Package: mesa (via r-universe)

July 23, 2026

Title Methylation Enrichment Sequencing Analysis

Version 0.99.5

Description A package for the analysis of methylation enrichment sequencing data (e.g. MBD-seq or MEDIP-seq). This allows for the window-based evaluation of methylation levels (using the 'qsea' package), including functions for determining differentially methylated regions. Many functions are provided for tidyverse style modification of the qseaSet objects originally defined in 'qsea'.

License GPL (>=2)

Encoding UTF-8

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports plyranges, magrittr, limma, glue, ChIPseeker, purrr, progress, tidyselect, rlang, GenomeInfoDb, GenomicRanges, BiocGenerics, Biostrings, gtools, RColorBrewer, pheatmap, HMMcopy, ggplot2, janitor, stats, stringr, methods, graphics, grDevices, tibble, tidyr, BSgenome, BiocParallel, matrixStats, IRanges, GenomicAlignments, data.table, Rsamtools, tools, ComplexHeatmap, readr, memoise, biomaRt, hues, circlize, scales

Depends qsea, R (>= 4.5.0), dplyr

Suggests rmarkdown, knitr, openxlsx, testthat (>= 3.0.0), UpSetR, rtracklayer, ggrepel, MEDIPS, workflows, MEDIPSData, uwot, BSgenome.Hsapiens.NCBI.GRCh38, BSgenome.Hsapiens.UCSC.hg19, BSgenome.Scerevisiae.UCSC.sacCer3, org.Hs.eg.db, org.Mm.eg.db, TxDb.Hsapiens.UCSC.hg38.knownGene, TxDb.Hsapiens.UCSC.hg19.knownGene, TxDb.Mmusculus.UCSC.mm10.knownGene, plotly, ragg

VignetteBuilder knitr

biocViews Sequencing, DNAMethylation, CpGIsland, Preprocessing, Normalization, QualityControl, Visualization, CopyNumberVariation, DifferentialMethylation, Coverage, ChIPSeq, ChipOnChip

Config/testthat/edition 3

URL https://github.com/cruk-mi/mesa

BugReports https://github.com/cruk-mi/mesa

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libglpk-dev make libbz2-dev libicu-dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev perl libx11-dev xz-utils zlib1g-dev

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

Date/Publication 2026-06-26 10:57:27 UTC

RemoteUrl https://github.com/BiocStaging/mesa

RemoteRef HEAD

RemoteSha 61c890a5836b061eb9bd1b3f077d24768faa1469

Contents

`addBamCoveragePairedAndUnpaired` *Add coverage to a qseaSet using proper pairs and/or high-quality R1 reads*

Description

Scan BAM files listed in a `qseaSet` and add per-region counts and library summaries using proper pairs (with MAPQ and insert-size filters) and, optionally, high-quality unpaired R1 reads extended to a fragment length. Parallelisation via **BiocParallel** is supported.

Usage

```
addBamCoveragePairedAndUnpaired(
  qs,
  fragmentLength = NULL,
  minMapQual = 30,
  maxInsertSize = 1000,
```

```
  minInsertSize = 100,
  minReferenceLength = 30,
  parallel = getMesaParallel(),
  properPairsOnly = FALSE
)
```

Arguments

qs	qseaSet Input object with valid file_name in its sample table and regions accessible via qsea::getRegions().
fragmentLength	integer(1) or NULL Length to extend unpaired R1 reads. If NULL, estimated as the median proper-pair width (when available). **Default:** NULL.
minMapQual	integer(1) Minimum MAPQ for either end (proper pairs) or R1 (unpaired). **Default:** 30.
maxInsertSize	integer(1) Maximum absolute insert size for proper pairs. **Default:** 1000.
minInsertSize	integer(1) Minimum absolute insert size for proper pairs. **Default:** 100.
minReferenceLength	integer(1) Minimum reference span for R1s (unpaired). **Default:** 30.
parallel	logical(1) If TRUE and a multi-core BPPARAM is registered, BAMs are processed in parallel (see **BiocParallel**). **Default:** getMesaParallel().
properPairsOnly	logical(1) If TRUE, only proper pairs (plus rescued near-proper pairs) are used. If FALSE, combine proper pairs with high-quality unpaired R1 reads. **Default:** FALSE.

Details

Per-region counts are computed from fragment midpoints to align with qsea's counting scheme. When properPairsOnly = FALSE, unpaired R1 reads are extended from the 5' end by fragmentLength. If fragmentLength is not supplied, it is estimated from proper pairs in the same BAM.

Value

A qseaSet with updated slots:

- **Counts**: updated count matrix.
- **Library**: updated library table with summary metrics.
- **Parameters**: appended record of MAPQ/size thresholds.

Parallelisation

Parallel execution uses BiocParallel::bplapply() with the currently registered backend (see BiocParallel::register()). Pass parallel = TRUE and set a multi-core/cluster backend for speed.

See Also

`getBamCoveragePairedAndUnpairedR1()` (internal low-level worker), `qsea::getRegions()`, `qsea::getSampleTable()`, `BiocParallel::register()`

`addHMMcopyCNV` *Add per-sample CNV calls to a qseaSet using HMMcopy*

Description

Runs a CNV pipeline on each sample: counts fragments in fixed-size genomic windows, normalises for GC and mappability (HMMcopy), segments, and stores per-window copy-number in the `qseaSet`. Supports parallel execution. Only GRCh38/hg38 is currently supported.

Usage

```
addHMMcopyCNV(
  qs,
  inputColumn = "input_file",
  windowSize = 1e+06,
  fragmentLength = NULL,
  plotDir = NULL,
  parallel = getMesaParallel(),
  maxInsertSize = 1000,
  minInsertSize = 50,
  minReferenceLength = 30,
  minMapQual = 30,
  properPairsOnly = FALSE,
  hmmCopyGC = NULL,
  hmmCopyMap = NULL
)
```

Arguments

`qs`	`qseaSet`. Input object whose samples will receive CNV tracks.
`inputColumn`	`character(1)`. Column in the sample table containing BAM paths (e.g. `"input_file"`). **Default:** `"input_file"`.
`windowSize`	`integer(1)`. CNV bin size in bp. Allowed values: `50000`, `500000`, `1000000`. **Default:** `1000000`.
`fragmentLength`	`integer(1)` or `NULL`. Length to extend unpaired R1 reads; if `NULL`, estimate from proper pairs. **Default:** `NULL`.
`plotDir`	`character(1)` or `NULL`. Directory to write per-sample PDF diagnostics; if `NULL`, no plots are saved. **Default:** `NULL`.
`parallel`	`logical(1)`. Use the registered **BiocParallel** backend when `TRUE`; otherwise run serially. **Default:** `getMesaParallel()`.

`maxInsertSize` integer(1). Maximum insert size to accept for proper pairs (bp). **Default:** 1000.

`minInsertSize` integer(1). Minimum insert size to accept for proper pairs (bp). **Default:** 50.

`minReferenceLength` integer(1). Minimum aligned reference span for unpaired R1 (bp). **Default:** 30.

`minMapQual` integer(1). Minimum MAPQ; for pairs, either end meeting the cutoff is accepted. **Default:** 30.

`properPairsOnly` logical(1). If TRUE, ignore unpaired R1; if FALSE, combine proper pairs with high-quality R1. **Default:** FALSE.

`hmmCopyGC` GRanges or NULL. Precomputed GC content track binned at windowSize. **Default:** NULL.

`hmmCopyMap` GRanges or NULL. Precomputed mappability track binned at windowSize. **Default:** NULL.

Details

Coverage is obtained (pairs + optional R1) over tiled windows, corrected with HMMcopy (`HMMcopy::correctReadcount()` followed by `HMMcopy::HMMsegment()`), and merged back into the qseaSet. You must supply GC/mappability tracks binned at the same windowSize (or use package-provided defaults where applicable). When parallel = TRUE, computation uses the currently registered backend (see `BiocParallel::register()`).

Value

The input qseaSet with CNV information attached, including:

- per-window CNV tracks accessible via downstream CNV accessors,
- library/sample tables updated with relevant CNV/coverage summaries,
- (optionally) per-sample diagnostic PDFs if plotDir is provided.

See Also

runHMMcopy(), plotCNVheatmap(), getBamCoveragePairedAndUnpairedR1() (internal low-level worker), HMMcopy::correctReadcount(), HMMcopy::HMMsegment(), BiocParallel::register()

Other CNV: plotCNVheatmap(), removeCNV(), runHMMCopy()

Examples

```
## Not run:
data(exampleTumourNormal, package = "mesa")
exampleTumourNormal %>%

    addHMMcopyCNV(
        inputColumn = "input_file",
        windowSize = 1e6,
        hmmCopyGC = gc_hg38_1000kb, # GRanges with 'gc' column
        hmmCopyMap = map_hg38_1000kb, # GRanges with 'map' column
```

```
        properPairsOnly = TRUE,
        parallel = FALSE,
        plotDir = tempdir()
    )

## End(Not run)
```

---

addHyperStableFraction *Summarise the fraction of hyper-stable methylated regions (GRCh38 only)*

Description

For each sample, compute the proportion of *Edgar et al.* "ultra-stable" windows that are clearly methylated (beta >= `minBeta`) after filtering by CpG density (>= `minDensity`). The result is added to the sample table as `hyperStableEdgar` (overwriting any existing column of that name).

Usage

```
addHyperStableFraction(qseaSet, minDensity = 5, minBeta = 0.8)
```

Arguments

`qseaSet`	`qseaSet`. Input object (must be GRCh38). **Default:** none (must be supplied).
`minDensity`	`numeric(1)`. Minimum `CpG_density` to keep a window. **Default:** `5`.
`minBeta`	`numeric(1)`. Beta threshold to call a window methylated. **Default:** `0.8`.

Details

- **Genome restriction.** Only supported for `"BSgenome.Hsapiens.NCBI.GRCh38"`; the function stops otherwise (checked via `qsea::getParameters()`).
- **Regions used.** Uses `mesa::hg38UltraStableProbes`, a GRCh38 set derived from array data (*Edgar et al.*, 2014).
- **Computation.** Windows overlapping the ultra-stable set are extracted, beta values are computed via `qsea::makeTable(norm_methods = "beta")`, windows with `CpG_density` < `minDensity` are dropped, `NA` beta are treated as `0` for thresholding, and the per-sample fraction with beta >= `minBeta` is returned.

Value

The input `qseaSet` with its `sampleTable` augmented by a numeric column `hyperStableEdgar` (range 0–1) giving, per sample, the fraction of ultra-stable windows called methylated.

References

Edgar R, Tan PPC, Portales-Casamar E, Pavlidis P (2014). *Meta-analysis of human methylomes reveals stably methylated sequences surrounding CpG islands associated with high gene expression*. https://pubmed.ncbi.nlm.nih.gov/25493099/

See Also

qsea::makeTable(), qsea::getSampleTable(), qsea::getSampleNames(), hg38UltraStableProbes, getSampleQCSummary()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Run only if the example windows overlap the ultra-stable probes
exampleTumourNormal %>%
    {
        has_ov <- sum(
            GenomicRanges::countOverlaps(
                qsea::getRegions(.),
                mesa::hg38UltraStableProbes
            )
        ) > 0

        if (has_ov) addHyperStableFraction(.) else {
            message(
                "No overlap with hg38UltraStableProbes in this toy dataset;",
                " skipping."
            )
            .
        }
    } %>%
    qsea::getSampleTable() %>%
    dplyr::select(sample_name, dplyr::any_of("hyperStableEdgar")) %>%
    head()
```

`addLibraryInformation` *Add library metrics to the qseaSet sample table*

Description

Append library information to the `sampleTable` of a `qseaSet`. Columns are taken from `qseaSet@libraries$file_name`, and when available, from `qseaSet@libraries$input_file` (prefixed with `"input_"`). Existing columns are not duplicated.

Usage

```
addLibraryInformation(qseaSet)
```

Arguments

qseaSet	qseaSet A qseaSet object containing sequencing data.

Value

A qseaSet object:

- **addLibraryInformation**(): returns the same qseaSet with its sampleTable augmented by library-level columns.

Examples

```
data(exampleTumourNormal, package = "mesa")
qs <- addLibraryInformation(exampleTumourNormal)
head(qsea::getSampleTable(qs))
```

---

addMedipsEnrichmentFactors

Add MEDIPS-style enrichment metrics to a qseaSet sample table

---

Description

For each sample, compute MEDIPS-style CpG enrichment metrics (**relH**, **GoGe**, and read counts) from its BAM and append the results to the qseaSet sample metadata. Supports both pulldown (default) and input libraries.

Usage

```
addMedipsEnrichmentFactors(
  qseaSet,
  exportPath = NULL,
  nonEnrich = FALSE,
  extend = 0,
  shift = 0,
  uniq = 0,
  chr.select = NULL,
  paired = TRUE,
  file_name = "file_name",
  nCores = 1
)
```

Arguments

qseaSet	qseaSet Input object whose sample table contains BAM paths.
exportPath	character(1) or NULL Directory to export per-sample fragment-length PDFs and read-GRanges RDS files. If NULL, no files are written. **Default:** NULL.

nonEnrich	logical(1) If TRUE, treat samples as **Input** libraries (columns appended under @libraries$input_file). If FALSE, treat as **Pulldown** (columns appended under @libraries$file_name). **Default:** FALSE.
extend	integer(1) Passed to MEDIPS::getGRange(). Extension length for unpaired reads. **Default:** 0.
shift	integer(1) Passed to MEDIPS::getGRange(). Shift applied to read positions. **Default:** 0.
uniq	integer(1) Passed to MEDIPS::getGRange(). Minimum mapping uniqueness. **Default:** 0.
chr.select	character() or NULL Passed to MEDIPS range extraction; subset of chromosomes to analyse. **Default:** NULL (all chromosomes).
paired	logical(1) Whether BAMs are paired-end (uses MEDIPS::getPairedGRange()). **Default:** TRUE.
file_name	character(1) Column name in the sample table holding BAM paths when nonEnrich = FALSE. When nonEnrich = TRUE, the column input_file is used instead. **Default:** "file_name".
nCores	integer(1) Number of parallel cores for parallel::mclapply(). Set to 1 for serial. **Default:** 1.

Details

Internally calls calculateCGEnrichment() per sample to derive relH/GoGe and counts, then merges the results into the @libraries slot ($file_name for pulldown, $input_file for input when nonEnrich = TRUE). Parallelisation uses parallel::mclapply(); on non-Unix systems, nCores > 1 is ignored.

Value

A qseaSet with new MEDIPS-style metrics appended:

- **Sample/library metrics** (added under the appropriate library frame): relH, GoGe, nReads, nReadsWithoutPattern, n100bpReads, n100bpReadsWithoutPattern.
- **Provenance**: any export artefacts (PDF/RDS) written to exportPath if provided.

See Also

calculateCGEnrichment(), calculateCGEnrichmentGRanges()

Examples

```
# Requires BAM files and the MEDIPS package; see \dontrun{} for a full
# usage example.
## Not run:
if (requireNamespace("MEDIPS", quietly = TRUE) &&
    requireNamespace("BSgenome.Hsapiens.UCSC.hg19", quietly = TRUE)) {
    bam <- system.file(
        "extdata",
        "hESCs.Input.chr22.bam",
```

```
        package = "MEDIPSData"
    )

    data.frame(
        sample_name = "hESC_Input_chr22",
        file_name = bam,
        group = "Input",
        input_file = bam,
        stringsAsFactors = FALSE
    ) %>%
        qsea::createQseaSet("BSgenome.Hsapiens.UCSC.hg19") %>%
        addMedipsEnrichmentFactors(
            exportPath = tempdir(),
            chr.select = paste0("chr22"),
            paired     = FALSE
            # If your qsea returns a BSgenome object and as.character()
            # doesn't help, uncomment:
            # , BSgenome_pkg = "BSgenome.Hsapiens.UCSC.hg19"
        ) %>%
        qsea::getSampleTable() %>%
        print()
}

## End(Not run)
```

`addNormalisation` *Add qsea normalisation steps with defaults*

Description

Apply a set of qsea normalisation steps to a `qseaSet`, including:

- setting library factors to 1 (disabling TMM),
- adding offsets based on an enrichment pattern, and
- configuring enrichment parameters across a CpG-density window range.

Usage

```
addNormalisation(
  qseaSet,
  enrichmentMethod = "blind1-15",
  maxPatternDensity = 0.05
)
```

Arguments

`qseaSet`	`qseaSet` Input object containing methylation-enriched sequencing data.

enrichmentMethod
: character(1) Method used to calculate enrichment. Options are described in **Details**. Recorded in qseaSet@parameters$enrichmentMethod. **Default:** "blind1-15".

maxPatternDensity
: numeric(1) Maximum pattern density in a window to consider it for the background calculation, passed to qsea::addOffset(). **Default:** 0.05.

Details

This function encapsulates a recommended default pipeline for qsea normalisation. It ensures consistency across runs and simplifies code by bundling commonly applied steps.

#' Methods available for enrichmentMethod:

- **"blind1-15"**: the blind calibration method detailed in the qsea paper, fitting a straight line of decreasing expected average methylation levels from ~76% at CpG density = 1 to ~25% at CpG density = 15.
- **"none"**: no enrichment normalisation is performed.

Value

A qseaSet with updates to:

- **Library factors** — set to 1 via qsea::addLibraryFactors().
- **Offsets** — added via qsea::addOffset() using getPattern() and maxPatternDensity.
- **Enrichment parameters** — configured over windows with CpG density in [1, 15] using a linear signal schedule and bins = seq(0.5, 40.5, 0.5).
- **Parameters slot** — enrichmentMethod recorded in qseaSet@parameters$enrichmentMethod.

See Also

qsea::addLibraryFactors(), qsea::addOffset(), qsea::addEnrichmentParameters(), getPattern()

Examples

```
# addNormalisation requires windows spanning a range of CpG densities for
# background estimation; the pre-filtered example datasets lack these, so
# a synthetic qseaSet is used here.

# Run normalisation on a toy qseaSet with ~100k reads
qsea::getExampleQseaSet(expSamplingDepth = 100000) %>%
    addNormalisation(maxPatternDensity = 0.5)

# Run with no enrichment normalisation
qsea::getExampleQseaSet(expSamplingDepth = 1e5) %>%
    addNormalisation(enrichmentMethod = "none", maxPatternDensity = 0.5)

# Access the recorded enrichment method
qsea::getExampleQseaSet(expSamplingDepth = 1e5) %>%
    addNormalisation(maxPatternDensity = 0.5) %>%
```

```
  getParameters() %>%
  purrr::pluck("enrichmentMethod")
```

---

addSummaryAcrossWindows

Append window summaries to the sample table

Description

Compute per-sample summaries across windows and left-join them to the qseaSet sample table.

Usage

```
addSummaryAcrossWindows(
  qseaSet,
  regionsToOverlap = NULL,
  fn = mean,
  suffix = "",
  normMethod = c("nrpm", "beta"),
  naMethod = "impute",
  minEnrichment = 3
)
```

Arguments

qseaSet	qseaSet. Input object providing window-level signal. **Default:** none (must be supplied).
regionsToOverlap	GRanges, data.frame, or NULL. Regions used to restrict the windows before summarising. Data frames must be coercible to GRanges (e.g., have seqnames/start/end or chr/window_start/window_end). If NULL, use all windows in qseaSet. **Default:** NULL.
fn	function. Summary function applied **per sample** over the selected windows (e.g., mean, median, sd). Provide NA handling inside fn if needed, e.g., function(x) mean(x, na.rm = TRUE). **Default:** mean.
suffix	character(1). Optional string appended to output column names (e.g., a region label) so you can call this repeatedly for different regions. **Default:** "".
normMethod	character(). One or more measures to summarise. Typically a subset of c("nrpm","beta"). **Default:** c("nrpm","beta").
naMethod	character(1). How to treat missing values prior to/within summarisation. Supported values: "na.rm" — call fn with na.rm = TRUE (when supported); "drop" — drop windows with any NA across selected methods; "impute" — replace missing values via the package's internal strategy before summarising. **Default:** "impute".
minEnrichment	integer(1). Minimum reads per window required for a non-NA beta (relevant when normMethod includes "beta"). **Default:** 3.

Details

Internally, this calls summariseAcrossWindows() to compute per-sample summaries for the requested normMethod(s), then left-joins those summaries to the sampleTable. Output column names typically encode the method and the summary function (and include suffix when provided).

Value

A qseaSet with new summary columns appended to its sampleTable.

See Also

summariseAcrossWindows(), getDataTable()

Other window-summaries: countWindowsAboveCutoff(), summariseAcrossWindows()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Add mean NRPM and beta across all windows
exampleTumourNormal %>%
  addSummaryAcrossWindows(
    fn         = mean,
    normMethod = c("nrpm","beta")
  ) %>%
  getSampleTable()

# Maximum NRPM across a small genomic slice
exampleTumourNormal %>%
  addSummaryAcrossWindows(
    regionsToOverlap = data.frame(
        seqnames = 7,
        start = 25002001,
        end = 25017900
    ),
    fn         = max,
    normMethod = "nrpm"
  ) %>%
  getSampleTable()

# Add summaries repeatedly with different regions using a suffix
exampleTumourNormal %>%
  addSummaryAcrossWindows(
    regionsToOverlap = data.frame(
        seqnames = 7,
        start = 25002001,
        end = 25017900
    ),
    fn         = median,
    normMethod = "nrpm",
    suffix     = "subset"
  ) %>%
```

```
  addSummaryAcrossWindows(
    fn         = median,
    normMethod = "nrpm",
    suffix     = "all"
  ) %>%
  getSampleTable()
```

`annotateWindows` *Annotate genomic windows using ChIPseeker (with optional CpG/FANTOM context)*

Description

Uses `ChIPseeker::annotatePeak()` to assign transcript-relative annotations (promoter, exon, intron, intergenic, etc.) to genomic windows in `dataTable`. If a genome is specified or globally set via `setMesaGenome()`, defaults and convenience datasets are applied for GRCh38. CpG island context and FANTOM enhancer overlaps can be added if the corresponding ranges are provided (or left `NULL` to use mesa defaults for GRCh38).

Usage

```
annotateWindows(
  dataTable,
  genome = .getMesaGenome(),
  TxDb = .getMesaTxDb(),
  annoDb = .getMesaAnnoDb(),
  CpGislandsGR = NULL,
  FantomRegionsGR = NULL
)
```

Arguments

`dataTable`	`data.frame`/`tibble` (coercible to `GRanges`) or `GRanges`. Windows to annotate. Data frames must include `seqnames`, `start`, and `end` (or columns convertible by `qseaTableToChrGRanges()`).
`genome`	`character(1)` or `NULL`. Guides annotation defaults. Currently supports `"hg38"`/`"GRCh38"`. **Default:** value set by `setMesaGenome()` (via internal `.getMesaGenome()`), or `NULL`.
`TxDb`	TxDb object or `character(1)`. Either an unquoted TxDb object or a string like `"TxDb.Hsapiens.UCSC.hg38.knownGene"`. If character, it is resolved at runtime. **Default:** value set by `setMesaTxDb()` (via `.getMesaTxDb()`), or for GRCh38/hg38 when `NULL`, use `TxDb.Hsapiens.UCSC.hg38.knownGene::TxDb.Hsapiens.UCSC.hg38.` if installed.
`annoDb`	`character(1)` or `NULL`. OrgDb package name (e.g., `"org.Hs.eg.db"`). **Default:** value set by `setMesaAnnoDb()` (via `.getMesaAnnoDb()`), or for GRCh38/hg38 when `NULL`, use `"org.Hs.eg.db"` if installed.

CpGislandsGR GRanges or NULL. CpG island regions for island/shore/shelf context. **Default:** NULL (for GRCh38/hg38, uses mesa::hg38CpGIslands).

FantomRegionsGR GRanges or NULL. FANTOM enhancer regions for overlap counts. **Default:** NULL (for GRCh38/hg38, uses mesa::FantomRegions).

Details

- **Conversion:** Non-GRanges inputs are converted via qseaTableToChrGRanges().
- **Genome-aware defaults (GRCh38/hg38):** Missing TxDb/annoDb/contexts are filled using the packages noted above when installed.
- **Seqlevels style:** Output seqnames have the "chr" prefix removed (e.g., "chr1" → "1").
- **ChIPseeker call:** Uses tssRegion = c(-2000, 500), level = "transcript", overlap = "all", verbose = FALSE.
- **Output columns:** Includes annotation (full label), shortAnno (without the trailing parenthetical), and—when contexts are provided—landscape (Island/Shore/Shelf/Open Sea) and inFantom (overlap count with FANTOM).

