

# Package ‘raer’

April 18, 2025

**Type** Package

**Title** RNA editing tools in R

**Version** 1.7.0

**Description** Toolkit for identification and statistical testing of RNA editing signals from within R. Provides support for identifying sites from bulk-RNA and single cell RNA-seq datasets, and general methods for extraction of allelic read counts from alignment files. Facilitates annotation and exploratory analysis of editing signals using Bioconductor packages and resources.

**License** MIT + file LICENSE

**Imports** stats, methods, GenomicRanges, IRanges, Rsamtools, BSgenome, Biostrings, SummarizedExperiment, SingleCellExperiment, S4Vectors, GenomeInfoDb, GenomicAlignments, GenomicFeatures, BiocGenerics, BiocParallel, rtracklayer, Matrix, cli

**Suggests** testthat (>= 3.0.0), knitr, DESeq2, edgeR, limma, rmarkdown, BiocStyle, ComplexHeatmap, TxDb.Hsapiens.UCSC.hg38.knownGene, SNPlocs.Hsapiens.dbSNP144.GRCh38, BSgenome.Hsapiens.NCBI.GRCh38, scater, scan, scuttle, AnnotationHub, covr, raerdata, txdbmaker

**LinkingTo** Rhtslib

**SystemRequirements** GNU make

**VignetteBuilder** knitr

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**URL** <https://rnabioco.github.io/raer>, <https://github.com/rnabioco/raer>

**BugReports** <https://github.com/rnabioco/raer/issues>

**biocViews** MultipleComparison, RNASeq, SingleCell, Sequencing, Coverage, Epitranscriptomics, FeatureExtraction, Annotation, Alignment

**Config/Needs/website** pkgdown, rnabioco/rbitemplate

**Config/testthat/edition** 3

**git\_url** <https://git.bioconductor.org/packages/raer>

**git\_branch** devel

**git\_last\_commit** d7ecf03

**git\_last\_commit\_date** 2025-04-15

**Repository** Bioconductor 3.22

**Date/Publication** 2025-04-17

**Author** Kent Riemondy [aut, cre] (ORCID:

<<https://orcid.org/0000-0003-0750-1273>>),

Kristen Wells-Wrasman [aut] (ORCID:

<<https://orcid.org/0000-0002-7466-8164>>),

Ryan Sheridan [ctb] (ORCID: <<https://orcid.org/0000-0003-4012-3147>>),

Jay Hesselberth [ctb] (ORCID: <<https://orcid.org/0000-0002-6299-179X>>),

RNA Bioscience Initiative [cph, fnd]

**Maintainer** Kent Riemondy <[kent.riemondy@gmail.com](mailto:kent.riemondy@gmail.com)>

## Contents

|                                     |    |
|-------------------------------------|----|
| annot_from_gr . . . . .             | 2  |
| annot_snps . . . . .                | 3  |
| calc_AEI . . . . .                  | 5  |
| calc_confidence . . . . .           | 6  |
| calc_edit_frequency . . . . .       | 7  |
| calc_scAEI . . . . .                | 8  |
| correct_strand . . . . .            | 10 |
| filter_clustered_variants . . . . . | 11 |
| filter_multiallelic . . . . .       | 12 |
| filter_splice_variants . . . . .    | 13 |
| find_de_sites . . . . .             | 14 |
| find_mispriming_sites . . . . .     | 15 |
| find_scde_sites . . . . .           | 17 |
| get_overlapping_snps . . . . .      | 18 |
| get_splice_sites . . . . .          | 18 |
| make_de_object . . . . .            | 19 |
| mock_rse . . . . .                  | 20 |
| pileup_cells . . . . .              | 21 |
| pileup_sites . . . . .              | 23 |
| raer . . . . .                      | 27 |
| raer_example . . . . .              | 27 |
| read_sparray . . . . .              | 28 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>30</b> |
|--------------|-----------|

---

annot\_from\_gr

*Annotate sites using GRanges object*

---

## Description

Utility function to map annotations from GRanges to rowData of SummarizedExperiment or to mcols of GRanges object. If multiple features overlap then they will be concatenated with the specified separator string.

**Usage**

```
annot_from_gr(obj, gr, cols_to_map, RLE = TRUE, sep = ",", ...)
```

**Arguments**

|             |   |
|-------------|---|
| obj         | RangedSummarizedExperiment or GRanges object  |
| gr          | GRanges with annotations to map to obj  |
| cols_to_map | character vector of columns from GRanges to map to SummarizedExperiment. If the vector has names, the names will be the column names in the output. |
| RLE         | If TRUE, columns added will returned as <code>S4Vectors::Rle()</code> vectors to reduce memory  |
| sep         | separator string, defaults to comma.  |
| ...         | additional arguments to pass to <code>GenomicRanges::findOverlaps()</code>  |

**Value**

Either a SummarizedExperiment or GRanges object with additional annotations provided by the supplied GRanges object.

**Examples**

```
library(SummarizedExperiment)
rse_adar_ifn <- mock_rse()
gr <- GRanges(rep(c("SSR3", "SPCS3"), c(5, 15)),
  IRanges(seq(1, 500, by = 25), width = 50),
  strand = "+")
)

gr$feature <- sample(1:100, size = 20)
gr$id <- sample(LETTERS, size = 20)

rse <- annot_from_gr(rse_adar_ifn, gr, c(feature_set = "feature", "id"))
rowData(rse)
```

---

annot\_snps

---

*Annotate known SNP positions*


---

**Description**

This function will annotate a [GRanges](#) or the rowRanges of a [SummarizedExperiment](#) with SNPs from a SNP package.

**Usage**

```
annot_snps(obj, ...)

## S3 method for class 'GRanges'
annot_snps(
  obj,
  dbsnp,
```

```

    chrom = NULL,
    col_to_aggr = "RefSNP_id",
    drop = FALSE,
    genome = NULL,
    RLE = TRUE,
    ...
)

## S3 method for class 'SummarizedExperiment'
annot_snps(obj, ...)
```

### Arguments

|             |   |
|-------------|---|
| obj         | GRanges or SummarizedExperiment object  |
| ...         | For the generic, further arguments to pass to specific methods. Unused for now.   |
| dbSNP       | SNPlocs package, see available packages from <a href="#">BSgenome::available.SNPs()</a>   |
| chrom       | only operate on a specified chromosome  |
| col_to_aggr | column from SNPlocs package to add to input. If multiple SNPs overlap these values will be concatenated as comma separated values.  |
| drop        | If TRUE, remove sites overlapping SNPs  |
| genome      | A BSgenome object, which if supplied, will be used to provide additional snp_ref_allele and snp_alt_alleles columns containing the reference and alt allele sequences, with respect to the positive strand. Additionally the snp sequences will be checked against the allele at the site if a column named ALT is present in object. The strand of the site will be used to determine if the ALT allele needs to be complemented prior to comparing against the SNP db (which always returns sequences w.r.t the plus strand). |
| RLE         | If TRUE, columns added will returned as <a href="#">S4Vectors::Rle()</a> vectors to reduce memory usage.  |

### Value

Either a GRanges or SummarizedExperiment object with a new column added with information from col\_to\_aggr and optionally snp\_ref\_allele, snp\_alt\_alleles, and snp\_matches\_site annotations.

