Package: PhysioExperiment (via r-universe)

May 28, 2026

Title Unified Analysis of Physiological Signals

Version 1.0.0

Description A comprehensive R/Bioconductor framework for physiological signal analysis. Provides a unified data model (PhysioExperiment class extending SummarizedExperiment) for multi-modal sensor signals (EEG, EMG, ECG, IMU, MoCap). Includes file I/O (EDF, BDF, BrainVision, GDF, HDF5, BIDS, CSV, MATLAB), database integration (DuckDB), signal preprocessing (filtering, artifact removal, epoching), time-frequency analysis, connectivity analysis, network metrics, statistical testing (SPM1D, permutation tests), and publication-quality visualization.

Depends R (>= 4.2)

Imports methods, SummarizedExperiment, S4Vectors, DelayedArray, HDF5Array, rhdf5, DBI, duckdb, signal, abind, ggplot2, jsonlite, parallel, Rcpp (>= 1.0.0), RcppArmadillo

Suggests R.matlab, testthat (>= 3.2.0), knitr, rmarkdown, pkgdown, plumber (>= 1.2.0), later, callr, covr

biocViews Software, TimeCourse, Preprocessing

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/matsui-lab/PhysioExperiment

BugReports https://github.com/matsui-lab/PhysioExperiment/issues

Collate 'PhysioExperiment-class.R' 'PhysioExperiment-accessors.R' 'PhysioExperiment-methods.R' 'channels.R' 'events.R' 'event-query.R' 'utils-signal.R' 'utils-na.R' 'io-readwrite.R' 'io-edf.R' 'io-brainvision.R' 'io-gdf.R' 'io-hdf5.R'

'io-bids.R' 'io-csv.R' 'io-matlab.R' 'db-interface.R'
'db-schema.R' 'ops-filter.R' 'ops-filters-advanced.R'
'ops-fft.R' 'ops-rereference.R' 'ops-artifact.R' 'ops-epoch.R'
'ops-epoch-sliding.R' 'ops-resample.R' 'ops-timefreq.R'
'ops-connectivity.R' 'ops-network.R' 'stats-tests.R'
'stats-spm.R' 'vis-plot.R' 'vis-multichannel.R' 'vis-topomap.R'
'vis-network.R' 'gui-launcher.R' 'RcppExports.R' 'zzz.R'

Config/pak/sysreqs libssl-dev xz-utils zlib1g-dev

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

Date/Publication 2026-03-16 11:30:50 UTC

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

RemoteRef HEAD

RemoteSha a6d6c3eca2f1f3d077f194fd0711fe10095e4290

Contents

`[,PhysioExperiment,ANY,ANY,ANY-method`
Subset PhysioExperiment by time indices

Description

Subset PhysioExperiment by time indices

Usage

```
## S4 method for signature 'PhysioExperiment,ANY,ANY,ANY'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	A PhysioExperiment object.
`i`	Time indices (rows).
`j`	Channel indices (columns in first non-time dimension).
`...`	Additional arguments (not used).
`drop`	Logical. If TRUE, drops dimensions of size 1.

Value

A subsetted PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)

# Subset by time
pe_subset <- pe[1:50, ]
dim(pe_subset)  # 50 4

# Subset by channels
pe_channels <- pe[, 1:2]
dim(pe_channels)  # 100 2
```

addEvents *Add events to a PhysioExperiment object*

Description

Add events to a PhysioExperiment object

Usage

```
addEvents(x, onset, duration = 0, type = "event", value = "")
```

Arguments

x	A PhysioExperiment object.
onset	Numeric vector of event onset times in seconds.
duration	Numeric vector of event durations in seconds.
type	Character vector of event types.
value	Character vector of event values/labels.

Value

The modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100)),
  samplingRate = 100
)

# Add stimulus events
pe <- addEvents(pe, onset = c(1, 2, 3), type = "stimulus")

# Add response events
pe <- addEvents(pe, onset = c(1.5, 2.5), type = "response", value = c("hit", "hit"))
nEvents(pe)  # 5 events total
```

adjacencyMatrix *Network Analysis for PhysioExperiment*

Description

Functions for graph-theoretic analysis of functional connectivity networks, including network construction, topology measures, and spectral analysis. Create adjacency matrix from connectivity

Usage

```
adjacencyMatrix(connectivity, threshold = NULL, absolute = FALSE)
```

Arguments

connectivity	A connectivity matrix or result from connectivityMatrix().
threshold	Threshold value for binarization. If NULL, keeps weighted edges.
absolute	If TRUE, uses absolute values before thresholding.

Details

Converts a connectivity matrix into an adjacency matrix for network analysis.

Value

A square adjacency matrix.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500, ncol = 4)),
  samplingRate = 100
)
conn <- correlationMatrix(pe)
adj <- adjacencyMatrix(conn$correlation, threshold = 0.3)
```

anovaEpochs *ANOVA across conditions*

Description

Performs one-way ANOVA at each time point and channel across multiple conditions.

Usage

```
anovaEpochs(x, groups)
```

Arguments

x	An epoched PhysioExperiment object (4D data).
groups	Factor or character vector indicating group membership for each epoch. Can also be a column name from epoch_info metadata.

Value

A list containing:

f_values	Matrix of F-statistics (time x channel)
p_values	Matrix of p-values (time x channel)
df_between	Between-group degrees of freedom
df_within	Within-group degrees of freedom
group_means	Array of group means (time x channel x group)

Examples

```
# Create example epoched data with conditions
set.seed(123)
epochs <- array(rnorm(100 * 4 * 30 * 1), dim = c(100, 4, 30, 1))
pe <- PhysioExperiment(
  assays = list(epoched = epochs),
  samplingRate = 100,
  metadata = list(
    epoch_info = S4Vectors::DataFrame(
      condition = rep(c("A", "B", "C"), each = 10)
    )
  )
)
# ANOVA across conditions
result <- anovaEpochs(pe, groups = "condition")
```

---

applyMontage *Apply standard montage*

Description

Applies a standard electrode montage (e.g., 10-20 system).

Usage

```
applyMontage(x, system = c("10-20", "10-10", "10-5"))
```

Arguments

x	A PhysioExperiment object.
system	Montage system: "10-20", "10-10", or "10-5".

Value

Modified PhysioExperiment object with electrode positions.

Examples

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

# Apply 10-20 system positions
pe <- applyMmontage(pe, "10-20")
getElectrodePositions(pe)
```

---

as.data.frame,PhysioExperiment-method

Coerce to data.frame

Description

Converts the default assay to a data.frame.

Usage

```
## S4 method for signature 'PhysioExperiment'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A PhysioExperiment object.
row.names	Unused.
optional	Unused.
...	Additional arguments.

Value

A data.frame.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(12), nrow = 3, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
df <- as.data.frame(pe)
head(df)
```

assaySamplingRates *Get sampling rates for all assays*

Description

Returns sampling rates associated with each assay.

Usage

```
assaySamplingRates(x)
```

Arguments

x	A PhysioExperiment object.

Value

Named numeric vector of sampling rates.

averageEpochs *Average epochs*

Description

Computes the average across epochs, optionally by condition.

Usage

```
averageEpochs(x, by = NULL)
```

Arguments

x	An epoched PhysioExperiment object.
by	Optional column name in epoch_info to group by.

Value

A PhysioExperiment object with averaged epochs.

bandPower *Compute band power*

Description

Extracts power in specified frequency bands.

Usage

```
bandPower(x, bands = NULL, method = c("welch", "wavelet"), relative = FALSE)
```

Arguments

x	A PhysioExperiment object.
bands	Named list of frequency bands. Each element should be c(low, high). Default includes standard EEG bands.
method	Method: "welch" (PSD) or "wavelet".
relative	If TRUE, returns relative power (proportion of total).

Value

A data.frame with band powers for each channel.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(2560), nrow = 256, ncol = 10)),
  rowData = S4Vectors::DataFrame(label = paste0("Ch", 1:10)),
  samplingRate = 256
)

# Compute band power for standard EEG bands
bp <- bandPower(pe)
head(bp)

# Compute relative band power
bp_rel <- bandPower(pe, relative = TRUE)
```

baselineCorrect *Baseline correction*

Description

Subtracts baseline from epochs.

Usage

```
baselineCorrect(
  x,
  baseline = c(-0.2, 0),
  method = c("mean", "median"),
  output_assay = "baseline_corrected"
)
```

Arguments

x	An epoched PhysioExperiment object.
baseline	Numeric vector of length 2 (tmin, tmax) for baseline period.
method	Correction method: "mean" or "median".
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with baseline-corrected data.

betweennessCentrality *Compute betweenness centrality*

Description

Calculates betweenness centrality for each node.

Usage

```
betweennessCentrality(
  adjacency,
  normalized = TRUE,
  use_cpp = TRUE,
  n_cores = 1L
)
```

Arguments

adjacency	An adjacency matrix.
normalized	If TRUE, normalizes by (n-1)(n-2)/2.

Value

A numeric vector of betweenness centrality values.

Examples

```
# Star network - center node should have highest betweenness
adj <- matrix(0, 4, 4)
adj[1, 2:4] <- 1
adj[2:4, 1] <- 1
betweennessCentrality(adj)
```

---

binarizeNetwork *Binarize network*

Description

Converts a weighted adjacency matrix to a binary network.

Usage

```
binarizeNetwork(adjacency, threshold = 0)
```

Arguments

adjacency	A weighted adjacency matrix.
threshold	Threshold for binarization. Default 0 (any non-zero edge).

Value

A binary adjacency matrix (0s and 1s).

Examples

```
set.seed(123)
adj <- matrix(c(0, 0.5, 0.3, 0.5, 0, 0.8, 0.3, 0.8, 0), 3, 3)
bin <- binarizeNetwork(adj)
```

bootstrapCI *Bootstrap confidence interval for ERP*

Description

Computes bootstrap confidence intervals for averaged epochs.

Usage

```
bootstrapCI(
  x,
  n_bootstrap = 1000L,
  ci_level = 0.95,
  condition = NULL,
  seed = NULL
)
```

Arguments

x	An epoched PhysioExperiment object (4D data).
n_bootstrap	Number of bootstrap iterations (default: 1000).
ci_level	Confidence interval level (default: 0.95).
condition	Epoch indices to include. If NULL, uses all epochs.
seed	Random seed for reproducibility.

Value

A list containing:

mean	Mean across epochs (time x channel)
ci_lower	Lower CI bound (time x channel)
ci_upper	Upper CI bound (time x channel)
se	Standard error (time x channel)

Examples

```
# Create example epoched data
set.seed(123)
epochs <- array(rnorm(100 * 4 * 20 * 1), dim = c(100, 4, 20, 1))
pe <- PhysioExperiment(
  assays = list(epoched = epochs),
  samplingRate = 100
)
# Bootstrap CI
result <- bootstrapCI(pe, n_bootstrap = 500)
```

`butterworthFilter` *Advanced signal filtering functions*

Description

This file provides advanced filtering operations including Butterworth, FIR, and notch filters for physiological signal processing. Butterworth filter

Usage

```
butterworthFilter(
  x,
  low = NULL,
  high = NULL,
  order = 4L,
  type = c("pass", "low", "high", "stop"),
  output_assay = "filtered"
)
```

Arguments

`x`	A `PhysioExperiment` object.
`low`	Lower cutoff frequency in Hz. Required for highpass and bandpass.
`high`	Upper cutoff frequency in Hz. Required for lowpass and bandpass.
`order`	Filter order. Default is 4.
`type`	Filter type: "low", "high", "pass" (bandpass), or "stop" (bandstop).
`output_assay`	Name for the output assay. Default is "filtered".