Value

A tibble with the input windows augmented by ChIPseeker annotations and, when available, CpG island landscape and Fantom overlap.

See Also

ChIPseeker::annotatePeak(), setMesaGenome(), setMesaTxDb(), setMesaAnnoDb(), qseaTableToChrGRanges(), liftOverHg19()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Derive some regions (e.g., DMRs) then annotate using GRCh38 defaults
exampleTumourNormal %>%
  calculateDMRs(variable = "tumour", contrasts = "first") %>%
  annotateWindows(genome = "hg38")

# Or specify TxDb/annoDb explicitly
exampleTumourNormal %>%
  calculateDMRs(variable = "tumour", contrasts = "first") %>%
  annotateWindows(
    TxDb   = "TxDb.Hsapiens.UCSC.hg38.knownGene",
    annoDb = "org.Hs.eg.db"
  )

# Mouse example (mm10): supply mouse TxDb and OrgDb
data(exampleMouse, package = "mesa")
exampleMouse %>%
  getRegions() %>%
  annotateWindows(
```

```
    TxDb   = "TxDb.Mmusculus.UCSC.mm10.knownGene",
    annoDb = "org.Mm.eg.db"
  )

# You can also set defaults globally using setMesaTxDb and setMesaAnnoDb:
  setMesaGenome("hg38");
  setMesaTxDb("TxDb.Hsapiens.UCSC.hg38.knownGene");
  setMesaAnnoDb("org.Hs.eg.db")
```

`arrange.qseaSet` *Arrange (reorder) samples in a qseaSet via dplyr syntax*

Description

Extends `dplyr::arrange()` to reorder **samples** of a `qseaSet` based on columns in its `sampleTable`.

Usage

```
## S3 method for class 'qseaSet'
arrange(.data, ..., .by_group = FALSE)
```

Arguments

`.data`	`qseaSet`. The qseaSet whose samples will be reordered.
`...`	Expressions. Variables or helper functions passed to `dplyr::arrange()` to define the ordering. **Default:** none (no reordering).
`.by_group`	`logical(1)`. Grouped arrangement is not implemented for `qseaSet`; this argument is ignored. **Default:** `FALSE`.

Value

A `qseaSet` object with samples reordered. Both the `count_matrix` and the `sampleTable` are updated consistently.

See Also

`qsea::getSampleTable()`, `subsetQset()`, `sort.qseaSet()` (for natural sorting of sample names).

Examples

```
data(exampleTumourNormal, package = "mesa")

# Reorder samples by age
exampleTumourNormal %>% arrange(age)

# Reorder by tissue
```

```
exampleTumourNormal %>% arrange(tissue)

# Reorder by tissue (descending), then tumour type
exampleTumourNormal %>% arrange(desc(tissue), tumour)
```

---

asValidGranges *Coerce common tabular inputs to GRanges*

Description

Accept a GRanges or a data frame containing window coordinates and return a valid GenomicRanges::GRanges. Supported data-frame schemas:

- seqnames, start, end
- chr, start, end (renamed to seqnames)
- chr, window_start, window_end (renamed to seqnames/start/end)

Usage

```
asValidGranges(object)
```

Arguments

object	GRanges **or** data.frame with window coordinates. **Default:** none (must be supplied).

Value

A GenomicRanges::GRanges built from object. Errors if no supported schema is found.

See Also

as_granges, qseaTableToChrGRanges

Examples

```
# From GRanges (passes through)
GenomicRanges::GRanges("chr1", IRanges::IRanges(100, 200)) %>%
    asValidGranges()

# From data.frame (seqnames/start/end)
data.frame(seqnames = "chr2", start = 10, end = 20) %>%
    asValidGranges()

# From data.frame (chr/window_start/window_end)
data.frame(chr = "1", window_start = 1000, window_end = 1100) %>%
    asValidGranges()
```

BSgenome.Hsapiens.UCSC.hg19.CpG.distribution *Pre-calculated CpG distribution values for BSgenomes*

Description

A data frame containing cached relH and GoGe values for different BSgenomes, generated with calculateGenomicCGDistribution

Usage

```
BSgenome.Hsapiens.UCSC.hg19.CpG.distribution

BSgenome.Hsapiens.NCBI.GRCh38.CpG.distribution

BSgenome.Mmusculus.UCSC.mm10.CpG.distribution
```

Format

A data frame with 1 row and 2 columns

A data frame with 1 row and 2 columns

A data frame with 1 row and 2 columns

calculateCGEnrichment *CpG enrichment from a BAM file (MEDIPS-style)*

Description

Compute CpG enrichment metrics (relH and GoGe) from aligned reads in a BAM file. Uses **MEDIPS** to obtain fragment ranges and **Biostrings** to interrogate the reference genome. Optionally exports a fragment-length density plot (PDF) and a serialized RDS with the GRanges of reads.

Usage

```
calculateCGEnrichment(
  file = NULL,
  BSgenome = NULL,
  exportPath = NULL,
  extend = 0,
  shift = 0,
  uniq = 0,
  chr.select = NULL,
  paired = TRUE
)
```

Arguments

`file`	`character(1)` Path to the BAM file. **Default:** `NULL` (must be supplied).
`BSgenome`	`character(1)` Name of a BSgenome package, e.g. `"BSgenome.Hsapiens.NCBI.GRCh38"`. For GRCh38 and hg19, precomputed distributions are cached within **mesa** for speed; otherwise, `calculateGenomicCGDistribution()` is used. **Default:** `NULL`.
`exportPath`	`character(1)` or `NULL` Directory in which to write a fragment-length density PDF and an RDS file containing the read GRanges. If `NULL`, no files are written. **Default:** `NULL`.
`extend`	`integer(1)` Passed to `MEDIPS::getGRange()`. Extension length for unpaired reads. **Default:** `0`.
`shift`	`integer(1)` Passed to `MEDIPS::getGRange()`. Shift applied to read positions. **Default:** `0`.
`uniq`	`integer(1)` Passed to `MEDIPS::getGRange()`. Minimum mapping uniqueness. **Default:** `0`.
`chr.select`	`character()` or `NULL` Passed to `MEDIPS::getGRange()`. Subset of chromosomes to use. **Default:** `NULL` (all chromosomes).
`paired`	`logical(1)` Whether BAM contains paired-end reads (passed to `MEDIPS::getPairedGRange()`). **Default:** `TRUE`.

Details

Reads are scanned for `"CG"` dinucleotides. Counts are normalised against genome-wide expectations (relH, GoGe).

- For **GRCh38** and **hg19**, cached genomic distributions are bundled with **mesa** for efficiency.
- For other genomes, `calculateGenomicCGDistribution()` is invoked.

Value

A `data.frame` with columns:

- **file** — input BAM path.
- **relH** — sample relH normalised by genome relH.
- **GoGe** — sample GoGe normalised by genome GoGe.
- **nReads** — total reads considered.
- **nReadsWithoutPattern** — reads lacking `"CG"` motif.
- **n100bpReads** — reads with length >= 100 bp.
- **n100bpReadsWithoutPattern** — reads >= 100 bp lacking `"CG"`.

See Also

`calculateCGEnrichmentGRanges()`, `calculateGenomicCGDistribution()`, **MEDIPS**, **BSgenome**

Examples

```
if (requireNamespace("MEDIPS", quietly = TRUE) &&
    requireNamespace("MEDIPSData", quietly = TRUE) &&
    requireNamespace("BSgenome.Hsapiens.UCSC.hg19", quietly = TRUE) &&
    require("GenomicRanges", quietly = TRUE)) {
    calculateCGEnrichment(
        file = system.file(
            "extdata",
            "hESCs.Input.chr22.bam",
            package = "MEDIPSData"
        ),
        BSgenome   = "BSgenome.Hsapiens.UCSC.hg19",
        exportPath = tempdir(),
        paired     = FALSE
    )
}
```

---

calculateCGEnrichmentGRanges

CpG enrichment from GRanges of reads (MEDIPS-style)

Description

Compute CpG enrichment metrics (**relH** and **GoGe**) from a GenomicRanges::GRanges of read spans, using the specified BSgenome. Useful when reads are already represented as genomic ranges rather than extracted directly from a BAM file.

Usage

```
calculateCGEnrichmentGRanges(
  readGRanges = NULL,
  BSgenome = NULL,
  chr.select = NULL
)
```

Arguments

`readGRanges`	`GRanges` Genomic ranges representing fragments (each range = one fragment). **Default:** `NULL` (must be supplied).
`BSgenome`	`character(1)` Name of a BSgenome package, e.g. `"BSgenome.Hsapiens.NCBI.GRCh38"`. The package must be installed and loadable. **Default:** `NULL`.
`chr.select`	`character()` or `NULL` Vector of chromosomes to restrict motif calculation. **Default:** `NULL` (use all chromosomes).

Details

relH and GoGe are computed by counting "CG" dinucleotides in the provided fragments and normalising by genome-wide expectations (from calculateGenomicCGDistribution()). Unlike calculateCGEnrichment(), this function assumes reads are already summarised as genomic ranges.

Value

A tibble with one row and the following columns:

- **relH** — sample relH normalised by genome relH.
- **GoGe** — sample GoGe normalised by genome GoGe.
- **nReads** — total reads (fragments).
- **nReadsWithoutPattern** — reads lacking "CG" motif.
- **n100bpReads** — reads with length >= 100 bp.
- **n100bpReadsWithoutPattern** — reads >= 100 bp lacking "CG".

See Also

calculateCGEnrichment(), calculateGenomicCGDistribution(), **GenomicRanges**, **BSgenome**

Examples

```
# Runnable toy example with synthetic reads over chr1 (requires a BSgenome)
if (requireNamespace("BSgenome.Hsapiens.NCBI.GRCh38", quietly = TRUE)) {
    n <- 200
    gr <- GenomicRanges::GRanges(
        seqnames = rep(1, n),
        ranges = IRanges::IRanges(
            start = sample(1e6:2e6, n),
            width = sample(80:180, n, replace = TRUE)
        ),
        strand = "*"
    )
    calculateCGEnrichmentGRanges(
        readGRanges = gr,
        BSgenome    = "BSgenome.Hsapiens.NCBI.GRCh38",
        chr.select  = 1
    )
}
```

---

calculateDMRs *Fit GLM and return DMR data in one step*

Description

Calculate one or more contrasts to find Differentially Methylated Regions (DMRs).

Usage

```
calculateDMRs(
  qseaSet,
  variable = NULL,
  covariates = NULL,
  contrasts = NULL,
  minReadCount = 0,
  minNRPM = 1,
  checkPVals = TRUE,
  FDRthres = 0.05,
  keepPvals = FALSE,
  formula = NULL,
  keepContrastMeans = TRUE,
  keepData = FALSE,
  keepGroupMeans = FALSE,
  direction = "both",
  calcDispersionAll = FALSE
)
```

Arguments

qseaSet	qseaSet. Input object containing counts and sample metadata.
variable	character(1) or NULL. Sample-table column used as the primary explanatory variable for contrasts. **Default:** NULL (must be supplied).
covariates	character() or NULL. Additional covariate column names (e.g., batch, patient). **Default:** NULL.
contrasts	data.frame, character(1), or NULL. Contrast specification. If a data frame, must contain columns group1 and group2. If a string, supports: • "A_vs_B" — a single explicit contrast, • "All" / "all" — all pairwise contrasts among levels of variable, • "All_vs_X" — each level vs X, • "X_vs_All" — X vs each other level, • "first" — only the first possible contrast (based on factor order). **Default:** NULL (fit the GLM without adding contrasts).
minReadCount	numeric(1). Minimum raw count required in at least one sample to keep a window. **Default:** 0.
minNRPM	numeric(1). Minimum NRPM required in at least one sample to keep a window. **Default:** 1.
checkPVals	logical(1). If TRUE, stop/warn when an excessive proportion of p-values are exactly zero (instability check). **Default:** TRUE.
FDRthres	numeric(1). False discovery rate threshold for calling significance. **Default:** 0.05.
keepPvals	logical(1). If TRUE, include raw (unadjusted) p-values in the output. **Default:** FALSE.

formula — formula or NULL. Optional model formula overriding variable/covariates. **Default:** NULL.

keepContrastMeans — logical(1). If FALSE, drop contrast mean columns from the output. **Default:** TRUE.

keepData — logical(1). If TRUE, include per-sample data columns for significant windows. **Default:** FALSE.

keepGroupMeans — logical(1). If TRUE, include group mean columns (based on variable). **Default:** FALSE.

direction — character(1). Direction for calling significance, passed to qsea::isSignificant(): one of "up", "down", or "both". **Default:** "both".

calcDispersionAll — logical(1). If TRUE, samples not present in any contrast still contribute to the initial GLM fit/dispersion estimation (so adding samples can change DMRs). **Default:** FALSE.

Details

Contrast strings are parsed as follows:

- "A_vs_B" creates a single contrast *A vs B*;
- "All"/"all" expands to all pairwise contrasts among levels of variable (via makeAllContrasts());
- "All_vs_X" creates *each level vs X*;
- "X_vs_All" creates *X vs each other level*;
- "first" constructs only the first possible pair (based on factor order). Internally, this function:

1. filters windows by minReadCount/minNRPM, 2) fits a Negative Binomial Generalised Linear Model including dispersion estimates, 3) adds contrasts as requested, calculating adjusted p-values (adjPval), log2 fold changes (log2FC) and differences in beta values (deltaBeta), 4) creates an output table showing any window which is differentially methylated (with adjusted p-value below FDRthres) in at least one contrast. Additional columns will be added containing data values if keepData and keepGroupMeans are true.

Value

A tibble with one row per **significant** region/window containing:

- per-contrast statistics (e.g., log2FC, deltaBeta, test statistic),
- multiple-testing results (FDR; optionally raw p-values if keepPvals),
- optional group means (if keepGroupMeans = TRUE),
- optional per-sample columns (if keepData = TRUE),
- metadata columns for genomic coordinates and window attributes.

See Also

Other DMR-detection: fitQseaGLM(), getDMRsData(), makeAllContrasts()

Examples

```
data(exampleTumourNormal, package = "mesa")

# A single explicit contrast
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "LUAD_vs_NormalLung")

# All pairwise contrasts among levels of 'type'
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "all")

# Tighter FDR threshold
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "all", FDRthres = 1e-4)

# Return all per-sample columns for the first available contrast on 'tumour'
exampleTumourNormal %>%
    calculateDMRs(variable = "tumour", contrasts = "first", keepData = TRUE)
```

---

calculateFractionReadsInGRanges

Fraction of thresholded windows overlapping a set of regions

Description

For each sample, compute the proportion of **windows** with counts >= numCountsNeeded that overlap regionsToOverlap, relative to all windows with counts >= numCountsNeeded in that sample.

Usage

```
calculateFractionReadsInGRanges(qseaSet, regionsToOverlap, numCountsNeeded)
```

Arguments

qseaSet	qseaSet. Input qseaSet. **Default:** none (must be supplied).
regionsToOverlap	GRanges or data.frame. Regions to consider for overlap. Data frames must be coercible to GRanges (e.g., have seqnames/start/end or chr/window_start/window_end). **Default:** none (must be supplied).
numCountsNeeded	integer(1). Minimum reads per window for it to count toward the fraction. **Default:** none (must be supplied).

Details

Operates on the window × sample count matrix from `qsea::getCounts()`. Counts are converted to a logical indicator (count >= `numCountsNeeded`) and summed per sample:

- `initialOverBackNum`: windows meeting the threshold genome-wide.
- `afterOverBackNum`: windows meeting the threshold **within** `regionsToOverlap` (via `filterByOverlaps()`). The reported `fraction` is `afterOverBackNum / initialOverBackNum`.

Note: despite the function name, this computes a **window-based** fraction using a count threshold, not a direct fraction of raw reads.

Value

A tibble with one row per sample containing:

- `sample_name`
- `initialOverBackNum` — windows >= threshold genome-wide
- `afterOverBackNum` — windows >= threshold within `regionsToOverlap`
- `fraction` — `afterOverBackNum / initialOverBackNum` followed by columns from the sample table (left-joined by `sample_name`).

See Also

`qsea::getCounts()`, `qsea::getSampleTable()`, `filterByOverlaps()`, hg38CpGIslands, `subsetWindowsBySignal()`

Other window-helpers: `convertToArrayBetaTable()`, `downSample()`, `subsetWindowsBySignal()`, `subsetWindowsOverBackground()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Using the shipped ultra-stable probes (GRCh38) as the region set
exampleTumourNormal %>%
  calculateFractionReadsInGRanges(
    regionsToOverlap = mesa::hg38CpGIslands,
    numCountsNeeded   = 5
  ) %>%
  dplyr::select(sample_name, initialOverBackNum, afterOverBackNum, fraction)

# Define a small GRanges subset from the object's own windows
gr <- exampleTumourNormal %>%
  qsea::getRegions() %>%
  head(n = 100)

calculateFractionReadsInGRanges(
      qseaSet           = exampleTumourNormal,
      regionsToOverlap  = gr,
      numCountsNeeded   = 3) %>%
  dplyr::select(sample_name, initialOverBackNum, afterOverBackNum, fraction)
```

`calculateGenomicCGDistribution` *Genome-wide CpG statistics (relH and GoGe)*

Description

Compute genome-wide CpG metrics for a given BSgenome:

- **relH**: relative CpG frequency (% of dinucleotides that are `"CG"`).
- **GoGe**: enrichment statistic $(nCG * genome_length)/(nC * nG)$.

Usage

```
calculateGenomicCGDistribution(BSgenome)
```

Arguments

`BSgenome`	`character(1)` The name of a BSgenome package, e.g. `"BSgenome.Hsapiens.NCBI.GRCh38"`. The package must be installed and loadable in the current session.

Details

These values are used as denominators to normalise sample-level CpG enrichment.

- Chromosomes with names containing `"rand"` or `"chrUn"` are excluded.
- The BSgenome indicated by `BSgenome` is loaded dynamically.
- Uses **Biostrings** and **BSgenome** utilities to count CpG, C, and G occurrences across the genome.

Value

A `data.frame` with two numeric columns:

- **genome.relH** — percent of `"CG"` dinucleotides genome-wide.
- **genome.GoGe** — GoGe enrichment statistic.

See Also

`calculateCGEnrichment()`, `calculateCGEnrichmentGRanges()`, **BSgenome**, **Biostrings**

Examples

```
# Example: compute genome-wide CpG metrics for a S. cerevisae genome.
# It could be done for GRCh38 genome instead, but it will take more time to
# run (package must be installed)
if (requireNamespace("BSgenome.Scerevisiae.UCSC.sacCer3", quietly = TRUE)) {
    calculateGenomicCGDistribution("BSgenome.Scerevisiae.UCSC.sacCer3")
}
```

colnames,qseaSet-method *Column names of a qseaSet sample table*

Description

S4 method for `colnames` that returns the column names of a `qseaSet`'s sample metadata table (i.e., `qsea::getSampleTable()`).

Usage

```
## S4 method for signature 'qseaSet'
colnames(x)
```

Arguments

`x`	`qseaSet`. The qseaSet object whose `sampleTable` column names are to be retrieved.

Value

`character()`. A character vector containing the column names of the sample table.

See Also

`qsea::getSampleTable()`, `qsea::getSampleNames()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Retrieve the column names of the sample metadata
colnames(exampleTumourNormal)
```

combineQsets *Combine two qseaSets*

Description

Merge two `qseaSet` objects (or `.rds` files containing them) into a single `qseaSet`. This is the pairwise worker used internally by `combineQsetsList()`.

Usage

```
combineQsets(
  qseaSet1,
  qseaSet2,
  checkParams = FALSE,
  regionsToKeep = NULL,
  dropDuplicates = FALSE
)
```

Arguments

qseaSet1	qseaSet or character(1) Either a qseaSet object, or path to an .rds file containing one.
qseaSet2	qseaSet or character(1) Either a qseaSet object, or path to an .rds file containing one.
checkParams	logical(1) If TRUE, enforce identical global parameters across inputs. **Default:** FALSE.
regionsToKeep	GRanges or coercible If supplied, restrict both qseaSet1 and qseaSet2 to these genomic regions before combining. Accepts a GRanges or a data frame with seqnames, start, end. **Default:** NULL.
dropDuplicates	logical(1) If TRUE, drop duplicate sample names. If FALSE, duplicates are retained but renamed with suffix "_Dup". **Default:** FALSE.

Value

A qseaSet object containing:

- Samples from both inputs.
- Regions restricted by regionsToKeep if supplied.
- Duplicates handled according to dropDuplicates.
- Global parameters optionally checked for consistency.

See Also

combineQsetsList() for merging more than two inputs.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Split a cohort into Tumour and Normal, then combine back
tumours <- exampleTumourNormal %>% filter(tumour == "Tumour")
normals <- exampleTumourNormal %>% filter(tumour == "Normal")
combineQsets(tumours, normals)
```

`combineQsetsList` *Combine multiple qseaSets*

Description

Merge a list of `qseaSet` objects (or `.rds` files containing them) into a single `qseaSet`. Useful when cohorts were processed separately (e.g., tumour/normal, multiple runs) and need to be combined into one container.

Usage

```
combineQsetsList(
  qseaSets,
  firstQset = NULL,
  dropDuplicates = TRUE,
  checkParams = TRUE,
  regionsToKeep = NULL
)
```

Arguments

`qseaSets`	`list` List of `qseaSet` objects, or `character()` paths to `.rds` files containing `qseaSet` objects. **Default:** none (must be supplied).
`firstQset`	`qseaSet` or `character(1)` Optional initial `qseaSet` to merge into. If a `character(1)` ending in `.rds`, it is loaded via `readr::read_rds()`. **Default:** `NULL`.
`dropDuplicates`	`logical(1)` If `TRUE`, drop samples with duplicated names across inputs. If `FALSE`, duplicates are renamed by appending `"_Dup"`. **Default:** `TRUE`.
`checkParams`	`logical(1)` Verify that global parameters are identical across inputs. **Default:** `TRUE`.
`regionsToKeep`	`GRanges` or coercible If supplied, restrict each `qseaSet` to these genomic regions *before* combining (saves memory). Accepts `GRanges` or a data frame with `seqnames`, `start`, `end`. **Default:** `NULL` (keep all regions).

Value

A `qseaSet` object:

- **Combined samples** from all inputs (or from `firstQset` plus inputs).
- **Duplicates** handled according to `dropDuplicates`.
- **Regions** optionally prefiltered by `regionsToKeep` prior to merge.

See Also

`combineQsets()` for pairwise merging.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Split a cohort into Tumour and Normal, create a list, then combine back
tumours <- exampleTumourNormal %>% filter(tumour == "Tumour")
normals <- exampleTumourNormal %>% filter(tumour == "Normal")
qsetList <- list(tumours, normals)
combineQsetsList(qsetList)
```

---

convertToArrayBetaTable

Convert qsea beta values to array-like probe matrix

Description

Build a probe × sample beta table by mapping `qseaSet` windows to array probe coordinates. Currently supports Illumina **Infinium 450k** probes on GRCh38; a custom `GRanges` of probes can also be supplied.

Usage

```
convertToArrayBetaTable(qseaSet, arrayDetails = "Infinium450k")
```

Arguments

`qseaSet`	`qseaSet`. Input object providing per-window beta values. **Default:** none (must be supplied).
`arrayDetails`	`character(1)` or `GRanges`. Either a recognised keyword (currently `"Infinium450k"`) or a `GRanges` of probe loci with an `ID` metadata column of probe identifiers. Coordinates must match the `qseaSet` genome (e.g., GRCh38). **Default:** `"Infinium450k"`.

Details

Probe coordinates are matched to `qseaSet` windows (e.g., via overlap). When multiple windows are relevant for a probe, the package's implementation is used to derive a single beta value per probe. Ensure the probe genome build matches the `qseaSet` (e.g., GRCh38) to avoid mismatches.

Value

A `data.frame` with one row per probe (including an `ID` column) and one column per sample containing the corresponding beta values. Probes with no matching/overlapping window receive `NA`.

See Also

`getDataTable()`, dataset `mesa::hg38_450kArrayGR`

Other window-helpers: `calculateFractionReadsInGRanges()`, `downSample()`, `subsetWindowsBySignal()`, `subsetWindowsOverBackground()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Use the built-in keyword for 450k probes (GRCh38)
exampleTumourNormal %>%
  convertToArrayBetaTable(arrayDetails = "Infinium450k") %>%
  head()

# Supply an explicit GRanges of 450k probes (must have metadata column 'ID')
exampleTumourNormal %>%
  convertToArrayBetaTable(arrayDetails = mesa::hg38_450kArrayGR) %>%
  head()
```

`countWindowsAboveCutoff`

Count windows above a cutoff

Description

For a given set of genomic windows, count (per sample) how many windows exceed a chosen threshold using the selected normalisation/measure.