### See Also

[SNPlocs.Hsapiens.dbSNP144.GRCh38](#)

### Examples

```

if (require(SNPlocs.Hsapiens.dbSNP144.GRCh38)) {
  gr <- GRanges(rep("22", 10),
    IRanges(
      seq(10510077,
        10610077,
        by = 1000
      )[1:10],
      width = 250
    ),
    strand = "+"
  )
}
```

```

    )
    genome(gr) <- "GRCh38.p2"
    annot_snps(gr, SNPlocs.Hsapiens.dbSNP144.GRCh38)
}

```

calc\_AEI

*Calculate the Adenosine Editing Index (AEI)*

## Description

The Adenosine Editing Index describes the magnitude of A-to-I editing in a sample. The index is a weighted average of editing events (G bases) observed at A positions. The vast majority A-to-I editing occurs in ALU elements in the human genome, and these regions have a high A-to-I editing signal compared to other regions such as coding exons. This function will perform pileup at specified repeat regions and return a summary AEI metric.

## Usage

```

calc_AEI(
  bamfiles,
  fasta,
  alu_ranges = NULL,
  txdb = NULL,
  snp_db = NULL,
  param = FilterParam(),
  BPPARAM = SerialParam(),
  verbose = FALSE
)

```

## Arguments

|            |  |
|------------|--|
| bamfiles   | character vector of paths to indexed bam files. If a named character vector is supplied the names will be used in the output.  |
| fasta      | fasta filename   |
| alu_ranges | <a href="#">GRanges</a> with regions to query for calculating the AEI, typically ALU repeats.  |
| txdb       | A <a href="#">TxDb</a> object, if supplied, will be used to subset the alu_ranges to those found overlapping genes. Alternatively a <a href="#">GRanges</a> object with gene coordinates. If the library_type, specified by FilterParam, is unstranded then the <a href="#">TxDb</a> will be used to correct the strandness relative to the reference and is a required parameter.   |
| snp_db     | either a <a href="#">SNPlocs</a> , <a href="#">GPos</a> , or <a href="#">GRanges</a> object. If supplied, will be used to exclude polymorphic positions prior to calculating the AEI. If calc_AEI() will be used many times, one will save time by first identifying SNPs that overlap the supplied alu_ranges, and passing these as a <a href="#">GRanges</a> to snp_db rather than supplying all known SNPs (see <a href="#">get_overlapping_snps()</a> ). |
| param      | object of class <a href="#">FilterParam()</a> which specify various filters to apply to reads and sites during pileup.   |
| BPPARAM    | A <a href="#">BiocParallelParam</a> object for specifying parallel options for operating over chromosomes.   |
| verbose    | report progress on each chromosome?  |

**Value**

A named list containing:

- AEI: a matrix of AEI index values computed for all allelic combinations, one row for each supplied bam file.
- AEI\_per\_chrom: a data.frame containing values computed for each chromosome

**References**

Roth, S.H., Levanon, E.Y. & Eisenberg, E. Genome-wide quantification of ADAR adenosine-to-inosine RNA editing activity. Nat Methods 16, 1131–1138 (2019). <https://doi.org/10.1038/s41592-019-0610-9>

**Examples**

```
suppressPackageStartupMessages(library(Rsamtools))

bamfn <- raer_example("SRR5564269_Aligned.sortedByCoord.out.md.bam")
bam2fn <- raer_example("SRR5564277_Aligned.sortedByCoord.out.md.bam")
bams <- c(bamfn, bam2fn)
names(bams) <- c("ADAR1KO", "WT")

fafn <- raer_example("human.fasta")
mock_alu_ranges <- scanFaIndex(fafn)

res <- calc_AEI(bams, fafn, mock_alu_ranges)
res$AEI
```

---

calc\_confidence

---

*Calculate confidence score for observing editing*


---

**Description**

Calculate a confidence score based on a Bayesian inverse probability model as described by Washburn et al. Cell Reports. 2015, and implemented in the SAILOR pipeline.

**Usage**

```
calc_confidence(
  se,
  edit_to = "G",
  edit_from = "A",
  per_sample = FALSE,
  exp_fraction = 0.01,
  alpha = 0L,
  beta = 0L
)
```

**Arguments**

|              |   |
|--------------|---|
| se           | SummarizedExperiment::SummarizedExperiment containing editing sites   |
| edit_to      | edited base   |
| edit_from    | non-edited base   |
| per_sample   | if TRUE, calculate confidence per sample, otherwise edited and non-edited counts will be summed across all samples. |
| exp_fraction | Numeric value between 0 and 1, specifying the expected error rate   |
| alpha        | Pseudo-count to add to non-edited base counts   |
| beta         | Pseudo-count to add to edited base counts   |

**Value**

SummarizedExperiment::SummarizedExperiment with either a new assay or rowData column named "confidence" depending on whether confidence is calculated per\_sample.

**References**

Washburn MC, Kakaradov B, Sundararaman B, Wheeler E, Hoon S, Yeo GW, Hundley HA. The dsRBP and inactive editor ADR-1 utilizes dsRNA binding to regulate A-to-I RNA editing across the *C. elegans* transcriptome. *Cell Rep.* 2014 Feb 27;6(4):599-607. doi: 10.1016/j.celrep.2014.01.011. Epub 2014 Feb 6. PMID: 24508457; PMCID: PMC3959997.

SAILOR pipeline: <https://github.com/YeoLab/sailor>

**Examples**

```
rse_adar_ifn <- mock_rse()
calc_confidence(rse_adar_ifn)
calc_confidence(rse_adar_ifn, per_sample = TRUE)
```

---

calc\_edit\_frequency     *Adds editing frequencies*

---

**Description**

Adds editing frequencies to an existing [RangedSummarizedExperiment](#) object (created by [pileup\\_sites\(\)](#)). The [RangedSummarizedExperiment](#) with a new assay for editing frequencies for each site (edit\_freq), depth of coverage computed using the indicated edited nucleotides (depth) and new colData columns with the number of edited sites (n\_sites) and the fraction of edits (edit\_idx) is returned.