Details

Applies a Butterworth filter (lowpass, highpass, bandpass, or bandstop) along the time axis of the specified assay.

Value

The input object with a new assay containing filtered data.

Examples

```
# Create example EEG data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 250
)

# Bandpass filter (1-40 Hz) - common for EEG
pe <- butterworthFilter(pe, low = 1, high = 40, type = "pass")
```

```
# Lowpass filter (30 Hz)
pe <- butterworthFilter(pe, high = 30, type = "low",
                        output_assay = "lowpass")

# Highpass filter (0.5 Hz) to remove DC drift
pe <- butterworthFilter(pe, low = 0.5, type = "high",
                        output_assay = "highpass")
```

---

cbindPhysio *Combine PhysioExperiment objects by channels*

Description

Combines two PhysioExperiment objects by adding channels.

Usage

```
cbindPhysio(x, y)
```

Arguments

x	A PhysioExperiment object.
y	A PhysioExperiment object to combine.

Value

Combined PhysioExperiment object.

Examples

```
pe1 <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(200), nrow = 100, ncol = 2)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz")),
  samplingRate = 100
)
pe2 <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(200), nrow = 100, ncol = 2)),
  colData = S4Vectors::DataFrame(label = c("Pz", "Oz")),
  samplingRate = 100
)

# Combine channels
pe_combined <- cbindPhysio(pe1, pe2)
nChannels(pe_combined)  # 4
```

channelInfo *Channel information management for PhysioExperiment*

Description

Functions for managing channel metadata including labels, types, units, and electrode positions. Get channel information

Usage

```
channelInfo(x)
```

Arguments

x	A PhysioExperiment object.

Details

Returns channel metadata as a DataFrame. Channel information is stored in colData (columns = channels).

Value

A DataFrame with channel information.

Examples

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

# Get channel information
channelInfo(pe)

# Get channel names
channelNames(pe)

# Get number of channels
nChannels(pe)
```

channelInfo<- *Set channel information*

Description

Updates channel metadata. Channel information is stored in colData (columns = channels).

Usage

```
channelInfo(x) <- value
```

Arguments

x	A PhysioExperiment object.
value	A DataFrame with channel information.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)

# Set channel info
channelInfo(pe) <- S4Vectors::DataFrame(
  label = c("Fz", "Cz", "Pz", "Oz"),
  type = rep("EEG", 4)
)
```

channelNames *Get channel names/labels*

Description

Get channel names/labels

Usage

```
channelNames(x)
```

Arguments

x	A PhysioExperiment object.

Value

Character vector of channel names.

Examples

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

---

channelNames<- *Set channel names/labels*

Description

Set channel names/labels

Usage

```
channelNames(x) <- value
```

Arguments

x	A PhysioExperiment object.
value	Character vector of channel names.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
channelNames(pe) <- c("Fz", "Cz", "Pz", "Oz")
channelNames(pe)
```

checkGUIDependencies *Check GUI Dependencies*

Description

Checks if all required dependencies for the GUI are available.

Usage

```
checkGUIDependencies()
```

Value

A list with the status of each dependency.

Examples

```
## Not run:
checkGUIDependencies()

## End(Not run)
```

checkNA *Check for NA values in assay data*

Description

Validates assay data for NA values and reports statistics.

Usage

```
checkNA(x, action = c("warn", "error", "none"))
```

Arguments

x	A PhysioExperiment object or numeric array.
action	Action to take: "warn" (default), "error", or "none".

Value

Invisibly returns a list with NA statistics.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(c(1, NA, 3, 4), nrow = 2)),
  samplingRate = 100
)
checkNA(pe)
```

`classifyICAComponents` *Classify ICA components as brain or artifact*

Description

Automatically identifies artifact ICA components using statistical criteria.

Usage

```
classifyICAComponents(
  x,
  method = c("autocorrelation", "kurtosis", "frequency"),
  eog_channels = NULL,
  threshold = NULL
)
```

Arguments

`x`	A PhysioExperiment object with ICA decomposition (from `icaDecompose`).
`method`	Classification method: "autocorrelation", "kurtosis", or "frequency".
`eog_channels`	Integer vector of EOG channel indices for correlation-based detection. If provided, also computes EOG correlation scores.
`threshold`	Classification threshold. If NULL, uses method-specific defaults: autocorrelation=0.9, kurtosis=3.0, frequency=0.5.

Value

A list with:

- `artifact_indices`: Integer vector of artifact component indices
- `scores`: Named numeric vector of scores per component
- `threshold`: The threshold used
- `method`: The method used

`cleanData` *Clean data using an artifact removal pipeline*

Description

Runs a configurable sequence of artifact removal steps.

Usage

```
cleanData(
  x,
  steps = c("bad_channels", "ica"),
  bad_channel_method = "zscore",
  bad_channel_threshold = NULL,
  interpolation_method = "average",
  ica_method = "fastica",
  ica_classify_method = "kurtosis",
  output_assay = "cleaned"
)
```

Arguments

`x`	A PhysioExperiment object.
`steps`	Character vector of steps to run. Options: "bad_channels", "ica".
`bad_channel_method`	Method for `detectBadChannels`.
`bad_channel_threshold`	Threshold for bad channel detection.
`interpolation_method`	Method for `interpolateBadChannels`.
`ica_method`	Method for `icaDecompose`.
`ica_classify_method`	Method for `classifyICAComponents`.
`output_assay`	Name for the output assay.

Value

Modified PhysioExperiment with cleaned data.

clusteringCoefficient *Compute clustering coefficient*

Description

Calculates the local clustering coefficient for each node.

Usage

```
clusteringCoefficient(
  adjacency,
  weighted = FALSE,
  use_cpp = TRUE,
  n_cores = 1L
)
```

Arguments

adjacency	An adjacency matrix.
weighted	If TRUE, uses weighted clustering coefficient.

Value

A numeric vector of clustering coefficients.

Examples

```
# Fully connected triangle
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
clusteringCoefficient(adj)  # Should be 1 for all nodes
```

clusterPermutationTest *Cluster-based permutation test*

Description

Performs cluster-based permutation testing for multiple comparison correction. Identifies clusters of significant effects and computes cluster-level p-values.

Usage

```
clusterPermutationTest(
  x,
  condition1 = NULL,
  condition2 = NULL,
  n_permutations = 1000L,
  cluster_threshold = 0.05,
  tail = 0L,
  seed = NULL,
  n_cores = NULL
)
```

Arguments

x	An epoched PhysioExperiment object (4D data).
condition1	Indices for first condition.
condition2	Indices for second condition. If NULL, tests against zero.
n_permutations	Number of permutations (default: 1000).
cluster_threshold	Initial threshold for cluster formation (p-value).
tail	Test type: 0 = two-tailed, 1 = right tail, -1 = left tail.
seed	Random seed for reproducibility.
n_cores	Number of cores for parallel processing. Default NULL uses sequential processing. Set to parallel::detectCores() - 1 for maximum speed.

Value

A list containing:

clusters	List of identified clusters with their statistics
cluster_p	P-values for each cluster
t_obs	Observed t-values matrix
cluster_mask	Logical matrix indicating cluster membership

Examples

```
# Create example epoched data
set.seed(123)
epochs <- array(rnorm(50 * 4 * 20 * 1), dim = c(50, 4, 20, 1))
# Add effect to condition 2
epochs[20:30, 1:2, 11:20, 1] <- epochs[20:30, 1:2, 11:20, 1] + 1
pe <- PhysioExperiment(
  assays = list(epoched = epochs),
  samplingRate = 100
)
# Cluster permutation test
result <- clusterPermutationTest(pe, 1:10, 11:20, n_permutations = 100)
```

```
# With parallel processing (faster for large datasets)
# result <- clusterPermutationTest(pe, 1:10, 11:20, n_cores = 4)
```

---

coherence *Connectivity Analysis for PhysioExperiment*

Description

Functions for computing functional connectivity between channels including coherence, phase synchrony measures, and correlation-based metrics. Compute coherence between channels

Usage

```
coherence(
  x,
  channels = NULL,
  freq_range = NULL,
  nperseg = 256L,
  noverlap = NULL,
  assay_name = NULL
)
```

Arguments

x	A PhysioExperiment object.
channels	Integer vector of channel indices to analyze. If NULL, uses all.
freq_range	Numeric vector of length 2 specifying frequency range (Hz).
nperseg	Number of samples per segment for Welch's method. Default is 256.
noverlap	Number of overlapping samples. Default is nperseg/2.
assay_name	Input assay name. If NULL, uses default assay.

Details

Calculates the magnitude-squared coherence between pairs of channels, which measures the linear correlation between signals as a function of frequency.

Coherence is computed using Welch's averaged periodogram method. Values range from 0 (no linear relationship) to 1 (perfect linear relationship).

Value

A list with components:

coherence	3D array (freq x channel x channel) of coherence values
frequencies	Frequency vector
channel_names	Channel names

Examples

```
# Create example with 4 channels
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 256
)

# Compute coherence
coh <- coherence(pe, freq_range = c(1, 50))
dim(coh$coherence)  # frequencies x channels x channels
```

---

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.

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)
```

`connectivityMatrix` *Compute connectivity matrix for a frequency band*

Description

High-level function to compute connectivity using various metrics.

Usage

```
connectivityMatrix(
  x,
  method = c("coherence", "plv", "pli", "wpli", "correlation"),
  freq_band = NULL,
  channels = NULL,
  assay_name = NULL
)
```

Arguments

`x`	A PhysioExperiment object.
`method`	Connectivity method: "coherence", "plv", "pli", "wpli", "correlation".
`freq_band`	Frequency band for phase-based methods.
`channels`	Integer vector of channel indices.
`assay_name`	Input assay name.

Value

A connectivity matrix or array depending on the method.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  samplingRate = 256
)

# Compute PLV connectivity in alpha band
conn <- connectivityMatrix(pe, method = "plv", freq_band = c(8, 12))
```

correctPValues *Multiple comparison correction*

Description

Applies multiple comparison correction to p-values.

Usage

```
correctPValues(p_values, method = c("fdr", "bonferroni", "holm", "bh", "none"))
```

Arguments

p_values	Matrix or vector of p-values.
method	Correction method: "bonferroni", "holm", "fdr" (Benjamini-Hochberg), "bh" (alias for fdr), or "none".

Value

Corrected p-values in the same format as input.

Examples

```
# Example p-values
p <- matrix(runif(100), nrow = 10)
# Apply FDR correction
p_corrected <- correctPValues(p, method = "fdr")
```

correlationMatrix *Compute correlation matrix between channels*

Description

Calculates the Pearson correlation coefficient between all channel pairs.

Usage

```
correlationMatrix(
  x,
  channels = NULL,
  method = c("pearson", "spearman", "kendall"),
  assay_name = NULL
)
```

Arguments

x	A PhysioExperiment object.
channels	Integer vector of channel indices.
method	Correlation method: "pearson", "spearman", or "kendall".
assay_name	Input assay name.

Value

A correlation matrix (channel x channel).

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 256
)

# Compute Pearson correlation
cor_matrix <- correlationMatrix(pe)
```

---

crossSpectrum *Compute cross-spectral density*

Description

Calculates the cross-spectral density between channel pairs.

Usage

```
crossSpectrum(
  x,
  channels = NULL,
  nperseg = 256L,
  noverlap = NULL,
  assay_name = NULL
)
```

Arguments

x	A PhysioExperiment object.
channels	Integer vector of channel indices. If NULL, uses all.
nperseg	Number of samples per segment. Default is 256.
noverlap	Number of overlapping samples.
assay_name	Input assay name.

Value

A list with components:

csd	3D complex array (freq x channel x channel)
frequencies	Frequency vector
channel_names	Channel names

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(2000), nrow = 500, ncol = 4)),
  samplingRate = 256
)

# Compute cross-spectral density
csd <- crossSpectrum(pe)
```

---

dbStats *Get database statistics*

Description

Get database statistics

Usage

```
dbStats(con)
```

Arguments

con	A DuckDB connection.