Usage

```
countWindowsAboveCutoff(
  qseaSet,
  GRanges,
  samples = NULL,
  cutoff = 0,
  normMethod = "nrpm"
)
```

Arguments

`qseaSet`	`qseaSet`. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
`GRanges`	`GenomicRanges::GRanges`. Windows to count over. **Default:** none (must be supplied).
`samples`	`character()` or `NULL`. Sample names to include, or a single string used as a pattern to match sample names. If `NULL`, all samples are used. **Default:** `NULL`.
`cutoff`	`numeric(1)`. Threshold; windows with value `>= cutoff` are counted. **Default:** `0`.
`normMethod`	`character(1)`. Measure to use. One of `"nrpm"` or `"beta"`. **Default:** `"nrpm"`.

Details

The function builds a window × sample matrix over GRanges for the chosen normMethod, optionally restricts to samples, and counts, per sample, the number of windows with value >= cutoff. It also reports how many windows were evaluated (totalWindowsUsed). Results are joined with the sample table.

Value

A tibble with one row per sample containing:

- sample_name
- numOverCutoff — number of windows >= cutoff
- totalWindowsUsed — number of windows evaluated followed by columns from the sample table (left-joined).

See Also

getDataTable(), summariseAcrossWindows()

Other window-summaries: addSummaryAcrossWindows(), summariseAcrossWindows()

Examples

```
data(exampleTumourNormal, package = "mesa")

#' # Count windows with NRPM >= 1 over the first 100 regions
countWindowsAboveCutoff(exampleTumourNormal,
                                qsea::getRegions(exampleTumourNormal)[1:100],
                                cutoff = 1,
                                normMethod = "nrpm")
```

---

downSample *Downsample reads in a qseaSet*

Description

Randomly subsample reads to a fixed depth across all samples in a qseaSet. Useful for equalising library sizes prior to comparison.

Usage

```
downSample(qseaSet, nReads)
```

Arguments

qseaSet	qseaSet. Input object whose per-window counts will be downsampled. **Default:** none (must be supplied).
nReads	integer(1). Target number of reads to retain **per sample**. If a sample has fewer than nReads, it is left unchanged. **Default:** none (must be supplied).

Details

Downsampling is performed independently per sample. To obtain reproducible results across runs, call `set.seed()` **before** invoking `downSample()`. Samples with total reads < `nReads` are not upsampled.

Value

A `qseaSet` with downsampled counts. Library metadata (`valid_fragments`, `offset`, `library_factor`) are updated accordingly.

See Also

`getDataTable()`, `subsetWindowsBySignal()`, `mixSamples()`

Other window-helpers: `calculateFractionReadsInGRanges()`, `convertToArrayBetaTable()`, `subsetWindowsBySignal()`, `subsetWindowsOverBackground()`

Examples

```
data(exampleTumourNormal, package = "mesa")

set.seed(1)
# Downsample to 100 fragments per sample, then check per-sample totals
exampleTumourNormal %>%
  downSample(100) %>%
  getCountTable() %>%
  dplyr::select(tidyselect::matches("_[NT]")) %>%
  colSums() %>%
  range()  # should be ~ c(100, 100) if all samples had >= 100 reads

# Also inspect the updated library sizes recorded in the sample table
exampleTumourNormal %>%
  downSample(100) %>%
  qsea::getSampleTable() %>%
  dplyr::select(patient, type, gender)
```

ENCODEbadRegions *2019 ENCODE list of poorly mapped regions in hg38*

Description

The 2019 ENCODE list of regions that are poorly mapped (2019 https://doi.org/10.1038/s41598-019-45839-z). List downloaded from https://github.com/Boyle-Lab/Blacklist/blob/master/lists/ hg38-blacklist.v2.bed.gz and edited to remove "chr".

Usage

```
ENCODEbadRegions
```

Format

A Granges object with 636 ranges:

seqnames chromosome

ranges position on the chromosome

Source

https://github.com/Boyle-Lab/Blacklist/blob/master/lists/ hg38-blacklist.v2.bed.gz

---

exampleMouse *A small example qseaSet with 5 paired tumour/normal samples.*

Description

A qseaSet object containing a small region of chromosome 5 for a mouse cell line, 2 replicates of 2 conditions (processed within CRUK-MI CBC).

Usage

```
exampleMouse
```

Format

A qseaSet object with 153 regions and 4 samples:

---

exampleTumourNormal *A small example qseaSet with 5 paired tumour/normal samples.*

Description

A qseaSet object containing a small region of chromosome 7 for 5 paired tumour/normal samples (commercially bought from Origene and processed within CRUK-MI CBC).

Usage

```
exampleTumourNormal
```

Format

A qseaSet object with 819 regions and 10 samples:

FantomRegions *FANTOM5 regions for GRCh38*

Description

A data frame containing FANTOM5 enhancer regions, hg38

Usage

```
FantomRegions
```

Format

A data frame with 63285 rows and 3 variables:

filter.qseaSet *Filter samples in a qseaSet*

Description

Extends `dplyr::filter()` to subset **samples** in a `qseaSet` based on predicates applied to its `sampleTable`.

Usage

```
## S3 method for class 'qseaSet'
filter(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	`qseaSet` A `qseaSet` object whose samples are to be filtered.
`...`	Predicate expressions passed to `dplyr::filter()`, evaluated on the `sampleTable`. For example, `age > 65`, `tissue == "Colon"`. **Default:** none (no filtering).
`.preserve`	`logical(1)` Placeholder argument for compatibility with `dplyr::filter()`. Ignored here since grouping is not implemented. **Default:** `FALSE`.

Details

Filtering is performed on the result of `qsea::getSampleTable(.data)`. Matching rows determine the `sample_names` retained in the `qseaSet`. Grouped filtering (`group_by`) is not supported for `qseaSet`.

Value

A `qseaSet` object containing only the samples whose `sampleTable` rows satisfied the filter expressions. If no samples remain, an empty `qseaSet` is returned (with a message but not an error).

See Also

qsea::getSampleTable() to inspect metadata, subsetQset() for explicit sample subsetting, selectQset() for column selection in the sampleTable.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep only older patients
exampleTumourNormal %>% filter(age >= 70)

# Restrict to colon tissue
exampleTumourNormal %>% filter(tissue == "Colon")

# Select only tumour samples
exampleTumourNormal %>% filter(tumour != "Normal")

# Combine predicates: lung tumours only
exampleTumourNormal %>%
    filter(stringr::str_detect(sample_name, "Lung"), tumour != "Normal")

# If no rows match, returns empty qseaSet (with a message)
exampleTumourNormal %>% filter(tissue == "Liver")
```

---

filterByOverlaps *Subset a qseaSet by overlaps / non-overlaps with genomic regions*

Description

filterByOverlaps() retains windows that **overlap**, while filterByNonOverlaps() retains windows that **do not overlap** a set of genomic regions. Inputs may use different chromosome styles (with or without the "chr" prefix); styles are harmonized automatically before overlap computation.

Usage

```
filterByOverlaps(qseaSet, regionsToOverlap)

filterByNonOverlaps(qseaSet, regionsToOverlap)
```

Arguments

qseaSet	qseaSet A qseaSet object containing methylation-enriched sequencing data.
regionsToOverlap	GRanges or data.frame Genomic regions used for filtering. If a data.frame, it must contain columns seqnames (character()), start (integer()), and end (integer()); the object is coerced internally to a GRanges.

Details

Chromosome naming is aligned using GenomeInfoDb::seqlevelsStyle() so that overlap computation is consistent even when one input uses "chr1" and the other uses "1". Only the **window set** is filtered; sample metadata and counts are subset accordingly.

Value

A qseaSet object:

- **filterByOverlaps**(): returns the input qseaSet restricted to windows that **overlap** regionsToOverlap.
- **filterByNonOverlaps**(): returns the input qseaSet restricted to windows that **do not overlap** regionsToOverlap.

See Also

filter_by_overlaps, filter_by_non_overlaps, GRanges-class

Examples

```
data(exampleTumourNormal, package = "mesa")

# Data frame input
rgns <- data.frame(seqnames = 7L, start = 25002001L, end = 25017900L)
filterByOverlaps(exampleTumourNormal, regionsToOverlap = rgns)
filterByNonOverlaps(exampleTumourNormal, regionsToOverlap = rgns)

# GRanges input
gr <- GenomicRanges::GRanges("chr7", IRanges::IRanges(25002001L, 25017900L))
filterByOverlaps(exampleTumourNormal, regionsToOverlap = gr)
filterByNonOverlaps(exampleTumourNormal, regionsToOverlap = gr)
```

---

filterWindows *Filter regions (windows) inside a qseaSet*

Description

Filters the **regions** of a qseaSet using dplyr::filter() predicates applied to its regions (as a data frame), then subsets the object accordingly.

Usage

```
filterWindows(qseaSet, ...)
```

Arguments

qseaSet	qseaSet. The qseaSet object whose regions (windows) will be filtered.
...	Expressions. Predicates passed to dplyr::filter() and evaluated on the regions data frame. **Default:** none (no filtering).

Value

A `qseaSet` object with only regions matching the filter conditions. Both the `count_matrix` and `@regions` slot are updated consistently.

See Also

`qsea::getRegions()`, `filter.qseaSet()` (for filtering samples instead of regions), `dplyr::filter()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep regions starting before position 25,020,000
exampleTumourNormal %>% filterWindows(start <= 25020000)

# Keep regions with CpG density > 10
exampleTumourNormal %>% filterWindows(CpG_density > 10)

# Combine multiple conditions
exampleTumourNormal %>%
    filterWindows(seqnames == "chr7", CpG_density > 5)
```

---

fitQseaGLM *Fit a generalized linear model (GLM) to a qseaSet*

Description

Fit a negative-binomial GLM using `qsea::fitNBglm()` on counts from a qsea::qseaSet, and add one or more contrasts. This is the low-level worker used by `calculateDMRs()`.

Usage

```
fitQseaGLM(
  qseaSet,
  variable = NULL,
  covariates = NULL,
  contrasts = NULL,
  keepIndex = NULL,
  minReadCount = 0,
  minNRPM = 1,
  checkPVals = TRUE,
  formula = NULL,
  calcDispersionAll = FALSE
)
```

Arguments

qseaSet	qseaSet. Input object containing counts and sample metadata.
variable	character(1) Column name in the sample table used as the primary explanatory variable. **Default:** NULL (must be provided unless formula is supplied).
covariates	character() Additional column names (e.g., batch, patient ID) to include as covariates. **Default:** NULL.
contrasts	data.frame or NULL A data frame with columns group1 and group2 specifying contrasts to compute. If NULL, a fitted GLM with no contrasts added is returned. **Default:** NULL.
keepIndex	integer() or NULL Row indices (windows) to retain **before** fitting; when supplied, this overrides filtering by minReadCount / minNRPM. **Default:** NULL.
minReadCount	numeric(1) Minimum raw count required in at least one sample to retain a window. **Default:** 0.
minNRPM	numeric(1) Minimum NRPM required in at least one sample to retain a window. **Default:** 1.
checkPVals	logical(1) If TRUE, stop or warn if an excessive proportion of p-values are exactly zero (possible instability). **Default:** TRUE.
formula	formula or NULL Optional model formula overriding variable/covariates. **Default:** NULL.
calcDispersionAll	logical(1) If TRUE, samples not present in any specified contrast are still used to fit the initial GLM and dispersion estimates. Note this means adding samples to qseaSet can change DMRs even when they are not directly contrasted. **Default:** FALSE.

Details

- Dispersion is estimated across the windows retained after filtering.
- When calcDispersionAll = TRUE, all samples contribute to dispersion estimation even if not part of any contrast.
- Contrasts are added sequentially with qsea::addContrast().
- By default, beta-normalised counts are used as the outcome (as in qsea).

Value

A qseaGLM object containing:

- the fitted negative-binomial GLM;
- (optionally) one or more added contrasts (group1 vs group2);
- filtering metadata (based on keepIndex, minReadCount, minNRPM);
- model specification (variable, covariates, or formula).

See Also

Other DMR-detection: calculateDMRs(), getDMRsData(), makeAllContrasts()

Examples

```
data(exampleTumourNormal, package = "mesa")
qs <- exampleTumourNormal
contr <- tibble::tibble(group1 = "LUAD", group2 = "NormalLung")
fitQseaGLM(qs, variable = "type", contrasts = contr)
```

---

gc_hg38_1000kb *GC content across the human genome (hg38/GRCh38)*

Description

Fixed-size bins with average GC content per bin. Bins are ordered by chromosome size (not strictly numeric).

Usage

```
data(gc_hg38_1000kb); data(gc_hg38_500kb); data(gc_hg38_50kb)

gc_hg38_50kb

gc_hg38_500kb
```

Format

One of:

1. A `data.table`/`data.frame` with columns:

   chrom Chromosome (e.g., "chr1", ..., "chrX", "chrY").

   start 0-based start of the bin.

   end 1-based end of the bin.

   gc Average GC fraction in the bin (0–1).

2. A `GRanges` with genomic bins and `mcols(.)$gc` storing the average GC fraction (0–1) per bin.

An object of class `data.table` (inherits from `data.frame`) with 61775 rows and 4 columns.

An object of class `data.table` (inherits from `data.frame`) with 6188 rows and 4 columns.

Value

`gc_hg38_1000kb`, `gc_hg38_500kb`, and `gc_hg38_50kb` return objects as described in `@format` with bin sizes of 1000 kb, 500 kb, and 50 kb, respectively.

Source

https://github.com/broadinstitute/ichorCNA

getBetaTable *Get beta per window*

Description

Convenience wrapper around `getDataTable()` to extract **beta** values (fraction methylated). Returns a window × sample (or group) table for downstream summaries/plots.

Usage

```
getBetaTable(
  qseaSet,
  useGroupMeans = FALSE,
  minEnrichment = 3,
  addMethodSuffix = FALSE,
  verbose = TRUE
)
```

Arguments

`qseaSet`	`qseaSet`. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
`useGroupMeans`	`logical(1)`. If `TRUE`, average replicates by the `group` column in the sample table and return group means; if `FALSE`, return per-sample betas. **Default:** `FALSE`.
`minEnrichment`	`integer(1)`. Minimum reads per window required for a non-`NA` beta (below this threshold, qsea sets beta to `NA`). **Default:** `3`.
`addMethodSuffix`	`logical(1)`. If `TRUE`, keep the method suffix in column names (e.g., `Sample1_beta` or `Group_beta_means`); if `FALSE`, return bare sample/group names. **Default:** `FALSE`.
`verbose`	`logical(1)`. Print progress/messages. **Default:** `TRUE`.

Value

A tibble/data.frame with one row per genomic window and columns for each sample (or group), plus any window metadata included by `getDataTable()`.

See Also

`getDataTable()`, `getCountTable()`, `getNRPMTable()`

Other table-helpers: `getCountTable()`, `getDataTable()`, `getNRPMTable()`, `getSampleGroups2()`, `makeTransposedTable()`, `removeLibraryFactors()`, `removeNormMethodSuffix()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Per-sample beta (first rows/columns), with a minimum read threshold
exampleTumourNormal %>%
  getBetaTable(minEnrichment = 3) %>%
  dplyr::select(1:6) %>%
  head()

# Group means instead of individual samples
exampleTumourNormal %>%
  getBetaTable(useGroupMeans = TRUE, minEnrichment = 3) %>%
  dplyr::select(1:6) %>%
  head()

# Keep the method suffix in column names
exampleTumourNormal %>%
  getBetaTable(addMethodSuffix = TRUE) %>%
  names() %>%
  head()

# Inspect how many NAs arise from the minEnrichment filter
exampleTumourNormal %>%
  getBetaTable(minEnrichment = 5) %>%
  dplyr::select(dplyr::where(is.numeric)) %>%
  is.na() %>%
  colSums() %>%
  head()
```

---

getCGPositions *Genomic positions of a motif (CG) from MEDIPS*

Description

Convenience wrapper that returns genomic positions of the `"CG"` motif for the specified BSgenome and chromosomes, via `MEDIPS::MEDIPS.getPositions()`.

Usage

```
getCGPositions(BSgenome, chr.select)
```

Arguments

`BSgenome`	Character(1). BSgenome package name.
`chr.select`	Character vector of chromosome names to include (e.g., `paste0("chr", 1:22)`).

Value

A GRanges-class of motif positions.

See Also

calculateCGEnrichment, calculateCGEnrichmentGRanges, **MEDIPS**

Examples

```
# Requires MEDIPS and a BSgenome package
# if (requireNamespace("BSgenome.Hsapiens.NCBI.GRCh38", quietly = TRUE)) {
#   getCGPositions("BSgenome.Hsapiens.NCBI.GRCh38", chr.select = 22)
# }
```

---

getCountTable *Get counts per window*

Description

Convenience wrapper around `getDataTable()` to extract **raw counts**. Returns a window × sample (or group) table for downstream summaries/plots.

Usage

```
getCountTable(
  qseaSet,
  useGroupMeans = FALSE,
  addMethodSuffix = FALSE,
  verbose = TRUE
)
```

Arguments

`qseaSet`	`qseaSet`. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
`useGroupMeans`	`logical(1)`. If `TRUE`, average replicates by the `group` column in the sample table and return group means; if `FALSE`, return per-sample counts. **Default:** `FALSE`.
`addMethodSuffix`	`logical(1)`. If `TRUE`, keep the method suffix in column names (e.g., `Sample1_counts` or `Group_counts_means`); if `FALSE`, return bare sample/group names. **Default:** `FALSE`.
`verbose`	`logical(1)`. Print progress/messages. **Default:** `TRUE`.

Value

A tibble/data.frame with one row per genomic window and columns for each sample (or group), plus any window metadata included by `getDataTable()`.

See Also

getDataTable(), getNRPMTable(), getBetaTable()

Other table-helpers: getBetaTable(), getDataTable(), getNRPMTable(), getSampleGroups2(), makeTransposedTable(), removeLibraryFactors(), removeNormMethodSuffix()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Per-sample counts (first rows/columns)
exampleTumourNormal %>%
  getCountTable() %>%
  dplyr::select(1:6) %>%
  head()

# Group means instead of individual samples
exampleTumourNormal %>%
  getCountTable(useGroupMeans = TRUE) %>%
  dplyr::select(1:6) %>%
  head()

# Keep the method suffix in column names
exampleTumourNormal %>%
  getCountTable(addMethodSuffix = TRUE) %>%
  names() %>%
  head()
```

---

getDataTable *Extract per-window data (counts / NRPM / beta)*

Description

Build a per-window table for one or more normalisation methods, returning values per **sample** or **group means**.

Usage

```
getDataTable(
  qseaSet,
  normMethod = "nrpm",
  useGroupMeans = FALSE,
  minEnrichment = 3,
  addMethodSuffix = FALSE,
  verbose = TRUE
)
```

Arguments

qseaSet	qseaSet. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
normMethod	character(). One or more of "counts", "nrpm", "beta". **Default:** "nrpm".
useGroupMeans	logical(1). If TRUE, average replicates by the group column and return group means; if FALSE, return per-sample values. **Default:** FALSE.
minEnrichment	integer(1). Minimum reads per window required for a non-NA beta (applies only when normMethod includes "beta"). **Default:** 3.
addMethodSuffix	logical(1). Keep a method suffix in column names (e.g., Sample1_nrpm, Group_beta_means). If multiple methods are requested, suffixes are kept regardless to avoid collisions. **Default:** FALSE.
verbose	logical(1). Print progress/messages. **Default:** TRUE.

Details

The result is a **window × sample (or group)** table. Window coordinates and any available region metadata (e.g., seqnames, start, end, densities) are kept alongside the requested measures. Column naming: when a single method is requested and addMethodSuffix = FALSE, columns are bare sample / group names; otherwise a _{method} (and for group means _{method}_means) suffix is appended.

For "beta", windows with reads < minEnrichment are set to NA before aggregation. Use removeNormMethodSuffix() afterwards if you want to strip method tags in wide tables.

Value

A tibble/data.frame with one row per genomic window and one column per sample (or group), plus window metadata columns.

See Also

getCountTable(), getNRPMTable(), getBetaTable(), removeNormMethodSuffix()

Other table-helpers: getBetaTable(), getCountTable(), getNRPMTable(), getSampleGroups2(), makeTransposedTable(), removeLibraryFactors(), removeNormMethodSuffix()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Default: NRPM per window (first few columns)
exampleTumourNormal %>%
  getDataTable() %>%
  dplyr::select(1:6) %>%
  head()

# Multiple methods together (suffixes retained automatically)
exampleTumourNormal %>%
  getDataTable(normMethod = c("nrpm","beta"), minEnrichment = 3) %>%
```

```
  dplyr::select(1:8) %>%
  head()

# Group means instead of individual samples
exampleTumourNormal %>%
  getDataTable(useGroupMeans = TRUE, normMethod = "nrpm") %>%
  dplyr::select(1:6) %>%
  head()

# Keep (or later remove) the method suffix
exampleTumourNormal %>%
  getDataTable(normMethod = "nrpm", addMethodSuffix = TRUE) %>%
  names() %>%
  head()
```

`getDMRsData` *Extract DMR-level results from a fitted GLM*

Description

Given a `qsea::qseaGLM` object with one or more contrasts, extract a *wide* results table of statistics, adjusted p-values, and (optionally) group means and per-sample values.

Usage

```
getDMRsData(
  qseaSet,
  qseaGLM,
  sampleNames = NULL,
  variable = NULL,
  keepData = FALSE,
  keepGroupMeans = FALSE,
  FDRthres = 0.05,
  keepPvals = FALSE,
  keepFragmentInfo = FALSE,
  direction = "both"
)
```

Arguments

`qseaSet`	`qseaSet`. Input object used to obtain metadata and counts.
`qseaGLM`	`qseaGLM`. Fitted model returned by `fitQseaGLM()`.
`sampleNames`	`character()` or `NULL`. Sample names to include as per-sample columns. **Default:** `NULL` (chosen automatically when `keepData = TRUE`, otherwise per-sample columns are omitted).

variable — character(1) or NULL. Sample-table column on which contrasts were based (used to compute group means). **Default:** NULL (required if keepGroupMeans = TRUE).

keepData — logical(1). If TRUE, include per-sample values (beta/NRPM etc.) as columns. **Default:** FALSE.

keepGroupMeans — logical(1). If TRUE, include group mean columns for each contrast side (based on variable). **Default:** FALSE.

FDRthres — numeric(1). False discovery rate threshold used to flag significance. **Default:** 0.05.

keepPvals — logical(1). If TRUE, include raw (unadjusted) p-values in addition to FDR. **Default:** FALSE.

keepFragmentInfo — logical(1). If TRUE, include fragment/MAPQ metrics where available. **Default:** FALSE.

direction — character(1). Direction for significance calling, passed to qsea::isSignificant(): one of "up", "down", or "both". **Default:** "both".

Details

Significant rows are determined using FDRthres and direction via qsea::isSignificant(). When keepData = TRUE and sampleNames = NULL, per-sample columns are included for all samples in the fitted model (or the subset relevant to each contrast, depending on the internal representation). Group means require variable to identify the groupings used in contrasts.

Value

A tibble with one row per DMR (window) containing:

- **Per-contrast statistics** (e.g., log2FC, deltaBeta, test statistic).
- **Multiple-testing output** (FDR; optionally raw p-values if keepPvals).
- **Optional group summaries** if keepGroupMeans = TRUE.
- **Optional per-sample columns** if keepData = TRUE (for sampleNames).
- **Optional fragment/MAPQ metrics** if keepFragmentInfo = TRUE.

See Also

Other DMR-detection: calculateDMRs(), fitQseaGLM(), makeAllContrasts()

Examples

```
data(exampleTumourNormal, package = "mesa")
qs <- exampleTumourNormal
contr <- tibble::tibble(group1 = "LUAD", group2 = "NormalLung")
glmfit <- fitQseaGLM(qs, variable = "type", contrasts = contr)
getDMRsData(qs, glmfit, variable = "type", FDRthres = 0.1)
```

`getGenomicFeatureDistribution` *Summarise signal by genomic context*

Description

Annotate windows, then summarise signal and counts by CpG landscape (Island/Shore/Shelf/Open Sea) and short genomic annotation (e.g., Promoter/Exon/Intron/Intergenic).