**Usage**

```
calc_edit_frequency(
  rse,
  edit_from = "A",
  edit_to = "G",
  drop = FALSE,
  replace_na = TRUE,
  edit_frequency = 0,
  min_count = 1
)
```

**Arguments**

|                |   |
|----------------|---|
| rse            | A <a href="#">RangedSummarizedExperiment</a> object created by <a href="#">pileup_sites()</a>   |
| edit_from      | This should correspond to a nucleotide or assay (A, C, G, T, Ref, or Alt) you expect in the reference. Ex. for A to I editing events, this would be A.                            |
| edit_to        | This should correspond to a nucleotide or assay (A, C, G, T, Ref, or Alt) you expect in the editing site. Ex. for A to I editing events, this would be G.                         |
| drop           | If TRUE, the <a href="#">RangedSummarizedExperiment</a> returned will only retain sites matching the specified edit_from and edit_to bases.                                       |
| replace_na     | If TRUE, NA and NaN editing frequencies will be coerced to 0.   |
| edit_frequency | The edit frequency cutoff used when calculating the number of sites. Set to 0 to require any non-zero editing frequency. The number of sites is stored as n_sites in the colData. |
| min_count      | The minimum number of reads required when enumerating number of editing sites detected.   |

**Value**

[RangedSummarizedExperiment](#) supplemented with edit\_freq and depth assay.

**Examples**

```
library(SummarizedExperiment)
rse_adar_ifn <- mock_rse()
rse <- calc_edit_frequency(rse_adar_ifn)
assay(rse, "edit_freq")[1:5, ]
```

---

calc\_scAEI

---

*Calculate the Adenosine Editing Index (AEI) in single cells*


---

**Description**

The Adenosine Editing Index describes the magnitude of A-to-I editing in a sample. The index is a weighted average of editing events (G bases) observed at A positions. The vast majority A-to-I editing occurs in ALU elements in the human genome, and these regions have a high A-to-I editing signal compared to other regions such as coding exons. This function will examine enumerate edited and non-edited base counts at the supplied sites and return summary AEI metric per cell. Potential editing sites within repeat regions can be generated using [get\\_scAEI\\_sites\(\)](#).

**Usage**

```
calc_scAEI(
  bamfiles,
  sites,
  cell_barcodes,
  param = FilterParam(),
  edit_from = "A",
  edit_to = "G",
  output_dir = NULL,
```



```

    return_sce = FALSE,
    ...
)

get_sCAEI_sites(fasta, genes, alus, edit_from = "A", edit_to = "G")

```

### Arguments

|               |   |
|---------------|---|
| bamfiles      | a path to a BAM file (for 10x libraries), or a vector of paths to BAM files (smart-seq2). Can be supplied as a character vector, BamFile, or BamFileList.   |
| sites         | a GRanges object produced by <code>get_sCAEI_sites()</code> containing sites to process.  |
| cell_barcodes | A character vector of single cell barcodes to process. If processing multiple BAM files (e.g. smart-seq-2), provide a character vector of unique identifiers for each input BAM, to name each BAM file in the output files. |
| param         | object of class <code>FilterParam()</code> which specify various filters to apply to reads and sites during pileup.   |
| edit_from     | This should correspond to the base (A, C, G, T) you expect in the reference. Ex. for A to I editing events, this would be A.  |
| edit_to       | This should correspond to the base (A, C, G, T) you expect in an edited site. Ex. for A to I editing events, this would be G.   |
| output_dir    | Output directory for nRef and nAlt sparseMatrix files. If NULL, a temporary directory will be used.   |
| return_sce    | if TRUE, data is returned as a SingleCellExperiment, if FALSE a DataFrame containing computed AEI values will be returned.  |
| ...           | additional arguments to <code>pileup_cells()</code>   |
| fasta         | Path to a genome fasta file   |
| genes         | A GRanges object with gene coordinates. Alternatively a TxDb object, which if supplied, will be used extract gene coordinates.  |
| alus          | GRanges with repeat regions to query for calculating the AEI, typically ALU repeats. The strand of the supplied intervals will be ignored for defining repeat regions.  |

### Value

A DataFrame containing computed AEI values, count of editing events (n\_alt), and count of reference events (n\_ref) per cell. If return\_sce is TRUE, then a SingleCellExperiment is returned with the AEI values stored in the colData.

### References

Roth, S.H., Levanon, E.Y. & Eisenberg, E. Genome-wide quantification of ADAR adenosine-to-inosine RNA editing activity. Nat Methods 16, 1131–1138 (2019). <https://doi.org/10.1038/s41592-019-0610-9>

### Examples

```

suppressPackageStartupMessages(library(Rsamtools))
library(GenomicRanges)

bam_fn <- raer_example("5k_neuron_mouse_possort.bam")

```

```

bai <- indexBam(bam_fn)

# cell barcodes to query
cbs <- c("TGTTTGTTCATCCGT-1", "CAACCAACATAATCGC-1", "TGGAAGCTCAAGCTGTT-1")

# genes used to infer transcribed strand
genes_gr <- GRanges(c(
  "2:100-400:-",
  "2:500-605:-",
  "2:600-680:+"
))

# alu intervals
alus_gr <- GRanges(c(
  "2:110-380",
  "2:510-600",
  "2:610-670"
))

# genome fasta file, used to find A bases
fa_fn <- raer_example("mouse_tiny.fasta")

# get positions of potential A -> G changes in alus
sites <- get_scAEI_sites(fa_fn, genes_gr, alus_gr)

fp <- FilterParam(
  library_type = "fr-second-strand",
  min_mapq = 255
)
calc_scAEI(bam_fn, sites, cbs, fp)

```

---

correct\_strand

*Apply strand correction using gene annotations*


---

## Description

Gene annotations are used to infer the likely strand of editing sites. This function will operate on unstranded datasets which have been processed using "unstranded" library type which reports variants with respect to the + strand for all sites. The strand of the editing site will be assigned the strand of overlapping features in the genes\_gr object. Sites with no-overlap, or overlapping features with conflicting strands (+ and -) will be removed.

## Usage

```
correct_strand(rse, genes_gr)
```

## Arguments

|          |  |
|----------|--|
| rse      | RangedSummarizedExperiment object containing editing sites processed with "unstranded" setting |
| genes_gr | GRanges object containing reference features to annotate the strand of the editing sites.      |

**Value**

RangedSummarizedExperiment object containing pileup assays, with strand corrected based on supplied genomic intervals.