Value

A list with database statistics.

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)
```

`decimate` *Decimate signal*

Description

Downsamples by an integer factor with anti-aliasing filter.

Usage

```
decimate(x, factor, filter_order = 8L, output_assay = "decimated")
```

Arguments

`x`	A PhysioExperiment object.
`factor`	Integer decimation factor.
`filter_order`	Order of the anti-aliasing lowpass filter.
`output_assay`	Name for the output assay.

Value

A new PhysioExperiment with decimated data.

`defaultAssay` *Retrieve the default assay name*

Description

Retrieve the default assay name

Usage

```
defaultAssay(x)
```

Arguments

`x`	A `PhysioExperiment` instance.

Value

Character scalar naming the first assay or `NA_character_` when absent.

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.

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)
```

detectArtifacts *Detect artifacts in continuous data*

Description

Scans continuous data for artifacts using amplitude or gradient criteria.

Usage

```
detectArtifacts(
  x,
  method = c("amplitude", "gradient", "joint"),
  threshold = NULL,
  window_sec = 1
)
```

Arguments

x	A PhysioExperiment object.
method	Detection method: "amplitude", "gradient", or "joint".
threshold	Detection threshold. If NULL, computed as median + 5 * MAD.
window_sec	Detection window in seconds (default: 1.0).

Value

A data.frame with columns: onset (seconds), offset (seconds), channel (index), type (detection method).

---

detectBadChannels *Detect bad channels*

Description

Identifies channels with abnormal characteristics.

Usage

```
detectBadChannels(
  x,
  method = c("zscore", "correlation", "flatline"),
  threshold = NULL
)
```

Arguments

x	A PhysioExperiment object.
method	Detection method: "zscore", "correlation", or "flatline".
threshold	Threshold for detection (depends on method).

Value

Integer vector of bad channel indices.

detrendSignal *Detrend signal*

Description

Removes linear or polynomial trends from the signal.

Usage

```
detrendSignal(x, type = c("linear", "constant"), output_assay = "detrended")
```

Arguments

x	A PhysioExperiment object.
type	Type of detrending: "linear" or "constant" (mean removal).
output_assay	Name for the output assay. Default is "detrended".

Value

The input object with a new assay containing detrended data.

dim,PhysioExperiment-method *Dim method for PhysioExperiment*

Description

Returns dimensions of the default assay.

Usage

```
## S4 method for signature 'PhysioExperiment'
dim(x)
```

Arguments

x	A PhysioExperiment object.

Value

Integer vector of dimensions.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
dim(pe)  # 100 4
```

`dropChannels` *Drop channels*

Description

Creates a new PhysioExperiment without specified channels.

Usage

```
dropChannels(x, channels)
```

Arguments

`x`	A PhysioExperiment object.
`channels`	Integer indices or character names of channels to drop.

Value

A new PhysioExperiment without specified channels.

Examples

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

# Drop by index
pe_dropped <- dropChannels(pe, 1)
nChannels(pe_dropped)  # 3

# Drop by name
pe_dropped <- dropChannels(pe, c("Fz", "Oz"))
```

duration *Get signal duration*

Description

Get signal duration

Usage

```
duration(x)
```

Arguments

x	A PhysioExperiment object.

Value

Duration in seconds.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 1000, ncol = 4)),
  samplingRate = 100
)
duration(pe)  # 10 seconds
```

effectSize *Compute effect size (Cohen's d)*

Description

Calculates Cohen's d effect size at each time point and channel.

Usage

```
effectSize(x, condition1 = NULL, condition2 = NULL, pooled = TRUE)
```

Arguments

x	An epoched PhysioExperiment object (4D data).
condition1	Indices for first condition.
condition2	Indices for second condition. If NULL, computes d against zero.
pooled	If TRUE (default for two-sample), uses pooled standard deviation.

Value

A list containing:

d	Matrix of Cohen's d values (time x channel)
ci_lower	Lower 95% CI for d
ci_upper	Upper 95% CI for d

Examples

```
# Create example epoched data
set.seed(123)
epochs <- array(rnorm(100 * 4 * 20 * 1), dim = c(100, 4, 20, 1))
pe <- PhysioExperiment(
  assays = list(epoched = epochs),
  samplingRate = 100
)
# Effect size for one-sample
result <- effectSize(pe, condition1 = 1:10)
# Effect size between conditions
result2 <- effectSize(pe, condition1 = 1:10, condition2 = 11:20)
```

---

eigenvectorCentrality *Compute eigenvector centrality*

Description

Calculates eigenvector centrality for each node.

Usage

```
eigenvectorCentrality(adjacency, max_iter = 100, tol = 1e-06, use_cpp = TRUE)
```

Arguments

adjacency	An adjacency matrix.
max_iter	Maximum iterations for power method.
tol	Convergence tolerance.

Value

A numeric vector of eigenvector centrality values.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
eigenvectorCentrality(adj)
```

epochData *Epoching operations for PhysioExperiment*

Description

Functions for segmenting continuous data into epochs/trials based on events. Epoch data around events

Usage

```
epochData(
  x,
  tmin = -0.2,
  tmax = 0.8,
  event_type = NULL,
  baseline = NULL,
  reject = NULL,
  events = NULL,
  min_length = NULL
)
```

Arguments

x	A PhysioExperiment object.
tmin	Time before event onset in seconds (negative for pre-stimulus).
tmax	Time after event onset in seconds.
event_type	Character vector of event types to epoch around. If NULL, uses all events. Ignored if events is provided.
baseline	Numeric vector of length 2 specifying baseline period (tmin, tmax) for baseline correction. NULL for no correction.
reject	Amplitude threshold for epoch rejection. NULL to keep all.
events	An EventQuery object for advanced event filtering. If provided, overrides event_type.
min_length	Minimum epoch length in seconds when using variable-length epochs (tmax as event name). Epochs shorter than this are excluded.

Details

Extracts epochs (segments) of data around specified events.

Value

A new PhysioExperiment object with epoched data. The assay becomes 4D: (time x channel x epoch x sample).

Examples

```
# Create continuous data with events
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 100
)
pe <- addEvents(pe, onset = c(1, 2, 3, 4, 5), type = "stimulus")

# Extract epochs: 200ms before to 800ms after stimulus
epochs <- epochData(pe, tmin = -0.2, tmax = 0.8)

# With baseline correction
epochs_bl <- epochData(pe, tmin = -0.2, tmax = 0.8,
                       baseline = c(-0.2, 0))

# With artifact rejection
epochs_clean <- epochData(pe, tmin = -0.2, tmax = 0.8,
                          reject = 100)
```

---

epochSliding *Create epochs using sliding window*

Description

Segments continuous data into overlapping epochs using a sliding window approach. This is useful for time-frequency analysis or when events are not available.

Usage

```
epochSliding(x, window, step = NULL, baseline = NULL)
```

Arguments

x	A PhysioExperiment object
window	Window size in seconds
step	Step size in seconds (default: window/2 for 50% overlap)
baseline	Optional baseline correction window as c(start, end) in seconds relative to epoch start. NULL for no correction.

Value

PhysioExperiment with epoched data

Examples

```
pe <- make_pe_2d(n_time = 1000, n_channels = 4, sr = 100)
# Create 0.5 second windows with 0.1 second step
epoched <- epochSliding(pe, window = 0.5, step = 0.1)
```

epochTimes *Get epoch time vector*

Description

Returns the time vector for epoched data relative to event onset.

Usage

```
epochTimes(x)
```

Arguments

x	An epoched PhysioExperiment object.

Value

Numeric vector of times in seconds.

eventQuery *Create an EventQuery from a PhysioExperiment*

Description

Create an EventQuery from a PhysioExperiment

Usage

```
eventQuery(x)
```

Arguments

x	A PhysioExperiment object

Value

An EventQuery object

Examples

```
pe <- make_pe_2d()
pe <- addEvents(pe, onset = c(1, 2, 3), type = "stimulus")
q <- eventQuery(pe)
```

EventQuery-class *EventQuery class for composable event filtering*

Description

EventQuery class for composable event filtering

Slots

`source` PhysioExperiment object

`filters` List of filter functions to apply

`resolved` Cached resolved events (NULL if not yet resolved)

extractWindow *Extract time window*

Description

Extracts a time window from the signal.

Usage

```
extractWindow(x, tmin, tmax)
```

Arguments

`x`	A PhysioExperiment object.
`tmin`	Start time in seconds.
`tmax`	End time in seconds.

Value

A PhysioExperiment with the extracted time window.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 1000, ncol = 4)),
  samplingRate = 100
)

# Extract 2 to 5 seconds
pe_window <- extractWindow(pe, tmin = 2, tmax = 5)
duration(pe_window)  # approximately 3 seconds
```

fftSignals *Fast Fourier transform helper*

Description

Computes the discrete Fourier transform along the time axis of the default assay and stores the magnitude spectrum in a new assay named `"fft"`.

Usage

```
fftSignals(x)
```

Arguments

`x`	A `PhysioExperiment` object.

Value

The modified object containing an FFT assay.

fillEdgeNA *Fill NA values at edges*

Description

Fills NA values at the beginning and end of a signal that may result from filtering operations.

Usage

```
fillEdgeNA(x, method = c("extend", "zero"))
```

Arguments

`x`	Numeric vector.
`method`	Fill method: "extend" (extend nearest valid value) or "zero" (fill with zeros).

Value

Vector with edge NA values filled.

Examples

```
x <- c(NA, NA, 1, 2, 3, NA, NA)
fillEdgeNA(x, method = "extend")
```

filterSignals *Moving average filter*

Description

Applies a moving average filter along the first dimension (time axis) of the default assay.

Usage

```
filterSignals(x, window = 5L, na.rm = FALSE, output_assay = "filtered")
```

Arguments

x	A PhysioExperiment object.
window	Integer window length for the moving average.
na.rm	Logical. If TRUE, NA values are ignored in the filter computation.
output_assay	Name for the output assay. Default is "filtered".

Value

The input object with a new assay containing filtered data.

filterType *Filter events by type*

Description

Filter events by type

Usage

```
filterType(q, types)
```

Arguments

q	An EventQuery object
types	Character vector of event types to keep

Value

Modified EventQuery

filterValue *Filter events by value*

Description

Filter events by value

Usage

```
filterValue(q, values)
```

Arguments

q	An EventQuery object
values	Character vector of event values to keep

Value

Modified EventQuery

findSignificantWindows *Find significant time windows*

Description

Identifies contiguous time periods with significant effects.

Usage

```
findSignificantWindows(p_values, times = NULL, alpha = 0.05, min_duration = 0)
```

Arguments

p_values	Vector of p-values across time.
times	Vector of time points.
alpha	Significance threshold (default: 0.05).
min_duration	Minimum duration of significant window in time units.

Value

A data.frame with columns: start, end, duration, min_p.

Examples

```
# Example p-values
times <- seq(-0.2, 0.8, length.out = 100)
p <- c(rep(0.5, 30), rep(0.01, 20), rep(0.5, 50))
windows <- findSignificantWindows(p, times)
```

`firFilter` *FIR filter*

Description

Applies a Finite Impulse Response (FIR) filter along the time axis.

Usage

```
firFilter(
  x,
  low = NULL,
  high = NULL,
  order = 100L,
  type = c("pass", "low", "high", "stop"),
  window = "hamming",
  output_assay = "filtered"
)
```

Arguments

`x`	A `PhysioExperiment` object.
`low`	Lower cutoff frequency in Hz.
`high`	Upper cutoff frequency in Hz.
`order`	Filter order (number of taps - 1). Default is 100.
`type`	Filter type: "low", "high", "pass" (bandpass), or "stop" (bandstop).
`window`	Window function for FIR design. Default is "hamming".
`output_assay`	Name for the output assay. Default is "filtered".

Value

The input object with a new assay containing filtered data.

getChannelsByType *Get channels by type*

Description

Returns indices of channels matching specified types.

Usage

```
getChannelsByType(x, types)
```

Arguments

x	A PhysioExperiment object.
types	Character vector of channel types to match.

Value

Integer vector of matching channel indices.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(
    label = c("Fz", "EOG1", "EMG1", "Oz"),
    type = c("EEG", "EOG", "EMG", "EEG")
  ),
  samplingRate = 100
)

# Get EEG channels
eeg_idx <- getChannelsByType(pe, "EEG")  # c(1, 4)
```

getCurrentReference *Get current reference*

Description

Returns the current reference electrode information.