Usage

```
getGenomicFeatureDistribution(
  qseaSet,
  cutoff = 1,
  normMethod = "nrpm",
  minEnrichment = 3
)
```

Arguments

`qseaSet`	`qseaSet`. Input object containing windows and signal. **Default:** none (must be supplied).
`cutoff`	`numeric(1)`. Threshold used to call a window "over cutoff" for counting. **Default:** `1`.
`normMethod`	`character(1)`. Measure to summarise. One of `"nrpm"` or `"beta"`. **Default:** `"nrpm"`.
`minEnrichment`	`integer(1)`. Minimum reads per window required for a non-`NA` beta (relevant when `normMethod = "beta"`). **Default:** `3`.

Details

Windows are annotated via `annotateWindows()` (which can add CpG landscape and a concise `shortAnno` label). The function then groups by `sample_name`, `landscape`, and `shortAnno` to compute:

- `nWindows`: number of windows in the group,
- `sum`: sum of the selected measure across those windows,
- `nOverCutoff`: number of windows with value `>= cutoff`. When using beta, windows failing `minEnrichment` are treated as `NA` before summarisation.

This function requires a transcript database and OrgDb to annotate windows (via `annotateWindows()`). Either:

- set them globally with `setMesaGenome()` (recommended for GRCh38/hg38), or
- provide compatible `TxDb`/`annoDb` packages yourself.

Value

A tibble with columns: `sample_name`, `landscape`, `shortAnno`, `nWindows`, `sum`, `nOverCutoff`, followed by sample-table metadata joined from `qseaSet`.

See Also

`annotateWindows()`, `summariseAcrossWindows()`, `getDataTable()`

Other annotation-summaries: `plotGenomicFeatureDistribution()`

Examples

```
# Ensure annotation defaults are available (GRCh38/hg38)
setMesaGenome("hg38")

data(exampleTumourNormal, package = "mesa")

# NRPM-based context summary
exampleTumourNormal %>%
  getGenomicFeatureDistribution(cutoff = 1, normMethod = "nrpm") %>%
  head()

# Beta-based summary (apply a minimum enrichment for non-NA betas)
exampleTumourNormal %>%
  getGenomicFeatureDistribution(
      cutoff = 0.7,
      normMethod = "beta",
      minEnrichment = 3
  ) %>%
  head()
```

`getMesaAnnoDb` *Get annotation DB for current or specified genome*

Description

Get annotation DB for current or specified genome

Usage

```
getMesaAnnoDb(genome = NULL)
```

Arguments

`genome`	Genome build, defaults to current setting

Value

Character string of annotation database name (e.g., "org.Hs.eg.db", "org.Mm.eg.db")

getMesaGenome *Get current mesa genome setting*

Description

Get current mesa genome setting

Usage

```
getMesaGenome()
```

Value

Character string of current genome setting (e.g., "hg38", "hg19", "mm10")

getMesaTxDb *Get TxDb for current or specified genome*

Description

Get TxDb for current or specified genome

Usage

```
getMesaTxDb(genome = NULL)
```

Arguments

genome	Genome build, defaults to current setting

Value

A TxDb object for the specified genome

getNRPMTable *Get NRPM per window*

Description

Convenience wrapper around `getDataTable()` to extract **NRPM** values (normalised reads per million). Returns a window × sample (or group) table for downstream summaries/plots.

Usage

```
getNRPMTable(
  qseaSet,
  useGroupMeans = FALSE,
  addMethodSuffix = FALSE,
  verbose = TRUE
)
```

Arguments

qseaSet	`qseaSet`. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
useGroupMeans	`logical(1)`. If `TRUE`, average replicates by the `group` column in the sample table and return group means; if `FALSE`, return per-sample NRPM. **Default:** `FALSE`.
addMethodSuffix	`logical(1)`. If `TRUE`, keep the method suffix in column names (e.g., `Sample1_nrpm` or `Group_nrpm_means`); if `FALSE`, return bare sample/group names. **Default:** `FALSE`.
verbose	`logical(1)`. Print progress/messages. **Default:** `TRUE`.

Value

A tibble/data.frame with one row per genomic window and columns for each sample (or group), plus any window metadata included by `getDataTable()`.

See Also

`getDataTable()`, `getCountTable()`, `getBetaTable()`

Other table-helpers: `getBetaTable()`, `getCountTable()`, `getDataTable()`, `getSampleGroups2()`, `makeTransposedTable()`, `removeLibraryFactors()`, `removeNormMethodSuffix()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Per-sample NRPM (first rows/columns)
exampleTumourNormal %>%
```

```
  getNRPMTable() %>%
  dplyr::select(1:6) %>%
  head()

# Group means instead of individual samples
exampleTumourNormal %>%
  getNRPMTable(useGroupMeans = TRUE) %>%
  dplyr::select(1:6) %>%
  head()

# Keep the method suffix in column names
exampleTumourNormal %>%
  getNRPMTable(addMethodSuffix = TRUE) %>%
  names() %>%
  head()
```

`getPattern` *Infer pattern names from region density columns*

Description

Scan the region metadata of a `qseaSet` for columns ending in `"_density"` (e.g., `"CpG_density"`) and return the base pattern names (e.g., `"CpG"`).

Usage

```
getPattern(qseaSet)
```

Arguments

`qseaSet`	`qseaSet`. Object whose regions (via `qsea::getRegions()`) may contain `*_density` columns produced by `qsea::addPatternDensity()`. **Default:** none (must be supplied).

Details

This function inspects `mcols(qsea::getRegions(qseaSet))`, selects column names matching the regex `"_density$"`, and strips that suffix. It does not compute densities or modify the object. If no matching columns exist, a zero-length character vector is returned.

Value

`character()` vector of pattern names discovered (length 0 if none).

See Also

`qsea::addPatternDensity()`, `qsea::getRegions()`, `S4Vectors::mcols()`

Examples

```
data(exampleTumourNormal, package = "mesa")
# Returns character(0) if no *_density columns are present
exampleTumourNormal %>% getPattern()
```

---

getPCA *Generate a PCA from a qseaSet*

Description

Convenience wrapper around `getDimRed()` with `method = "PCA"`.

Convenience wrapper around `getDimRed()` with `method = "UMAP"`.

Run PCA or UMAP on methylation signal matrices extracted from a qsea::qseaSet. This is the core workhorse for dimensionality reduction; wrappers such as `getPCA()` and `getUMAP()` provide simplified access.

Usage

```
getPCA(
  qseaSet,
  dataTable = NULL,
  regionsToOverlap = NULL,
  normMethod = "beta",
  minEnrichment = 3,
  useGroupMeans = FALSE,
  minDensity = 0,
  topVarNum = 1000,
  topVarSamples = NULL,
  center = TRUE,
  scale = FALSE,
  nPC = 5,
  returnDataTable = FALSE,
  verbose = TRUE
)

getUMAP(
  qseaSet,
  dataTable = NULL,
  regionsToOverlap = NULL,
  normMethod = "beta",
  minEnrichment = 3,
  useGroupMeans = FALSE,
  minDensity = 0,
  topVarNum = 1000,
  topVarSamples = NULL,
```

```
  returnDataTable = FALSE,
  verbose = TRUE,
  ...
)

getDimRed(
  qseaSet,
  dataTable = NULL,
  method = "PCA",
  regionsToOverlap = NULL,
  normMethod = "beta",
  minEnrichment = 3,
  useGroupMeans = FALSE,
  minDensity = 0,
  topVarNum = 1000,
  topVarSamples = NULL,
  center = TRUE,
  scale = FALSE,
  nPC = 5,
  returnDataTable = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

qseaSet	qseaSet. Input object providing windows, counts and sampleTable.
dataTable	data.frame or GRanges or NULL. Normalised values with windows in rows and samples in columns. Must contain seqnames, start, end (if data.frame), or metadata columns with values (if GRanges). If NULL, the matrix is derived internally (see normMethod, minEnrichment). **Default:** NULL.
regionsToOverlap	GRanges, coercible data.frame, or NULL. If supplied, only windows overlapping these regions are used. **Default:** NULL.
normMethod	character(1). Name of predefined normalisation (e.g., "beta", "nrpm"); passed to qsea::normMethod() when dataTable = NULL. **Default:** "beta".
minEnrichment	numeric(1). Minimum reads required for beta values to be non-NA; forwarded to getDataTable() when building data internally. **Default:** 3.
useGroupMeans	logical(1). If TRUE, average samples within the group column (combine replicates) before DR. **Default:** FALSE.
minDensity	numeric(1). Minimum CpG density; windows below this are removed. **Default:** 0.
topVarNum	numeric(1) or integer(1) or Inf or NULL **or** a vector. Keep the topVarNum most variable windows (by SD). If NA, NULL, Inf, or >= available windows, use all. If a vector, DR is run once per value and results are returned in a list. **Default:** 1000.

topVarSamples	NULL, character(), list, or regex string. Samples used to compute variability. If NULL/NA, use all samples. If a vector/regex, select matching samples. If a list, perform DR per list element (should match length(topVarNum) when vectorised). **Default:** NULL.
center	logical(1). Centre features to mean zero (PCA only). **Default:** TRUE.
scale	logical(1). Scale features to unit variance (PCA only). **Default:** FALSE.
nPC	integer(1). Number of principal components to compute (PCA only). **Default:** 5.
returnDataTable	logical(1). If TRUE, include the matrix used for DR in the return object. **Default:** FALSE.
verbose	logical(1). If TRUE, print messages regarding the function execution. **Default:** TRUE.
...	Additional arguments passed to uwot::umap() (when method="UMAP"), e.g. n_neighbors, min_dist, metric.
method	character(1). One of "PCA" or "UMAP". **Default:** "PCA".

Details

:

- For method = "PCA", DR is performed via stats::prcomp() with center and scale controls.
- For method = "UMAP", DR is performed via uwot::umap().
- Prior to DR, optional filtering is applied: enrichment cut-off (minEnrichment), CpG density (minDensity), region restriction (regionsToOverlap), and most-variable windows (topVarNum).
- When topVarNum or topVarSamples are vectorised/lists, multiple DR runs are performed and collected in res.

Value

A mesaDimRed object with:

- **res**: Named list of DR results; each element is a mesaPCA (for PCA) or mesaUMAP (for UMAP). Names correspond to entries in params$topVar.
- **samples**: Character vector of sample IDs used.
- **dataTable** (optional): The numeric matrix used for DR when returnDataTable = TRUE. It reflects filtering by regionsToOverlap and minDensity, removal of rows with missing values, and may include columns of window SDs when applicable (named per params$topVar).
- **params**: List of parameters used (method, selection/filter settings, etc.).
- **sampleTable**: Copy of qsea::getSampleTable(qseaSet).

Functions

- getPCA(): Principal component analysis of a qseaSet.
- getUMAP(): Uniform manifold approximation and projection of a qseaSet.

See Also

getPCA(), getUMAP(), mesaDimRed, mesaPCA, mesaUMAP, qsea::normMethod(), uwot::umap(), stats::prcomp()

Other dimred-helpers: plotDimRed(), plotUMAP()

Other dimred-helpers: plotDimRed(), plotUMAP()

Other dimred-helpers: plotDimRed(), plotUMAP()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Default PCA (beta-normalised, top 1000 most variable windows, 5 PCs)
exampleTumourNormal %>%
    getPCA()

# Request only the top 3 PCs
exampleTumourNormal %>%
    getPCA(nPC = 3)

# Use multiple cutoffs for most-variable windows
# (returns list of PCA results)
exampleTumourNormal %>%
    getPCA(topVarNum = c(10, 100, 500, NA)) %>%
    slot("res") %>%
    names()
# Default UMAP
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5)

# Alter UMAP hyperparameters
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, min_dist = 1)

# Use top 500 most variable windows only
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, topVarNum = 500)
data(exampleTumourNormal, package = "mesa")

# PCA on default beta-normalised matrix, keep 3 PCs
exampleTumourNormal %>%
    getDimRed(method = "PCA", nPC = 3)

# UMAP on the top 500 most variable windows
exampleTumourNormal %>%
    getDimRed(method = "UMAP", topVarNum = 500, n_neighbors = 5)

# Restrict DR to a set of regions (coerced from data.frame)
regs <- data.frame(seqnames = "chr7", start = 25002001, end = 25027900)
exampleTumourNormal %>%
    getDimRed(method = "PCA", regionsToOverlap = regs)
```

```
# Vectorised topVarNum runs (returns a list of results)
exampleTumourNormal %>%
    getDimRed(method = "UMAP", topVarNum = c(500, 2000), n_neighbors = 5) %>%
    slot("res") %>%
    names()
```

`getSampleQCSummary` *Summarise key QC fields per sample*

Description

Return a tidy summary of the most relevant QC fields from a `qseaSet` `sampleTable`. If core library metrics are missing, they are first added via `addLibraryInformation()`.

Usage

```
getSampleQCSummary(qseaSet)
```

Arguments

`qseaSet`	`qseaSet`. Input object from which to extract QC fields. **Default:** none (must be supplied).

Details

If the `sampleTable` lacks library metrics (e.g., `valid_fragments`), the function calls `addLibraryInformation()` to populate them. The output keeps `sample_name` and any columns whose names match the regex `"valid_fragment|relH|hyperStable|ichorTumo"` (if present), then orders rows by `sample_name`.

Value

A tibble with one row per sample containing:

- `sample_name`
- any available QC columns matching `valid_fragment`, `relH`, `hyperStable`, or `ichorTumo`. Columns not present in the `sampleTable` are silently omitted.

See Also

`qsea::getSampleTable()`, `addLibraryInformation()`, `addHyperStableFraction()`

Examples

```
data("exampleTumourNormal", package = "mesa")

# QC summary (adds library info if missing), ordered by sample_name
exampleTumourNormal %>%
    getSampleQCSummary() %>%
    head()
```

`getWindowNames` *Get window names from a qseaSet or ranges/table*

Description

Return character labels of the form `"seqnames:start-end"` for each genomic window/region.

Usage

```
getWindowNames(x)
```

Arguments

`x`	`qseaSet` **or** `GRanges` **or** `data.frame`. If a data frame, it must be coercible to `GRanges` (accepted columns include `seqnames/start/end`, or `chr/start/end`, or `chr/window_start/window_end`). **Default:** none (must be supplied).

Details

If `x` is a `qseaSet`, regions are taken from `qsea::getRegions(x)`. Otherwise the input is coerced to `GRanges` (see `asValidGranges()`) and labels are constructed as `"seqnames:start-end"`.

Value

`character()` vector of window labels, one per region.

See Also

`qsea::getRegions()`, `asValidGranges()`

Examples

```
# From a GRanges (no intermediate objects)
GenomicRanges::GRanges(c("chr1", "chr2"),
    IRanges::IRanges(c(10, 20), c(15, 30))) %>%
    getWindowNames()

# From a data.frame (no intermediate objects)
data.frame(seqnames = c("chr1", "chr2"),
    start = c(100, 200),
    end = c(120, 220)) %>%
    getWindowNames()

# From a qseaSet shipped with mesa (if available)
data(exampleTumourNormal, package = "mesa")
exampleTumourNormal %>% getWindowNames() %>% head()
```

getWindows *Extract the regions (windows) used in a qseaSet*

Description

Convenience wrapper for `qsea::getRegions()`, returning the genomic windows.

Usage

```
getWindows(qseaSet)
```

Arguments

qseaSet	qseaSet. Input object. **Default:** none (must be supplied).

Value

A GenomicRanges::GRanges of windows.

See Also

`qsea::getRegions()`, `getWindowNames()`

Examples

```
data(exampleTumourNormal, package = "mesa")
exampleTumourNormal %>% getWindows() %>% head()
```

hg19ToHg38.over.chain *hg19tohg38 liftover chain*

Description

A chain object to liftOver hg19 to hg38

Usage

```
hg19ToHg38.over.chain
```

Format

A chain object to liftOver hg19 to hg38

hg38_450kArrayGR *Infinium 450k probe locations for hg38.*

Description

A GRanges object containing the Illumina Infinium 450k array probe locations for hg38

Usage

```
hg38_450kArrayGR
```

Format

A GRanges object with 485541 ranges and 2 metadata columns:

ID Array probe ID

MASK_general A field suggesting whether the probe could be unreliable (if TRUE) due to some kind of technical issue

hg38CpGIslands *CpG islands for hg38.*

Description

A GRanges object containing CpG islands in hg38 coordinates. Generated from http://hgdownload.cse.ucsc.edu/goldenpath/h

Usage

```
hg38CpGIslands
```

Format

A GRanges object with 31144 ranges and 7 metadata columns:

length Number of bases in each island

cpgNum Number of CGs in each island

gcNum Number of Cs or Gs in the island

perCpg Percent CGs in the island

perGc Percent of Cs or Gs in the island

obsExp Unclear what this column means

`hg38UltraStableProbes` *Ultra-stable methylated regions (GRCh38)*

Description

Genomic loci that are consistently methylated ("ultra-stable") across many human tissues and diseases, mapped to GRCh38. Regions correspond to Illumina array probe loci (cg identifiers) derived as in Edgar et al. (2014), and are intended for QC and summary metrics (e.g., `addHyperStableFraction()`).

Usage

```
hg38UltraStableProbes
```

Format

A `GRanges` object with 974 ranges and 1 metadata column:

Probe_ID Character cg identifier for the probe/locus (e.g., `"cg11733071"`).

Details

- **Seqnames:** numeric chromosomes `"1"`–`"22"` (no `"chr"` prefix; UCSC style can be set if needed).
- **Strand:** `"*"` for all ranges.
- **Widths:** 3 bp windows (probe CpG ±1 bp).
- **Genome metadata:** currently unset; set to `"hg38"` if you need an explicit tag.

When combining with other `GRanges`, harmonise styles/metadata as needed, e.g. use `GenomeInfoDb::seqlevelsStyle()` to convert to `"UCSC"` (`"chr1"`, ...) and `GenomeInfoDb::genome()` to set `"hg38"`.

Source

Processed from public array resources following Edgar et al. (2014).

References

Edgar R, Tan PPC, Portales-Casamar E, Pavlidis P (2014). *Meta-analysis of human methylomes reveals stably methylated sequences surrounding CpG islands associated with high gene expression*. https://pubmed.ncbi.nlm.nih.gov/25493099/

See Also

`addHyperStableFraction()`

Examples

```
data(hg38UltraStableProbes)

# Basic overview
hg38UltraStableProbes
length(hg38UltraStableProbes) # 974
all(GenomicRanges::width(hg38UltraStableProbes) == 3) # TRUE (3-bp windows)

# The object uses numeric chromosomes and an unset genome tag
GenomeInfoDb::seqlevels(hg38UltraStableProbes)[1:5]
GenomeInfoDb::genome(hg38UltraStableProbes) # empty

# Harmonise style/metadata if needed
gr <- hg38UltraStableProbes
GenomeInfoDb::seqlevelsStyle(gr) <- "UCSC" # or "NCBI" / "Ensembl"
#' #   (for 1,2... instead of chr1, chr2...)
GenomeInfoDb::genome(gr) <- "hg38" # or "GRCh38"

# Count overlaps with a toy region
toy <- GenomicRanges::GRanges("1", IRanges::IRanges(951160, 951170))
GenomicRanges::countOverlaps(toy, hg38UltraStableProbes)

# With UCSC-style example
toy_ucsc <- GenomicRanges::GRanges("chr1", IRanges::IRanges(951160, 951170))
GenomicRanges::countOverlaps(toy_ucsc, gr) # after style harmonisation above
```

---

is.qseaSet *Check whether an object is a qseaSet*

Description

Predicate helper used in input validation and branching logic.

Usage

```
is.qseaSet(x)
```

Arguments

x	ANY. Object to test.

Value

logical(1): TRUE if x inherits from class "qseaSet", otherwise FALSE.

See Also

qsea::createQseaSet(), qsea::getSampleTable(), base::inherits()

Examples

```
data(exampleTumourNormal, package = "mesa")

# TRUE for qseaSet objects
exampleTumourNormal %>% is.qseaSet()

# FALSE for non-qseaSet objects
iris %>% is.qseaSet()
```

`left_join.mesaDimRed` *Left-join onto the sample table of a mesaDimRed*

Description

Extend `dplyr::left_join()` to operate on the `sampleTable` slot of a mesaDimRed object.

Usage

```
## S3 method for class 'mesaDimRed'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  ...
)
```

Arguments

`x`	`mesaDimRed` Object whose `sampleTable` will be joined.
`y`	`data.frame` Table to join.
`by`	`character()` or `NULL` Join columns; see `dplyr::left_join()`. **Default:** `NULL`.
`copy`	`logical(1)` See `dplyr::left_join()`. **Default:** `FALSE`.
`suffix`	`character(2)` Suffixes appended to overlapping non-join column names. **Default:** `c(".x", ".y")`.
`keep`	`logical(1)` or `NULL` Retain join keys from both tables. **Default:** `NULL`.
`...`	Additional arguments to `dplyr::left_join()`.

Value

A `mesaDimRed` object:

- **sampleTable** updated with `y` via a left join.

Examples

```
new <- data.frame(sample_name = c("Colon1_N", "Colon2_N"), foo = c(1, 2))

exampleTumourNormal %>%
    getPCA() %>%
    left_join(new, by = join_by(sample_name)) %>%
    getSampleTable()
```

`left_join.qseaSet` *Left join data onto a qseaSet sample table*

Description

Extends `dplyr::left_join()` to merge a data frame into a `qseaSet`'s `sampleTable`. Useful for adding annotations or metadata to samples.

Usage

```
## S3 method for class 'qseaSet'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  ...
)
```

Arguments

`x`	`qseaSet`. A qseaSet object whose `sampleTable` will receive additional columns.
`y`	`data.frame`. A data frame or tibble to join onto the `sampleTable`.
`by`	`character()` or `named character()`. Join keys, as in `dplyr::left_join()`. For qseaSets, the safest choice is `by = "sample_name"`. **Default:** `NULL` (all common variables between `x@sampleTable` and `y`).
`copy`	`logical(1)`. For remote backends only. If `TRUE`, copies `y` into the same data source as `x`. Ignored for in-memory data frames. **Default:** `FALSE`.
`suffix`	`character(2)`. Suffixes appended to overlapping non-join column names from `x` and `y`. **Default:** `c(".x", ".y")`.
`keep`	`logical(1)` or `NULL`. If `TRUE`, keeps join keys from both `x` and `y`. If `FALSE`, only keys from `x` are kept. **Default:** `NULL` (deferred to `dplyr::left_join()` behaviour).
`...`	Additional arguments passed to `dplyr::left_join()`, e.g. `relationship = "one-to-one"` (dplyr >= 1.1). **Default:** none.

Value

A qseaSet object with its sampleTable updated:

- existing rows preserved,
- additional columns from y merged by sample name or other keys,
- duplicate column names disambiguated with suffix.

See Also

qsea::getSampleTable() to view metadata, mutate.qseaSet(), filter.qseaSet() for other dplyr verbs, dplyr::left_join() for join semantics.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Join 'patient' (y) to 'sample_name' (x)
newData <- tibble::tibble(
    patient = c("Colon1", "Colon2", "Lung1", "Lung3"),
    new = 1:4
)
exampleTumourNormal %>%
    left_join(newData, by = c("sample_name" = "patient")) %>%
    qsea::getSampleTable()
```

---

liftOverHg19 *Lift over genomic ranges from hg19 to hg38*

Description

Convert genomic intervals from UCSC **hg19** to **hg38/GRCh38** using a preloaded chain object.

Usage

```
liftOverHg19(grOrDf)
```

Arguments

grOrDf	GRanges or data.frame. Input intervals. Data frames must contain at least seqnames, start, and end. seqnames may include or omit the "chr" prefix; it is added automatically if missing. **Default:** none (must be supplied).

Details

Internally, the input is coerced to a GRanges (adding "chr" if needed), then lifted over via **rtracklayer** with the chain stored in mesa::hg19ToHg38.over.chain. Unmappable or split mappings are handled by rtracklayer::liftOver(), and unmapped intervals are discarded when the result is unlisted. The returned object is tagged as "hg38".

Value

A GRanges-class in hg38 coordinates (with `GenomeInfoDb::genome(x) <- "hg38"`). Intervals that cannot be mapped by the chain are dropped (i.e., may return fewer ranges than provided).