**Examples**

```
suppressPackageStartupMessages(library("GenomicRanges"))

bamfn <- raer_example("SRR5564269_Aligned.sortedByCoord.out.md.bam")
fafn <- raer_example("human.fasta")
fp <- FilterParam(library_type = "unstranded")
rse <- pileup_sites(bamfn, fafn, param = fp)

genes <- GRanges(c(
  "DHFR:200-400:+",
  "SPCS3:100-200:-",
  "SSR3:3-10:-",
  "SSR3:6-12:+"
))

correct_strand(rse, genes)
```

---

filter\_clustered\_variants

*Filter out clustered sequence variants*

---

**Description**

Sequence variants of multiple allele types (e.g., A -> G, A -> C) proximal to a putative editing site can be indicative of a region prone to mis-alignment artifacts. Sites will be removed if variants of multiple allele types are present within a given distance in genomic or transcriptome coordinate space.

**Usage**

```
filter_clustered_variants(
  rse,
  txdb,
  regions = c("transcript", "genome"),
  variant_dist = 100
)
```

**Arguments**

|              |   |
|--------------|---|
| rse          | SummarizedExperiment::SummarizedExperiment containing editing sites                                       |
| txdb         | GenomicFeatures::TxDb   |
| regions      | One of transcript or genome, specifying the coordinate system for calculating distances between variants. |
| variant_dist | distance in nucleotides for determining clustered variants  |

**Value**

SummarizedExperiment::SummarizedExperiment with sites removed from object dependent on filtering applied.

**See Also**

Other se-filters: [filter\\_multiallelic\(\)](#), [filter\\_splice\\_variants\(\)](#)

**Examples**

```
if(require("txdbmaker")){
  rse_adar_ifn <- mock_rse()
  rse <- rse_adar_ifn[seqnames(rse_adar_ifn) == "SPCS3"]

  # mock up a txdb with genes
  gr <- GRanges(c(
    "SPCS3:100-120:-",
    "SPCS3:325-350:-"
  ))
  gr$source <- "raer"
  gr$type <- "exon"
  gr$source <- NA
  gr$phase <- NA_integer_
  gr$gene_id <- c(1, 2)
  gr$transcript_id <- c("1.1", "2.1")
  txdb <- txdbmaker::makeTxDbFromGRanges(gr)

  rse <- filter_multiallelic(rse)
  filter_clustered_variants(rse, txdb, variant_dist = 10)
}
```

---

|                     |                                       |
|---------------------|---------------------------------------|
| filter_multiallelic | <i>Filter out multi-allelic sites</i> |
|---------------------|---------------------------------------|

---

**Description**

Remove sites with multiple variant bases from a SummarizedExperiment. rowData() gains a new column, ALT, that contains the variant allele detected at each site.

**Usage**

```
filter_multiallelic(se)
```

**Arguments**

se SummarizedExperiment::SummarizedExperiment

**Value**

SummarizedExperiment::SummarizedExperiment with multiallelic sites removed. A new column, ALT will be added to rowData() indicating the single allele present at the site.

**See Also**

Other se-filters: [filter\\_clustered\\_variants\(\)](#), [filter\\_splice\\_variants\(\)](#)

**Examples**

```
rse_adar_ifn <- mock_rse()
filter_multiallelic(rse_adar_ifn)
```

---

filter\_splice\_variants

*Filter out sites near splice sites*

---

**Description**

Remove editing sites found in regions proximal to annotated splice junctions.

**Usage**

```
filter_splice_variants(rse, txdb, splice_site_dist = 4, ignore.strand = FALSE)
```

**Arguments**

|                  |   |
|------------------|---|
| rse              | SummarizedExperiment::SummarizedExperiment with editing sites       |
| txdb             | GenomicFeatures::TxDb   |
| splice_site_dist | distance to splice site   |
| ignore.strand    | if TRUE, ignore strand when comparing editing sites to splice sites |

**Value**

SummarizedExperiment::SummarizedExperiment with sites adjacent to splice sites removed.

**See Also**

Other se-filters: [filter\\_clustered\\_variants\(\)](#), [filter\\_multiallelic\(\)](#)

**Examples**

```
if(require("txdbmaker")) {
  rse_adar_ifn <- mock_rse()

  # mock up a txdb with genes
  gr <- GRanges(c(
    "DHFR:310-330:-",
    "DHFR:410-415:-",
    "SSR3:100-155:-",
    "SSR3:180-190:-"
  ))
  gr$source <- "raer"
  gr$type <- "exon"
  gr$source <- NA
```

```

gr$phase <- NA_integer_
gr$gene_id <- c(1, 1, 2, 2)
gr$transcript_id <- rep(c("1.1", "2.1"), each = 2)
txdb <- txdbmaker::makeTxDbFromGRanges(gr)

filter_splice_variants(rse_adar_ifn, txdb)
}

```

---

find\_de\_sites

*Perform differential editing*


---

## Description

Use edgeR or DESeq2 to perform differential editing analysis. This will work for designs that have 1 treatment and 1 control group. For more complex designs, we suggest you perform your own modeling.

## Usage

```

find_de_sites(
  deobj,
  test = c("edgeR", "DESeq2"),
  sample_col = "sample",
  condition_col = "condition",
  condition_control = NULL,
  condition_treatment = NULL
)

```

## Arguments

|                     |  |
|---------------------|--|
| deobj               | A <a href="#">RangedSummarizedExperiment</a> object prepared for differential editing analysis by <a href="#">make_de_object()</a>   |
| test                | Indicate if edgeR or DESeq2 should be run.   |
| sample_col          | The name of the column from <code>colData(deobj)</code> that contains your sample information. Default is sample. If you do not have a column named "sample", you must provide the appropriate sample column             |
| condition_col       | The name of the column from <code>colData(deobj)</code> that contains your treatment information. Default is condition. If you do not have a column named "condition", you must provide the appropriate condition column |
| condition_control   | The name of the control condition. This must be a variable in your <code>condition_col</code> of <code>colData(deobj)</code> . No default provided.  |
| condition_treatment | The name of the treatment condition. This must be a variable in your <code>condition_col</code> of <code>colData(deobj)</code> .   |

**Value**

A named list:

- `de_obj`: The edgeR or dseq object used for differential editing analysis
- `results_full`: Unfiltered differential editing results
- `sig_results`: Filtered differential editing (FDR < 0.05)
- `model_matrix`: The model matrix used for generating DE results