Usage

```
getCurrentReference(x)
```

Arguments

x	A PhysioExperiment object.

Value

Character string describing the current reference, or NULL if not set.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100), nrow = 10, ncol = 10)),
  samplingRate = 100
)
pe <- setReference(pe, "Cz")
getCurrentReference(pe)  # "Cz"
```

---

getElectrodePositions *Get electrode positions*

Description

Get electrode positions

Usage

```
getElectrodePositions(x)
```

Arguments

x	A PhysioExperiment object.

Value

A data.frame with x, y, z columns or NULL if not set.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(300), nrow = 100, ncol = 3)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz")),
  samplingRate = 100
)
pe <- applyMontage(pe, "10-20")
getElectrodePositions(pe)
```

getEvents *Get events from a PhysioExperiment object*

Description

Get events from a PhysioExperiment object

Usage

```
getEvents(x, type = NULL)
```

Arguments

x	A PhysioExperiment object.
type	Optional character vector of event types to filter.

Value

A PhysioEvents object or DataFrame of events.

Examples

```
# Create PhysioExperiment with events
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100)),
  samplingRate = 100
)
events <- PhysioEvents(
  onset = c(1, 2, 3),
  type = c("stimulus", "response", "stimulus")
)
pe <- setEvents(pe, events)

# Get all events
getEvents(pe)

# Get only stimulus events
getEvents(pe, type = "stimulus")
```

getReference *Get reference electrode*

Description

Get reference electrode

Usage

```
getReference(x)
```

Arguments

x	A PhysioExperiment object.

Value

Character string of reference electrode or NULL.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
pe <- setReference(pe, "Cz")
getReference(pe)  # "Cz"
```

globalEfficiency *Compute global efficiency*

Description

Calculates the global efficiency of a network.

Usage

```
globalEfficiency(adjacency, weighted = FALSE, use_cpp = TRUE)
```

Arguments

adjacency	An adjacency matrix.
weighted	If TRUE, uses weighted paths.

Value

Global efficiency value (0-1).

Examples

```
# Fully connected - maximum efficiency
adj <- matrix(1, 4, 4)
diag(adj) <- 0
globalEfficiency(adj)
```

---

grandAverage *Grand average across subjects/samples*

Description

Computes grand average across multiple PhysioExperiment objects.

Usage

```
grandAverage(...)
```

Arguments

...	PhysioExperiment objects or a list of them.

Value

A PhysioExperiment object with grand averaged data.

---

graphLaplacian *Compute graph Laplacian*

Description

Calculates the graph Laplacian matrix.

Usage

```
graphLaplacian(adjacency, normalized = FALSE)
```

Arguments

adjacency	An adjacency matrix.
normalized	If TRUE, returns normalized Laplacian.

Value

The Laplacian matrix.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
L <- graphLaplacian(adj)
```

handleNA *Handle NA values in signal data*

Description

Provides various strategies for handling NA values in signal data.

Usage

```
handleNA(
  x,
  method = c("interpolate", "omit", "zero", "mean", "locf", "none"),
  ...
)
```

Arguments

x	Numeric vector or matrix.
method	Method for handling NA: "omit" (remove), "interpolate" (linear), "zero" (replace with 0), "mean" (replace with mean), "locf" (last observation carried forward), or "none" (no action).
...	Additional arguments passed to interpolation methods.

Value

Data with NA values handled according to the specified method.

Examples

```
x <- c(1, NA, 3, NA, 5)

# Linear interpolation
handleNA(x, method = "interpolate")

# Replace with mean
handleNA(x, method = "mean")

# Last observation carried forward
handleNA(x, method = "locf")
```

hasNA *Check if data contains any NA values*

Description

Quick check for NA presence in PhysioExperiment data.

Usage

```
hasNA(x, assay_name = NULL)
```

Arguments

x	A PhysioExperiment object.
assay_name	Optional specific assay to check.

Value

Logical indicating presence of NA values.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(1:4, nrow = 2)),
  samplingRate = 100
)
hasNA(pe)
```

hilbertTransform *Hilbert transform for instantaneous amplitude/phase*

Description

Computes the analytic signal using the Hilbert transform. The analytic signal can be used to extract instantaneous amplitude and phase.

Usage

```
hilbertTransform(x, output_assay = "analytic")
```

Arguments

x	A PhysioExperiment object.
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with analytic signal assay.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500)),
  samplingRate = 100
)

# Compute Hilbert transform
pe <- hilbertTransform(pe)

# Extract amplitude and phase
pe <- instantaneousAmplitude(pe)
pe <- instantaneousPhase(pe)
```

---

icaDecompose *ICA and Artifact Removal for PhysioExperiment*

Description

Functions for Independent Component Analysis (ICA) and artifact removal from physiological signals. Perform ICA decomposition

Usage

```
icaDecompose(
  x,
  n_components = NULL,
  method = c("fastica", "jade"),
  max_iter = 200L,
  tol = 1e-04
)
```

Arguments

x	A PhysioExperiment object.
n_components	Number of components to extract. If NULL, uses number of channels.
method	ICA method: "fastica" (default) or "jade".
max_iter	Maximum iterations for convergence.
tol	Tolerance for convergence.

Details

Decomposes the signal into independent components using FastICA algorithm.

Value

A list containing:

- components: The independent components (time x component matrix)
- mixing: The mixing matrix (channel x component)
- unmixing: The unmixing matrix (component x channel)
- object: Modified PhysioExperiment with ICA components as assay

---

icaRemove *Remove ICA components*

Description

Removes specified ICA components and reconstructs the signal.

Usage

```
icaRemove(x, components, output_assay = "ica_cleaned")
```

Arguments

x	A PhysioExperiment object with ICA decomposition.
components	Integer vector of component indices to remove.
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with cleaned signal.

---

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.

Examples

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

# Check statistics
dbStats(con)

disconnectDatabase(con)

## End(Not run)
```

---

instantaneousAmplitude

Extract instantaneous amplitude (envelope)

Description

Extracts the instantaneous amplitude (envelope) from the analytic signal.

Usage

```
instantaneousAmplitude(x, assay_name = "analytic", output_assay = "amplitude")
```

Arguments

x	A PhysioExperiment object with analytic signal.
assay_name	Name of the analytic signal assay.
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with amplitude assay.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500)),
  samplingRate = 100
)

# First compute Hilbert transform, then extract amplitude
pe <- hilbertTransform(pe)
pe <- instantaneousAmplitude(pe)
```

---

instantaneousPhase *Extract instantaneous phase*

Description

Extracts the instantaneous phase from the analytic signal.

Usage

```
instantaneousPhase(x, assay_name = "analytic", output_assay = "phase")
```

Arguments

x	A PhysioExperiment object with analytic signal.
assay_name	Name of the analytic signal assay.
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with phase assay.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500)),
  samplingRate = 100
)

# First compute Hilbert transform, then extract phase
pe <- hilbertTransform(pe)
pe <- instantaneousPhase(pe)
```

interpolate *Interpolate signal*

Description

Upsamples by an integer factor with interpolation.

Usage

```
interpolate(
  x,
  factor,
  method = c("linear", "spline"),
  output_assay = "interpolated"
)
```

Arguments

x	A PhysioExperiment object.
factor	Integer interpolation factor.
method	Interpolation method: "linear" or "spline".
output_assay	Name for the output assay.

Value

A new PhysioExperiment with interpolated data.

interpolateBadChannels *Interpolate bad channels*

Description

Replaces bad channels with interpolated values from neighboring channels.

Usage

```
interpolateBadChannels(
  x,
  bad_channels,
  method = c("average", "spline"),
  output_assay = "interpolated"
)
```

Arguments

x	A PhysioExperiment object.
bad_channels	Integer vector of channel indices to interpolate.
method	Interpolation method: "average" or "spline".
output_assay	Name for the output assay.

Value

Modified PhysioExperiment with interpolated channels.

---

isAverageReferenced *Check if data is average referenced*

Description

Check if data is average referenced

Usage

```
isAverageReferenced(x)
```

Arguments

x	A PhysioExperiment object.

Value

Logical indicating if data is average referenced.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100), nrow = 10, ncol = 10)),
  samplingRate = 100
)
pe <- rereference(pe, ref_type = "average")
isAverageReferenced(pe)  # TRUE
```

isHDF5Backed *S4 Methods for PhysioExperiment*

Description

Standard S4 methods for PhysioExperiment objects including show, subsetting, and combining. Check if object is HDF5-backed

Usage

```
isHDF5Backed(x)

isHDF5Backed(x)
```

Arguments

x	A PhysioExperiment object.

Details

Returns FALSE by default in PhysioCore. PhysioIO provides the full implementation.

Value

Logical indicating if backed by HDF5.

Logical indicating if assays are HDF5-backed DelayedArrays.

Examples

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

launchGUI *Launch PhysioExperiment GUI*

Description

Starts the PhysioExperiment graphical user interface. This launches a local web server with the Plumber API backend and opens the React-based GUI in either a browser or Electron desktop application.

Usage

```
launchGUI(
  port = 8000L,
  host = "127.0.0.1",
  desktop = FALSE,
  browser = TRUE,
  quiet = FALSE
)
```

Arguments

port	Integer. Port number for the API server. Default is 8000.
host	Character. Host address to bind to. Default is "127.0.0.1" for local access only. Use "0.0.0.0" to allow external access.
desktop	Logical. If TRUE, attempts to launch the Electron desktop application. If FALSE (default), opens in the default web browser.
browser	Logical. If TRUE (default), automatically opens the GUI in the browser. Set to FALSE to start only the API server.
quiet	Logical. If TRUE, suppresses startup messages. Default is FALSE.

Details

The GUI provides a modern, responsive interface for:

- Data import and management (EDF, BDF, BrainVision, HDF5, CSV)
- Signal visualization with multi-channel display
- Preprocessing (filtering, referencing, resampling)
- Time-frequency analysis (spectrograms, wavelets)
- Connectivity analysis (coherence, PLV, PLI)
- Statistical testing (t-test, ANOVA, cluster permutation)
- Workflow building for batch processing

The GUI communicates with R through a REST API built on plumber. All computations are performed in R using the PhysioExperiment package.

Value

Invisibly returns the plumber object. The function blocks while the server is running unless run in background mode.

Desktop Mode

Desktop mode requires Electron to be installed. The Electron application provides native file access and a more integrated desktop experience. If Electron is not available, the function falls back to browser mode.