See Also

`rtracklayer::liftOver()`, GenomicRanges::GRanges

Examples

```
# Data.frame input; mix of 'chr' and no 'chr' is fine
data.frame(
    seqnames = c("1", "chr2"),
    start    = c(100000, 200000),
    end      = c(100100, 200100)
) %>%
    liftOverHg19() %>%
    {
        GenomeInfoDb::genome(.)
    }

# GRanges input (some intervals may not map and can be dropped)
gr <- GenomicRanges::GRanges("chr1", IRanges::IRanges(100000, 100300))
liftOverHg19(gr)
```

`makeAllContrasts` *Generate all possible pairwise contrasts*

Description

Construct a tibble of all pairwise contrasts between levels of a categorical variable in the sample table of a qsea::qseaSet.

Usage

```
makeAllContrasts(qseaSet, variable)
```

Arguments

`qseaSet`	`qseaSet`. Input object providing the sample table from which factor levels are taken.
`variable`	`character(1)`. Column name in the sample table whose distinct levels define the contrasts. **Default:** none (must be supplied).

Details

Levels are inferred from `unique(qsea::getSampleTable(qseaSet)[[variable]])`. Pairs are formed between all distinct levels; no self-contrasts are included.

Value

A tibble with two columns:

- group1 – the first level of the pair,
- group2 – the second level of the pair. Each row represents one **unordered** pair of distinct levels (i.e., all combinations without repetition).

See Also

Other DMR-detection: calculateDMRs(), fitQseaGLM(), getDMRsData()

Examples

```
data(exampleTumourNormal, package = "mesa")

# All pairwise contrasts among levels of 'type'
exampleTumourNormal %>%
    makeAllContrasts("type")
```

---

makeQset *Construct an initial qseaSet from BAMs and metadata*

Description

Build a qseaSet from a sampleTable that lists BAM files and sample metadata, tile the genome into fixed windows, optionally remove blacklisted regions, and load coverage and (optionally) CNV information according to the selected methods. This is typically the first step in a mesa/qsea workflow.

Usage

```
makeQset(
  sampleTable,
  BSgenome,
  chrSelect,
  windowSize = 300,
  CNVwindowSize = 1e+06,
  fragmentType = NULL,
  fragmentLength = NULL,
  fragmentSD = NULL,
  CNVmethod = "HMMdefault",
  coverageMethod = "PairedAndR1s",
  minMapQual = 10,
  minInsertSize = 70,
  maxInsertSize = 1000,
  minReferenceLength = 30,
```

```
  badRegions = NULL,
  properPairsOnly = FALSE,
  hmmCopyGC = NULL,
  hmmCopyMap = NULL,
  maxPatternDensity = 0.05,
  enrichmentMethod = "blind1-15",
  parallel = getMesaParallel()
)
```

Arguments

sampleTable	data.frame. Must contain sample_name, file_name, and group. If CNVmethod requires inputs, also input_file. Additional columns are preserved as sample metadata. **Default:** none (must be supplied).
BSgenome	character(1) or NULL. BSgenome package string (e.g., "BSgenome.Hsapiens.NCBI.GRCh38"). See BSgenome::available.genomes(). **Default:** NULL (must be supplied).
chrSelect	integer() or character(). Chromosomes to include. **Default:** 1:22.
windowSize	integer(1). Window size in bp for tiling the genome. **Default:** 300.
CNVwindowSize	integer(1). Copy number variation (CNV) window size (bp). **Default:** 1e6.
fragmentType	character(1) or NULL. If "Sheared" or "cfDNA", sets defaults for fragmentLength/fragmentSD; otherwise both must be supplied explicitly. **Default:** NULL.
fragmentLength	numeric(1) or NULL. Average fragment length (bp). This is used to extend single-end reads if used, but also strongly affects the CpG density calculation, which will strongly impact the beta value estimation. **Default:** NULL (inferred from fragmentType or must be provided).
fragmentSD	numeric(1) or NULL. Standard deviation of fragment length (bp). Used to calculate the window based CpG density calculation, which will strongly impact the beta value estimation. **Default:** NULL (inferred from fragmentType or must be provided).
CNVmethod	character(1). One of "HMMdefault", "qseaInput", "MeCap", "None". **Default:** "HMMdefault".
coverageMethod	character(1). One of "PairedAndR1s" (mesa) or "qseaPaired" (qsea). **Default:** "PairedAndR1s".
minMapQual	integer(1). Minimum MAPQ to retain a read. For "PairedAndR1s", a pair is kept if either end passes when properly paired and MAPQ tags are set. **Default:** 10.
minInsertSize	integer(1). Minimum absolute insert size for proper pairs (bp). Only applies to coverageMethod = "PairedAndR1s", and applies to Input samples as well as MeCap. **Default:** 70.
maxInsertSize	integer(1). Maximum absolute insert size for proper pairs (bp). Only applies to coverageMethod = "PairedAndR1s", and applies to Input samples as well as MeCap. **Default:** 1000.
minReferenceLength	integer(1). A minimum mapped distance on the genome to keep an read. This is to avoid very short reads that may be due to mapping artefacts. Note that with

defaultparameters bwa will allow reads where only 19bps have mapped to the genome. Only applies to coverageMethod = "PairedAndR1s", and applies to Input samples as well as MeCap. **Default:** `30`.

`badRegions` `GRanges` or `NULL`. Genomic regions to exclude (blacklist). These regions will be removed from the output. Note a set of GRCh38 windows identified by ENCODE are provided in the ENCODEbadRegions data object. **Default:** `NULL`.

`properPairsOnly`
`logical(1)`. If `TRUE`, use only proper pairs (stricter size selection). **Default:** `FALSE`.

`hmmCopyGC` `data.frame` or `NULL`. GC content per CNV bin (size = `CNVwindowSize`) for HMMcopy. Required if using the `CNVmethod = "HMMdefault"` option *unless* you are using hg38/GRCh38 with provided default bin sizes (50kb, 500kb, 1Mb). Must be a dataframe with columns `chr`, `start`, `end`, and `gc`. **Default:** `NULL`.

`hmmCopyMap` `data.frame` or `NULL`. Mappability per CNV bin (size = `CNVwindowSize`) for HMMcopy. Required if using the `CNVmethod = "HMMdefault"` option *unless* you are using hg38/GRCh38 with provided default bin sizes (50kb, 500kb, 1Mb). Must be a dataframe with columns `chr`, `start`, `end`, and `map`. **Default:** `NULL`.

`maxPatternDensity`
`numeric(1)`. Maximum pattern density (e.g., CpG) to consider a window for background offset calculation. Regions above this are excluded from the calculation. This may need to be set higher if you are not considering many windows or have a high average genomic CG content. **Default:** `0.05`.

`enrichmentMethod`
`character(1)`. Method for calculating enrichment for target pattern (e.g. CpG). See `addNormalisation()` for details. **Default:** `"blind1-15"`.

`parallel` `logical(1)`. Use the registered BiocParallel backend (`TRUE`) or run serially (`FALSE`). See `BiocParallel::register()`. **Default:** `getMesaParallel()`.

Details

Inputs & requirements

- `sampleTable` must include at least:
  - `sample_name` (unique per sample),
  - `file_name` (path to MeDIP/capture BAM),
  - `group` (string label). If the selected `CNVmethod` uses inputs (`"HMMdefault"`, `"qseaInput"`), also include `input_file` (path to matched Input BAM).
- BAMs should be coordinate-sorted. For paired data, running `samtools fixmate` is recommended so MAPQ tags are available (see `minMapQual` note below).
- `BSgenome` must be an installed genome package string (e.g. `"BSgenome.Hsapiens.NCBI.GRCh38"`), which provides chromosome lengths and sequences for tiling windows.

Windows & blacklist

- Chromosomes in `chrSelect` are tiled into non-overlapping windows of size `windowSize` (bp). Windows overlapping `badRegions` (if provided) are removed.

Coverage loading

- coverageMethod = "PairedAndR1s" loads proper pairs and, optionally, high-MAPQ R1 singletons; governed by minMapQual, minInsertSize, maxInsertSize, properPairsOnly, minReferenceLength.
- coverageMethod = "qseaPaired" delegates to qsea::addCoverage() with paired settings.
- fragmentType can set defaults for fragmentLength/fragmentSD ("Sheared" or "cfDNA"); otherwise supply both explicitly.

CNV options

- "qseaInput": run qsea::addCNV() using input_file BAMs.
- "HMMdefault": use mesa's addHMMcopyCNV() with supplied defaults for GRCh38 GC/mappability tracks (objects like gc_hg38_1000kb, map_hg38_1000kb, chosen by CNVwindowSize).
- "MeCap": estimate CNV from MeCap BAMs (file_name).
- "None": do not compute CNV.

Parallelisation

- If parallel = TRUE, computation uses the currently registered BiocParallel backend (see BiocParallel::register()) otherwise runs serially. Use setMesaParallel() to toggle package-wide.

Additional processing

- Adds CpG density via qsea::addPatternDensity(pattern = "CG"), and enrichment parameters, allowing for estimation of beta values.

Value

A qseaSet containing:

- tiled (and optionally blacklisted) genomic windows;
- the input sampleTable as sample metadata;
- loaded coverage according to coverageMethod;
- CNV results depending on CNVmethod;
- CpG density and enrichment parameters;
- recorded processing parameters in @parameters.

See Also

qsea::createQseaSet(), qsea::addCoverage(), qsea::addCNV(), addHMMcopyCNV(), setMesaParallel(), BSgenome::available.genomes()

Examples

```
# Minimal runnable sketch if MEDIPSData is installed
if (requireNamespace("MEDIPSData", quietly = TRUE)) {
    sampleTable <- data.frame(
        sample_name = c("Normal1", "Tumour1"),
        group = c("Normal", "Tumour"),
        file_name = c(
            system.file("extdata", "NSCLC_MeDIP_1N_fst_chr_20_21_22.bam",
                package = "MEDIPSData", mustWork = TRUE),
            system.file("extdata", "NSCLC_MeDIP_1T_fst_chr_20_21_22.bam",
                package = "MEDIPSData", mustWork = TRUE)
        )
    )

    sampleTable %>%
        makeQset(
            BSgenome          = "BSgenome.Hsapiens.UCSC.hg19",
            chrSelect         = paste0("chr", 20:22),
            windowSize        = 300,
            fragmentLength    = 200,
            fragmentSD        = 50,
            CNVmethod         = "None",
            coverageMethod    = "PairedAndR1s",
            properPairsOnly   = FALSE,
            minMapQual        = 10
        )
}
```

---

makeTransposedTable *Make a wide sample-by-window table*

Description

Create a wide table for ML/visualisation where **rows are samples** and **columns are genomic windows**, using one normalisation/measure.

Usage

```
makeTransposedTable(qseaSet, normMethod = "nrpm", ...)
```

Arguments

qseaSet	qseaSet. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
normMethod	character(1). Measure to extract. Common choices include "nrpm" or "beta". **Default:** "nrpm".
...	tidyselect specification or character(). Optional columns from the sample table to append as additional features (e.g., group, tumour or c("group","tumour")). **Default:** none.

Details

Internally obtains a window × sample matrix for `normMethod`, transposes it to sample × window, and (if requested) appends selected sample-table columns. Window columns are typically named `"chr:start-end"`. If measure suffixes are present in column names, they may be cleaned to bare sample/window names (see `removeNormMethodSuffix()`).

Value

A tibble with one row per sample and one column per genomic window, plus any appended sample-level metadata.

See Also

`getDataTable()`, `removeNormMethodSuffix()`, `countWindowsAboveCutoff()`

Other table-helpers: `getBetaTable()`, `getCountTable()`, `getDataTable()`, `getNRPMTable()`, `getSampleGroups2()`, `removeLibraryFactors()`, `removeNormMethodSuffix()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# NRPM features (sample × first few windows) with a group column appended
exampleTumourNormal %>%
  makeTransposedTable(normMethod = "nrpm", group) %>%
  dplyr::select(1:6) %>%
  head()

# Beta features; inspect dimensions
exampleTumourNormal %>%
  makeTransposedTable(normMethod = "beta") %>%
  dim()
```

---

map_hg38_1000kb *Mappability across the human genome (hg38/GRCh38)*

Description

Fixed-size bins with average mappability per bin. Bins are ordered by chromosome size (not strictly numeric).

Usage

```
data(map_hg38_1000kb); data(map_hg38_500kb); data(map_hg38_50kb)

map_hg38_50kb

map_hg38_500kb
```

Format

One of:

1. A `data.table`/`data.frame` with columns:
   chrom Chromosome (e.g., "chr1", ..., "chrX", "chrY").
   start 0-based start of the bin.
   end 1-based end of the bin.
   map Average mappability in the bin (0–1).
2. A `GRanges` with genomic bins and `mcols(.)$map` storing the average mappability (0–1) per bin.

An object of class `data.table` (inherits from `data.frame`) with 61775 rows and 4 columns.

An object of class `data.table` (inherits from `data.frame`) with 6188 rows and 4 columns.

Value

`map_hg38_1000kb`, `map_hg38_500kb`, and `map_hg38_50kb` return objects as described in `@format` with bin sizes of 1000 kb, 500 kb, and 50 kb, respectively.

Source

https://github.com/broadinstitute/ichorCNA

---

mesaDimRed-class *Dimensionality reduction results container*

Description

Aggregates one or more dimensionality reduction (DR) results (e.g., PCA/UMAP) computed on a common set of samples.

Usage

```
mesaDimRed(res, sampleTable, samples, params, dataTable = data.frame())

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

Arguments

`res`	`list` DR result objects. **Default:** none. `res` must be supplied.
`sampleTable`	`data.frame` Sample annotations; row names must be sample IDs. **Default:** none.
`samples`	`character()` Sample IDs included. **Default:** none.
`params`	`list` Parameters used to compute `res`. **Default:** none.
`dataTable`	`data.frame` Optional numeric matrix/data used for DR. **Default:** `data.frame()`.
`object`	`mesaDimRed`

Value

A mesaDimRed object:

- stores DR results in `res`,
- carries sample metadata in `sampleTable`,
- records the samples in `samples`,
- and persists parameters/data in `params` / `dataTable`.

Slots

`res list` Individual DR result objects (e.g., mesaPCA, mesaUMAP).

`sampleTable data.frame` Sample annotations; row names are sample IDs.

`samples character()` Vector of sample IDs used.

`params list` Parameters used to generate results in `res`.

`dataTable data.frame` Optional matrix/data used to compute DR (for reproducibility).

See Also

mesaPCA, mesaUMAP

Examples

```
st <- data.frame(sample_name = c("S1", "S2"),
    group = c("A", "B"),
    row.names = "sample_name")
md <- mesaDimRed(res = list(), sampleTable = st,
    samples = rownames(st), params = list(),
    dataTable = data.frame())
md
```

---

mesaPCA-class *PCA results container*

Description

Stores a PCA fit (from `stats::prcomp()`) computed over methylation windows.

Usage

```
mesaPCA(prcomp, windows)

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

Arguments

prcomp	prcomp PCA fit from stats::prcomp(). **Default:** none.
windows	character() Window IDs used. **Default:** none.
object	mesaPCA

Value

A mesaPCA object containing the PCA fit and window IDs.

Slots

prcomp prcomp A PCA fit returned by stats::prcomp().

windows character() Window IDs used in the PCA.

See Also

mesaDimRed

Examples

```
set.seed(1)
x <- matrix(rnorm(20), nrow = 5, ncol = 4,
    dimnames = list(paste0("S", 1:5), paste0("W", 1:4)))
pc <- stats::prcomp(x, center = TRUE, scale. = FALSE)
mp <- mesaPCA(prcomp = pc, windows = colnames(x))
mp
```

---

mesaUMAP-class *UMAP results container*

Description

Stores per-sample coordinates from a UMAP embedding computed over methylation windows.

Usage

```
mesaUMAP(points, windows)

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

Arguments

points	data.frame One row per sample with positions in UMAP space (row names = sample IDs). **Default:** none.
windows	character() Window IDs used. **Default:** none.
object	mesaUMAP

Value

A mesaUMAP object containing UMAP coordinates and window IDs.

Slots

`points` `data.frame` One row per sample with UMAP coordinates (e.g., `UMAP1`, `UMAP2`). Row names are sample IDs.

`windows` `character()` Window IDs used to compute the embedding.

See Also

mesaDimRed

Examples

```
pts <- data.frame(UMAP1 = c(0.1, -0.2, 0.0),
    UMAP2 = c(0.3, 0.1, -0.1))
rownames(pts) <- paste0("S", 1:3)
mu <- mesaUMAP(points = pts, windows = c("w1", "w2", "w3"))
mu
```

---

mixSamples *Mix two samples to generate a synthetic qseaSet sample*

Description

Create a new synthetic sample in a `qseaSet` by mixing reads from two existing samples in user-specified proportions. Useful for benchmarking, downsampling experiments, or simulating tumour–normal mixtures.

Usage

```
mixSamples(
  qseaSet,
  sample1,
  sample2,
  nReadsTotal,
  proportion,
  newName = NULL,
  groupName = NULL,
  onlyNew = FALSE,
  renormalise = TRUE
)
```

Arguments

qseaSet	qseaSet. The input object containing the two source samples.
sample1	character(1). First sample name, contributing proportion * nReadsTotal reads.
sample2	character(1). Second sample name, contributing the remainder of reads.
nReadsTotal	integer(1). Total number of reads to simulate in the new mixed sample. Must be >= 0.
proportion	numeric(1). Proportion of reads taken from sample1. Must be in [0, 1]. The remaining 1 - proportion reads are taken from sample2.
newName	character(1) or NULL. Name for the new synthetic sample. **Default:** NULL (a name is generated automatically, e.g., "Mix_<sample1>_<sample2>_<proportion>").
groupName	character(1) or NULL. Group name for the new sample recorded in the sampleTable. **Default:** NULL (uses newName).
onlyNew	logical(1). If TRUE, return only the synthetic sample. If FALSE, return the original qseaSet with the new sample appended. **Default:** FALSE.
renormalise	logical(1). Whether to run addNormalisation() on the result. For efficiency, set to FALSE when creating multiple mixtures and run normalisation once at the end. **Default:** TRUE.

Details

Note that if the qseaSet windows have been filtered prior to using this, the fraction of fragments present in the current qseaSet is calculated, such that the total number of reads would have been approximately nReadsTotal. E.g. if there are 50% of the total fragments in the windows present in the current samples, 0.5*nReadsTotal fragments will be sampled.

- Input validation ensures sample1/sample2 exist, proportion in [0,1], and nReadsTotal is non-negative.
- The mixing strategy (e.g., proportional allocation vs sampling) follows the package's implementation; set renormalise = TRUE to recompute offsets/enrichment after adding the new sample if required by your workflow.

Value

A qseaSet containing the new synthetic sample:

- If onlyNew = TRUE, a qseaSet with **one** (mixed) sample.
- If onlyNew = FALSE, the input qseaSet **plus** the mixed sample. The sampleTable, counts, and relevant slots are updated consistently.

See Also

mixThreeQsetSamples(), downSample(), addNormalisation()

Other sample-simulation: mixThreeQsetSamples()

Examples

```
data(exampleTumourNormal, package = "mesa")


# Mix two samples 50/50 into ~100k total fragments; append to the qseaSet
exampleTumourNormal %>%
    mixSamples("Colon1_T",
        "Colon1_N",
        nReadsTotal = 100000,
        proportion = 0.5,
        renormalise = FALSE
    ) %>%
    # Only a few reads are included in this small subset of windows (82):
    getCountTable() %>%
    pull(Mix_Colon1_T_Colon1_N_0.5) %>%
    sum()


# Mix two samples 80/20 and return only the synthetic sample, with an
# explicit name and group
exampleTumourNormal %>%
    mixSamples("Colon1_T", "Colon1_N",
        nReadsTotal = 50000,
        proportion  = 0.8,
        newName     = "Mix_Colon1_T_0.8_Colon1_N_0.2",
        groupName   = "Synthetic",
        onlyNew     = TRUE,
        renormalise = FALSE
    ) %>%
    # Only a few reads are included in this small subset of windows (45):
    getCountTable() %>%
    dplyr::pull(Mix_Colon1_T_0.8_Colon1_N_0.2) %>%
    sum()
```

`mutate.mesaDimRed` *Mutate the sample table of a mesaDimRed*

Description

Extend `dplyr::mutate()` to operate on the `sampleTable` slot of a mesaDimRed object.

Usage

```
## S3 method for class 'mesaDimRed'
mutate(.data, ...)
```

Arguments

.data	mesaDimRed Object to modify.
...	Arguments passed to dplyr::mutate().

Value

A mesaDimRed object:

- **sampleTable** updated with the mutated columns.
- Sample identity is preserved; attempts to alter sample_name are rejected.

Examples

```
data(exampleTumourNormal, package = "mesa")

md <- exampleTumourNormal %>%
    getPCA() %>%
    mutate(group2 = paste0(group, "_2"))

stopifnot("group2" %in% colnames(getSampleTable(md)))
```

---

mutate.qseaSet *Mutate columns in a qseaSet sample table*

Description

Extends dplyr::mutate() to add or modify columns in a qseaSet's sampleTable.

Usage

```
## S3 method for class 'qseaSet'
mutate(.data, ...)
```

Arguments

.data	qseaSet A qseaSet object whose sampleTable will be mutated.
...	Column transformations passed to dplyr::mutate(). For example, over70 = age > 70, group = paste(tissue, tumour, sep = "_"). **Default:** none.

Details

The order of samples is preserved. The sample_name column **cannot** be changed with this function; use renameQsetNames() or renameSamples() if sample names must be updated.

Value

A `qseaSet` object with its `sampleTable` updated:

- new columns are added,
- existing columns are modified,
- `sample_name` remains unchanged.

See Also

`qsea::getSampleTable()` to inspect metadata, `renameSamples()` and `renameQsetNames()` for renaming, `filter.qseaSet()` and `selectQset()` for other dplyr verbs.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Add a new logical column based on age
exampleTumourNormal %>%
    mutate(over70 = age > 70) %>%
    qsea::getSampleTable()

# Recode gender (better practice: use dplyr::case_when() or
# stringr::str_replace())
exampleTumourNormal %>%
    mutate(gender = ifelse(gender == "F", "Female", "Male")) %>%
    qsea::getSampleTable()

# Change group by removing digits from sample_name
exampleTumourNormal %>%
    mutate(group = stringr::str_remove(sample_name, "[0-9]")) %>%
    qsea::getSampleTable()
```

---

pivotDMRsLonger *Transform DMR results to long format*

Description

Convert wide-format DMR results into a long-format table with explicit columns for contrasts, effect sizes, and significance values.

Usage

```
pivotDMRsLonger(DMRtable, FDRthres = 0.05, makePositive = FALSE)
```

Arguments

DMRtable	data.frame DMR results in wide format, typically produced by calculateDMRs().
FDRthres	numeric(1) False discovery rate threshold for filtering significant windows. **Default:** 0.05.
makePositive	logical(1) If TRUE, reverse the direction of contrasts so that all retained windows are positively associated with group1. **Default:** FALSE.

Value

A tibble:

- **pivotDMRsLonger()**: returns a long-format table with one row per DMR–contrast combination, including columns group1, group2, deltaBeta, log2FC, and adjPval.

See Also

tidyr::pivot_longer(), summariseDMRsByContrast()

Other DMR-helpers: summariseDMRsByContrast(), summariseDMRsByGene(), writeDMRsToBed(), writeDMRsToExcel()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Convert results to long format and filter at custom FDR threshold
exampleTumourNormal %>%
    calculateDMRs(
        variable = "type",
        contrasts = "LUAD_vs_NormalLung",
        FDRthres = 0.1
    ) %>%
    pivotDMRsLonger(FDRthres = 0.1)
```

---

plotCNVheatmap *CNV heatmap across samples*

Description

Plot per-window copy-number estimates stored in a qseaSet as a heatmap using **ComplexHeatmap**. Each column is a sample; rows are genome windows.

Usage

```
plotCNVheatmap(
  qseaSet,
  sampleAnnotation = NULL,
  annotationColors = NA,
  clusterRows = TRUE
)
```

Arguments

qseaSet	qseaSet. Input object containing per-window CNV tracks (e.g., created by addHMMcopyCNV() or qsea::addCNV()).
sampleAnnotation	tidyselect specification, character(), or NULL. Sample-table columns to display as column annotations (e.g., c("tumour","type") or bare helpers tumour, type). **Default:** NULL.
annotationColors	list or NA. Optional mapping of levels → colours to override auto palettes, e.g. list(tumour = c(Tumour = "firebrick4", Normal = "blue")). **Default:** NA.
clusterRows	logical(1). Cluster rows (windows) before plotting. **Default:** TRUE.

