Package: PhysioIO (via r-universe)

May 16, 2026

Title I/O Functions for PhysioExperiment Objects

Version 0.2.0

Author Yusuke Matsui

Maintainer Yusuke Matsui <you@example.com>

Description Provides comprehensive I/O capabilities for PhysioExperiment objects. Supports EDF/EDF+, HDF5, BIDS, CSV, MATLAB formats, and DuckDB database integration. Includes out-of-memory processing for large datasets via HDF5Array.

Depends R (>= 4.2), PhysioCore

Imports methods, SummarizedExperiment, S4Vectors, HDF5Array, rhdf5, jsonlite, DBI

Suggests duckdb, arrow, R.matlab, knitr, rmarkdown, testthat (>= 3.2.0)

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'io-readwrite.R' 'io-edf.R' 'io-hdf5.R' 'io-bids.R' 'io-csv.R' 'io-clinical.R' 'io-matlab.R' 'db-interface.R' 'db-schema.R' 'zzz.R'

RoxygenNote 7.3.3

Config/pak/sysreqs libssl-dev zlib1g-dev

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

Date/Publication 2026-03-16 11:30:59 UTC

RemoteUrl https://github.com/x-biosignal/PhysioIO

RemoteRef HEAD

RemoteSha 73c35a084e51581776f9416feb4f855624fea630

Contents

`.onLoad` *Package on-load hook*

Description

Package on-load hook

Usage

```
.onLoad(libname, pkg)
```

Arguments

libname	Library path.
pkg	Package name.

`connectDatabase` *Lightweight database interface*

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

```
## Not run:
# Connect to an in-memory database
con <- connectDatabase()

# Connect to a file-based database
con <- connectDatabase("experiments.duckdb")

# Always disconnect when done
disconnectDatabase(con)

## End(Not run)
```

dbStats *Get database statistics*

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

```
## Not run:
con <- connectDatabase("experiments.duckdb")
initPhysioSchema(con)

# Get database statistics
stats <- dbStats(con)
cat("Experiments:", stats$n_experiments, "\n")
cat("Channels:", stats$n_channels, "\n")
cat("Subjects:", paste(stats$subjects, collapse = ", "), "\n")

disconnectDatabase(con)

## End(Not run)
```

`deleteExperiment` *Delete experiment from database*

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

```
## Not run:
con <- connectDatabase("experiments.duckdb")

# Delete an experiment
deleteExperiment(con, "exp_20240101120000_1234")

# Skip existence check (for batch deletion)
deleteExperiment(con, "exp_id", confirm = FALSE)

disconnectDatabase(con)

## End(Not run)
```

initPhysioSchema *DuckDB Schema Management for PhysioExperiment*

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

```
## Not run:
# Set up a new database
con <- connectDatabase("experiments.duckdb")
initPhysioSchema(con)

# Check statistics
dbStats(con)

disconnectDatabase(con)

## End(Not run)
```

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
```

`listBIDSSessions` *List sessions for a subject in BIDS dataset*

Description

List sessions for a subject in BIDS dataset

Usage

```
listBIDSSessions(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

```
## Not run:
# List sessions for a specific subject
sessions <- listBIDSSessions("path/to/bids", subject = "01")
print(sessions)  # e.g., c("baseline", "followup")

## End(Not run)
```

`listBIDSSubjects` *List subjects in a BIDS dataset*

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

```
## Not run:
# List all subjects in a BIDS dataset
subjects <- listBIDSSubjects("path/to/bids")
print(subjects)  # e.g., c("01", "02", "03")

## End(Not run)
```

`loadExperiment` *Load experiment from database*

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

```
## Not run:
con <- connectDatabase("experiments.duckdb")

# Load experiment metadata only
pe <- loadExperiment(con, "exp_20240101120000_1234")

# Load with signal data
pe_full <- loadExperiment(con, "exp_20240101120000_1234",
  load_signals = TRUE
)

disconnectDatabase(con)

## End(Not run)
```

