------------------------------------------------|
| registerExperiment | <i>Register a PhysioExperiment in the database</i> |
|--------------------|----------------------------------------------------|

---

## Description

Stores experiment metadata and optionally signal data in the database.

## Usage

```
registerExperiment(
  con,
  x,
  experiment_id = NULL,
  subject_id = NULL,
  session_id = NULL,
  task = NULL,
  store_signals = FALSE,
  chunk_size = 10000L,
  parquet_dir = NULL
)
```

**Arguments**

|               |                                                                                                                                                                                                                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| con           | A DuckDB connection.                                                                                                                                                                                                                                                                                                                |
| x             | A <code>PhysioExperiment</code> object.                                                                                                                                                                                                                                                                                             |
| experiment_id | Unique identifier for the experiment.                                                                                                                                                                                                                                                                                               |
| subject_id    | Subject identifier.                                                                                                                                                                                                                                                                                                                 |
| session_id    | Session identifier.                                                                                                                                                                                                                                                                                                                 |
| task          | Task name.                                                                                                                                                                                                                                                                                                                          |
| store_signals | If TRUE, stores signal data in chunks.                                                                                                                                                                                                                                                                                              |
| chunk_size    | Number of samples per chunk for signal storage.                                                                                                                                                                                                                                                                                     |
| parquet_dir   | Optional directory for a Parquet backend. When supplied, the experiment is written with <code>writeParquet()</code> and its default assay is registered as a DuckDB view named <experiment_id>_signals via <code>registerParquetAssay()</code> , enabling out-of-core queries over the Parquet file without loading signals into R. |

**Value**

The `experiment_id`.

**References**

Raasveldt M, Muehleisen H (2019). "DuckDB: an embeddable analytical database." Proceedings of the 2019 International Conference on Management of Data (SIGMOD). doi:10.1145/3299869.3320212

**See Also**

[loadExperiment](#), [queryExperiments](#), [deleteExperiment](#), [initPhysioSchema](#)

**Examples**

```
# DuckDB is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("duckdb", quietly = TRUE)) {
  tf <- tempfile(fileext = ".duckdb")
  con <- connectDatabase(tf)
  initPhysioSchema(con)
  # Build a small deterministic PhysioExperiment
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256
  )
  exp_id <- registerExperiment(con, pe,
    experiment_id = "demo1", subject_id = "sub01", task = "rest")
  disconnectDatabase(con)
  unlink(tf)
}
```

---

|                      |                                                                                 |
|----------------------|---------------------------------------------------------------------------------|
| registerParquetAssay | <i>Register a Parquet-backed assay as a DuckDB view for out-of-core queries</i> |
|----------------------|---------------------------------------------------------------------------------|

---

### Description

Creates (or replaces) a DuckDB view over an assay Parquet file written by `writeParquet()`, using DuckDB's `read_parquet()` so that aggregates are computed directly over the file without loading it into R. The view columns are `ch1`, `ch2`, ... (one per channel).

### Usage

```
registerParquetAssay(con, dir, view_name = "signals", assay = NULL)
```

### Arguments

|                        |                                                             |
|------------------------|-------------------------------------------------------------|
| <code>con</code>       | A DuckDB connection (from <code>connectDatabase()</code> ). |
| <code>dir</code>       | A directory written by <code>writeParquet()</code> .        |
| <code>view_name</code> | Name of the DuckDB view to create (default "signals").      |
| <code>assay</code>     | Assay to expose; defaults to the manifest's default assay.  |

### Value

The `view_name`, invisibly.

### See Also

`writeParquet()`, `registerExperiment()`

---

|       |                                                             |
|-------|-------------------------------------------------------------|
| toMNE | <i>Convert a PhysioExperiment to an MNE-Python RawArray</i> |
|-------|-------------------------------------------------------------|

---

### Description

Builds an `mne.io.RawArray` from a `PhysioExperiment`: channel names, channel types and sampling rate populate the `mne.Info`; any electrode positions (`getElectrodePositions()`) become a `DigMontage`; and any events (`getEvents()`) become `mne.Annotations`. The signal matrix is passed through unchanged, so `fromMNE(toMNE(x))` reproduces it exactly.

### Usage

```
toMNE(x, assay = NULL)
```

**Arguments**

`x` A `PhysioExperiment`.

`assay` Assay to export (default: the object's default assay).

**Details**

Channel types not recognised by MNE are mapped to "misc" (and are lower-cased to MNE's convention). Channel names must be unique. An event's type and (when present) value are packed into the annotation description so both survive the round-trip; a missing (NA) value becomes "" because MNE annotations cannot represent NA. Events whose onset falls outside the recording are dropped by MNE (with a warning).