Details

Assumes CNV values are available per window and sample. Rows are windows (typically in genomic order if clusterRows = FALSE), columns are samples. Sample annotations (and optional colours) are added when sampleAnnotation and annotationColors are provided.

Value

Draws a heatmap on the active device and (invisibly) returns the underlying **ComplexHeatmap** object for further composition with draw.

See Also

addHMMcopyCNV(), runHMMcopy(), Heatmap, qsea::addCNV()

Other CNV: addHMMcopyCNV(), removeCNV(), runHMMCopy()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Basic CNV heatmap with sample annotations
exampleTumourNormal %>%
    plotCNVheatmap(sampleAnnotation = c(tumour, type))

# Custom annotation colours; disable row clustering
exampleTumourNormal %>%
    plotCNVheatmap(
        sampleAnnotation = tumour,
```

```
    annotationColors = list(
        tumour = c(Tumour = "firebrick4", Normal = "blue")
    ),
    clusterRows = FALSE
)
```

`plotCorrelationMatrix` *Sample correlation heatmap*

Description

Compute and plot the correlation matrix across samples (or group means) using the selected normalisation measure and optional region filters.

Usage

```
plotCorrelationMatrix(
  qseaSet,
  regionsToOverlap = NULL,
  useGroupMeans = FALSE,
  sampleAnnotation = NULL,
  normMethod = "nrpm",
  minEnrichment = 3,
  annotationColors = NA,
  minDensity = 0,
  ...
)
```

Arguments

`qseaSet`	`qseaSet`. Input object containing windows and signal (beta or nrpm).
`regionsToOverlap`	`GRanges`, `data.frame` (coercible to `GRanges`), or `NULL`. If provided, restrict computation to windows overlapping these regions; otherwise use all windows. **Default:** `NULL`.
`useGroupMeans`	`logical(1)`. If `TRUE`, average replicates by the `group` column and compute correlations on group means; if `FALSE`, use individual samples. **Default:** `FALSE`.
`sampleAnnotation`	tidyselect specification or `character()` or `NULL`. Columns from the sample table to display as annotations on the heatmap (e.g., `c("tumour","patient")` or bare helpers `tumour, patient`). **Default:** `NULL`.
`normMethod`	`character(1)`. Measure to correlate. One of `"nrpm"` or `"beta"`. **Default:** `"nrpm"`.
`minEnrichment`	`numeric(1)`. For `"beta"`, values with enrichment `< minEnrichment` are set to `NA` before correlation. **Default:** `3`.

annotationColors
: `list` or `NA`. Optional named colour maps for annotations (passed to **pheatmap**), e.g. `list(tumour = c(Tumour = "firebrick4", Normal = "blue"))`. **Default:** `NA`.

minDensity
: `numeric(1)`. Minimum `CpG_density` required to keep a window. **Default:** `0`.

...
: Additional arguments forwarded to `pheatmap::pheatmap()`.

Details

Windows are optionally restricted by `regionsToOverlap` and filtered by `minDensity`. Correlations are computed on the chosen `normMethod` after any group averaging. For `"beta"`, entries below `minEnrichment` are set to `NA` to down-weight low-enrichment windows.

Value

A `pheatmap::pheatmap` object.

See Also

`plotRegionsHeatmap()`, `getDataTable()`, `pheatmap::pheatmap()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# By default uses NRPM
exampleTumourNormal %>% plotCorrelationMatrix()

# Using beta values and adding sample annotations
exampleTumourNormal %>%
    plotCorrelationMatrix(normMethod = "beta",
        sampleAnnotation = c(tumour, patient))
```

---

plotDimRed *Plot dimensionality reduction results*

Description

Visualise PCA or UMAP coordinates stored in a mesaDimRed object. Wrappers such as `BiocGenerics::plotPCA()` and `plotUMAP()` provide convenient shortcuts for common use cases.

Usage

```
plotDimRed(
  object,
  components = list(c(1, 2), c(2, 3)),
  colour = NULL,
  colourPalette = NULL,
  NAcolour = "grey50",
  symDivColourScale = FALSE,
  shape = NULL,
  shapePalette = NULL,
  NAshape = NULL,
  showSampleNames = FALSE,
  pointSize = 2,
  alpha = 1,
  plotlyAnnotations = ""
)
```

Arguments

`object`	`mesaDimRed`. A dimensionality reduction container returned by `getPCA()` or `getUMAP()`.
`components`	`integer(2)` **or** `list` of `integer(2)`. Component indices to plot (e.g. `c(1, 2)` for PC1 vs PC2). A list of pairs generates multiple plots. **Default:** `list(c(1, 2), c(2, 3))` for PCA, `list(c(1, 2))` for UMAP.
`colour`	`NULL` or `character()`. Name(s) of sample-table variable(s) mapped to point colour. One plot is produced per variable. **Default:** `NULL` (no colouring).
`colourPalette`	`NULL` or `character()`. Palette for point colours. If `NULL`, defaults are selected automatically. **Default:** `NULL`.
`NAcolour`	`character(1)`. Colour for missing values in `colour`. **Default:** `"grey50"`.
`symDivColourScale`	`logical(1)`. If `TRUE`, diverging colour scales are centred symmetrically around zero. Ignored for non-diverging scales. **Default:** `FALSE`.
`shape`	`NULL` or `character(1)`. Sample-table variable mapped to point shape. Only one variable supported. **Default:** `NULL`.
`shapePalette`	`NULL`, `character(1)`, or `numeric()`. Shapes used for categories of the `shape` variable. Options: • A numeric vector of base-R shape codes: – 0–20: line/filled shapes. – 21–25: filled shapes with borders (border colour = black). • A keyword: • `"line-first"`: 15 line shapes, then 4 filled shapes (**max 19 categories**). • `"filled-first"`: 4 filled shapes, then 15 line shapes (**max 19 categories**). – `"mixture"`: mixture of line and filled shapes (**max 19 categories**). • `"filled+border"`: filled+border shapes (**max 5 categories; 4 if NAs in** `shape`).

  - NULL: automatic choice—"mixture" unless a diverging colour scale is used, in which case "filled+border". **Default:** NULL.

NAshape NULL or numeric(1). Shape used for missing values in shape. If NULL, defaults to 7, or 25 when "filled+border" is active. **Default:** NULL.

showSampleNames
logical(1). If TRUE, overlay sample names on points. **Default:** FALSE.

pointSize numeric(1). Size of plotted points. **Default:** 2.

alpha numeric(1). Transparency of points in [0, 1]. **Default:** 1.

plotlyAnnotations
character(). Column names from the sample table used as tooltips when converting plots to interactive plotly. **Default:** "" (empty string).

Value

A list of ggplot2 objects, one for each combination of:

- component pairs in components, and
- variables specified in colour.

Each plot depicts the requested dimensions with aesthetics mapped as specified.

See Also

Other dimred-helpers: getPCA(), plotUMAP()

Examples

```
data(exampleTumourNormal, package = "mesa")

# PCA: colour by group
exampleTumourNormal %>%
    getPCA(nPC = 3) %>%
    plotDimRed(colour = "group")

# UMAP: colour by group, shape by tissue
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, min_dist = 1) %>%
    plotDimRed(
        colour = "group",
        shape = "tissue"
    )

# Show sample names
exampleTumourNormal %>%
    getPCA(nPC = 3) %>%
    plotDimRed(
        colour = "group",
        showSampleNames = TRUE
    )
```

plotDMRUpset *UpSet plot of DMR overlaps*

Description

Visualise the overlap of significant DMR sets across contrasts (columns ending with `*_adjPval`) using an UpSet plot.

Usage

```
plotDMRUpset(DMRtable, string = NULL, removeVS = FALSE, minAdjPval = 0.05, ...)
```

Arguments

`DMRtable`	`data.frame`. Table as returned by `calculateDMRs()` (optionally pre-filtered).
`string`	`character(1)` or `NULL`. Optional regular expression to subset the set/contrast names (applied after stripping suffixes). **Default:** `NULL` (use all sets).
`removeVS`	`logical(1)`. If `TRUE`, remove the `"_vs_"` substring and everything after it from set names (e.g., `"Tumour_vs_Normal"` → `"Tumour"`). **Default:** `FALSE`.
`minAdjPval`	`numeric(1)`. Adjusted P-value threshold; windows with `adjPval <= minAdjPval` are included in each set. **Default:** `0.05`.
`...`	Additional arguments forwarded to `UpSetR::upset()`.

Details

The function discovers DMR sets by locating columns that end with `"_adjPval"`. For each such column, a logical membership is formed at the chosen FDR cutoff (`minAdjPval`). Optionally, set names are simplified with `removeVS`, and then subset with `string` if provided. The resulting membership matrix is visualised with **UpSetR**.

Value

Draws an UpSet plot on the active device and **invisibly** returns the result from `UpSetR::upset()` (for further customization if needed).

See Also

`calculateDMRs()`, `UpSetR::upset()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Overlap of significant DMR sets across contrasts (default FDR 0.05)
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "all_vs_NormalLung") %>%
    plotDMRUpset()
```

```
# Clean set names by dropping the '_vs_*' suffix
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "all_vs_NormalLung") %>%
    plotDMRUpset(removeVS = TRUE)
```

`plotGeneHeatmap` *Plot sample signal over windows spanning a gene (± flanks) as a heatmap using* **ComplexHeatmap**. *The gene can be given as a HGNC symbol or Ensembl ID.*

Description

Plot sample signal over windows spanning a gene (± flanks) as a heatmap using **ComplexHeatmap**. The gene can be given as a HGNC symbol or Ensembl ID.

Usage

```
plotGeneHeatmap(
  qseaSet,
  gene,
  normMethod = "beta",
  useGroupMeans = FALSE,
  sampleAnnotation = NULL,
  minDensity = 0,
  minEnrichment = 3,
  maxScale = 1,
  clusterNum = NULL,
  annotationColors = NA,
  upstreamDist = 3000,
  scaleRows = FALSE,
  clusterCols = TRUE,
  mart = NULL,
  showSampleNames = NULL,
  downstreamDist = 1000,
  idType = NULL,
  ...
)
```

Arguments

`qseaSet`	`qseaSet`. Input object containing windows and counts/betas.
`gene`	`character(1)`. Gene identifier to plot (HGNC symbol like `"HOXA9"` or Ensembl ID like `"ENSG000..."`).
`normMethod`	`character(1)`. Which measure to plot. One of `"beta"` or `"nrpm"`. **Default:** `"beta"`.

useGroupMeans `logical(1)`. If `TRUE`, average samples by the `group` column in the sample table (combine replicates). **Default:** `FALSE`.

sampleAnnotation
: tidyselect specification or `character()` or `NULL`. Columns from the sample table to display as column annotations (e.g., `c("tumour","tissue")` or bare helpers `tumour, tissue`). **Default:** `NULL`.

minDensity `numeric(1)`. Minimum `CpG_density` required to keep a window. **Default:** `0`.

minEnrichment `numeric(1)`. For `"beta"`, values with enrichment `< minEnrichment` are set to `NA`. **Default:** `3`.

maxScale `numeric(1)`. Upper limit of the colour scale (used for `"nrpm"`; not applied to `"beta"`). **Default:** `1`.

clusterNum `integer(1)` or `NULL`. If set, cut the **column** dendrogram into this many clusters and display cluster labels. **Default:** `NULL`.

annotationColors
: `list` or `NA`. Optional named colour maps for annotations, e.g. `list(tumour = c(Tumour = "firebrick4", Normal = "blue"))`. **Default:** `NA`.

upstreamDist `integer(1)`. Number of base pairs upstream of the gene to include. **Default:** `3000`.

scaleRows `logical(1)`. Whether to z-score scale rows (windows) before plotting. **Default:** `FALSE`.

clusterCols `logical(1)`. Whether to cluster columns (samples). **Default:** `TRUE`.

mart `biomaRt::Mart` or `NULL`. Ensembl mart used to resolve gene coordinates. If `NULL`, attempts to use a mart stored on the `qseaSet` (see `setMart()`); otherwise falls back to a default for GRCh38/hg38 or hg19. **Default:** `NULL`.

showSampleNames
: `logical(1)` or `NULL`. If `NULL`, names are shown when there are fewer than 50 samples; set `TRUE`/`FALSE` to force. **Default:** `NULL`.

downstreamDist `integer(1)`. Number of base pairs downstream of the gene to include. **Default:** `1000`.

idType `character(1)` or `NULL`. Column in the mart to match `gene` against (needed for non-human/mouse setups). **Default:** `NULL`.

... Additional arguments forwarded to **ComplexHeatmap** constructors.

Details

The function resolves the gene to genomic coordinates (using `mart`, or a mart stored on the object, or a built-in default for human builds), expands the interval by `upstreamDist`/`downstreamDist`, filters windows by overlap and `minDensity`, then draws a heatmap of `"beta"` or `"nrpm"`. For `"beta"`, values below `minEnrichment` are set to `NA`. If `useGroupMeans = TRUE`, samples are aggregated by `group` prior to plotting. Column annotations can be added via `sampleAnnotation`; colours can be customized via `annotationColors`.

Value

Draws a heatmap on the active device and (invisibly) returns the underlying **ComplexHeatmap** object (e.g., a `Heatmap`/`HeatmapList`) for further composition with `draw`.

See Also

plotRegionsHeatmap(), qsea::getSampleTable(), setMart(), biomaRt::useMart(), ComplexHeatmap::Heatmap()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Basic gene heatmap (beta) with defaults
tryCatch(
    exampleTumourNormal %>% plotGeneHeatmap("HOXA9"),
    error = function(e) message("Ensembl unavailable: ", conditionMessage(e))
)

# Add sample annotations (tidyselect bare names) and cluster columns into
# 2 groups
tryCatch(
    exampleTumourNormal %>%
        plotGeneHeatmap("HOXA9",
            sampleAnnotation = c(tumour, tissue),
            clusterNum = 2),
    error = function(e) message("Ensembl unavailable: ", conditionMessage(e))
)

# Custom colours and wider flanks
tryCatch(
    exampleTumourNormal %>%
        plotGeneHeatmap("HOXA9",
            sampleAnnotation = tumour,
            annotationColors = list(
                tumour = c(Tumour = "firebrick4", Normal = "blue")
            ),
            upstreamDist = 1000,
            downstreamDist = 2000),
    error = function(e) message("Ensembl unavailable: ", conditionMessage(e))
)
```

---

plotGenomicFeatureDistribution

Distribution of windows across genomic features

Description

Annotate windows (e.g., promoters/exons/introns) and plot the distribution per sample as stacked/dodged/filled bars.

Usage

```
plotGenomicFeatureDistribution(
  qseaSet,
  cutoff = 1,
  barType = "stack",
  normMethod = "nrpm",
  genome = NULL,
  TxDb = NULL,
  annoDb = NULL
)
```

Arguments

qseaSet	qseaSet. Input object containing windows and signal (beta or nrpm).
cutoff	numeric(1). Threshold applied to the chosen normalisation measure per window. **Default:** 1.
barType	character(1). Bar layout: one of "stack", "dodge", or "fill" (relative proportions). **Default:** "stack".
normMethod	character(1). Normalisation/measure to threshold. One of "nrpm" or "beta". **Default:** "nrpm".
genome	character(1) or NULL. Genome build (e.g., "hg38", "hg19", "mm10"). If provided, uses mesa's genome system via getMesaTxDb() and getMesaAnnoDb(). **Default:** NULL (uses TxDb and annoDb parameters).
TxDb	TxDb object or NULL. Transcript database for gene annotation. Ignored if genome is provided. **Default:** TxDb.Hsapiens.UCSC.hg38.knownGene for backward compatibility.
annoDb	character(1) or NULL. Annotation database name (e.g., "org.Hs.eg.db"). Ignored if genome is provided. **Default:** "org.Hs.eg.db".

Details

Feature classes are taken from region annotations available on the windows (e.g., columns produced by annotateWindows() such as shortAnno/annotation, if present). Bars are positioned according to barType (stack/dodge/fill).

Value

A ggplot2 object showing counts (or proportions, for "fill") of feature classes per sample above cutoff.

See Also

annotateWindows(), qsea::makeTable(), ggplot2::geom_bar()

Other annotation-summaries: getGenomicFeatureDistribution()

Examples

```
# Recommended workflow: set genome globally
setMesaGenome("hg38")
data(exampleTumourNormal, package = "mesa")

# Uses global hg38 setting
exampleTumourNormal %>%
    plotGenomicFeatureDistribution(normMethod = "beta", cutoff = 0.75)

# Override for specific analysis
exampleTumourNormal %>%
    plotGenomicFeatureDistribution(
        genome = "mm10",
        normMethod = "nrpm",
        cutoff = 2
    )

# Advanced: manual control (bypasses mesa genome system)
exampleTumourNormal %>%
    plotGenomicFeatureDistribution(
        TxDb = TxDb.Mmusculus.UCSC.mm10.knownGene::
            TxDb.Mmusculus.UCSC.mm10.knownGene,
        annoDb = "org.Mm.eg.db"
    )
```

---

plotPCA.mesaDimRed *Plot principal component analysis (PCA) results*

Description

Generate publication-ready plots from the output of `getPCA()`. The function supports flexible visual encodings (colour, shape, size) of sample-level metadata and can return multiple plots across different PC pairs and annotation variables.

Usage

```
plotPCA.mesaDimRed(
  object,
  components = list(c(1, 2), c(2, 3)),
  colour = NULL,
  colourPalette = NULL,
  NAcolour = "grey50",
  symDivColourScale = FALSE,
  shape = NULL,
  shapePalette = NULL,
  NAshape = NULL,
```

```
  showSampleNames = FALSE,
  pointSize = 2,
  alpha = 1,
  plotlyAnnotations = ""
)
```

Arguments

object	mesaDimRed. A dimensionality-reduction container returned by getPCA().
components	integer(2) **or** list of integer(2). PC indices to plot (e.g. c(1, 2) for PC1 vs PC2), or a list of such pairs to produce multiple plots. **Default:** list(c(1, 2), c(2, 3)).
colour	NULL or character() . Name(s) of sample-table variable(s) mapped to point colour; one plot per variable. **Default:** NULL (no colour mapping).
colourPalette	NULL or character(). Palette used for colouring points. If NULL, a suitable default is chosen. **Default:** NULL.
NAcolour	character(1). Colour for missing values in colour. **Default:** "grey50".
symDivColourScale	logical(1). If TRUE, diverging colour scales are centred symmetrically around zero. Ignored for non-diverging scales. **Default:** FALSE.
shape	NULL or character(1). Sample-table variable mapped to point shape (single variable only). **Default:** NULL.
shapePalette	NULL, character(1), or numeric(). Shapes used for categories of the shape variable. Options: • A numeric vector of base-R shape codes. – 0–20: line/filled shapes. – 21–25: filled shapes with a border (border colour = black). • A keyword, with internally defined sets: • "line-first": 15 line shapes, then 4 filled shapes (**max 19 categories**). • "filled-first": 4 filled shapes, then 15 line shapes (**max 19 categories**). – "mixture": mixture of line and filled shapes (**max 19 categories**). • "filled+border": filled+border shapes (**max 5 categories, or 4 if NAs in** shape). • NULL: choose automatically ("mixture" for non-diverging/none; "filled+border" for diverging colour scales). **Default:** NULL.
NAshape	NULL or numeric(1). Shape for missing values in shape. If NULL, uses 7, or 25 when "filled+border" is active. **Default:** NULL.
showSampleNames	logical(1). Overlay sample names on points. **Default:** FALSE.
pointSize	numeric(1). Point size. **Default:** 2.
alpha	numeric(1). Point transparency in [0, 1]. **Default:** 1.
plotlyAnnotations	character() . Column names from the sample table to add as tooltips when converting to interactive plotly. **Default:** "" (empty string).

Details

Shape defaults

- `"mixture"` is used for non-diverging or discrete colour scales (or no colour).
- `"filled+border"` is used for diverging colour scales. These defaults aim to keep categories visually separable for larger cohorts.

Value

A `list` of `ggplot` objects—one per combination of `components` pairs and `colour` variables—each depicting the chosen PCs with requested aesthetics. Specifically:

- one plot per `components` pair × per `colour` variable;
- shapes/labels applied per `shape`/`showSampleNames`;
- consistent axis labelling (PC variance if available).

See Also

`getPCA()`, `plotUMAP()`, `plotDimRed()`, `qsea::getSampleTable()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Basic PCA plot (PC1 vs PC2) with defaults
exampleTumourNormal %>%
    getPCA() %>%
    plotPCA()

# Colour by sample group; show names
exampleTumourNormal %>%
    getPCA() %>%
    plotPCA(
        colour = "group",
        showSampleNames = TRUE
    )

# Plot PC1 vs PC3; shape by tissue
exampleTumourNormal %>%
    getPCA() %>%
    plotPCA(
        components = list(c(1, 3)),
        colour = "group",
        shape = "tissue"
    )
```

`plotRegionsHeatmap` *Heatmap of signal across selected genomic regions*

Description

Plot sample signal over a set of regions as a heatmap using **ComplexHeatmap**. Regions are selected by overlap with `regionsToOverlap`, values are taken from the `qseaSet`, and optional annotations from the sample/region metadata can be added.

Usage

```
plotRegionsHeatmap(
  qseaSet,
  regionsToOverlap = NULL,
  normMethod = "beta",
  sampleAnnotation = NULL,
  windowAnnotation = NULL,
  annotationColors = NA,
  useGroupMeans = FALSE,
  clusterRows = FALSE,
  clusterCols = TRUE,
  minEnrichment = 3,
  maxScale = 5,
  clusterNum = NULL,
  clip = 1e+09,
  minDensity = 0,
  annotationPosition = "right",
  title = NULL,
  showSampleNames = NULL,
  clusterMethod = "ward.D2",
  ...
)
```

Arguments

`qseaSet`	`qseaSet`. Input object containing windows and counts/betas.
`regionsToOverlap`	`GRanges`, `data.frame` (coercible to `GRanges`), or `NULL`. If provided, only windows overlapping these regions are plotted; otherwise all windows are considered. **Default:** `NULL`.
`normMethod`	`character(1)`. Which measure to plot, `"beta"` or `"nrpm"`. **Default:** `"beta"`.
`sampleAnnotation`	`character()` or `NULL`. Names of columns from the sample table to display as column annotations (e.g., `c("tumour","tissue")`). **Default:** `NULL`.

windowAnnotation
: character() or NULL. Names of columns from region metadata (from the qseaSet regions or regionsToOverlap) to display as row annotations (e.g., c("CpG_density")). **Default:** NULL.

annotationColors
: list or NA. Optional named colour maps for annotations, e.g. list(tumour = c(Tumour = "firebrick4", Normal = "blue")). **Default:** NA.

useGroupMeans
: logical(1). If TRUE, average samples by the group column in the sample table (combine replicates). **Default:** FALSE.

clusterRows
: logical(1). Whether to cluster rows (windows). **Default:** FALSE.

clusterCols
: logical(1). Whether to cluster columns (samples). **Default:** TRUE.

minEnrichment
: numeric(1). For "beta", values with enrichment < minEnrichment are set to NA. **Default:** 3.

maxScale
: numeric(1). Upper limit of the colour scale (used for "nrpm"; not applied to "beta"). **Default:** 5.

clusterNum
: integer(1) or NULL. If set, cut the **column** dendrogram into this many clusters and display cluster labels. **Default:** NULL.

clip
: numeric(1). Cap values above this threshold before plotting (applies to "nrpm", ignored for "beta"). **Default:** 1e9.

minDensity
: numeric(1). Minimum CpG_density required to keep a window. **Default:** 0.

annotationPosition
: character(1). Where to place column annotations (e.g., "right" or "bottom"). **Default:** "right".

title
: character(1) or NULL. Optional plot title. **Default:** NULL.

showSampleNames
: logical(1) or NULL. If NULL, names are shown when there are fewer than 50 samples; set TRUE/FALSE to force. **Default:** NULL.

clusterMethod
: character(1). Clustering method for dendrograms (e.g., "ward.D2"). **Default:** "ward.D2".

...
: Additional arguments forwarded to **ComplexHeatmap** constructors.

Details

Windows are filtered by overlap (if regionsToOverlap is given) and by minDensity. For "beta", low-enrichment values are set to NA using minEnrichment. For "nrpm", clip and maxScale help stabilise colour scaling for outliers. If useGroupMeans = TRUE, samples are aggregated by group before plotting.

Value

Draws a heatmap on the active device and (invisibly) returns the underlying **ComplexHeatmap** object (e.g., a Heatmap/HeatmapList) for further composition with draw.

See Also

qsea::makeTable(), ComplexHeatmap::Heatmap(), draw, calculateDMRs(), qsea::getSampleTable(), getWindows()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Compute regions (DMRs) to plot
DMRs <- exampleTumourNormal %>%
    calculateDMRs(variable = "tumour", contrasts = "first")

# Basic heatmap of beta values over DMRs
exampleTumourNormal %>% plotRegionsHeatmap(DMRs)

# Cluster rows, add sample and window annotations
exampleTumourNormal %>%
    plotRegionsHeatmap(DMRs,
        clusterRows = TRUE,
        sampleAnnotation = c(tumour, tissue))

# Group means, 2 column clusters, and custom annotation colours
exampleTumourNormal %>%
    plotRegionsHeatmap(regionsToOverlap = DMRs,
        clusterRows = TRUE,
        clusterNum = 2,
        sampleAnnotation = tumour,
        windowAnnotation = CpG_density,
        annotationColors = list(
            tumour = c("Tumour" = "firebrick4", "Normal" = "blue")
        )
    )
```

---

plotUMAP *Plot UMAP results*

Description

Convenience wrapper around `plotDimRed()` for UMAP embeddings returned by `getUMAP()`. Supports flexible visual encodings (colour, shape, size) of sample-level metadata, and can return multiple plots across UMAP component pairs and annotation variables.