Requirements

The following packages are required and will be loaded automatically:

- plumber - for the REST API server
- jsonlite - for JSON serialization

Examples

```
## Not run:
# Launch GUI in browser
launchGUI()

# Launch on a different port
launchGUI(port = 3000)

# Start API server without opening browser
launchGUI(browser = FALSE)

# Launch in desktop mode (requires Electron)
launchGUI(desktop = TRUE)

## End(Not run)
```

---

length,PhysioExperiment-method

Length method for PhysioExperiment

Description

Returns the number of time points.

Usage

```
## S4 method for signature 'PhysioExperiment'
length(x)
```

Arguments

x	A PhysioExperiment object.

Value

Integer number of time points.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
length(pe)  # 100
```

`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).

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).

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.

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)
```

---

localEfficiency *Compute local efficiency*

Description

Calculates the local efficiency for each node.

Usage

```
localEfficiency(adjacency, weighted = FALSE)
```

Arguments

adjacency	An adjacency matrix.
weighted	If TRUE, uses weighted paths.

Value

A numeric vector of local efficiency values.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
localEfficiency(adj)
```

modularity *Compute network modularity*

Description

Calculates the modularity of a network given a community partition.

Usage

```
modularity(adjacency, communities)
```

Arguments

adjacency	An adjacency matrix.
communities	A vector of community assignments for each node.

Value

Modularity value (-0.5 to 1).

Examples

```
# Two clear communities
adj <- matrix(0, 6, 6)
adj[1:3, 1:3] <- 1
adj[4:6, 4:6] <- 1
diag(adj) <- 0
communities <- c(1, 1, 1, 2, 2, 2)
modularity(adj, communities)
```

naSummary *Get NA summary for all assays*

Description

Returns a summary of NA values across all assays.

Usage

```
naSummary(x)
```

Arguments

x	A PhysioExperiment object.

Value

A data.frame with NA statistics for each assay.

Examples

```
pe <- PhysioExperiment(
  assays = list(
    raw = matrix(c(1, NA, 3, 4), nrow = 2),
    filtered = matrix(1:4, nrow = 2)
  ),
  samplingRate = 100
)
naSummary(pe)
```

---

nChannels *Get number of channels*

Description

Get number of channels

Usage

```
nChannels(x)
```

Arguments

x	A PhysioExperiment object.

Value

Integer number of channels.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
nChannels(pe)  # 4
```

nEvents *Get number of events*

Description

Get number of events

Usage

```
nEvents(x)
```

Arguments

x	A PhysioEvents object.

Value

Integer count of events.

nodeDegree *Compute node degree*

Description

Calculates the degree (number of connections) for each node.

Usage

```
nodeDegree(adjacency, weighted = FALSE, use_cpp = TRUE)
```

Arguments

adjacency	An adjacency matrix (binary or weighted).
weighted	If TRUE, returns weighted degree (strength).

Value

A numeric vector of node degrees.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 0, 1, 0, 0), 3, 3)
nodeDegree(adj)
```

`notchFilter` *Notch filter (power line noise removal)*

Description

Applies a notch filter to remove power line noise (50 Hz or 60 Hz) and optionally its harmonics.

Usage

```
notchFilter(
  x,
  freq = 50,
  bandwidth = 2,
  harmonics = 1L,
  output_assay = "filtered"
)
```

Arguments

`x`	A `PhysioExperiment` object.
`freq`	Center frequency to remove in Hz. Default is 50 (European power line).
`bandwidth`	Bandwidth of the notch in Hz. Default is 2.
`harmonics`	Number of harmonics to remove. Default is 1 (only fundamental).
`output_assay`	Name for the output assay. Default is "filtered".

Value

The input object with a new assay containing filtered data.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 250
)

# Remove 50 Hz power line noise (Europe/Asia)
pe <- notchFilter(pe, freq = 50)

# Remove 60 Hz and harmonics (Americas)
pe <- notchFilter(pe, freq = 60, harmonics = 2)
```

pathLength *Compute shortest path length*

Description

Calculates the shortest path length between all node pairs using Floyd-Warshall.

Usage

```
pathLength(adjacency, weighted = FALSE, use_cpp = TRUE)
```

Arguments

adjacency	An adjacency matrix.
weighted	If TRUE, uses edge weights as distances.

Value

A matrix of shortest path lengths.

Examples

```
adj <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
pathLength(adj)
```

PhysioEvents *Create a PhysioEvents object*

Description

Create a PhysioEvents object

Usage

```
PhysioEvents(
  onset = numeric(0),
  duration = numeric(0),
  type = character(0),
  value = character(0)
)
```

Arguments

onset	Numeric vector of event onset times in seconds.
duration	Numeric vector of event durations in seconds.
type	Character vector of event types (e.g., "stimulus", "response").
value	Character vector of event values/labels.

Value

A PhysioEvents object.

Examples

```
# Create events for a simple experiment
events <- PhysioEvents(
  onset = c(1.0, 2.5, 4.0, 5.5),
  duration = c(0.5, 0.5, 0.5, 0.5),
  type = c("stimulus", "response", "stimulus", "response"),
  value = c("target", "hit", "distractor", "false_alarm")
)
events

# Create events with single type
stim_events <- PhysioEvents(
  onset = c(1, 2, 3, 4, 5),
  type = "stimulus"
)
```

PhysioEvents-class *Event management for PhysioExperiment*

Description

Functions for managing experimental events (triggers, markers, annotations) within PhysioExperiment objects. PhysioEvents class

Details

A simple S4 class to store event information as a DataFrame.

Slots

`events` A DataFrame containing event information with columns: onset (numeric), duration (numeric), type (character), value (character).

PhysioExperiment *Construct a PhysioExperiment object*

Description

Construct a PhysioExperiment object

Usage

```
PhysioExperiment(
  assays = S4Vectors::SimpleList(),
  rowData = NULL,
  colData = NULL,
  metadata = list(),
  samplingRate = as.numeric(NA)
)
```

Arguments

assays	A SimpleList (or coercible object) of assay arrays.
rowData	Feature-level metadata as a DataFrame.
colData	Sample-level metadata as a DataFrame.
metadata	Optional experiment-level metadata list.
samplingRate	Numeric scalar sampling rate in Hz.

Value

A PhysioExperiment instance.

Examples

```
# Create a simple PhysioExperiment with random EEG-like data
# 1000 time points, 4 channels
eeg_data <- matrix(rnorm(1000 * 4), nrow = 1000, ncol = 4)
colnames(eeg_data) <- c("Fz", "Cz", "Pz", "Oz")

pe <- PhysioExperiment(
  assays = list(raw = eeg_data),
  colData = S4Vectors::DataFrame(
    label = c("Fz", "Cz", "Pz", "Oz"),
    type = rep("EEG", 4)
  ),
  samplingRate = 250
)
pe

# Access sampling rate
samplingRate(pe)

# Create with multiple assays
pe2 <- PhysioExperiment(
  assays = list(raw = eeg_data, filtered = eeg_data * 0.5),
  samplingRate = 500
)
```

PhysioExperiment-class *PhysioExperiment class definition*

Description

The PhysioExperiment class extends SummarizedExperiment to store multi-modal physiological signal data alongside metadata such as sampling rate. This file defines the class, its validity checks, and the user-facing constructor.

Slots

samplingRate Numeric scalar describing the acquisition frequency in Hz.

pickChannels *Pick specific channels*

Description

Creates a new PhysioExperiment with only selected channels.

Usage

```
pickChannels(x, channels)
```

Arguments

x	A PhysioExperiment object.
channels	Integer indices or character names of channels to keep.

Value

A new PhysioExperiment with selected channels.

Examples

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

# Pick by index
pe_subset <- pickChannels(pe, c(1, 3))
nChannels(pe_subset)  # 2

# Pick by name
pe_frontal <- pickChannels(pe, c("Fz", "Cz"))
```

pli *Compute Phase Lag Index (PLI)*

Description

Calculates the Phase Lag Index between channel pairs, a measure of asymmetry in the phase difference distribution.

Usage

```
pli(x, freq_band, channels = NULL, assay_name = NULL)
```

Arguments

x	A PhysioExperiment object.
freq_band	Numeric vector of length 2 specifying frequency band (Hz).
channels	Integer vector of channel indices.
assay_name	Input assay name.

Details

PLI measures the asymmetry of the distribution of phase differences. Unlike PLV, PLI is insensitive to volume conduction effects that lead to zero-lag synchronization.

PLI = |mean(sign(Im(S_xy)))|, where S_xy is the cross-spectrum.

Value

A matrix of PLI values (channel x channel), values 0-1.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 256
)

# Compute PLI in alpha band (8-12 Hz)
pli_matrix <- pli(pe, freq_band = c(8, 12))
```

`plotAdjacencyMatrix` *Plot adjacency matrix heatmap*

Description

Creates a heatmap visualization of an adjacency or connectivity matrix.

Usage

```
plotAdjacencyMatrix(
  adjacency,
  node_names = NULL,
  symmetric = TRUE,
  color_palette = "default",
  show_values = FALSE,
  title = "Connectivity Matrix"
)
```

Arguments

`adjacency`	An adjacency matrix or connectivity result.
`node_names`	Optional vector of node names.
`symmetric`	If TRUE, only shows lower triangle.
`color_palette`	Color palette: "default", "viridis", "heat", or custom.
`show_values`	If TRUE, displays values in cells.
`title`	Plot title.

Value

A ggplot object.

Examples

```
set.seed(123)
adj <- matrix(runif(16), 4, 4)
adj <- (adj + t(adj)) / 2
diag(adj) <- 1
plotAdjacencyMatrix(adj, node_names = c("Fz", "Cz", "Pz", "Oz"))
```

plotDynamicConnectivity *Plot dynamic connectivity*

Description

Creates a visualization of time-varying connectivity.

Usage

```
plotDynamicConnectivity(
  dyn_conn,
  node_pair = "mean",
  title = "Dynamic Connectivity"
)
```

Arguments

dyn_conn	Result from slidingWindowConnectivity().
node_pair	Vector of two node indices to plot, or "mean" for average.
title	Plot title.