`mapClinicalCodes` *Harmonize clinical codes across sites*

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: 1. named character vector (`names` = source code, `values` = target code), or 2. data frame with columns defined by `from_col` and `to_col`.
`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")
```

---

queryExperiments *Query experiments from database*

Description

Query experiments from database

Usage

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

Arguments

con	A DuckDB connection.
subject_id	Optional filter by subject.
task	Optional filter by task.
date_range	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

```
## Not run:
con <- connectDatabase("experiments.duckdb")

# Query all experiments
all_exps <- queryExperiments(con)

# Filter by subject
sub01_exps <- queryExperiments(con, subject_id = "sub01")

# Filter by task
```

```
rest_exps <- queryExperiments(con, task = "rest")

# Filter by date range
recent <- queryExperiments(con,
  date_range = c("2024-01-01", "2024-12-31")
)

disconnectDatabase(con)

## End(Not run)
```

---

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

bids_root	Path to the BIDS dataset root directory.
subject	Subject identifier (without 'sub-' prefix).
session	Session identifier (without 'ses-' prefix). Optional.
task	Task name.
run	Run number. Optional.
modality	Data modality: "eeg" or "ieeg".
load_events	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

```
## Not run:
# Read EEG data from BIDS dataset
pe <- readBIDS("path/to/bids", subject = "01", task = "rest")

# Read with session and run specified
pe <- readBIDS("path/to/bids", subject = "01", session = "01",
               task = "oddball", run = 1, modality = "eeg")

# Read without loading events
pe <- readBIDS("path/to/bids", subject = "02", task = "rest",
               load_events = FALSE)

## End(Not run)
```

---

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

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

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

```
## Not run:
# Read wide-format CSV with time column
pe <- readCSV("signals.csv", time_col = "time", sampling_rate = 256)

# Read without time column (generate from sampling rate)
pe <- readCSV("signals.csv", sampling_rate = 256)

# Read TSV file
pe <- readCSV("signals.tsv", sep = "\t", sampling_rate = 256)

## End(Not run)
```

---

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)
```

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.

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 sampling rates, data is resampled to the highest rate.

Value

A PhysioExperiment object with the EDF signal data stored in the "raw" assay. Channel metadata (label, transducer, physical dimensions, digital/physical min/max) are stored in colData, and recording metadata (patient ID, start date/time) 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

```
## Not run:
# Read an EDF file
pe <- readEDF("recording.edf")

# Read only specific channels
pe <- readEDF("recording.edf", channels = c("Fp1", "Fp2", "C3", "C4"))

# Read a time window (10 to 60 seconds)
pe <- readEDF("recording.edf", start_time = 10, end_time = 60)

## End(Not run)
```

---

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

```
## Not run:
# Read electrode positions
positions <- readElectrodePositionsCSV("electrodes.csv")

# Apply to PhysioExperiment
pe <- setElectrodePositions(pe, positions)

## End(Not run)
```

readEventsCSV *Read events from CSV/TSV file*

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

```
## Not run:
# Read events from CSV
events <- readEventsCSV("events.csv")

# Add to PhysioExperiment
pe <- setEvents(pe, events)

## End(Not run)
```

`readMAT` *Read PhysioExperiment from MATLAB .mat file*

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

See Also

writeMAT, readEDF, readCSV, readPhysioHDF5

Examples

```
## Not run:
# Read MATLAB file with auto-detection
pe <- readMAT("eeg_data.mat")

# Specify variable names
pe <- readMAT("data.mat", data_var = "signal", sr_var = "fs")

# Read EEGLAB format
pe <- readMAT("EEG.set", data_var = "EEG")

## End(Not run)
```

---

readPhysioHDF5 *Read PhysioExperiment from HDF5*

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

path	Path to the HDF5 file.
as_delayed	Logical. If TRUE (default), returns DelayedArray-backed assays. If FALSE, 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

```
## Not run:
# Read HDF5 file with DelayedArray backend (out-of-memory)
pe <- readPhysioHDF5("data.h5")
isHDF5Backed(pe)  # TRUE

# Read HDF5 file into memory
pe_mem <- readPhysioHDF5("data.h5", as_delayed = FALSE)
isHDF5Backed(pe_mem)  # FALSE

## End(Not run)
```

---

realizeHDF5 *Realize HDF5-backed data to memory*

Description

Loads HDF5-backed assays into memory as regular arrays.