**Examples**

```
library(SummarizedExperiment)
bamfn <- raer_example("SRR5564269_Aligned.sortedByCoord.out.md.bam")
bam2fn <- raer_example("SRR5564277_Aligned.sortedByCoord.out.md.bam")
fafn <- raer_example("human.fasta")

bams <- rep(c(bamfn, bam2fn), each = 3)
sample_ids <- paste0(rep(c("KO", "WT"), each = 3), 1:3)
names(bams) <- sample_ids

fp <- FilterParam(only_keep_variants = TRUE)
rse <- pileup_sites(bams, fafn, param = fp)
rse$condition <- substr(rse$sample, 1, 2)

rse <- calc_edit_frequency(rse)
dse <- make_de_object(rse)
res <- find_de_sites(dse,
  condition_control = "WT",
  condition_treatment = "KO"
)
res$sig_results[1:3, ]
```

---

`find_mispriming_sites` *Find regions with oligodT mispriming*

---

**Description**

OligodT will prime at A-rich regions in an RNA. Reverse transcription from these internal priming sites will install an oligodT sequence at the 3' end of the cDNA. Sequence variants within these internal priming sites are enriched for variants converting the genomic sequence to the A encoded by the oligodT primer. Trimming poly(A) from the 3' ends of reads reduces but does not eliminate these signals

This function will identify regions that are enriched for mispriming events. Reads that were trimmed to remove poly(A) (encoded in the pa tag by 10x Genomics) are identified. The aligned 3' positions of these reads are counted, and sites passing thresholds (at least 2 reads) are retained as possible sites of mispriming. By default regions 5 bases upstream and 20 bases downstream of these putative mispriming sites are returned.

**Usage**

```
find_mispriming_sites(
  bamfile,
  fasta,
  pos_5p = 5,
  pos_3p = 20,
  min_reads = 2,
  tag = "pa",
  tag_values = 3:300,
  n_reads_per_chunk = 1e+06,
  verbose = TRUE
)
```

**Arguments**

|                   |  |
|-------------------|--|
| bamfile           | path to bamfile  |
| fasta             | path to fasta file   |
| pos_5p            | distance 5' of mispriming site to define mispriming region                     |
| pos_3p            | distance 3' of mispriming site to define mispriming region                     |
| min_reads         | minimum required number of reads at a mispriming site                          |
| tag               | bam tag containing number of poly(A) bases trimmed                             |
| tag_values        | range of values required for read to be considered                             |
| n_reads_per_chunk | number of reads to process in memory, see <a href="#">Rsamtools::BamFile()</a> |
| verbose           | if true report progress  |

**Value**

A `GenomicsRanges` containing regions enriched for putative mispriming events. The `n_reads` column specifies the number of polyA trimmed reads overlapping the mispriming region. `mean_pal` indicates the mean length of polyA sequence trimmed from reads overlapping the region. The `n_regions` column specifies the number overlapping independent regions found in each chunk (dictated by `n_reads_per_chunk`). The `A_freq` column indicates the frequency of A bases within the region.

**Examples**

```
bam_fn <- raer_example("5k_neuron_mouse_possort.bam")
fa_fn <- raer_example("mouse_tiny.fasta")
find_mispriming_sites(bam_fn, fa_fn)
```



---

|                 |   |
|-----------------|---|
| find_scde_sites | <i>Identify sites with differential editing between cells in single cell datasets</i> |
|-----------------|---|

---

## Description

Compare editing frequencies between clusters or celltypes. REF and ALT counts from each cluster are pooled to create pseudobulk estimates. Each pair of clusters are compared using fisher exact tests. Statistics are aggregated across each pairwise comparison using [scrn::combineMarkers](#).

## Usage

```
find_scde_sites(sce, group, rowData = FALSE, BPPARAM = SerialParam(), ...)
```

## Arguments

|         |  |
|---------|--|
| sce     | <a href="#">SingleCellExperiment</a> object with nRef and nAlt assays.   |
| group   | column name from colData used to define groups to compare.   |
| rowData | if TRUE, <a href="#">rowData</a> from the input <a href="#">SingleCellExperiment</a> will be included in the output DataFrames |
| BPPARAM | BiocParallel backend for control how parallel computations are performed.  |
| ...     | Additional arguments passed to <a href="#">scrn::combineMarkers</a>  |

## Value

A named list of [DataFrames](#) containing results for each cluster specified by group. The difference in editing frequencies between cluster pairs are denoted as dEF. See [scrn::combineMarkers](#) for a description of additional output fields.

## Examples

```
### generate example data ###

library(Rsamtools)
library(GenomicRanges)
bam_fn <- raer_example("5k_neuron_mouse_possort.bam")

gr <- GRanges(c("2:579:-", "2:625:-", "2:645:-", "2:589:-", "2:601:-"))
gr$REF <- c(rep("A", 4), "T")
gr$ALT <- c(rep("G", 4), "C")

cbs <- unique(scanBam(bam_fn, param = ScanBamParam(tag = "CB"))[[1]]$tag$CB)
cbs <- na.omit(cbs)

outdir <- tempdir()
bai <- indexBam(bam_fn)

fp <- FilterParam(library_type = "fr-second-strand")
sce <- pileup_cells(bam_fn, gr, cbs, outdir, param = fp)

# mock some clusters
set.seed(42)
```

```
sce$clusters <- paste0("cluster_", sample(1:3, ncol(sce), replace = TRUE))
res <- find_scde_sites(sce, "clusters")
res[[1]]
```

---

get\_overlapping\_snps     *Retrieve SNPs overlapping intervals*

---

### Description

This function will find SNPs overlapping supplied intervals using a SNPlocs package. The SNPs can be returned in memory (as GPos objects) or written to disk as a bed-file (optionally compressed).

### Usage

```
get_overlapping_snps(gr, snpDb, output_file = NULL)
```

### Arguments

|             |   |
|-------------|---|
| gr          | Intervals to query  |
| snpDb       | A reference of a SNPlocs database   |
| output_file | A path to an output file. If supplied the file can be optionally compressed by including a ".gz" suffix. If not supplied, SNPS will be returned as a <a href="#">GenomicRanges::GPos</a> object |

### Value

GPos object containing SNPs overlapping supplied genomic intervals

### Examples

```
if (require(SNPlocs.Hsapiens.dbSNP144.GRCh38)) {
  gr <- GRanges(rep("22", 10),
    IRanges(seq(10510077, 10610077, by = 1000)[1:10], width = 250),
    strand = "+")
  get_overlapping_snps(gr, SNPlocs.Hsapiens.dbSNP144.GRCh38)
}
```

---

get\_splice\_sites     *Extract regions surrounding splice sites*

---

### Description

Extract intervals at splice sites and their adjacent regions.