Value

A ggplot object.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000, ncol = 4)),
  samplingRate = 100
)
dyn_conn <- slidingWindowConnectivity(pe, window_size = 200, step = 50)
plotDynamicConnectivity(dyn_conn, node_pair = c(1, 2))
```

plotERP *Plot event-related potential (ERP) waveform*

Description

Generates an ERP plot from epoched data.

Usage

```
plotERP(x, channel = 1L, ci = 0.95, show_epochs = FALSE, epoch_alpha = 0.2)
```

Arguments

x	An epoched PhysioExperiment object.
channel	Integer index or character name of the channel.
ci	Confidence interval level (0-1). NULL for no CI.
show_epochs	Logical. If TRUE, shows individual epoch traces.
epoch_alpha	Alpha value for individual epoch traces.

Value

A ggplot object.

---

plotMultiChannel *Multi-channel visualization functions*

Description

Functions for visualizing multiple channels simultaneously. Plot multiple channels (butterfly or stacked)

Usage

```
plotMultiChannel(
  x,
  channels = NULL,
  sample = 1L,
  style = c("butterfly", "stacked"),
  offset = NULL,
  assay_name = NULL,
  colors = NULL
)
```

Arguments

x	A PhysioExperiment object.
channels	Integer vector of channel indices to plot. If NULL, plots all.
sample	Integer index for the sample (for 3D data).
style	Plot style: "butterfly" (overlaid) or "stacked" (offset).
offset	Numeric offset between channels for stacked plot.
assay_name	Optional assay name. If NULL, uses the default assay.
colors	Optional color vector for channels.

Details

Generates a plot showing multiple channels from the signal data.

Value

A ggplot object.

Examples

```
# Create example data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)

# Butterfly plot (all channels overlaid)
plotMultiChannel(pe, style = "butterfly")

# Stacked plot (channels offset vertically)
plotMultiChannel(pe, style = "stacked")

# Plot specific channels
plotMultiChannel(pe, channels = c(1, 3), style = "butterfly")
```

---

plotNetwork *Network Visualization for PhysioExperiment*

Description

Functions for visualizing functional connectivity networks including network graphs, adjacency matrices, and network metrics. Plot network graph

Usage

```
plotNetwork(
  adjacency,
  node_names = NULL,
  node_size = "degree",
  edge_threshold = 0,
  layout = c("circle", "spring", "grid"),
  node_color = "#3498db",
  edge_color = "#7f8c8d",
  title = "Network Graph"
)
```

Arguments

adjacency	An adjacency matrix or connectivity result.
node_names	Optional vector of node names.
node_size	Node size. Can be "degree", "betweenness", "eigenvector", or numeric.

edge_threshold	Minimum edge weight to display.
layout	Layout algorithm: "circle", "spring", or "grid".
node_color	Node color or vector of colors.
edge_color	Edge color.
title	Plot title.

Details

Creates a network graph visualization with nodes and edges.

Value

A ggplot object.

Examples

```
set.seed(123)
adj <- matrix(runif(16), 4, 4)
adj <- (adj + t(adj)) / 2
diag(adj) <- 0
plotNetwork(adj, node_names = c("Fz", "Cz", "Pz", "Oz"))
```

---

plotNetworkMetrics *Plot network metrics*

Description

Creates a bar plot of network metrics for each node.

Usage

```
plotNetworkMetrics(
  adjacency,
  metrics = c("degree", "clustering", "betweenness"),
  node_names = NULL,
  title = "Network Metrics"
)
```

Arguments

adjacency	An adjacency matrix.
metrics	Vector of metrics to compute: "degree", "clustering", "betweenness", "eigenvector", "local_efficiency".
node_names	Optional vector of node names.
title	Plot title.

Value

A ggplot object.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0), 4, 4)
plotNetworkMetrics(adj, node_names = c("Fz", "Cz", "Pz", "Oz"))
```

---

plotNetworkStability *Plot network stability over time*

Description

Visualizes the temporal stability of network topology.

Usage

```
plotNetworkStability(stability, title = "Network Stability")
```

Arguments

stability	Result from temporalStability().
title	Plot title.

Value

A ggplot object.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000, ncol = 4)),
  samplingRate = 100
)
dyn_conn <- slidingWindowConnectivity(pe, window_size = 200, step = 50)
stab <- temporalStability(dyn_conn)
plotNetworkStability(stab)
```

plotPSD *Plot power spectral density*

Description

Generates a PSD plot for specified channels.

Usage

```
plotPSD(x, channels = NULL, sample = 1L, log_scale = TRUE, freq_range = NULL)
```

Arguments

x	A PhysioExperiment object.
channels	Integer vector of channel indices. If NULL, plots all.
sample	Integer index for the sample (for 3D data).
log_scale	Logical. If TRUE, uses log scale for power.
freq_range	Numeric vector of length 2 specifying frequency range.

Value

A ggplot object.

plotSignal *Plot a signal channel*

Description

Generates a simple line plot for the selected channel and sample from the default assay. Supports both 2D (time x channel) and 3D (time x channel x sample) assay arrays.

Usage

```
plotSignal(x, channel = 1L, sample = 1L, assay_name = NULL)
```

Arguments

x	A PhysioExperiment object.
channel	Integer index for the channel.
sample	Integer index for the sample (only used for 3D arrays).
assay_name	Optional assay name. If NULL, uses the default assay.

Value

A ggplot object.

plotSpectrogram *Plot spectrogram*

Description

Creates a visualization of a spectrogram result.

Usage

```
plotSpectrogram(spec, freq_range = NULL, log_power = TRUE)
```

Arguments

spec	Spectrogram result from spectrogram().
freq_range	Optional frequency range to display.
log_power	If TRUE, displays log power.

Value

A ggplot object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 250
)

# Compute spectrogram
spec <- spectrogram(pe, channel = 1)

# Plot with frequency range filter
plotSpectrogram(spec, freq_range = c(1, 50))
```

plotSPM *Plot SPM result*

Description

Visualizes SPM analysis results with significance thresholds.

Usage

```
plotSPM(
  x,
  time_axis = NULL,
  show_threshold = TRUE,
  show_clusters = TRUE,
  title = NULL
)
```

Arguments

x	An spm_result object.
time_axis	Optional time axis values.
show_threshold	Logical; if TRUE, shows significance threshold.
show_clusters	Logical; if TRUE, highlights significant clusters.
title	Plot title.

Value

A ggplot object.

Examples

```
# Create and plot SPM result
set.seed(123)
g1 <- matrix(rnorm(100 * 10), nrow = 100)
g2 <- matrix(rnorm(100 * 10), nrow = 100)
g2[40:60, ] <- g2[40:60, ] + 1.5

pe <- PhysioExperiment(
  assays = list(values = cbind(g1, g2)),
  samplingRate = 100
)
result <- spmTTest(pe, group1 = 1:10, group2 = 11:20)
plotSPM(result)
```

---

plotTopomap *Topographic Map Visualization*

Description

Functions for plotting scalp topography maps showing the spatial distribution of signal values across electrode positions. Plot topographic map (scalp topography)

Usage

```
plotTopomap(
  x,
  values = NULL,
  time = NULL,
  channel_values = NULL,
  assay_name = NULL,
  resolution = 100L,
  contours = TRUE,
  head_shape = TRUE,
  electrodes = TRUE,
  palette = "RdBu",
  limits = NULL,
  title = NULL
)
```

Arguments

x	A PhysioExperiment object with electrode positions.
values	Optional numeric vector of values to plot. If NULL, uses values from the specified time point.
time	Time point in seconds to extract values (if values is NULL).
channel_values	Named vector of channel values (alternative to values).
assay_name	Optional assay name. If NULL, uses the default assay.
resolution	Grid resolution for interpolation. Default is 100.
contours	Logical. If TRUE, adds contour lines. Default is TRUE
head_shape	Logical. If TRUE, draws head outline. Default is TRUE.
electrodes	Logical. If TRUE, shows electrode positions. Default is TRUE.
palette	Color palette name or vector of colors.
limits	Numeric vector of length 2 for color scale limits.
title	Plot title. If NULL, auto-generated.

Details

Creates a 2D topographic map showing the spatial distribution of values across electrode positions on the scalp.

Value

A ggplot object.

Examples

```
# Create example with 10-20 electrode positions
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)

# Apply 10-20 montage to get electrode positions
pe <- applyMontage(pe, "10-20")

# Plot topographic map at time = 0.5s
plotTopomap(pe, time = 0.5)

# Plot with custom values
plotTopomap(pe, values = c(1, 0.5, -0.5, -1))
```

---

plotTopomapSeries *Plot topographic map animation*

Description

Creates a series of topographic maps across time.

Usage

```
plotTopomapSeries(x, times, ...)
```

Arguments

x	A PhysioExperiment object with electrode positions.
times	Numeric vector of time points to plot.
...	Additional arguments passed to plotTopomap.

Value

A list of ggplot objects.

Examples

```
## Not run:
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 100
)
pe <- applyMontage(pe, "10-20")
```

```
# Create topomaps at multiple time points
plots <- plotTopomapSeries(pe, times = c(0.1, 0.2, 0.3, 0.4))

## End(Not run)
```

---

plv *Compute Phase Locking Value (PLV)*

Description

Calculates the Phase Locking Value between channel pairs, measuring the consistency of phase difference across time.

Usage

```
plv(x, freq_band, channels = NULL, assay_name = NULL)
```

Arguments

x	A PhysioExperiment object.
freq_band	Numeric vector of length 2 specifying frequency band (Hz).
channels	Integer vector of channel indices.
assay_name	Input assay name.

Details

PLV measures the consistency of the phase difference between two signals. A value of 1 indicates perfect phase locking, while 0 indicates random phase relationship.

The signals are first bandpass filtered to the specified frequency band, then the analytic signal is computed using the Hilbert transform.

Value

A matrix of PLV values (channel x channel), values 0-1.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  rowData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz", "Oz")),
  samplingRate = 256
)

# Compute PLV in alpha band (8-12 Hz)
plv_matrix <- plv(pe, freq_band = c(8, 12))
```

print.spm_result *Print SPM result*

Description

Print SPM result

Usage

```
## S3 method for class 'spm_result'
print(x, ...)
```

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.

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)
```

`rbindPhysio` *Combine PhysioExperiment objects by time*

Description

Concatenates two PhysioExperiment objects along the time axis.

Usage

```
rbindPhysio(x, y)
```

Arguments

x	A PhysioExperiment object.
y	A PhysioExperiment object to concatenate.

Value

Combined PhysioExperiment object.

Examples

```
pe1 <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)
pe2 <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)

# Concatenate in time
pe_concat <- rbindPhysio(pe1, pe2)
length(pe_concat)  # 200
```

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)
```

Arguments

path	Path to the BDF 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

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 object.

Examples

```
## Not run:
# Read a BDF file
pe <- readBDF("recording.bdf")

# Read only specific channels
pe <- readBDF("recording.bdf", channels = c("A1", "A2", "B1", "B2"))

## 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.

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)
```

`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

```
## Not run:
pe <- readBrainVision("recording.vhdr")

## End(Not run)
```

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.

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.

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.

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.

Examples

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

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

## End(Not run)
```

---

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

```
## Not run:
pe <- readGDF("recording.gdf")
pe <- readGDF("recording.gdf", channels = c("Fz", "Cz", "Pz"))

## End(Not run)
```

readMAT *MATLAB .mat File I/O for PhysioExperiment*

Description

Functions for reading and writing MATLAB .mat files. Requires the R.matlab package. Read PhysioExperiment from MATLAB .mat file

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

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

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.

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.

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 in-memory assays.

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.

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)
```

`rejectBadEpochs` *Reject bad epochs*

Description

Identifies and optionally removes epochs with artifacts.

Usage

```
rejectBadEpochs(
  x,
  threshold = 100,
  method = c("amplitude", "gradient", "variance"),
  remove = TRUE
)
```

Arguments

`x`	An epoched PhysioExperiment object.
`threshold`	Amplitude threshold for rejection.
`method`	Detection method: "amplitude", "gradient", or "variance".
`remove`	If TRUE, removes bad epochs. If FALSE, returns indices only.

Value

If remove=TRUE, modified object. If remove=FALSE, indices of bad epochs.

`removeEvents` *Remove events from a PhysioExperiment object*

Description

Remove events from a PhysioExperiment object

Usage

```
removeEvents(x, type = NULL, indices = NULL)
```

Arguments

x	A PhysioExperiment object.
type	Optional event types to remove. If NULL, removes all events.
indices	Optional integer indices of events to remove.

Value

The modified PhysioExperiment object.

---

renameChannels *Rename channels*

Description

Rename channels

Usage

```
renameChannels(x, old_names, new_names)
```

Arguments

x	A PhysioExperiment object.
old_names	Character vector of current names.
new_names	Character vector of new names.

Value

Modified PhysioExperiment object.

Examples

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

# Rename channels
pe <- renameChannels(pe, c("Fz", "Cz"), c("F3", "C3"))
channelNames(pe)
```

`replaceNA` *Replace NA values in assay*

Description

Creates a new assay with NA values handled according to the specified method.