Usage

```
plotUMAP(
  object,
  components = list(c(1, 2)),
  colour = NULL,
  colourPalette = NULL,
  NAcolour = "grey50",
  symDivColourScale = FALSE,
  shape = NULL,
  shapePalette = NULL,
```

```
  NAshape = NULL,
  showSampleNames = FALSE,
  pointSize = 2,
  alpha = 1,
  plotlyAnnotations = ""
)
```

Arguments

object	mesaDimRed. A dimensionality-reduction container returned by getUMAP().
components	integer(2) **or** list of integer(2). UMAP indices to plot (e.g. c(1, 2) for UMAP1 vs UMAP2), or a list of such pairs to produce multiple plots. **Default:** list(c(1, 2)).
colour	NULL or character(). Name(s) of sample-table variable(s) mapped to point colour; one plot per variable. **Default:** NULL (no colour mapping).
colourPalette	NULL or character(). Palette used for colouring points. If NULL, a suitable default is chosen. **Default:** NULL.
NAcolour	character(1). Colour for missing values in colour. **Default:** "grey50".
symDivColourScale	logical(1). If TRUE, diverging colour scales are centred symmetrically around zero. Ignored for non-diverging scales. **Default:** FALSE.
shape	NULL or character(1). Sample-table variable mapped to point shape (single variable only). **Default:** NULL.
shapePalette	NULL, character(1), or numeric(). Shapes used for categories of the shape variable. Options: • A numeric vector of base-R shape codes. – 0–20: line/filled shapes. – 21–25: filled shapes with a border (border colour = black). • A keyword, with internally defined sets: • "line-first": 15 line shapes, then 4 filled shapes (**max 19 categories**). • "filled-first": 4 filled shapes, then 15 line shapes (**max 19 categories**). – "mixture": mixture of line and filled shapes (**max 19 categories**). • "filled+border": filled+border shapes (**max 5 categories, or 4 if NAs in** shape). • NULL: choose automatically ("mixture" for non-diverging/none; "filled+border" for diverging colour scales). **Default:** NULL.
NAshape	NULL or numeric(1). Shape for missing values in shape. If NULL, uses 7, or 25 when "filled+border" is active. **Default:** NULL.
showSampleNames	logical(1). Overlay sample names on points. **Default:** FALSE.
pointSize	numeric(1). Point size. **Default:** 2.
alpha	numeric(1). Point transparency in [0, 1]. **Default:** 1.
plotlyAnnotations	character(). Column names from the sample table to add as tooltips when converting to interactive plotly. **Default:** "" (empty string).

Value

A `list` of `ggplot` objects—one per combination of `components` pairs and `colour` variables—each depicting the chosen UMAP dimensions with requested aesthetics. Specifically:

- one plot per `components` pair × per `colour` variable;
- shapes/labels applied per `shape`/`showSampleNames`;
- consistent axis labelling (UMAP1/UMAP2 etc.).

See Also

Other dimred-helpers: `getPCA()`, `plotDimRed()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Default UMAP (beta-normalised, top 1000 most variable windows)
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, min_dist = 1) %>%
    plotUMAP()

# Colour by group; show names
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, min_dist = 1) %>%
    plotUMAP(
        colour = "group",
        showSampleNames = TRUE
    )

# Custom components and shape mapping
exampleTumourNormal %>%
    getUMAP(n_neighbors = 5, min_dist = 1) %>%
    plotUMAP(
        components = list(c(1, 2)),
        colour = "group",
        shape = "tissue"
    )
```

`poolSamples` *Pool (merge) samples that share a common name prefix*

Description

Merge samples whose names share a common prefix by removing a suffix pattern (`mergeString`) and summing counts across the resulting groups. CNV and zygosity values are combined using fragment-count weights. The result is a new `qseaSet` with pooled samples and updated metadata.

Usage

```
poolSamples(qseaSet, mergeString)
```

Arguments

qseaSet	qseaSet A qseaSet object containing methylation-enriched sequencing data.
mergeString	character(1) Regular expression used to remove the **suffix** that distinguishes samples to be merged. The remaining prefix becomes the pooled sample name. After removal, pooled names are matched using startsWith(). For example, with names like "Patient1_T" and "Patient1_N", use mergeString = "_[TN]$" to pool them into "Patient1".

Details

Internally, pooled counts are simple sums. CNV values and zygosity are combined by a weighted average using total_fragments from @libraries$input_file. Library statistics are aggregated (sums for counts, weighted means for rates).

Value

A qseaSet object:

- **poolSamples()**: returns a new qseaSet where samples with names sharing a common prefix (after applying mergeString) are merged. Counts are summed, CNV and zygosity values are combined using weighted averages, and library statistics are aggregated.

Examples

```
## Not run:
data(exampleTumourNormal)
# Pool Tumour/Normal pairs by patient into a single sample per patient:
# e.g., "Colon1_T","Colon1_N" -> "Colon1"
poolSamples(exampleTumourNormal, mergeString = "_[TN]$")

## End(Not run)
```

---

pull.qseaSet *Pull a column from a qseaSet sample table*

Description

Extends dplyr::pull() to extract a column from the sampleTable of a qseaSet.

Usage

```
## S3 method for class 'qseaSet'
pull(.data, var = -1, name = NULL, ...)
```

Arguments

.data	qseaSet. A qseaSet object whose sampleTable column should be extracted.
var	Column to extract (name or position). **Default:** -1 (last column).
name	Optional column to use for names in the returned vector. **Default:** NULL (no names).
...	Additional arguments passed to dplyr::pull(). **Default:** none.

Value

A vector with length equal to the number of samples in the qseaSet. If name is provided, the vector is named accordingly.

See Also

qsea::getSampleTable(), dplyr::pull()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Pull the 'group' column as a vector
exampleTumourNormal %>% pull(group)

# Pull by column position (last column by default)
exampleTumourNormal %>% pull()

# Pull a column and name it by sample_name
exampleTumourNormal %>% pull(group, name = sample_name)
```

---

qseaTableToChrGRanges *Convert a makeTable-like data frame to GRanges (UCSC style)*

Description

Coerce a table of windows to a GRanges-class. Two input layouts are supported:

- chr, window_start, window_end (as in qsea::makeTable() output), or
- seqnames, start, end.

Usage

```
qseaTableToChrGRanges(dataTable)
```

Arguments

dataTable	data.frame. Table of windows to convert. Must contain either the trio chr/window_start/window_end **or** seqnames/start/end. **Default:** none (must be supplied).

Details

If chromosome labels lack the "chr" prefix, it is added automatically so that seqnames are in UCSC style (e.g., "chr1").

- When given chr/window_start/window_end, the columns are renamed to seqnames/start/end before coercion.
- When given seqnames/start/end, seqnames are first coerced to character and normalised to include the "chr" prefix.
- The returned GRanges does **not** set a genome tag; set it yourself if needed (e.g., GenomeInfoDb::genome(gr) <- "hg38").

Value

A GRanges with seqnames in UCSC style. The genome is unset.

See Also

qsea::makeTable(), as_granges, GenomeInfoDb::seqlevelsStyle()

Examples

```
# From makeTable-like columns
data.frame(chr = c("1", "2"),
    window_start = c(100, 500),
    window_end = c(200, 600)) %>%
    qseaTableToChrGRanges()

# From seqnames/start/end; adds 'chr' if missing
data.frame(seqnames = c("chr3", "4"),
    start = c(1000, 2000),
    end = c(1100, 2100)) %>%
    qseaTableToChrGRanges()
```

---

removeCNV *Remove (zero out) CNV data from a qseaSet*

Description

Set all per-window CNV values to zero while keeping genomic coordinates and sample metadata intact. Useful for resetting CNV prior to recomputation.

Usage

```
removeCNV(qseaSet)
```

Arguments

qseaSet	qseaSet. Input object whose CNV assay will be replaced by zeros.

Details

This operation preserves regions (windows) and sample metadata; only the per-window CNV values are replaced with zeros. Use this to "clear" existing CNV calls before running `addHMMcopyCNV()` or other CNV routines again.

Value

A `qseaSet` with the CNV matrix zeroed (same dimensions as before).

See Also

`addHMMcopyCNV()`, `plotCNVheatmap()`, `qsea::addCNV()`

Other CNV: `addHMMcopyCNV()`, `plotCNVheatmap()`, `runHMMCopy()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Zero out CNV and inspect the first rows
exampleTumourNormal %>%
    removeCNV() %>%
    getCNV() %>%
    utils::head()
```

`removeLibraryFactors` *Set library factors to 1*

Description

Overwrite all library-factor values in a `qseaSet` (and its `sampleTable`) with `1`. Useful when you want to avoid library scaling (e.g., TMM) and keep beta unchanged; NRPM may change depending on downstream use.

Usage

```
removeLibraryFactors(qseaSet)
```

Arguments

`qseaSet`	`qseaSet`. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).

Details

Setting library factors to `1` has **no effect on beta** (fractional measure) but can affect **NRPM** since scaling factors are neutralised. This is akin to calling `qsea::addLibraryFactors(qseaSet, factors = 1)` and synchronising the `sampleTable`.

Value

A qseaSet with all library factors set to 1 and sampleTable updated.

See Also

addLibraryInformation(), getNRPMTable(), qsea::addLibraryFactors()

Other table-helpers: getBetaTable(), getCountTable(), getDataTable(), getNRPMTable(), getSampleGroups2(), makeTransposedTable(), removeNormMethodSuffix()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Show current factors then reset to 1 and show again
exampleTumourNormal %>%
  addLibraryInformation() %>%
  pull(library_factor) %>%
  head()
```

---

removeNormMethodSuffix

Remove normalisation suffix from column names

Description

Clean wide tables (e.g., from getDataTable()) by stripping the trailing _{normMethod} or _{normMethod}_means suffixes from sample / group-mean columns.

Usage

```
removeNormMethodSuffix(dataTable, normMethod)
```

Arguments

dataTable	data.frame or tibble. Window × sample (or group) table whose column names may include a normalisation suffix (e.g., "sampleA_beta", "Tumour_beta_means"). **Default:** none (must be supplied).
normMethod	character(1). Normalisation/measure tag to remove from column names (e.g., "beta", "nrpm"). **Default:** none (must be supplied).

Details

Intended for post-processing tables where column names encode the measure, e.g., "sample_beta", "group_beta_means". The function drops those suffixes to yield bare sample/group names, facilitating downstream joins and plotting.

Value

A data.frame like dataTable but with cleaned column names (suffixes _{normMethod} and _{normMethod}_means removed where present).

See Also

getDataTable(), selectQset(), pull.qseaSet()

Other table-helpers: getBetaTable(), getCountTable(), getDataTable(), getNRPMTable(), getSampleGroups2(), makeTransposedTable(), removeLibraryFactors()

Examples

```
# Basic cleaning for 'beta'
data.frame(A_beta = 1:3, B_beta_means = 4:6) %>%
  removeNormMethodSuffix("beta") %>%
  names()

# Works similarly for other measures (e.g., 'nrpm')
data.frame(S1_nrpm = 1:2, Tumour_nrpm_means = 3:4) %>%
  removeNormMethodSuffix("nrpm") %>%
  names()
```

---

renameQsetNames *Rename samples in a qseaSet by regex*

Description

Apply a regular expression replacement to sample names and update all relevant slots consistently (counts, CNV, enrichment, zygosity, libraries, and the sampleTable).

Usage

```
renameQsetNames(qseaSet, pattern, replacement = "")
```

Arguments

qseaSet	qseaSet A qseaSet object containing methylation-enriched sequencing data.
pattern	character(1) Regular expression used to match substrings in sample names.
replacement	character(1) String to replace matches of pattern. Defaults to "" (empty string).

Value

A qseaSet object:

- **renameQsetNames()**: returns the input qseaSet with sample names updated according to pattern and replacement. All dependent slots are updated consistently.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Replace "T" with "Tumour" in sample names
renameQsetNames(exampleTumourNormal,
    pattern = "T",
    replacement = "Tumour"
)
```

`renameSamples` *Rename samples using a column from the sample table*

Description

Replace sample names with values from a user-supplied column in the `sampleTable` of a `qseaSet`, updating all dependent slots consistently.

Usage

```
renameSamples(qseaSet, newNameColumn)
```

Arguments

`qseaSet`	`qseaSet` A qseaSet object containing methylation-enriched sequencing data.
`newNameColumn`	`character(1)` The name of a column in the `sampleTable` containing new sample names. Values must be unique and contain no `NA`s. (Unquoted column names are supported via tidy evaluation.)

Value

A `qseaSet` object:

- **renameSamples()**: returns the input `qseaSet` with sample names replaced by values from `newNameColumn`. All associated slots are updated to reflect the new naming.

Examples

```
data(exampleTumourNormal, package = "mesa")

exampleTumourNormal %>%
    dplyr::mutate(
        newName = paste0(
            "Sample",
            seq_len(nrow(qsea::getSampleTable(.)))
        )
    ) %>%
    renameSamples(newNameColumn = "newName")
```

runHMMCopy *Run HMMcopy on per-window reads for a single sample*

Description

Given a table of per-window counts with GC and mappability, run HMMcopy correction and segmentation, and return a GRanges of CNV calls for the selected sample/column.

Usage

```
runHMMCopy(CNV_RegionsWithReads, colname, plotDir = NULL)
```

Arguments

CNV_RegionsWithReads	data.frame / data.table / GRanges. Coercible table with columns: chr, start, end, width, strand, gc, map, and one sample-specific count column referenced by colname.
colname	character(1). Name of the sample count column to use (e.g., "sampleA").
plotDir	character(1) or NULL. Directory to write a per-sample PDF diagnostic plot named <colname>.pdf; if NULL, no plot is produced. **Default:** NULL.

Details

Counts are GC/map corrected via HMMcopy::correctReadcount() and segmented with HMMcopy::HMMsegment(). The result is mapped back to the input windows and returned as a GRanges. Ensure that the binning (width) and covariate columns (gc, map) are consistent across the table.

Value

A GRanges with a metadata column named <colname> containing per-window CNV estimates (typically HMM state means on the log2 scale).

See Also

addHMMcopyCNV(), HMMcopy::correctReadcount(), HMMcopy::HMMsegment()

Other CNV: addHMMcopyCNV(), plotCNVheatmap(), removeCNV()

Examples

```
## Not run:
# Minimal synthetic example: build a table and pipe into runHMMCopy()
set.seed(1)
n <- 2000L; w <- 5e4
tibble::tibble(
    chr = factor(rep(c("chr1", "chr2"),
        each = n / 2),
    levels = c("chr1", "chr2")),
```

```
    start = rep(seq(1, by = w, length.out = n / 2), 2),
    end = start + w - 1L,
    width = w, strand = "*",
    gc = runif(n, 0.05, 0.95), #' wide spread => stable LOESS
    map = runif(n, 0.60, 0.98), #' avoid extremes / exact 1.0
    sampleA = rpois(n, 500)
) %>%
    runHMMCopy(colname = "sampleA", plotDir = NULL)

## End(Not run)
```

`select.qseaSet` *Select or rename columns in a qseaSet sample table*

Description

Extends `dplyr::select()` to keep or rename columns in a `qseaSet`'s `sampleTable`. This allows column subsetting, renaming, and use of tidyselect helpers while preserving key identifiers.

Usage

```
## S3 method for class 'qseaSet'
select(.data, ...)
```

Arguments

`.data`	`qseaSet`. A qseaSet object whose `sampleTable` columns will be selected or renamed.
`...`	Additional arguments passed to `dplyr::select()`, supports tidyselect helpers (e.g., `dplyr::matches()`, `dplyr::starts_with()`). Negative selection (`-col`) is supported, but `sample_name` and `group` cannot be removed. **Default:** none.

Details

This is a wrapper around `selectQset()`. Essential columns `sample_name` and `group` are always preserved (and appear first if present).

Value

A `qseaSet` with its `sampleTable` updated:

- contains only selected/renamed columns,
- always includes `sample_name` and `group` (if available).

See Also

`selectQset()` for the underlying implementation, `qsea::getSampleTable()` to view results, `mutate.qseaSet()`, `filter.qseaSet()` for related dplyr verbs.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep only specific columns (sample_name and group always preserved)
exampleTumourNormal %>%
    dplyr::select(type, tumour) %>%
    qsea::getSampleTable()

# Select and rename a column
exampleTumourNormal %>%
    dplyr::select(age, tumour_new = tumour) %>%
    qsea::getSampleTable()

# Use tidyselect helpers
exampleTumourNormal %>%
    dplyr::select(matches("s")) %>%
    qsea::getSampleTable()

# Negative selection (except essential columns)
exampleTumourNormal %>%
    dplyr::select(-matches("s")) %>%
    qsea::getSampleTable()
```

`selectQset` *Select or rename columns of a qseaSet sample table*

Description

Keeps or renames columns from the `sampleTable` of a `qseaSet`. Essential columns `sample_name` and `group` are always preserved (and placed at the front of the table if present).

Usage

```
selectQset(qseaSet, ...)
```

Arguments

`qseaSet`	`qseaSet`. A qseaSet object whose `sampleTable` columns will be selected or renamed.
`...`	Additional arguments passed to `dplyr::select()`. Supports tidyselect helpers (e.g., `dplyr::matches()`, `dplyr::starts_with()`) and renaming (`newName = oldName`). Negative selection is supported, but `sample_name` and `group` cannot be removed. **Default:** none.

Value

A qseaSet with its sampleTable updated:

- contains only the selected/renamed columns,
- always includes sample_name and group (if available).

See Also

select.qseaSet() for the dplyr S3 method, qsea::getSampleTable() to inspect the updated table.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep only sample_name and group
exampleTumourNormal %>%
    selectQset(sample_name, group) %>%
    qsea::getSampleTable()

# Select and rename a column
exampleTumourNormal %>%
    selectQset(age, tumour_type = tumour) %>%
    qsea::getSampleTable()

# Use tidyselect helpers
exampleTumourNormal %>%
    selectQset(matches("s")) %>%
    qsea::getSampleTable()
```

---

setMart *Set or get an Ensembl/BioMart handle on a qseaSet*

Description

Store and retrieve a "mart" handle inside a qseaSet for downstream annotation tasks (e.g., resolving gene coordinates).

Usage

```
setMart(object, ...)

## S4 method for signature 'qseaSet'
setMart(object, mart)

getMart(object, ...)

## S4 method for signature 'qseaSet'
getMart(object)
```

Arguments

object	qseaSet. The object to modify or query.
...	Additional arguments passed to methods or to biomaRt::useMart(). Not used in the default "qseaSet" method.
mart	biomaRt::Mart, character(1), or ANY. Connection object returned by **biomaRt** (recommended), or a string label you interpret elsewhere (e.g., "ENSEMBL_110"). Stored verbatim in object@parameters$mart. **Default:** none (must be supplied to setMart()).

Details

These are S4 generics with methods for class "qseaSet". The value is not validated or dereferenced here; functions that consume it (e.g., gene annotation helpers) should perform any needed checks.

Value

- setMart() returns the updated qseaSet (with @parameters$mart set).
- getMart() returns the stored value (whatever was set), or NULL if absent.

Methods

setMart(object, mart) Set the handle on a qseaSet.

getMart(object) Retrieve the stored handle from a qseaSet.

See Also

annotateWindows(), setMesaGenome(), setMesaTxDb(), setMesaAnnoDb(), biomaRt::useMart()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Store a label (or use a real biomaRt::Mart) and read it back
exampleTumourNormal %>%
   setMart("ENSEMBL_110") %>%
   getMart()
```

---

setMesaAnnoDb *Set default OrgDb (annoDb) for downstream annotation helpers*

Description

Record a preferred **OrgDb** package to use when functions such as annotateWindows() are called without an explicit annoDb.

Usage

```
setMesaAnnoDb(annoDb)
```

Arguments

annoDb	`character(1)` or `NULL`. Name of an installed OrgDb package (e.g., `"org.Hs.eg.db"` or `"org.Mm.eg.db"`), or `NULL` to clear the setting. The package must be installed if a string is supplied. **Default:** none (must be supplied).

Details

This sets the session option `options(mesa_annoDb = <annoDb>)`. When present, helpers like `annotateWindows()` will use this OrgDb by default (particularly for GRCh38 workflows) unless an explicit annoDb is provided. Use `setMesaGenome()` and `setMesaTxDb()` to configure complementary defaults for the genome and TxDb, respectively.

Value

Invisibly returns `TRUE` on success.

See Also

`annotateWindows()`, `setMesaGenome()`, `setMesaTxDb()`, `ChIPseeker::annotatePeak()`

Examples

```
# Set default annoDb for human (only if the package is installed)
if (requireNamespace("org.Hs.eg.db", quietly = TRUE)) {
    setMesaAnnoDb("org.Hs.eg.db")
    getOption("mesa_annoDb")
}

# Set default annoDb for mouse (only if the package is installed)
if (requireNamespace("org.Mm.eg.db", quietly = TRUE)) {
    setMesaAnnoDb("org.Mm.eg.db")
    getOption("mesa_annoDb")
}

# Unset the default annoDb
setMesaAnnoDb(NULL)
is.null(getOption("mesa_annoDb"))
```

setMesaGenome *Set default genome for downstream annotation helpers*

Description

Record a preferred genome string (e.g., "hg38" / "GRCh38") in the session options so that functions like annotateWindows() can use it when genome is not explicitly supplied.

Usage

```
setMesaGenome(genome)
```

Arguments

genome	character(1) or NULL. Genome identifier to store (typically "hg38" or "GRCh38"). Use NULL to clear the setting. **Default:** none (must be supplied).

Details

This function sets the session option options(mesa_genome = <genome>). It **does not** modify TxDb or OrgDb options directly; set those via setMesaTxDb() and setMesaAnnoDb() if you want global defaults for transcript and gene annotation packages. Helpers such as annotateWindows() consult getOption("mesa_genome") when genome is missing.

Value

Invisibly returns TRUE on success.

See Also

annotateWindows(), setMesaTxDb(), setMesaAnnoDb()

Examples

```
# Set the default genome to hg38
setMesaGenome("hg38")
getOption("mesa_genome")

# Use the GRCh38 alias
setMesaGenome("GRCh38")
getOption("mesa_genome")

# Reset/remove the default genome
setMesaGenome(NULL)
is.null(getOption("mesa_genome"))
```

`setMesaParallel` *Manage mesa parallelisation (set & query)*

Description

`setMesaParallel()` enables/disables parallelisation (and may register a backend); `getMesaParallel()` returns the current setting.