**Value**

An `mne.io.RawArray` Python object.

**References**

Gramfort A, et al. (2013). "MEG and EEG data analysis with MNE-Python." *Frontiers in Neuroscience*, 7, 267.

**See Also**

[fromMNE\(\)](#), [hasMNE\(\)](#)

**Examples**

```
if (hasMNE()) {
  pe <- PhysioExperiment(
    assays = S4Vectors::SimpleList(raw = matrix(rnorm(300), 100, 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz"),
                                   type = rep("eeg", 3)),
    samplingRate = 100)
  raw <- toMNE(pe)
}
```

---

 validateBIDS

*Validate BIDS dataset structure*


---

**Description**

Performs basic validation of BIDS compliance.

**Usage**

```
validateBIDS(bids_root)
```

**Arguments**

`bids_root` Path to the BIDS dataset root.

**Value**

A list with validation results.

**References**

Gorgolewski KJ, et al. (2016). "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments." *Scientific Data*, 3, 160044. doi:10.1038/sdata.2016.44

**See Also**

[readBIDS](#), [writeBIDS](#), [listBIDSSubjects](#)

**Examples**

```
# Write a minimal BIDS dataset, then validate it
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
bids_root <- file.path(tempdir(), "bids-validate-demo")
writeBIDS(pe, bids_root, subject = "01", task = "rest")

result <- validateBIDS(bids_root)
result$valid

unlink(bids_root, recursive = TRUE)
```

---

validateClinicalMetadata

*Validate clinical metadata*

---

**Description**

Validate clinical metadata

**Usage**

```
validateClinicalMetadata(
  x,
  required_cols = c("subject_id", "visit_id", "scale_name", "scale_score"),
  allowed_source_system = c("EDC", "EHR", "paper_crf"),
  allowed_assessor_role = c("PT", "OT", "MD", "RN", "researcher"),
  strict = FALSE
)
```

**Arguments**

`x` Data frame created from EDC/EHR exports.

`required_cols` Required columns that must exist and be non-missing.

`allowed_source_system` Allowed values for `source_system` when present.

`allowed_assessor_role` Allowed values for `assessor_role` when present.

`strict` Logical; stop when validation fails.

**Value**

A list with validation details and an overall valid flag.

**References**

Goldberger AL, et al. (2000). "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals." *Circulation*, 101(23), e215-e220. doi:10.1161/01.CIR.101.23.e215

**See Also**

[readClinicalMetadataCSV](#), [mapClinicalCodes](#), [validateBIDS](#)

**Examples**

```
df <- data.frame(
  subject_id = "S01",
  visit_id = "V01",
  scale_name = "FIM",
  scale_score = 88,
  assessment_date = "2026-01-01",
  stringsAsFactors = FALSE
)
validateClinicalMetadata(df)
```

---

writeAssayHDF5

*Write assay to HDF5 file*

---

**Description**

Writes a single assay to an existing HDF5 file.

**Usage**

```
writeAssayHDF5(x, path, assay_name, compression_level = 6L)
```

**Arguments**

|                   |                                             |
|-------------------|---------------------------------------------|
| x                 | A PhysioExperiment object.                  |
| path              | Path to the HDF5 file.                      |
| assay_name        | Character string naming the assay to write. |
| compression_level | Integer compression level (0-9).            |

**Value**

Invisible NULL. The assay is written to the HDF5 file as a side effect.

**References**

The HDF Group (1997-2024). "Hierarchical Data Format, version 5." <https://www.hdfgroup.org/HDF5/>

**See Also**

[writePhysioHDF5](#), [readPhysioHDF5](#), [realizeHDF5](#)

**Examples**

```
# Create a PhysioExperiment and write it to a temporary HDF5 file
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".h5")
writePhysioHDF5(pe, tmp)

# Add a second assay to the object and write only that assay to the file
SummarizedExperiment::assay(pe, "scaled") <-
  SummarizedExperiment::assay(pe, "raw") * 2
writeAssayHDF5(pe, tmp, "scaled")

unlink(tmp)
```

---

writeBDF

*Write BDF (BioSemi Data Format) file*

---

**Description**

Writes a PhysioExperiment object to BDF format (24-bit resolution).

**Usage**

```
writeBDF(x, path, patient_id = "X", recording_id = "X")
```

**Arguments**

|                           |                                         |
|---------------------------|-----------------------------------------|
| <code>x</code>            | A <code>PhysioExperiment</code> object. |
| <code>path</code>         | Output file path.                       |
| <code>patient_id</code>   | Patient identification string.          |
| <code>recording_id</code> | Recording identification string.        |

**Details**

BDF is a 24-bit extension of EDF providing higher resolution than the standard 16-bit EDF format. It is commonly used with BioSemi acquisition systems but can be used for any high-resolution physiological data.