Usage

```
replaceNA(
  x,
  method = "interpolate",
  input_assay = NULL,
  output_assay = "na_handled"
)
```

Arguments

`x`	A PhysioExperiment object.
`method`	Method for handling NA (see handleNA).
`input_assay`	Input assay name. If NULL, uses default assay.
`output_assay`	Output assay name. Default is "na_handled".

Value

Modified PhysioExperiment with new assay.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(c(1, NA, 3, NA, 5, 6), nrow = 3)),
  samplingRate = 100
)

# Interpolate NA values
pe <- replaceNA(pe, method = "interpolate")
```

`rereference` *Re-referencing Operations for EEG Data*

Description

Functions for changing the reference electrode in EEG recordings. Re-referencing is a common preprocessing step that affects the spatial distribution of the signal. Re-reference EEG data

Usage

```
rereference(
  x,
  ref_type = c("average", "channel", "channels", "REST"),
  ref_channels = NULL,
  exclude = NULL,
  input_assay = NULL,
  output_assay = "rereferenced",
  keep_ref = TRUE
)
```

Arguments

`x`	A PhysioExperiment object.
`ref_type`	Type of re-referencing: "average" (common average reference), "channel" (single channel), "channels" (average of specified channels), or "REST" (Reference Electrode Standardization Technique).
`ref_channels`	For "channel" or "channels" type, the channel name(s) or index/indices to use as reference.
`exclude`	Channels to exclude from average reference calculation (e.g., non-EEG channels like EOG, EMG).
`input_assay`	Input assay name. If NULL, uses default assay.
`output_assay`	Output assay name. Default is "rereferenced".
`keep_ref`	Logical. If TRUE, keeps the original reference channel(s) in the output (zeroed). If FALSE, removes them.

Details

Changes the reference electrode for EEG recordings. Supports common re-referencing schemes including average reference, linked mastoids, and single electrode reference.

Re-referencing transforms the data by subtracting a reference signal from each channel. The choice of reference affects the spatial distribution and interpretation of the signal.

Average reference ("average"): Subtracts the mean of all channels at each time point. This is commonly used for high-density EEG and provides a reference-independent measure, but requires good spatial sampling.

Single channel reference ("channel"): Subtracts the signal from a specified electrode. Common choices include Cz, linked mastoids (A1+A2)/2, or nose reference.

Multi-channel reference ("channels"): Subtracts the average of multiple specified channels. Useful for linked mastoids or other custom references.

Value

A PhysioExperiment object with re-referenced data.

Examples

```
# Create example EEG data
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  rowData = S4Vectors::DataFrame(
    label = c("Fp1", "Fp2", "F3", "F4", "C3", "C4", "P3", "P4", "O1", "O2"),
    type = rep("EEG", 10)
  ),
  samplingRate = 256
)

# Apply average reference
pe_avg <- rereference(pe, ref_type = "average")

# Re-reference to a single channel (Cz)
pe_cz <- rereference(pe, ref_type = "channel", ref_channels = "C3")

# Re-reference to linked mastoids (if available)
# pe_linked <- rereference(pe, ref_type = "channels",
#                          ref_channels = c("M1", "M2"))
```

---

resample *Resampling operations for PhysioExperiment*

Description

Functions for resampling signal data to different sampling rates. Resample signal data

Usage

```
resample(
  x,
  target_rate,
  method = c("linear", "spline", "fft"),
  assay_name = NULL,
  output_assay = "resampled"
)
```

Arguments

x	A PhysioExperiment object.
target_rate	Target sampling rate in Hz.
method	Resampling method: "linear" (default), "spline", or "fft".
assay_name	Optional assay name. If NULL, uses the default assay.
output_assay	Name for the output assay. Default is "resampled".

Details

Resamples the signal data to a target sampling rate using interpolation.

Value

A new PhysioExperiment object with resampled data.

---

resolveQuery *Resolve an EventQuery to get filtered events*

Description

Resolve an EventQuery to get filtered events

Usage

```
resolveQuery(q)
```

Arguments

q	An EventQuery object

Value

Data frame of filtered events

samplesToTime *Convert sample indices to times*

Description

Convert sample indices to times

Usage

```
samplesToTime(x, samples)
```

Arguments

x	A PhysioExperiment object.
samples	Integer vector of sample indices.

Value

Numeric vector of times in seconds.

samplingRate *Accessors for PhysioExperiment*

Description

These helper functions expose common slots and derived quantities for `PhysioExperiment` objects. Get or set sampling rate

Usage

```
samplingRate(x)
```

Arguments

x	A PhysioExperiment object.
value	Numeric scalar for the new sampling rate in Hz.

Value

The sampling rate in Hz.

Examples

```
# Create example data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(100), nrow = 10)),
  samplingRate = 250
)

# Get sampling rate
samplingRate(pe)

# Set sampling rate
samplingRate(pe) <- 500
samplingRate(pe)
```

---

setAssaySamplingRate *Set sampling rate for a specific assay*

Description

Set sampling rate for a specific assay

Usage

```
setAssaySamplingRate(x, assay_name, rate)
```

Arguments

x	A PhysioExperiment object.
assay_name	Name of the assay.
rate	Sampling rate for the assay.

Value

Modified PhysioExperiment object.

---

setChannelTypes *Set channel types*

Description

Assigns types (EEG, EMG, EOG, etc.) to channels.

Usage

```
setChannelTypes(x, types)
```

Arguments

x	A PhysioExperiment object.
types	Named character vector or list mapping channel names/indices to types. If unnamed, applies types in order.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  colData = S4Vectors::DataFrame(label = c("Fz", "EOG1", "EMG1", "Oz")),
  samplingRate = 100
)

# Set all channels to same type
pe <- setChannelTypes(pe, "EEG")

# Set specific channel types by name
pe <- setChannelTypes(pe, c(EOG1 = "EOG", EMG1 = "EMG"))
```

---

setChannelUnits *Set channel units*

Description

Assigns physical units to channels.

Usage

```
setChannelUnits(x, units)
```

Arguments

x	A PhysioExperiment object.
units	Character vector or named list of units.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)

# Set all channels to same unit
pe <- setChannelUnits(pe, "uV")
```

---

setElectrodePositions *Set electrode positions*

Description

Assigns 3D electrode positions to channels.

Usage

```
setElectrodePositions(x, positions)
```

Arguments

x	A PhysioExperiment object.
positions	A data.frame or matrix with columns x, y, z and rows matching channels. Row names should match channel names.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(300), nrow = 100, ncol = 3)),
  colData = S4Vectors::DataFrame(label = c("Fz", "Cz", "Pz")),
  samplingRate = 100
)

# Set electrode positions
positions <- data.frame(
  x = c(0, 0, 0),
  y = c(0.71, 0, -0.71),
  z = c(0.71, 1, 0.71)
)
pe <- setElectrodePositions(pe, positions)
```

setEvents *Set events for a PhysioExperiment object*

Description

Set events for a PhysioExperiment object

Usage

```
setEvents(x, events)
```

Arguments

x	A PhysioExperiment object.
events	A PhysioEvents object or a data.frame with columns: onset, duration, type, value.

Value

The modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100)),
  samplingRate = 100
)

# Set events using PhysioEvents object
events <- PhysioEvents(onset = c(1, 2, 3), type = "stimulus")
pe <- setEvents(pe, events)

# Set events using data.frame
pe <- setEvents(pe, data.frame(onset = c(1, 2), type = "response"))
```

setReference *Set reference electrode*

Description

Records the reference electrode used for the recording.

Usage

```
setReference(x, reference)
```

Arguments

x	A PhysioExperiment object.
reference	Character string naming the reference electrode.

Value

Modified PhysioExperiment object.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  samplingRate = 100
)

# Set reference electrode
pe <- setReference(pe, "average")
getReference(pe)  # "average"
```

---

show,PhysioEvents-method

Show method for PhysioEvents

Description

Show method for PhysioEvents

Usage

```
## S4 method for signature 'PhysioEvents'
show(object)
```

Arguments

object	A PhysioEvents object.

show,PhysioExperiment-method *Show method for PhysioExperiment*

Description

Displays a summary of the PhysioExperiment object.

Usage

```
## S4 method for signature 'PhysioExperiment'
show(object)
```

Arguments

object	A PhysioExperiment object.

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  # Displays summary
```

slidingWindowConnectivity *Sliding window connectivity*

Description

Computes connectivity matrices over sliding time windows.

Usage

```
slidingWindowConnectivity(
  x,
  window_size = 256L,
  step = 64L,
  method = c("correlation", "coherence", "plv"),
  ...
)
```

Arguments

x	A PhysioExperiment object.
window_size	Window size in samples.
step	Step size in samples.
method	Connectivity method: "correlation", "coherence", or "plv".
...	Additional arguments passed to connectivity function.

Value

A list with time-varying connectivity matrices.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000, ncol = 4)),
  samplingRate = 100
)
dyn_conn <- slidingWindowConnectivity(pe, window_size = 200, step = 50)
```

---

smallWorldness *Compute small-worldness*

Description

Calculates the small-world coefficient (sigma) of a network.

Usage

```
smallWorldness(adjacency, n_rand = 100, n_cores = NULL)
```

Arguments

adjacency	An adjacency matrix.
n_rand	Number of random networks for comparison.
n_cores	Number of cores for parallel processing. Default NULL uses sequential processing.

Value

A list with small-worldness metrics.

Examples

```
# Create a small-world-like network
set.seed(123)
n <- 20
adj <- matrix(0, n, n)
for (i in 1:n) {
  adj[i, ((i) %% n) + 1] <- 1
  adj[i, ((i + 1) %% n) + 1] <- 1
}
adj <- adj + t(adj)
adj[adj > 0] <- 1
# Add some random long-range connections
adj[sample(which(adj == 0 & row(adj) < col(adj)), 5)] <- 1
adj <- adj + t(adj)
adj[adj > 0] <- 1
diag(adj) <- 0
sw <- smallWorldness(adj, n_rand = 10)
```

---

spectralClustering *Spectral clustering of network nodes*

Description

Performs spectral clustering on the network.

Usage

```
spectralClustering(adjacency, n_clusters = 2, normalized = TRUE)
```

Arguments

adjacency	An adjacency matrix.
n_clusters	Number of clusters.
normalized	If TRUE, uses normalized Laplacian.

Value

A list with cluster assignments and spectral embedding.

Examples

```
# Create block-structured network
adj <- matrix(0, 6, 6)
adj[1:3, 1:3] <- 1
adj[4:6, 4:6] <- 1
adj[3, 4] <- adj[4, 3] <- 0.5
diag(adj) <- 0
clusters <- spectralClustering(adj, n_clusters = 2)
```

spectralDecomposition *Spectral decomposition of graph Laplacian*

Description

Computes eigenvalues and eigenvectors of the graph Laplacian.

Usage

```
spectralDecomposition(adjacency, normalized = TRUE, n_components = NULL)
```

Arguments

adjacency	An adjacency matrix.
normalized	If TRUE, uses normalized Laplacian.
n_components	Number of components to return. If NULL, returns all.

Value

A list with eigenvalues and eigenvectors.

Examples

```
adj <- matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), 3, 3)
spec <- spectralDecomposition(adj)
```

spectrogram *Time-Frequency Analysis for PhysioExperiment*

Description

Functions for time-frequency analysis including wavelet transforms, spectrograms, and band power extraction. Compute spectrogram (Short-Time Fourier Transform)

Usage

```
spectrogram(
  x,
  window_size = 256L,
  overlap = 0.5,
  window_type = c("hanning", "hamming", "blackman", "rectangular"),
  channel = 1L,
  sample = 1L
)
```

Arguments

x	A PhysioExperiment object.
window_size	Window size in samples.
overlap	Overlap between windows (0-1).
window_type	Window function: "hanning", "hamming", "blackman", or "rectangular".
channel	Channel index to analyze.
sample	Sample index (for 3D data).