Usage

```
setMesaParallel(nCores = NULL, useParallel = FALSE, verbose = TRUE)

getMesaParallel(verbose = FALSE)
```

Arguments

`nCores`	`integer(1)` or `NULL`. If `> 1` on Unix/macOS, registers `BiocParallel::MulticoreParam(workers = nCores)` and enables parallel mode. Ignored on Windows (see **Details**). **Default:** `NULL`.
`useParallel`	`logical(1)`. Explicitly turn mesa parallelisation on/off (sets the `"mesa_parallel"` option). If `nCores > 1`, this is treated as `TRUE`. **Default:** `FALSE`.
`verbose`	Boolean to determine whether to print messages or not.

Details

- This function stores the flag in `options(mesa_parallel = <TRUE/FALSE>)`.
- On Unix/macOS, supplying `nCores > 1` will **register** a `MulticoreParam` backend via `BiocParallel::register()` and enable parallelisation.
- On **Windows**, forked backends are unavailable; you should manually register a compatible backend (e.g., `SnowParam`) and then call `setMesaParallel(useParallel = TRUE)`.
- You can always inspect the active backend with `BiocParallel::bpparam()` and the worker count with `BiocParallel::bpworkers()`.

Value

- `setMesaParallel()` — logical scalar (`TRUE`/`FALSE`), returned **invisibly**; primarily used for its side effects (setting the option and possibly registering a backend).
- `getMesaParallel()` — logical scalar indicating whether mesa parallelisation is currently enabled; with `verbose = TRUE`, also prints a status message.

Functions

- `getMesaParallel()`: Get whether mesa is using parallelisation.

See Also

BiocParallel::register(), BiocParallel::MulticoreParam(), BiocParallel::SnowParam(), BiocParallel::bpworkers()

Examples

```
# Turn parallelisation OFF (serial evaluation)
setMesaParallel(useParallel = FALSE, verbose = FALSE)
getMesaParallel() # FALSE

# --- Unix/mac (MulticoreParam): enable and use 2 cores ---
## Not run:
if (.Platform$OS.type != "windows") {
    old <- BiocParallel::bpparam() # save current backend
    setMesaParallel(nCores = 2, verbose = FALSE)
    # registers MulticoreParam(2)
    BiocParallel::bpworkers() # e.g. 2
    getMesaParallel() # TRUE
    BiocParallel::register(old) # restore previous backend
    setMesaParallel(useParallel = FALSE, verbose = FALSE) # back to serial
}

## End(Not run)

# --- Windows: register SnowParam yourself, then enable ---
## Not run:
if (.Platform$OS.type == "windows") {
    old <- BiocParallel::bpparam()
    BiocParallel::register(BiocParallel::SnowParam(workers = 2))
    setMesaParallel(useParallel = TRUE, verbose = FALSE)
    BiocParallel::bpworkers() # e.g. 2
    getMesaParallel() # TRUE
    BiocParallel::register(old)
    setMesaParallel(useParallel = FALSE, verbose = FALSE)
}

## End(Not run)
```

---

setMesaTxDb *Set default TxDb for downstream annotation helpers*

Description

Record a preferred TxDb to use when functions such as annotateWindows() are called without an explicit TxDb.

Usage

```
setMesaTxDb(TxDb)
```

Arguments

TxDb	character(1), TxDb **object**, or NULL. Either the name of an installed TxDb package (e.g., "TxDb.Hsapiens.UCSC.hg38.knownGene"), the TxDb object itself, or NULL to clear the setting. If a string is supplied, the package must be installed. **Default:** none (must be supplied).

Details

This sets the session option options(mesa_TxDb = TxDb). When a string is stored, helpers like annotateWindows() will load the TxDb object on demand (e.g., "TxDb.Hsapiens.UCSC.hg38.knownGene"). Use setMesaGenome() and setMesaAnnoDb() to configure complementary defaults for genome and OrgDb.

Value

Invisibly returns TRUE on success.

See Also

annotateWindows(), setMesaGenome(), setMesaAnnoDb(), ChIPseeker::annotatePeak()

Examples

```
# Set default TxDb for human (GRCh38/hg38)
setMesaTxDb("TxDb.Hsapiens.UCSC.hg38.knownGene")
getOption("mesa_TxDb")

# Clear the default TxDb
setMesaTxDb(NULL)
getOption("mesa_TxDb")

# (Alternatively) you can store the object itself if already loaded:
# setMesaTxDb(
#   TxDb.Hsapiens.UCSC.hg38.knownGene::TxDb.Hsapiens.UCSC.hg38.knownGene
# )
```

---

sliceDMRs *Take the top most DMRs per contrast, based on those with the largest value of the selected metric*

Description

Take the top most DMRs per contrast, based on those with the largest value of the selected metric

Usage

```
sliceDMRs(
  DMRs,
  n = 1,
  metric = deltaBeta,
  makePositive = TRUE,
  FDRthres = 0.05
)
```

Arguments

DMRs	A data frame containing the output of calculateDMRs, potentially with multiple contrasts
n	How many DMRs to take of each contrast (if that many exists)
metric	Which metric to use to select the top DMRs. Options are deltaBeta, log2FC, adjPval, CpG_density, position (using seqnames and start columns) or any other column in the data frame. If `adjPval` or `position` are used, then the window with the smallest value will be chosen, otherwise the largest value will be used.
makePositive	Whether to reverse the contrast when the window is hypomethylated in the contrast.
FDRthres	Threshold on the adjusted p values

Value

A data frame with the DMRs with the largest value of the selected metrics

Examples

```
# calculate some DMRs
DMRs <- exampleTumourNormal %>%
    calculateDMRs(
        variable = "type", contrasts = "all",
        keepContrastMeans = FALSE
    )
# Find the DMRs with the largest deltaBeta between each comparison:
DMRs %>% sliceDMRs(n = 1)
# Or the windows with the largest log2FC:
DMRs %>% sliceDMRs(n = 1, metric = log2FC)
# Or the windows with the largest CpG_density:
DMRs %>% sliceDMRs(n = 1, metric = CpG_density)
# If adjPval is used, then the smallest value is chosen instead:
DMRs %>% sliceDMRs(n = 1, metric = CpG_density)
# If position is used, then the windows are sorted by genomic position:
DMRs %>% sliceDMRs(n = 1, metric = position)
```

`sort.qseaSet` *Sort samples in a qseaSet*

Description

Extends `base::sort()` to reorder the **samples** of a `qseaSet` using `gtools::mixedsort()` so that numeric parts sort naturally (e.g., `"2"` comes before `"10"`).

Usage

```
## S3 method for class 'qseaSet'
sort(x, decreasing = FALSE, ...)
```

Arguments

`x`	`qseaSet`. A qseaSet object to reorder.
`decreasing`	`logical(1)`. Whether to reverse the order. **Default:** `FALSE`.
`...`	Additional arguments passed on to `gtools::mixedsort()`. **Default:** none.

Value

A `qseaSet` object with reordered samples. The content of all slots (e.g., `count_matrix`, `sampleTable`) is updated consistently.

See Also

`qsea::getSampleNames()`, `subsetQset()`, `gtools::mixedsort()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Sort samples in natural (mixed) order
exampleTumourNormal %>% sort()

# Sort in reverse order
exampleTumourNormal %>% sort(decreasing = TRUE)
```

subsetQset *Subset a qseaSet by samples*

Description

Restrict a qseaSet to a specified set of samples. All internal matrices and slots are subset consistently. Use exactly one of samplesToKeep or samplesToDrop.

Usage

```
subsetQset(qseaSet, samplesToKeep = NULL, samplesToDrop = NULL)
```

Arguments

qseaSet	qseaSet A qseaSet object containing methylation-enriched sequencing data.
samplesToKeep	character() Names of samples to retain. Cannot be used together with samplesToDrop. **Default:** NULL.
samplesToDrop	character() Names of samples to remove. Cannot be used together with samplesToKeep. **Default:** NULL.

Value

A qseaSet object:

- **subsetQset()**: returns the input qseaSet restricted to the selected samples.

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep only two samples
subsetQset(exampleTumourNormal, samplesToKeep = c("Colon1_T", "Colon1_N"))

# Drop one sample
subsetQset(exampleTumourNormal, samplesToDrop = "Lung1_N")
```

subsetWindowsBySignal *Subset windows in a qseaSet by signal across samples*

Description

Filter genomic windows based on a summary function applied across selected samples. Typical uses include keeping windows with median NRPM above a threshold, or retaining windows where the minimum beta is below 0.5.

Usage

```
subsetWindowsBySignal(
  qseaSet,
  fn,
  threshold,
  aboveThreshold,
  samples = NULL,
  normMethod = "nrpm",
  useGroupMeans = FALSE
)
```

Arguments

qseaSet	qseaSet. Input object containing windows and signal.
fn	function. Summary function applied **row-wise** across the selected samples (e.g., median, min, max). If you need NA handling, provide it explicitly, e.g. function(x) median(x, na.rm = TRUE). **Default:** none (must be supplied).
threshold	numeric(1). Threshold applied to the summary value returned by fn. **Default:** none (must be supplied).
aboveThreshold	logical(1). Keep windows with values >= threshold when TRUE; keep windows with values < threshold when FALSE. **Default:** none (must be supplied).
samples	character() or NULL. Sample names to include, or a single string used as a pattern to match sample names. If NULL, use all samples. **Default:** NULL.
normMethod	character(1). Normalisation/measure to evaluate. One of "nrpm", "beta", or "counts". **Default:** "nrpm".
useGroupMeans	logical(1). If TRUE, aggregate replicates by the group column in the sample table and apply fn to group means; if FALSE, use individual samples. **Default:** FALSE.

Details

The function builds a window × sample matrix for the chosen normMethod, optionally restricts to samples via samples (exact names or pattern match), computes fn per row, and filters by threshold according to aboveThreshold. Supply a summary function that accepts a numeric vector and returns a scalar; include na.rm = TRUE inside fn if needed.

Value

A filtered qseaSet containing only the selected windows.

See Also

getDataTable(), filterByOverlaps()

Other window-helpers: calculateFractionReadsInGRanges(), convertToArrayBetaTable(), downSample(), subsetWindowsOverBackground()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep windows with median NRPM > 1 across all samples
exampleTumourNormal %>%
  subsetWindowsBySignal(
    fn = median,
    threshold = 1,
    aboveThreshold = TRUE
  )

# Keep windows where the MIN beta < 0.5 in any sample
exampleTumourNormal %>%
  subsetWindowsBySignal(
    fn = min,
    threshold = 0.5,
    aboveThreshold = FALSE,
    normMethod = "beta"
  )

# Restrict to samples whose names contain "Lung"
exampleTumourNormal %>%
  subsetWindowsBySignal(
    fn = median,
    threshold = 1,
    aboveThreshold = TRUE,
    samples = "Lung"
  )

# Use group means instead of individual samples
exampleTumourNormal %>%
  subsetWindowsBySignal(
    fn = median,
    threshold = 1,
    aboveThreshold = TRUE,
    useGroupMeans = TRUE
  )
```

`subsetWindowsOverBackground` *Subset windows above Poisson background*

Description

Identify genomic windows with significantly more reads than expected under a Poisson background model, given total reads per sample and the number of genome windows. If `numWindows` is provided, then that value will be used as the total original number of windows under consideration for the false discovery rate, otherwise it will use the number of windows currently in the qseaSet.

Usage

```
subsetWindowsOverBackground(
  qseaSet,
  keepAbove = FALSE,
  samples = NULL,
  numWindows = NULL,
  FDRthres = 0.01,
  numAbove = 1
)
```

Arguments

qseaSet	qseaSet. Input object containing per-window read counts.
keepAbove	logical(1). If TRUE, **keep** windows above background; if FALSE, **remove** them. **Default:** FALSE.
samples	character() or NULL. Vector of sample names to test, or a single string used as a pattern matched against sample names. If NULL, all samples are tested. **Default:** NULL.
numWindows	integer(1) or NULL. Total number of windows in the genome used to compute the expected background. If NULL, it will use all the windows currently in the qseaSet. **Default:** NULL.
FDRthres	numeric(1). FDR threshold used to call windows significantly above background. **Default:** 0.01.
numAbove	integer(1). Minimum number of tested samples that must exceed background in a window to keep/drop it (depending on keepAbove). **Default:** 1.

Details

For each tested sample, an expected count per window is derived from its total reads and the supplied/estimated numWindows, and a Poisson test is used to flag windows above background. This uses the valid_fragments column in the library information attached to the qseaSet, which is the number of fragments used in the generation of the original qseaSet. Windows are retained or removed based on keepAbove and the requirement that at least numAbove samples are significant at FDRthres.

Value

A filtered qseaSet containing only windows passing the criteria.

See Also

subsetWindowsBySignal()

Other window-helpers: calculateFractionReadsInGRanges(), convertToArrayBetaTable(), downSample(), subsetWindowsBySignal()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Keep windows above Poisson background in at least 2 samples whose names
# contain "Lung", assuming there were 9 million original windows
exampleTumourNormal %>%
  subsetWindowsOverBackground(keepAbove = TRUE, samples = "Lung",
                              numAbove = 2, numWindows = 9e6 )

# Drop windows above background (retain only background-like windows) across
# all samples, assuming there were 9 million original windows
exampleTumourNormal %>%
  subsetWindowsOverBackground(keepAbove = FALSE, numWindows = 9e6)
```

---

summariseAcrossWindows

Summarise across windows per sample

Description

Apply a summary function (e.g., mean, median, sd) over genomic windows per sample, optionally restricted to a set of regions and for one or more normalisation methods.

Usage

```
summariseAcrossWindows(
  qseaSet,
  regionsToOverlap = NULL,
  fn = mean,
  addSampleTable = TRUE,
  normMethod = c("nrpm", "beta"),
  naMethod = "na.rm",
  minEnrichment = 3,
  suffix = "",
  fnName = NULL
)
```

Arguments

qseaSet	qseaSet. Input object providing window-level signal. **Default:** none (must be supplied).
regionsToOverlap	GRanges, data.frame, or NULL. Regions used to restrict the windows before summarising. Data frames must be coercible to GRanges (e.g., have seqnames/start/end or chr/window_start/window_end). If NULL, use all windows in qseaSet. **Default:** NULL.

fn	function. Summary function applied **column-wise per sample** over the selected windows (e.g., mean, median, sd). Provide NA handling inside fn if needed, e.g., function(x) mean(x, na.rm = TRUE). **Default:** mean.
addSampleTable	logical(1). If TRUE, join sample-table columns to the result. **Default:** TRUE.
normMethod	character(). One or more measures to summarise. Typically a subset of c("nrpm","beta"). **Default:** c("nrpm","beta").
naMethod	character(1). How to treat missing values prior to/within summarisation. Common choices: "na.rm" — call fn with na.rm = TRUE when supported; "drop" — drop windows with any NA across the selected methods before summarising. **Default:** "na.rm".
minEnrichment	integer(1). Minimum reads per window required for a non-NA beta (relevant when normMethod includes "beta"). **Default:** 3.
suffix	character(1). Optional string appended to output column names (e.g., a region label). **Default:** "".
fnName	character(1) or NULL. Name of fn used to construct output column names; if NULL, inferred from fn. **Default:** NULL.

Details

The function builds a window × sample matrix for each normMethod, optionally restricts to regionsToOverlap, then applies fn over windows for each sample to produce a single summary value per sample (per method). Missing-value handling follows naMethod. When summarising betas, windows below minEnrichment reads are NA prior to summarisation.

Value

A tibble with one row per sample. For each requested normMethod, a summary column is added (column naming typically reflects <method> and fn and may include suffix). If addSampleTable = TRUE, sample-table columns are joined.

See Also

addSummaryAcrossWindows(), getDataTable()

Other window-summaries: addSummaryAcrossWindows(), countWindowsAboveCutoff()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Mean NRPM and beta across all windows
exampleTumourNormal %>%
  summariseAcrossWindows(
    fn         = mean,
    normMethod = c("nrpm","beta")
  )

# Maximum NRPM within a small genomic region
exampleTumourNormal %>%
```

```
  summariseAcrossWindows(
    regionsToOverlap = data.frame(
        seqnames = 7,
        start = 25002001,
        end = 25017900
    ),
    fn         = max,
    normMethod = "nrpm",
    suffix     = "_chr7_slice"
  )

# Median beta over all windows
exampleTumourNormal %>%
  summariseAcrossWindows(
    fn         = median,
    normMethod = "beta",
    minEnrichment = 3
  )
```

`summariseDMRsByContrast` *Summarise DMRs by contrast*

Description

Count the number of up- and down-regulated DMR windows per contrast. Internally, results are reshaped to long format before summarisation.

Usage

```
summariseDMRsByContrast(
  DMRtable,
  FDRthres = 0.05,
  log2FCthres = 0,
  deltaBetaThres = 0
)
```

Arguments

`DMRtable`	`data.frame` or `GRanges` DMR results, typically returned by `calculateDMRs()`. If in wide format, it will be converted with `pivotDMRsLonger()`.
`FDRthres`	`numeric(1)` False discovery rate threshold for significance. **Default:** `0.05`.
`log2FCthres`	`numeric(1)` Absolute log2 fold-change threshold for calling up/down regulation. **Default:** `0`.
`deltaBetaThres`	`numeric(1)` Absolute delta-beta threshold for calling up/down regulation. **Default:** `0`.

Value

A tibble:

- **summariseDMRsByContrast**(): returns one row per contrast, with counts of up-regulated (nUp) and down-regulated (nDown) DMRs that pass the specified thresholds.

See Also

pivotDMRsLonger(), summariseDMRsByGene()

Other DMR-helpers: pivotDMRsLonger(), summariseDMRsByGene(), writeDMRsToBed(), writeDMRsToExcel()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Summarise DMRs for a single contrast, using custom FDR threshold
exampleTumourNormal %>%
    calculateDMRs(
        variable = "type",
        contrasts = "LUAD_vs_NormalLung",
        FDRthres = 0.1
    ) %>%
    summariseDMRsByContrast(FDRthres = 0.1)
```

---

summariseDMRsByGene *Summarise DMRs by gene*

Description

Aggregate differentially methylated region (DMR) windows by gene annotation.

Usage

```
summariseDMRsByGene(DMRtable)
```

Arguments

DMRtable	data.frame or GRanges Annotated DMRs. Must include the columns ENSEMBL, SYMBOL, and GENENAME (typically added with annotateWindows()).

Value

A tibble:

- **summariseDMRsByGene**(): returns one row per gene, with counts of associated DMRs and any aggregated metadata.

See Also

annotateWindows(), summariseDMRsByContrast()

Other DMR-helpers: pivotDMRsLonger(), summariseDMRsByContrast(), writeDMRsToBed(), writeDMRsToExcel()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Summarise DMRs with explicit annotation databases
exampleTumourNormal %>%
    calculateDMRs(variable = "tumour", contrasts = "all") %>%
    annotateWindows(
        TxDb = "TxDb.Hsapiens.UCSC.hg38.knownGene",
        annoDb = "org.Hs.eg.db"
    ) %>%
    summariseDMRsByGene()
```

---

writeBigWigs *Write bigWig tracks per sample or group*

Description

Export bigWig files with per-window scores for each sample (or group means).

Usage

```
writeBigWigs(
  qseaSet,
  folderName,
  normMethod = "nrpm",
  useGroupMeans = FALSE,
  naVal = -1
)
```

Arguments

qseaSet	qseaSet. A qseaSet object containing methylation-enriched sequencing data. **Default:** none (must be supplied).
folderName	character(1). Output directory where .bw files will be written (created if missing). **Default:** none (must be supplied).
normMethod	character(1). Normalisation/measure to export. Common choices: "nrpm" or "beta". **Default:** "nrpm".
useGroupMeans	logical(1). If TRUE, export group-mean tracks instead of per-sample tracks. **Default:** FALSE.
naVal	numeric(1). Value used to replace NA scores prior to export (useful for beta). **Default:** 0.

Details

For each sample (or group), the function pairs the per-window scores from `qseaSet` with the corresponding window genomic coordinates and writes a bigWig via **rtracklayer**. When exporting beta, windows failing the minimum read threshold used to compute beta may be `NA`; these are replaced by `naVal` before writing.

File naming. Output files are typically named using the sample (or group) name and the method, e.g. `"<sample>_<method>.bw"` or `"<group>_<method>_means.bw"` when `useGroupMeans = TRUE`.

Seqinfo. bigWig export requires chromosome lengths; ensure the window GRanges in `qseaSet` carries valid `seqinfo` (this is the case for the example data on GRCh38).

Value

(Invisibly) `NULL`. Files are written to `folderName`.

See Also

`getDataTable()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Per-sample NRPM bigWigs
td <- tempfile()
dir.create(td)
exampleTumourNormal %>%
  writeBigWigs(folderName = td, normMethod = "nrpm", useGroupMeans = FALSE)
list.files(td, pattern = "\\\\.bw$")

# Group-mean beta bigWigs (replace NA beta with 0)
td2 <- tempfile()
dir.create(td2)
exampleTumourNormal %>%
  writeBigWigs(
      folderName = td2,
      normMethod = "beta",
      useGroupMeans = TRUE,
      naVal = 0
  )
list.files(td, pattern = "\\\\.bw$")
```

---

writeDMRsToBed *Write DMR results to BED files*

Description

Export differentially methylated region (DMR) results to a set of BED files, with one file generated per contrast.

Usage

```
writeDMRsToBed(dataTable, folder, FDRthres = 0.05)
```

Arguments

dataTable	data.frame DMR results, typically returned by calculateDMRs(). If a GRanges, it will be coerced to a data frame internally.
folder	character(1) Directory in which to write BED files. Must exist and be writable.
FDRthres	numeric(1) False discovery rate threshold used to filter DMRs before writing. **Default:** 0.05.

Value

(Invisibly) returns the input object:

- **writeDMRsToBed**(): returns dataTable (invisibly), enabling use in a pipeline.

See Also

writeDMRsToExcel()

Other DMR-helpers: pivotDMRsLonger(), summariseDMRsByContrast(), summariseDMRsByGene(), writeDMRsToExcel()

Examples

```
data(exampleTumourNormal, package = "mesa")

# Export DMRs for a single contrast to BED files in a temp directory
exampleTumourNormal %>%
    calculateDMRs(
        variable = "type",
        contrasts = "LUAD_vs_NormalLung"
    ) %>%
    writeDMRsToBed(folder = tempdir())

# Export DMRs with explicit annotation (requires TxDb/annotation packages)
exampleTumourNormal %>%
    calculateDMRs(variable = "tumour", contrasts = "all") %>%
    annotateWindows(
        TxDb = "TxDb.Hsapiens.UCSC.hg38.knownGene",
        annoDb = "org.Hs.eg.db"
    ) %>%
    writeDMRsToBed(folder = tempdir(), FDRthres = 0.1)
```

`writeDMRsToExcel` *Write DMR results to an Excel workbook*

Description

Export differentially methylated region (DMR) results to an Excel workbook, creating one worksheet per contrast.

Usage

```
writeDMRsToExcel(dataTable, path, FDRthres = 0.05)
```

Arguments

`dataTable`	`data.frame` or `GRanges` DMR results, typically produced by `calculateDMRs()`. If a `GRanges`, it will be coerced to a data frame internally.
`path`	`character(1)` File path of the Excel workbook to write (e.g., `"dmr_results.xlsx"`).
`FDRthres`	`numeric(1)` False discovery rate threshold used to filter DMRs before writing. **Default:** `0.05`.

Details

One worksheet is created per contrast found in `dataTable`. If no contrast column is present, all rows are written to a single worksheet. If file I/O is restricted on the system (e.g., some HPC build environments), consider writing to `tempfile(fileext = ".xlsx")` in examples or tests.

Value

(Invisibly) returns the input object:

- **writeDMRsToExcel()**: returns `dataTable` (invisibly), enabling use in a pipeline.

See Also

`writeDMRsToBed()`

Other DMR-helpers: `pivotDMRsLonger()`, `summariseDMRsByContrast()`, `summariseDMRsByGene()`, `writeDMRsToBed()`

Examples

```
data(exampleTumourNormal, package = "mesa")

# Minimal example: write results for a single contrast
exampleTumourNormal %>%
    calculateDMRs(variable = "type", contrasts = "LUAD_vs_NormalLung") %>%
    writeDMRsToExcel(path = file.path(tempdir(), "test.xlsx"))

# With multiple contrasts and annotation
```

```
exampleTumourNormal %>%
    calculateDMRs(variable = "tumour", contrasts = "all") %>%
    annotateWindows(
        TxDb = "TxDb.Hsapiens.UCSC.hg38.knownGene",
        annoDb = "org.Hs.eg.db"
    ) %>%
    writeDMRsToExcel(path = file.path(tempdir(), "test.xlsx"))
```

Index