**Value**

Invisible `NULL`.

**Examples**

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 100
)
tmp <- tempfile(fileext = ".bdf")
writeBDF(pe, tmp)
unlink(tmp)
```

---

writeBIDS

---

Write *PhysioExperiment* to BIDS format

---

**Description**

Writes data in BIDS-compliant directory structure.

**Usage**

```
writeBIDS(
  x,
  bids_root,
  subject,
  session = NULL,
  task,
  run = NULL,
  modality = c("eeg", "ieeg"),
  overwrite = FALSE
)
```

**Arguments**

|                        |                                                       |
|------------------------|-------------------------------------------------------|
| <code>x</code>         | A <code>PhysioExperiment</code> object.               |
| <code>bids_root</code> | Path to the BIDS dataset root directory.              |
| <code>subject</code>   | Subject identifier (without 'sub-' prefix).           |
| <code>session</code>   | Session identifier (without 'ses-' prefix). Optional. |
| <code>task</code>      | Task name.                                            |
| <code>run</code>       | Run number. Optional.                                 |
| <code>modality</code>  | Data modality: "eeg" or "ieeg".                       |
| <code>overwrite</code> | If TRUE, overwrites existing files.                   |

**Value**

Invisible path to the created files.

**References**

Gorgolewski KJ, et al. (2016). "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments." *Scientific Data*, 3, 160044. doi:10.1038/sdata.2016.44

**See Also**

[readBIDS](#), [writeEDF](#), [listBIDSSubjects](#), [validateBIDS](#)

**Examples**

```
# Create a PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 100
)

# Export to a BIDS dataset in a temporary directory
bids_root <- file.path(tempdir(), "bids-write-demo")
writeBIDS(pe, bids_root, subject = "01", task = "rest")

unlink(bids_root, recursive = TRUE)
```

---

|                     |                                          |
|---------------------|------------------------------------------|
| writeBIDSDerivative | <i>Write a BIDS derivative recording</i> |
|---------------------|------------------------------------------|

---

**Description**

Writes `x` as a derivative under `deriv_root` with a `desc-<label>` entity, and a derivative `dataset_description.json` whose `GeneratedBy` is populated from the object's provenance ([provenance](#)). The signals are stored as a gzipped `_physio.tsv.gz` plus JSON.

Usage

```
writeBIDSDerivative(  
  x,  
  deriv_root,  
  subject,  
  task,  
  desc,  
  session = NULL,  
  run = NULL,  
  pipeline_name = NULL,  
  overwrite = FALSE  
)
```

Arguments

- x                    A PhysioExperiment (typically a processed derivative).
- deriv\_root        Derivative dataset root (e.g. "<bids>/derivatives/physio-clean").
- subject, task, session, run        BIDS entities.
- desc                The desc- label (e.g. "clean").
- pipeline\_name    Name recorded in GeneratedBy (default basename(deriv\_root)).
- overwrite        Overwrite existing files (default FALSE).

Value

The written data path, invisibly.

See Also

[readBIDSDerivatives](#)

---

|                 |                                      |
|-----------------|--------------------------------------|
| writeBIDSMotion | <i>Write a Motion-BIDS recording</i> |
|-----------------|--------------------------------------|

---

Description

Writes a motion-capture PhysioExperiment in the BIDS motion datatype (Jeung et al. 2023): a headerless `_motion.tsv`, a `_channels.tsv` carrying tracked\_point/component, and a `_motion.json` sidecar with SamplingFrequency and TrackingSystemName.

Usage

```
writeBIDSMotion(  
  x,  
  bids_root,  
  subject,  
  task,  
  session = NULL,  
  run = NULL,  
  tracking_system = "unspecified",  
  overwrite = FALSE  
)
```

Arguments

|                 |                                                                                                                                                                                                                                   |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x               | A <code>PhysioExperiment</code> of motion data (2D assay). <code>colData</code> columns <code>label</code> , <code>type</code> , <code>unit</code> , <code>tracked_point</code> and <code>component</code> are used when present. |
| bids_root       | BIDS dataset root.                                                                                                                                                                                                                |
| subject, task   | Subject label and task label.                                                                                                                                                                                                     |
| session, run    | Optional session label and run number.                                                                                                                                                                                            |
| tracking_system | Tracking-system label (the <code>tracksys-</code> entity and <code>TrackingSystemName</code> ).                                                                                                                                   |
| overwrite       | Overwrite existing files (default <code>FALSE</code> ).                                                                                                                                                                           |

Value

The data directory path, invisibly.

References

Jeung, S., et al. (2023). Motion-BIDS. *Scientific Data*.

See Also

[readBIDSMotion](#)

---

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| writeBIDSPhysio | <i>Write a BIDS continuous physiological recording</i> |
|-----------------|--------------------------------------------------------|

---

Description

Writes a gzipped `_physio.tsv.gz` (headerless, columns defined in the JSON) plus a `_physio.json` sidecar with `SamplingFrequency`, `StartTime` and `Columns`.

**Usage**