### Usage

```
get_splice_sites(txdb, slop = 4)
```

**Arguments**

|      |   |
|------|---|
| txdb | GenomicFeatures::TxDb   |
| slop | The number of bases upstream and downstream of splice site to extract |

**Value**

GenomicRanges::GRanges containing positions of splice sites, with flanking bases.

**Examples**

```
if (require(TxDb.Hsapiens.UCSC.hg38.knownGene)) {
  txdb <- TxDb.Hsapiens.UCSC.hg38.knownGene
  res <- get_splice_sites(txdb)
  res[1:5]
}
```

---

|                |  |
|----------------|--|
| make_de_object | <i>Make summarized experiment object for differential editing analysis</i> |
|----------------|--|

---

**Description**

Generates a [RangedSummarizedExperiment](#) object for use with edgeR or DESeq2 . Will generate a counts assay with a matrix formatted with 2 columns per sample, representing the reference and editing allele counts.

**Usage**

```
make_de_object(
  rse,
  edit_from = "A",
  edit_to = "G",
  min_prop = 0,
  max_prop = 1,
  min_samples = 1
)
```

**Arguments**

|             |   |
|-------------|---|
| rse         | A <a href="#">RangedSummarizedExperiment</a> object   |
| edit_from   | This should correspond to a nucleotide or assay (A, C, G, T, Ref, or Alt) you expect in the reference. Ex. for A to I editing events, this would be A.    |
| edit_to     | This should correspond to a nucleotide or assay (A, C, G, T, Ref, or Alt) you expect in the editing site. Ex. for A to I editing events, this would be G. |
| min_prop    | The minimum required proportion of reads edited at a site. At least min_samples need to pass this to keep the site.                                       |
| max_prop    | The maximum allowable proportion of reads edited at a site. At least min_samples need to pass this to keep the site.                                      |
| min_samples | The minimum number of samples passing the min_prop and max_prop cutoffs to keep a site.   |

**Value**

[RangedSummarizedExperiment](#) for use with edgeR or DESeq2. Contains a counts assay with a matrix formatted with 2 columns per sample (ref and alt counts).

**Examples**

```
library(SummarizedExperiment)
rse_adar_ifn <- mock_rse()
rse <- calc_edit_frequency(rse_adar_ifn)
dse <- make_de_object(rse, min_samples = 1)
assay(dse, "counts")[1:5, ]
dse
```

---

mock\_rse

---

*Generate a small RangedSummarizedExperiment object for tests and examples*


---

**Description**

A RangedSummarizedExperiment containing a subset of data from an RNA-seq experiment to measure the effects of IFN treatment of cell lines with wild-type or ADAR1-KO.

**Usage**

```
mock_rse()
```

**Value**

RangedSummarizedExperiment populated with pileup data

**Source**

<https://www.ncbi.nlm.nih.gov/bioproject/PRJNA386593>

**References**

<https://pubmed.ncbi.nlm.nih.gov/29395325/>

**Examples**

```
mock_rse()
```

---

|              |                                      |
|--------------|--------------------------------------|
| pileup_cells | <i>Generate base counts per cell</i> |
|--------------|--------------------------------------|

---

## Description

This function processes scRNA-seq library to enumerate base counts for Reference (unedited) or Alternate (edited) bases at specified sites in single cells. `pileup_cells` can process droplet scRNA-seq libraries, from a BAM file containing a cell-barcode and UMI, or well-based libraries that do not contain cell-barcodes.

The `sites` parameter specifies sites to quantify. This must be a [GRanges](#) object with 1 base intervals, a strand (+ or -), and supplemented with metadata columns named REF and ALT containing the reference and alternate base to query. See examples for the required format.

At each site, bases from overlapping reads will be examined, and counts of each ref and alt base enumerated for each cell-barcode present. A single base will be counted once for each UMI sequence present in each cell.

## Usage

```
pileup_cells(
  bamfiles,
  sites,
  cell_barcodes,
  output_directory,
  chroms = NULL,
  umi_tag = "UB",
  cb_tag = "CB",
  param = FilterParam(),
  BPPARAM = SerialParam(),
  return_sce = TRUE,
  verbose = FALSE
)
```

## Arguments

|                               |  |
|-------------------------------|--|
| <code>bamfiles</code>         | a path to a BAM file (for droplet scRNA-seq), or a vector of paths to BAM files (Smart-seq2). Can be supplied as a character vector, <a href="#">BamFile</a> , or <a href="#">BamFileList</a> .                            |
| <code>sites</code>            | a <a href="#">GRanges</a> object containing sites to process. See examples for valid formatting.   |
| <code>cell_barcodes</code>    | A character vector of single cell barcodes to process. If processing multiple BAM files (e.g. Smart-seq2), provide a character vector of unique identifiers for each input BAM, to name each BAM file in the output files. |
| <code>output_directory</code> | Output directory for output matrix files. The directory will be generated if it doesn't exist.   |
| <code>chroms</code>           | A character vector of chromosomes to process. If supplied, only sites present in the listed chromosomes will be processed  |
| <code>umi_tag</code>          | tag in BAM containing the UMI sequence   |
| <code>cb_tag</code>           | tag in BAM containing the cell-barcode sequence  |

|            |   |
|------------|---|
| param      | object of class <code>FilterParam()</code> which specify various filters to apply to reads and sites during pileup. Note that the <code>min_depth</code> and <code>min_variant_reads</code> parameters if set > 0 specify the number of reads from any cell required in order to report a site. E.g. if <code>min_variant_reads</code> is set to 2, then at least 2 reads (from any cell) must have a variant in order to report the site. Setting <code>min_depth</code> and <code>min_variant_reads</code> to 0 reports all sites present in the <code>sites</code> object. The following options are not enabled for <code>pileup_cells()</code> : <code>max_mismatch_type</code> , <code>homopolymer_len</code> , and <code>min_allelic_freq</code> . |
| BPPARAM    | <code>BiocParallel</code> instance. Parallel computation occurs across chromosomes.   |
| return_sce | if TRUE, data is returned as a <code>SingleCellExperiment</code> , if FALSE a character vector of the output files, specified by <code>outfile_prefix</code> , will be returned.  |
| verbose    | Display messages  |

### Value

Returns either a `SingleCellExperiment` or character vector of paths to the sparseMatrix files produced. The `SingleCellExperiment` object is populated with two assays, `nRef` and `nAlt`, which represent base counts for the reference and alternate alleles. The `rowRanges()` will contain the genomic interval for each site, along with REF and ALT columns. The rownames will be populated with the format `site_[seqnames]_[position(1-based)]_[strand]_[allele]`, with strand being encoded as 1 = +, 2 = -, and 3 = \*, and allele being REF + ALT.