Usage

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

Arguments

x	A PhysioExperiment object.
assays	Optional character vector of assay names to realize. If NULL, 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

```
## Not run:
# Load HDF5-backed data
pe <- readPhysioHDF5("data.h5")

# Realize all assays to memory
pe_mem <- realizeHDF5(pe)

# Realize only specific assays
pe_partial <- realizeHDF5(pe, assays = c("raw", "filtered"))

## End(Not run)
```

---

registerExperiment *Register a PhysioExperiment in the database*

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
)
```

Arguments

con	A DuckDB connection.
x	A PhysioExperiment 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.

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

```
## Not run:
con <- connectDatabase()
initPhysioSchema(con)

# Create sample data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(10000), nrow = 1000, ncol = 10)),
  samplingRate = 256
)

# Register experiment (metadata only)
exp_id <- registerExperiment(con, pe,
  subject_id = "sub01",
  task = "rest"
)

# Register with signal data
exp_id <- registerExperiment(con, pe,
  subject_id = "sub02",
  task = "oddball",
  store_signals = TRUE
)

disconnectDatabase(con)

## End(Not run)
```

---

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

```
## Not run:
# Validate a BIDS dataset
result <- validateBIDS("path/to/bids")
if (result$valid) {
  message("Dataset is BIDS-compliant")
} else {
  message("Validation errors: ", paste(result$errors, collapse = ", "))
}

## End(Not run)
```

---

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

```
## Not run:
# Add a new processed assay to an existing HDF5 file
pe <- readPhysioHDF5("data.h5")
pe <- butterworthFilter(pe, low = 1, high = 30, type = "pass")

# Write the filtered assay back to the HDF5 file
writeAssayHDF5(pe, "data.h5", "filtered")

## End(Not run)
```

---

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

x	A PhysioExperiment object.
bids_root	Path to the BIDS dataset root directory.
subject	Subject identifier (without 'sub-' prefix).
session	Session identifier (without 'ses-' prefix). Optional.
task	Task name.
run	Run number. Optional.
modality	Data modality: "eeg" or "ieeg".
overwrite	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

```
## Not run:
# Create a PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(25600), nrow = 2560, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

# Export to BIDS format
writeBIDS(pe, "path/to/bids", subject = "01", task = "rest")

# With session and run
writeBIDS(pe, "path/to/bids", subject = "01", session = "01",
          task = "oddball", run = 1)

## End(Not run)
```

writeCSV *Write PhysioExperiment to CSV file*

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 PhysioExperiment 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 write.csv/write.table.

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

x	A PhysioExperiment object.
path	Output file path.
patient_id	Patient identification string.
recording_id	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

```
## Not run:
# Create a PhysioExperiment with EEG-like data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(25600), nrow = 2560, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

# Export to EDF format
writeEDF(pe, "output.edf", patient_id = "Subject01", recording_id = "Session1")

## End(Not run)
```

---

writeElectrodePositionsCSV

Write electrode positions to CSV

Description

Writes electrode positions from a PhysioExperiment to CSV.

Usage

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

Arguments

x	A PhysioExperiment object with electrode positions.
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

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 *Write events to CSV file*

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)
```

---

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,
  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.
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

See Also

readMAT, writeCSV, writeEDF, writePhysioHDF5

Examples

```
## Not run:
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  colData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

# Write to .mat file
writeMAT(pe, "output.mat")

# Custom variable name
writeMAT(pe, "output.mat", data_var = "EEG_data")

## End(Not run)
```

---

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

x	A PhysioExperiment object.
path	Path to the output HDF5 file.
overwrite	Logical. If TRUE, overwrites existing file.
chunk_dims	Optional integer vector specifying chunk dimensions for HDF5 storage. If NULL, defaults are chosen automatically.
compression_level	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

```
## Not run:
# Create a large PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1e6), nrow = 1e5, ncol = 10)),
  samplingRate = 1000
)

# Save to HDF5 with compression
writePhysioHDF5(pe, "data.h5", compression_level = 6)

# Overwrite existing file
writePhysioHDF5(pe, "data.h5", overwrite = TRUE)

## End(Not run)
```

Index