```
writeBIDSPhysio(
  x,
  bids_root,
  subject,
  task,
  session = NULL,
  run = NULL,
  start_time = 0,
  overwrite = FALSE
)
```

**Arguments**

`x` A `PhysioExperiment` (2D assay).  
`bids_root`, `subject`, `task`, `session`, `run` BIDS entities.  
`start_time` Recording start time relative to the main recording, in seconds (BIDS `StartTime`; default 0).  
`overwrite` Overwrite existing files (default FALSE).

**Value**

The written `.tsv.gz` path, invisibly.

**See Also**

[readBIDSPhysio](#), [attachBIDSPhysio](#)

---

|                  |                                                     |
|------------------|-----------------------------------------------------|
| writeBrainVision | <i>Write PhysioExperiment to BrainVision format</i> |
|------------------|-----------------------------------------------------|

---

**Description**

Saves a `PhysioExperiment` object to BrainVision format (three files: `.vhdr` header, `.vmrk` markers, `.eeg` binary data).

**Usage**

```
writeBrainVision(
  x,
  path,
  overwrite = FALSE,
  binary_format = c("IEEE_FLOAT_32", "INT_16", "INT_32"),
  assay_name = NULL
)
```

Arguments

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| x             | A <code>PhysioExperiment</code> object.                                   |
| path          | Output path (without extension, or with <code>.vhdr</code> extension).    |
| overwrite     | Logical. If <code>TRUE</code> , overwrites existing files.                |
| binary_format | Binary format for data: "IEEE_FLOAT_32" (default), "INT_16", or "INT_32". |
| assay_name    | Assay to export. If <code>NULL</code> , uses default assay.               |

Value

Invisible `NULL`.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 100
)
base <- tempfile()
writeBrainVision(pe, base)
unlink(paste0(base, c(".vhdr", ".vmrk", ".eeg")))
```

---

|          |                                           |
|----------|-------------------------------------------|
| writeCSV | <i>Write PhysioExperiment to CSV file</i> |
|----------|-------------------------------------------|

---

Description

Writes physiological signal data to a CSV file.

Usage

```
writeCSV(
  x,
  path,
  format = c("wide", "long"),
  include_time = TRUE,
  assay_name = NULL,
  sep = ",",
  ...
)
```

**Arguments**

|              |                                                                     |
|--------------|---------------------------------------------------------------------|
| x            | A <code>PhysioExperiment</code> object.                             |
| path         | Output file path.                                                   |
| format       | Output format: "wide" (time x channels) or "long".                  |
| include_time | Logical. If TRUE, includes a time column.                           |
| assay_name   | Assay to export. If NULL, uses default assay.                       |
| sep          | Column separator. Default is ",".                                   |
| ...          | Additional arguments passed to <code>write.csv/write.table</code> . |

**Value**

Invisible path to the created file.

**References**

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, 59(10), 1-23. doi:10.18637/jss.v059.i10

**See Also**

[readCSV](#), [writeEventsCSV](#), [writeEDF](#), [writePhysioHDF5](#)

**Examples**

```
# Create example data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)

# Write to temporary CSV file
tmp <- tempfile(fileext = ".csv")
writeCSV(pe, tmp)

# Write in long format
writeCSV(pe, tmp, format = "long")

# Clean up
unlink(tmp)
```

writeEDF

*Write EDF file***Description**

Writes a `PhysioExperiment` object to EDF format.

**Usage**

```
writeEDF(x, path, patient_id = "X", recording_id = "X")
```

**Arguments**

|                           |                                         |
|---------------------------|-----------------------------------------|
| <code>x</code>            | A <code>PhysioExperiment</code> object. |
| <code>path</code>         | Output file path.                       |
| <code>patient_id</code>   | Patient identification string.          |
| <code>recording_id</code> | Recording identification string.        |

**Value**

Invisible `NULL`. The EDF file is written to `path` as a side effect.