If `return_sce` is FALSE then a character vector of paths to the sparseMatrix files (`barcodes.txt.gz`, `sites.txt.gz`, `counts.mtx.gz`), will be returned. These files can be imported using `read_spararray()`.

### See Also

Other pileup: `pileup_sites()`

### Examples

```
library(Rsamtools)
library(GenomicRanges)
bam_fn <- raer_example("5k_neuron_mouse_possort.bam")

gr <- GRanges(c("2:579:-", "2:625:-", "2:645:-", "2:589:-", "2:601:-"))
gr$REF <- c(rep("A", 4), "T")
gr$ALT <- c(rep("G", 4), "C")

cbs <- unique(scanBam(bam_fn, param = ScanBamParam(tag = "CB"))[[1]]$tag$CB)
cbs <- na.omit(cbs)

outdir <- tempdir()
bai <- indexBam(bam_fn)

fp <- FilterParam(library_type = "fr-second-strand")
sce <- pileup_cells(bam_fn, gr, cbs, outdir, param = fp)
sce

# example of processing multiple Smart-seq2 style libraries

many_small_bams <- rep(bam_fn, 10)
bam_ids <- LETTERS[1:10]

# for unstranded libraries, sites and alleles should be provided on + strand
```

```

gr <- GRanges(c("2:579:+", "2:625:+", "2:645:+", "2:589:+", "2:601:+"))
gr$REF <- c(rep("T", 4), "A")
gr$ALT <- c(rep("C", 4), "G")

fp <- FilterParam(
  library_type = "unstranded",
  remove_overlaps = TRUE
)

sce <- pileup_cells(many_small_bams,
  sites = gr,
  cell_barcodes = bam_ids,
  cb_tag = NULL,
  umi_tag = NULL,
  outdir,
  param = fp
)
sce

unlink(bai)

```

---

pileup\_sites

*Generate base counts using pileup*


---

## Description

This function uses a pileup routine to examine numerate base counts from alignments at specified sites, regions, or across all read alignments, from one or more BAM files. Alignment and site filtering options are controlled by the `FilterParam` class. A [RangedSummarizedExperiment](#) object is returned, populated with base count statistics for each supplied BAM file.

## Usage

```

pileup_sites(
  bamfiles,
  fasta,
  sites = NULL,
  region = NULL,
  chroms = NULL,
  param = FilterParam(),
  BPPARAM = SerialParam(),
  umi_tag = NULL,
  verbose = FALSE
)

FilterParam(
  max_depth = 10000,
  min_depth = 1L,
  min_base_quality = 20L,
  min_mapq = 0L,
  library_type = "fr-first-strand",
  bam_flags = NULL,

```

```

only_keep_variants = FALSE,
trim_5p = 0L,
trim_3p = 0L,
ftrim_5p = 0,
ftrim_3p = 0,
indel_dist = 0L,
splice_dist = 0L,
min_splice_overhang = 0L,
homopolymer_len = 0L,
max_mismatch_type = c(0L, 0L),
read_bqual = c(0, 0),
min_variant_reads = 0L,
min_allelic_freq = 0,
report_multiallelic = TRUE,
remove_overlaps = TRUE
)

```

## Arguments

|                  |   |
|------------------|---|
| bamfiles         | a character vector, <a href="#">BamFile</a> or <a href="#">BamFileList</a> indicating 1 or more BAM files to process. If named, the names will be included in the <a href="#">colData</a> of the <a href="#">Ranged-SummarizedExperiment</a> as a sample column, otherwise the names will be taken from the basename of the BAM file. |
| fasta            | path to genome fasta file used for read alignment. Can be provided in compressed gzip or bgzip format.  |
| sites            | a <a href="#">GRanges</a> object containing regions or sites to process.  |
| region           | samtools region query string (i.e. chr1:100-1000). Can be combined with sites, in which case sites will be filtered to keep only sites within the region.   |
| chroms           | chromosomes to process, provided as a character vector. Not to be used with the region parameter.   |
| param            | object of class <a href="#">FilterParam()</a> which specify various filters to apply to reads and sites during pileup.  |
| BPPARAM          | A <a href="#">BiocParallel</a> class to control parallel execution. Parallel processing occurs per chromosome and is disabled when run on a single region.  |
| umi_tag          | The BAM tag containing a UMI sequence. If supplied, multiple reads with the same UMI sequence will only be counted once per position.   |
| verbose          | if TRUE, then report progress and warnings.   |
| max_depth        | maximum read depth considered at each site  |
| min_depth        | min read depth needed to report site  |
| min_base_quality | min base quality score to consider read for pileup  |
| min_mapq         | minimum required MAPQ score. Values for each input BAM file can be provided as a vector.  |
| library_type     | read orientation, one of fr-first-strand, fr-second-strand, and unstranded. Unstranded library type will be reported with variants w.r.t the + strand. Values for each input BAM file can be provided as a vector.  |
| bam_flags        | bam flags to filter or keep, use <a href="#">Rsamtools::scanBamFlag()</a> to generate.  |



|                     |   |
|---------------------|---|
| only_keep_variants  | if TRUE, then only variant sites will be reported (FALSE by default). Values for each input BAM file can be provided as a vector.   |
| trim_5p             | Bases to trim from 5' end of read alignments  |
| trim_3p             | Bases to trim from 3' end of read alignments  |
| ftrim_5p            | Fraction of bases to trim from 5' end of read alignments  |
| ftrim_3p            | Fraction of bases to trim from 3' end of read alignments  |
| indel_dist          | Exclude read if site occurs within given distance from indel event in the read  |
| splice_dist         | Exclude read if site occurs within given distance from splicing event in the read   |
| min_splice_overhang | Exclude read if site is located adjacent to splice site with an overhang less than given length.  |
| homopolymer_len     | Exclude site if occurs within homopolymer of given length   |
| max_mismatch_type   | Exclude read if it has X different mismatch types (e.g A-to-G, G-to-C, C-to-G, is 3 mismatch types) or Y # of mismatches, must be supplied as a integer vector of length 2. e.g. c(X, Y).   |
| read_bqual          | Exclude read if more than X percent of the bases have base qualities less than Y. Numeric vector of length 2. e.g. c(0.25, 20)  |
| min_variant_reads   | Required number of reads containing a variant for a site to be reported. Calculated per bam file, such that if 1 bam file has >= min_variant_reads, then the site will be reported.   |
| min_allelic_freq    | minimum allelic frequency required for a variant to be reported in ALT assay.   |
| report_multiallelic | if TRUE, report sites with multiple variants passing filters. If FALSE, site will not be reported.  |
| remove_overlaps     | if TRUE, enable read pair overlap detection, which will count only 1 read in regions where read pairs overlap using the htlib algorithm. In brief for each overlapping base pair the base quality of the base with the lower quality is set to 0, which discards it from being counted. |