Details

Computes the spectrogram using STFT.

Value

A list containing:

- power: Power spectrogram matrix (frequency x time)
- frequencies: Frequency vector
- times: Time vector

Examples

```
# Create example data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000)),
  samplingRate = 250
)

# Compute spectrogram for channel 1
spec <- spectrogram(pe, channel = 1)

# Plot spectrogram
plotSpectrogram(spec, freq_range = c(1, 40))
```

---

spmAnova *SPM ANOVA (F-test)*

Description

Performs SPM F-test for comparing multiple groups.

Usage

```
spmAnova(x, groups, alpha = 0.05)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
groups	Factor or list indicating group membership.
alpha	Significance level.

Value

A list of class "spm_result" containing F-statistics and clusters.

Examples

```
# Three-group comparison
set.seed(123)
g1 <- matrix(rnorm(100 * 8), nrow = 100)
g2 <- matrix(rnorm(100 * 8), nrow = 100) + 0.5
g3 <- matrix(rnorm(100 * 8), nrow = 100) + 1.0

data <- cbind(g1, g2, g3)
groups <- factor(rep(c("A", "B", "C"), each = 8))

pe <- PhysioExperiment(assays = list(values = data), samplingRate = 100)
result <- spmAnova(pe, groups = groups)
```

---

spmPairedTTest *SPM paired t-test*

Description

Performs SPM analysis for paired/repeated measures data.

Usage

```
spmPairedTTest(x, condition1, condition2, alpha = 0.05, two_tailed = TRUE)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
condition1	Indices for first condition.
condition2	Indices for second condition.
alpha	Significance level.
two_tailed	Logical; if TRUE, performs two-tailed test.

Value

A list of class "spm_result".

Examples

```
# Pre-post intervention comparison
set.seed(123)
pre <- matrix(rnorm(100 * 15), nrow = 100)
post <- pre + 0.8  # Effect across all time points
post[30:50, ] <- post[30:50, ] + 0.5  # Additional effect

data <- cbind(pre, post)
pe <- PhysioExperiment(assays = list(values = data), samplingRate = 100)

result <- spmPairedTTest(pe, condition1 = 1:15, condition2 = 16:30)
```

---

spmTTest *Statistical Parametric Mapping (SPM1D) for Biomechanics*

Description

Functions for statistical analysis of continuous waveform data using Statistical Parametric Mapping (SPM) methodology adapted from neuroimaging. These methods test hypotheses over entire waveforms rather than discrete points. SPM t-test for waveform comparison

Usage

```
spmTTest(x, group1 = NULL, group2 = NULL, alpha = 0.05, two_tailed = TRUE)
```

Arguments

x	A PhysioExperiment object or matrix (time x observations).
group1	Indices for first group (for two-sample test).
group2	Indices for second group. If NULL, performs one-sample test.
alpha	Significance level (default: 0.05).
two_tailed	Logical; if TRUE, performs two-tailed test.

Details

Performs a t-test at each time point and computes SPMt statistic with Random Field Theory (RFT) correction for multiple comparisons.

SPM analyzes continuous biomechanical waveforms (e.g., joint angles, moments) by computing t-statistics at each time point and using Random Field Theory to control family-wise error rate across the entire waveform.

Value

A list of class "spm_result" containing:

t	T-statistic at each time point
threshold	Critical threshold from RFT
clusters	Significant clusters (start, end, extent, p-value)
p_values	Pointwise p-values
alpha	Significance level used

References

Pataky TC (2012). One-dimensional statistical parametric mapping in Python. Computer Methods in Biomechanics and Biomedical Engineering.

Examples

```
# Create example gait data (100 time points x 20 subjects)
set.seed(123)
# Group 1: normal gait
g1 <- matrix(rnorm(100 * 10), nrow = 100, ncol = 10)
# Group 2: altered gait (effect at 40-60% of cycle)
g2 <- matrix(rnorm(100 * 10), nrow = 100, ncol = 10)
g2[40:60, ] <- g2[40:60, ] + 1.5

data <- cbind(g1, g2)
pe <- PhysioExperiment(
  assays = list(values = data),
  samplingRate = 100
)

# Two-sample SPM t-test
result <- spmTTest(pe, group1 = 1:10, group2 = 11:20)
print(result)
```

---

startAPIServer *Start PhysioExperiment API Server (Non-blocking)*

Description

Starts the API server in the background, allowing R to remain interactive. This is useful for development and testing.

Usage

```
startAPIServer(port = 8000L, host = "127.0.0.1", quiet = FALSE)
```

Arguments

port	Integer. Port number for the API server.
host	Character. Host address to bind to.
quiet	Logical. Suppress startup messages.

Value

A plumber background process handle that can be used to stop the server with $kill().

---

summary,PhysioExperiment-method

Summary statistics for PhysioExperiment

Description

Computes summary statistics for each channel.

Usage

```
## S4 method for signature 'PhysioExperiment'
summary(object, ...)
```

Arguments

object	A PhysioExperiment object.
...	Additional arguments (not used).

Value

A data.frame with summary statistics.

Examples

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

temporalStability *Compute temporal stability of network*

Description

Calculates how stable the network topology is over time.

Usage

```
temporalStability(dyn_conn, metric = c("correlation", "distance"))
```

Arguments

dyn_conn	Result from slidingWindowConnectivity().
metric	Stability metric: "correlation" or "distance".

Value

A list with stability metrics.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000 * 4), nrow = 1000, ncol = 4)),
  samplingRate = 100
)
dyn_conn <- slidingWindowConnectivity(pe, window_size = 200, step = 50)
stability <- temporalStability(dyn_conn)
```

thresholdNetwork *Threshold network by density*

Description

Thresholds a connectivity matrix to achieve a target network density.

Usage

```
thresholdNetwork(connectivity, density = 0.2, absolute = TRUE)
```

Arguments

connectivity	A connectivity matrix.
density	Target density (proportion of edges to keep, 0-1).
absolute	If TRUE, uses absolute values for ranking.

Value

A thresholded adjacency matrix.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500, ncol = 4)),
  samplingRate = 100
)
conn <- correlationMatrix(pe)
# Keep top 30% of connections
adj <- thresholdNetwork(conn$correlation, density = 0.3)
```

---

timeIndex *Time index helper*

Description

Computes a time vector for the default assay using the object's sampling rate.

Usage

```
timeIndex(x)
```

Arguments

x	A PhysioExperiment instance.

Value

Numeric vector of time points in seconds.

---

timeToSamples *Convert event times to sample indices*

Description

Convert event times to sample indices

Usage

```
timeToSamples(x, times)
```

Arguments

x	A PhysioExperiment object.
times	Numeric vector of times in seconds.

Value

Integer vector of sample indices.

---

tTestEpochs *Statistical Testing for PhysioExperiment*

Description

Functions for statistical analysis of physiological signal data, including t-tests, ANOVA, cluster-based permutation tests, and effect sizes. Pointwise t-test across epochs

Usage

```
tTestEpochs(
  x,
  condition1 = NULL,
  condition2 = NULL,
  mu = 0,
  paired = FALSE,
  alternative = c("two.sided", "less", "greater"),
  var.equal = FALSE
)
```

Arguments

x	An epoched PhysioExperiment object (4D data).
condition1	Indices or logical vector for first condition epochs.
condition2	Indices or logical vector for second condition epochs. If NULL, performs one-sample t-test against mu.
mu	Value to test against for one-sample t-test (default: 0).
paired	Logical; if TRUE, performs paired t-test.
alternative	Alternative hypothesis: "two.sided", "less", or "greater".
var.equal	Logical; if TRUE, assumes equal variances.

Details

Performs t-tests at each time point and channel, comparing epochs against a baseline or between two conditions.

Value

A list containing:

t_values	Matrix of t-statistics (time x channel)
p_values	Matrix of p-values (time x channel)
df	Degrees of freedom
n1, n2	Sample sizes for each condition

Examples

```
# Create example epoched data
set.seed(123)
epochs <- array(rnorm(100 * 4 * 20 * 1), dim = c(100, 4, 20, 1))
pe <- PhysioExperiment(
  assays = list(epoched = epochs),
  samplingRate = 100
)
# One-sample t-test against zero
result <- tTestEpochs(pe)
# Two-sample t-test comparing conditions
result2 <- tTestEpochs(pe, condition1 = 1:10, condition2 = 11:20)
```

---

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.

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)
```

`waveletTransform` *Wavelet transform*

Description

Computes the continuous wavelet transform using Morlet wavelets.

Usage

```
waveletTransform(
  x,
  frequencies = seq(1, 40, by = 1),
  n_cycles = 7,
  channel = 1L,
  sample = 1L
)
```

Arguments

`x`	A PhysioExperiment object.
`frequencies`	Numeric vector of frequencies to analyze.
`n_cycles`	Number of wavelet cycles (can be scalar or vector).
`channel`	Channel index to analyze.
`sample`	Sample index (for 3D data).

Value

A list containing:

- `power`: Power matrix (frequency x time)
- `phase`: Phase matrix (frequency x time)
- `frequencies`: Frequency vector
- `times`: Time vector

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(500 * 4), nrow = 500)),
  samplingRate = 100
)

# Compute wavelet transform (1-30 Hz)
wt <- waveletTransform(pe, frequencies = seq(1, 30), channel = 1)

# Access power and phase
dim(wt$power)  # frequency x time
```

---

wPLI *Compute weighted Phase Lag Index (wPLI)*

Description

Calculates the weighted Phase Lag Index, which is less sensitive to noise than standard PLI.

Usage

```
wPLI(x, freq_band, channels = NULL, assay_name = NULL)
```

Arguments

x	A PhysioExperiment object.
freq_band	Numeric vector of length 2 specifying frequency band (Hz).
channels	Integer vector of channel indices.
assay_name	Input assay name.

Details

wPLI weights the contribution of each phase difference by the magnitude of the imaginary component, reducing the influence of noise sources.

Value

A matrix of wPLI values (channel x channel), values 0-1.

Examples

```
set.seed(123)
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(4000), nrow = 1000, ncol = 4)),
  samplingRate = 256
)

# Compute wPLI in theta band (4-8 Hz)
wpli_matrix <- wPLI(pe, freq_band = c(4, 8))
```

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	Name of the assay to write.
compression_level	Compression level (0-9).

Value

Invisible NULL.

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)
```

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

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

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

## 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.

Examples

```
## Not run:
# Create a PhysioExperiment
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(25600), nrow = 2560, ncol = 10)),
  rowData = 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)
```

---

writeBrainVision *Write PhysioExperiment to BrainVision format*

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

Value

Invisible NULL.

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
)
writeBrainVision(pe, "output")

## 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.

Examples

```
# Create example data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  rowData = 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.

Examples

```
## Not run:
# Create a PhysioExperiment with EEG-like data
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(25600), nrow = 2560, ncol = 10)),
  rowData = 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.

Examples

```
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(400), nrow = 100, ncol = 4)),
  rowData = 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.

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

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

## End(Not run)
```

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.

Examples

```
## Not run:
pe <- PhysioExperiment(
  assays = list(raw = matrix(rnorm(1000), nrow = 100, ncol = 10)),
  rowData = 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")
```

```
# EEGLAB-compatible format
writeMAT(pe, "output.mat", eeglab = TRUE)

## 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` instance; `writePhysio()` returns the input object invisibly.

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 chunk dimensions for HDF5 storage.
`compression_level`	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.

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