**References**

Kemp, B., et al. (1992). "A simple format for exchange of digitized polygraphic recordings." *Electroencephalography and Clinical Neurophysiology*, 82(5), 391-393. doi:10.1016/0013-4694(92)90009-7

**See Also**

[readEDF](#), [writeBIDS](#), [writePhysioHDF5](#), [writeCSV](#)

**Examples**

```
# Create a PhysioExperiment with EEG-like data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 100
)

# Export to EDF format in a temporary file
tmp <- tempfile(fileext = ".edf")
writeEDF(pe, tmp, patient_id = "Subject01", recording_id = "Session1")

unlink(tmp)
```

---

```
writeElectrodePositionsCSV
```

*Write electrode positions to CSV*

---

## Description

Writes electrode positions from a `PhysioExperiment` to CSV.

## Usage

```
writeElectrodePositionsCSV(x, path, sep = ",", ...)
```

## Arguments

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| <code>x</code>    | A <code>PhysioExperiment</code> object with electrode positions. |
| <code>path</code> | Output file path.                                                |
| <code>sep</code>  | Column separator.                                                |
| <code>...</code>  | Additional arguments passed to <code>write.csv</code> .          |

## Value

Invisible path to the created file.

## References

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, 59(10), 1-23. doi:10.18637/jss.v059.i10

## See Also

[readElectrodePositionsCSV](#), [writeCSV](#), [writeBIDS](#)

## Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
pe <- applyMontage(pe, "10-20")

# Write electrode positions
tmp <- tempfile(fileext = ".csv")
writeElectrodePositionsCSV(pe, tmp)

# Clean up
unlink(tmp)
```

---

|                |                                 |
|----------------|---------------------------------|
| writeEventsCSV | <i>Write events to CSV file</i> |
|----------------|---------------------------------|

---

### Description

Writes PhysioEvents to a CSV file.

### Usage

```
writeEventsCSV(events, path, sep = ",", ...)
```

### Arguments

|        |                                           |
|--------|-------------------------------------------|
| events | A PhysioEvents object.                    |
| path   | Output file path.                         |
| sep    | Column separator.                         |
| ...    | Additional arguments passed to write.csv. |

### Value

Invisible path to the created file.

### References

Wickham H (2014). "Tidy Data." Journal of Statistical Software, 59(10), 1-23. doi:10.18637/jss.v059.i10

### See Also

[readEventsCSV](#), [writeCSV](#), [writeBIDS](#)

### Examples

```
# Create events
events <- PhysioEvents(
  onset = c(1.0, 2.5, 4.0),
  duration = c(0.5, 0.5, 0.5),
  type = c("stimulus", "stimulus", "response"),
  value = c("A", "B", "correct")
)

# Write to temporary file
tmp <- tempfile(fileext = ".csv")
writeEventsCSV(events, tmp)

# Clean up
unlink(tmp)
```

---

writeGDF*Write PhysioExperiment to GDF format*

---

## Description

Saves a PhysioExperiment object to GDF (General Data Format) version 2.x.

## Usage

```
writeGDF(  
  x,  
  path,  
  patient_id = "",  
  recording_id = "",  
  overwrite = FALSE,  
  assay_name = NULL  
)
```

## Arguments

|              |                                               |
|--------------|-----------------------------------------------|
| x            | A PhysioExperiment object.                    |
| path         | Output file path.                             |
| patient_id   | Patient identifier string.                    |
| recording_id | Recording identifier string.                  |
| overwrite    | Logical. If TRUE, overwrites existing file.   |
| assay_name   | Assay to export. If NULL, uses default assay. |

## Value

Invisible NULL.

## Examples

```
pe <- PhysioExperiment(  
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),  
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),  
  samplingRate = 100  
)  
tmp <- tempfile(fileext = ".gdf")  
writeGDF(pe, tmp)  
unlink(tmp)
```

writeMAT

*Write PhysioExperiment to MATLAB .mat file*

---

## Description

Saves a PhysioExperiment object to MATLAB .mat format.

## Usage

```
writeMAT(  
    x,  
    path,  
    data_var = "data",  
    include_metadata = TRUE,  
    eeglab = FALSE,  
    assay_name = NULL  
)
```

## Arguments

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| x                | A PhysioExperiment object.                                |
| path             | Output file path.                                         |
| data_var         | Name for the data variable in the .mat file.              |
| include_metadata | Logical. If TRUE, includes metadata variables.            |
| eeglab           | Logical. If TRUE, exports in EEGLAB-compatible structure. |
| assay_name       | Assay to export. If NULL, uses default assay.             |

## Value

Invisible NULL.

## References

MathWorks (2024). "MAT-File Format." Technical documentation. [https://www.mathworks.com/help/matlab/import\\_export/mat-file-versions.html](https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html)

## See Also

[readMAT](#), [writeCSV](#), [writeEDF](#), [writePhysioHDF5](#)

## Examples

```
# 'R.matlab' is an optional dependency; guard the example so it only runs
# when the package is installed.
if (requireNamespace("R.matlab", quietly = TRUE)) {
  pe <- PhysioExperiment(
    assays = list(raw = matrix(0, nrow = 100, ncol = 10)),
    colData = S4Vectors::DataFrame(label = paste0("Ch", seq_len(10))),
    samplingRate = 256
  )
  tf <- tempfile(fileext = ".mat")
  writeMAT(pe, tf)
  unlink(tf)
}
```

---

writeParquet

---

Write a PhysioExperiment to a Parquet dataset directory

---

## Description

Serialises a `PhysioExperiment` (or `MultiPhysioExperiment`) to a directory of Apache Parquet tables and JSON sidecars. Each assay is stored as a columnar Parquet table (one column per channel), and `colData`, `rowData`, `events`, `metadata`, `provenance` and per-assay sampling rates are stored alongside so that `readParquet()` reproduces the object exactly.

## Usage

```
writeParquet(x, dir, overwrite = FALSE)
```

## Arguments

|                        |                                                                         |
|------------------------|-------------------------------------------------------------------------|
| <code>x</code>         | A <code>PhysioExperiment</code> or <code>MultiPhysioExperiment</code> . |
| <code>dir</code>       | Output directory (created if absent).                                   |
| <code>overwrite</code> | Overwrite a non-empty directory (default FALSE).                        |

## Details

Parquet is a columnar, self-describing format that DuckDB can query in place via `read_parquet()`, enabling out-of-core aggregation over the assays without loading them into R (see [registerParquetAssay\(\)](#)).

A `MultiPhysioExperiment` is written as a `streams/` sub-directory (one Parquet dataset per stream) plus its common clock (`t0`, reference rate, per-stream offsets).

## Value

The output `dir`, invisibly.

## References

Apache Software Foundation. "Apache Parquet." <https://parquet.apache.org/>.

**See Also**

[readParquet\(\)](#), [registerParquetAssay\(\)](#), [writePhysioHDF5\(\)](#)

**Examples**

```
if (requireNamespace("arrow", quietly = TRUE)) {
  pe <- PhysioExperiment(
    assays = S4Vectors::SimpleList(raw = matrix(rnorm(30), 10, 3)),
    colData = S4Vectors::DataFrame(label = c("Fp1", "Fp2", "Cz")),
    samplingRate = 256)
  d <- tempfile("pe-parquet")
  writeParquet(pe, d)
  pe2 <- readParquet(d)
}
```

---

writePhysio

*Basic read/write helpers*

---

**Description**

These helper functions provide a lightweight interface to serialise and deserialise `PhysioExperiment` objects to RDS files. They act as placeholders for richer IO backends that can be developed later.

**Usage**

`writePhysio(x, path)`

`readPhysio(path)`

**Arguments**

`x`                      A `PhysioExperiment` object.  
`path`                    Path to an `.rds` file.

**Value**

`readPhysio()` returns a `PhysioExperiment` object. `writePhysio()` returns the input object invisibly.

**References**

Wickham, H. (2014). "Tidy Data." *Journal of Statistical Software*, 59(10), 1-23. doi:10.18637/jss.v059.i10

**See Also**

[readEDF](#), [readPhysioHDF5](#), [readCSV](#), [readMAT](#)

**Examples**

```
# Create a PhysioExperiment object
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  samplingRate = 256
)

# Write to a temporary file
tmp <- tempfile(fileext = ".rds")
writePhysio(pe, tmp)

# Read it back
pe_loaded <- readPhysio(tmp)
samplingRate(pe_loaded)

# Clean up
unlink(tmp)
```

writePhysioHDF5

*HDF5 Backend for PhysioExperiment***Description**

Functions for reading and writing PhysioExperiment objects to HDF5 format, supporting out-of-memory operations for large datasets. Write PhysioExperiment to HDF5

**Usage**

```
writePhysioHDF5(
  x,
  path,
  overwrite = FALSE,
  chunk_dims = NULL,
  compression_level = 6L
)
```

**Arguments**

|                                |                                                                                                                   |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <code>x</code>                 | A PhysioExperiment object.                                                                                        |
| <code>path</code>              | Path to the output HDF5 file.                                                                                     |
| <code>overwrite</code>         | Logical. If TRUE, overwrites existing file.                                                                       |
| <code>chunk_dims</code>        | Optional integer vector specifying chunk dimensions for HDF5 storage. If NULL, defaults are chosen automatically. |
| <code>compression_level</code> | Integer compression level (0-9). Default is 6.                                                                    |

## Details

Saves a `PhysioExperiment` object to HDF5 format, enabling out-of-memory access for large datasets.

## Value

Invisible `NULL`. The HDF5 file is written to `path` as a side effect.

## References

The HDF Group (1997-2024). "Hierarchical Data Format, version 5." <https://www.hdfgroup.org/HDF5/>

## See Also

[readPhysioHDF5](#), [isHDF5Backed](#), [realizeHDF5](#), [writeAssayHDF5](#)

## Examples

```
# Create a PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  samplingRate = 100
)

# Save to a temporary HDF5 file with compression
tmp <- tempfile(fileext = ".h5")
writePhysioHDF5(pe, tmp, compression_level = 6)

# Overwrite the existing file
writePhysioHDF5(pe, tmp, overwrite = TRUE)

unlink(tmp)
```

---

`writeProv`

*Serialize a provenance trail to a W3C PROV interchange file*

---

## Description

Writes the provenance / audit trail of a `PhysioExperiment` (or a provenance `data.frame` from [provenance](#)) to a W3C PROV document. Three interchange syntaxes are supported: PROV-JSON (default), PROV in JSON-LD, and PROV-N. The document round-trips exactly through [readProv](#).

## Usage

```
writeProv(x, path, format = c("prov-json", "json-ld", "prov-n"))
```

**Arguments**

|        |                                                             |
|--------|-------------------------------------------------------------|
| x      | A <code>PhysioExperiment</code> or a provenance data.frame. |
| path   | Output file path.                                           |
| format | One of "prov-json", "json-ld", "prov-n".                    |

**Value**

path, invisibly.

**References**

W3C PROV-JSON (Huynh et al. 2013); PROV-O; PROV-N.

**See Also**

[readProv](#), [provenance](#)

**Examples**

```
pe <- PhysioExperiment(
  S4Vectors::SimpleList(raw = matrix(rnorm(20), 10, 2)), samplingRate = 100)
pe <- logStep(pe, "filterSignals", params = list(low = 1, high = 40))
tf <- tempfile(fileext = ".json")
writeProv(pe, tf)
readProv(tf)$activity
```

---

writeWFDB

---

Write a WFDB record

---

**Description**

Writes a `PhysioExperiment` to a WFDB record (.hea + .dat).

**Usage**

```
writeWFDB(x, record, format = 16L, gain = 200, baseline = 0)
```

**Arguments**

|          |                                                                       |
|----------|-----------------------------------------------------------------------|
| x        | A <code>PhysioExperiment</code> (2D "raw" assay of physical values).  |
| record   | Output record path (without extension).                               |
| format   | WFDB storage format: one of 16, 61, 80, 212, 24, 32.                  |
| gain     | ADC gain (units per ADU); a scalar or one per channel.                |
| baseline | ADC baseline value (raw value for 0 physical); scalar or per channel. |

**Value**

record, invisibly.

**See Also**

[readWFDB](#)

---

|                     |                                     |
|---------------------|-------------------------------------|
| writeWFDBAnnotation | <i>Write a WFDB annotation file</i> |
|---------------------|-------------------------------------|

---

**Description**

Writes a `PhysioEvents` object (onset in samples, type = beat symbol, value = auxiliary string) to a WFDB binary annotation file.

**Usage**

```
writeWFDBAnnotation(events, path)
```

**Arguments**

|        |                                                            |
|--------|------------------------------------------------------------|
| events | A <code>PhysioEvents</code> object with onsets in samples. |
| path   | Output annotation file path.                               |

**Value**

path, invisibly.

**See Also**

[readWFDBAnnotation](#)

# Index

attachBIDSPhysio, [3](#), [29](#), [57](#)

bdfTriggerEvents, [4](#), [25](#), [26](#)

connectDatabase, [5](#), [6](#), [16](#)  
connectDatabase(), [48](#)

dbStats, [6](#), [16](#), [24](#)  
deleteExperiment, [7](#), [20](#), [47](#)  
disconnectDatabase (connectDatabase), [5](#)  
downloadFigshare, [8](#)  
downloadFigshare(), [9](#), [11](#), [13](#), [14](#)  
downloadFromDOI, [9](#)  
downloadFromDOI(), [9](#), [11](#), [13](#), [22](#)  
downloadMendeley, [10](#)  
downloadMendeley(), [9](#), [13](#), [22](#)  
downloadPhysioNet, [11](#)  
downloadPhysioNet(), [9](#), [13](#)  
downloadZenodo, [12](#)  
downloadZenodo(), [9](#), [11](#)

figshareCollectionArticles, [13](#)  
figshareCollectionArticles(), [8](#), [9](#)  
fromMNE, [14](#)  
fromMNE(), [15](#), [49](#)

hasMNE, [15](#)  
hasMNE(), [14](#), [23](#), [49](#)

initPhysioSchema, [5](#), [6](#), [15](#), [47](#)  
isHDF5Backed, [16](#), [42](#), [46](#), [68](#)

listBIDS Sessions, [17](#), [18](#)  
listBIDS Subjects, [18](#), [18](#), [27](#), [50](#), [54](#)  
listWFDBRecords, [19](#)  
loadExperiment, [7](#), [19](#), [24](#), [47](#)

mapClinicalCodes, [20](#), [32](#), [51](#)  
mendeleyDatasetFiles, [22](#)  
mendeleyDatasetFiles(), [11](#)  
MultiPhysioExperiment, [4](#), [25](#), [35](#)

PhysioExperiment, [35](#), [42](#), [45](#), [66](#)  
provenance, [43](#), [54](#), [68](#), [69](#)  
pyMNEVersion, [23](#)  
pyMNEVersion(), [15](#)

queryExperiments, [5–7](#), [20](#), [23](#), [47](#)

readBDF, [4](#), [24](#), [26](#), [38](#)  
readBDFStatus, [4](#), [26](#)  
readBIDS, [18](#), [26](#), [35–37](#), [50](#), [54](#)  
readBIDS Derivatives, [28](#), [55](#)  
readBIDSMotion, [28](#), [56](#)  
readBIDSPhysio, [4](#), [29](#), [57](#)  
readBrainVision, [30](#)  
readClinicalMetadataCSV, [21](#), [31](#), [51](#)  
readCSV, [32](#), [32](#), [35–37](#), [41](#), [59](#), [66](#)  
readEDF, [27](#), [33](#), [34](#), [38](#), [41](#), [60](#), [66](#)  
readElectrodePositionsCSV, [35](#), [61](#)  
readEventsCSV, [33](#), [37](#), [62](#)  
readFIF, [38](#)  
readGDF, [39](#)  
readMAT, [33](#), [40](#), [64](#), [66](#)  
readParquet, [41](#)  
readParquet(), [66](#)  
readPhysio (writePhysio), [66](#)  
readPhysioHDF5, [17](#), [35](#), [41](#), [42](#), [46](#), [52](#), [66](#), [68](#)  
readProv, [43](#), [68](#), [69](#)  
readWFDB, [44](#), [70](#)  
readWFDBAnnotation, [44](#), [45](#), [70](#)  
realizeHDF5, [17](#), [42](#), [45](#), [52](#), [68](#)  
registerExperiment, [5](#), [7](#), [16](#), [20](#), [24](#), [46](#)  
registerExperiment(), [48](#)  
registerParquetAssay, [48](#)  
registerParquetAssay(), [47](#), [65](#), [66](#)

toMNE, [48](#)  
toMNE(), [14](#), [15](#)

validateBIDS, [18](#), [27](#), [49](#), [51](#), [54](#)  
validateClinicalMetadata, [21](#), [32](#), [50](#)

writeAssayHDF5, [42](#), [51](#), [68](#)  
writeBDF, [52](#)  
writeBIDS, [27](#), [50](#), [53](#), [60–62](#)  
writeBIDSDerivative, [28](#), [54](#)  
writeBIDSMotion, [29](#), [55](#)  
writeBIDSPhysio, [29](#), [56](#)  
writeBrainVision, [57](#)  
writeCSV, [33](#), [58](#), [60–62](#), [64](#)  
writeEDF, [35](#), [54](#), [59](#), [60](#), [64](#)  
writeElectrodePositionsCSV, [36](#), [61](#)  
writeEventsCSV, [37](#), [59](#), [62](#)  
writeGDF, [63](#)  
writeMAT, [41](#), [64](#)  
writeParquet, [65](#)  
writeParquet(), [41](#), [47](#), [48](#)  
writePhysio, [66](#)  
writePhysioHDF5, [17](#), [42](#), [46](#), [52](#), [59](#), [60](#), [64](#),  
[67](#)  
writePhysioHDF5(), [66](#)  
writeProv, [43](#), [68](#)  
writeWFDB, [44](#), [69](#)  
writeWFDBAnnotation, [45](#), [70](#)