## Value

A [RangedSummarizedExperiment](#) object populated with multiple assays:

- ALT: Alternate base(s) found at each position
- nRef: # of reads supporting the reference base
- nAlt: # of reads supporting an alternate base
- nA: # of reads with A
- nT: # of reads with T
- nC: # of reads with C
- nG: # of reads with G

The [rowRanges\(\)](#) contains the genomic interval for each site, along with:

- REF: The reference base
- rpbz: Mann-Whitney U test of Read Position Bias from bcftools, extreme negative or positive values indicate more bias.
- vdb: Variant Distance Bias for filtering splice-site artifacts from bcftools, lower values indicate more bias.
- sor: Strand Odds Ratio Score, strand bias estimated by the Symmetric Odds Ratio test, based on GATK code. Higher values indicate more bias.

The rownames will be populated with the format `site_[seqnames]_[position(1-based)]_[strand]`, with strand being encoded as 1 = +, 2 = -, and 3 = \*.

### See Also

Other pileup: [pileup\\_cells\(\)](#)

### Examples

```
library(SummarizedExperiment)
bamfn <- raer_example("SRR5564269_Aligned.sortedByCoord.out.md.bam")
bam2fn <- raer_example("SRR5564277_Aligned.sortedByCoord.out.md.bam")
fafn <- raer_example("human.fasta")

rse <- pileup_sites(bamfn, fafn)

fp <- FilterParam(only_keep_variants = TRUE, min_depth = 55)
pileup_sites(bamfn, fafn, param = fp)

# using multiple bam files

bams <- rep(c(bamfn, bam2fn), each = 3)
sample_ids <- paste0(rep(c("KO", "WT"), each = 3), 1:3)
names(bams) <- sample_ids

fp <- FilterParam(only_keep_variants = TRUE)
rse <- pileup_sites(bams, fafn, param = fp)
rse

rse$condition <- substr(rse$sample, 1, 2)
assays(rse)

colData(rse)

rowRanges(rse)

# specifying regions to query using GRanges object

sites <- rowRanges(rse)
rse <- pileup_sites(bams, fafn, sites = sites)
rse

rse <- pileup_sites(bams, fafn, chroms = c("SPCS3", "DHFR"))
rse

rse <- pileup_sites(bams, fafn, region = "DHFR:100-101")
rse
```

raer

*raer: RNA editing tools in R***Description**

Toolkit for identification and statistical testing of RNA editing signals from within R. Provides support for identifying sites from bulk-RNA and single cell RNA-seq datasets, and general methods for extraction of allelic read counts from alignment files. Facilitates annotation and exploratory analysis of editing signals using Bioconductor packages and resources.

**Author(s)**

**Maintainer:** Kent Riemondy <kent.riemondy@gmail.com> ([ORCID](#))

Authors:

- Kristen Wells-Wrasman <kristen.wells-wrasman@cuanschutz.edu> ([ORCID](#))

Other contributors:

- Ryan Sheridan <ryan.sheridan@cuanschutz.edu> ([ORCID](#)) [contributor]
- Jay Hesselberth <jay.hesselberth@gmail.com> ([ORCID](#)) [contributor]
- RNA Bioscience Initiative [copyright holder, funder]

**See Also**

Useful links:

- <https://rnabioco.github.io/raer>
- <https://github.com/rnabioco/raer>
- Report bugs at <https://github.com/rnabioco/raer/issues>

raer\_example

*Provide working directory for raer example files.***Description**

Provide working directory for raer example files.

**Usage**

```
raer_example(path)
```

**Arguments**

path                      path to file

**Value**

Character vector with path to an internal package file.

**Examples**

```
raer_example("human.fasta")
```

---

|                |   |
|----------------|---|
| read_spararray | <i>Read sparseMatrix produced by pileup_cells()</i> |
|----------------|---|

---

**Description**

Read in tables produced by `pileup_cells()` which are an extension of the matrixMarket sparse matrix format to store values for more than 1 matrix.

The .mtx.gz files are formatted with columns:

1. row index (0 based)
2. column index (0 based)
3. values for sparseMatrix #1 (nRef)
4. values for sparseMatrix #2 (nAlt)

**Usage**

```
read_spararray(mtx_fn, sites_fn, bc_fn, site_format = c("coordinate", "index"))
```

**Arguments**

|             |  |
|-------------|--|
| mtx_fn      | .mtx.gz file path  |
| sites_fn    | sites.txt.gz file path   |
| bc_fn       | bcs.txt.gz file path   |
| site_format | one of coordinate or index, coordinate will populate a SingleCellExperiment with rowRanges and rownames corresponding to genomic intervals, whereas 'index' will only add row indices to the rownames. |

**Value**

a SingleCellExperiment object populated with nRef and nAlt assays.

**Examples**

```
library(Rsamtools)
library(GenomicRanges)
bam_fn <- raer_example("5k_neuron_mouse_possort.bam")

gr <- GRanges(c("2:579:-", "2:625:-", "2:645:-", "2:589:-", "2:601:-"))
gr$REF <- c(rep("A", 4), "T")
gr$ALT <- c(rep("G", 4), "C")

cbs <- unique(scanBam(bam_fn, param = ScanBamParam(tag = "CB"))[[1]]$tag$CB)
cbs <- na.omit(cbs)

outdir <- tempdir()
bai <- indexBam(bam_fn)
```

```
fp <- FilterParam(library_type = "fr-second-strand")
mtx_fns <- pileup_cells(bam_fn, gr, cbs, outdir, return_sce = FALSE)
sce <- read_spararray(mtx_fns[1], mtx_fns[2], mtx_fns[3])
sce

unlink(bai)
```

# Index

- \* **internal**
  - raer, [27](#)
- \* **pileup**
  - pileup\_cells, [21](#)
  - pileup\_sites, [23](#)
- \* **se-filters**
  - filter\_clustered\_variants, [11](#)
  - filter\_multiallelic, [12](#)
  - filter\_splice\_variants, [13](#)
- annot\_from\_gr, [2](#)
- annot\_snps, [3](#)
- BamFile, [21, 24](#)
- BamFileList, [21, 24](#)
- BiocParallel, [24](#)
- BiocParallelParam, [5](#)
- BSgenome::available.SNPs(), [4](#)
- calc\_AEI, [5](#)
- calc\_confidence, [6](#)
- calc\_edit\_frequency, [7](#)
- calc\_scAEI, [8](#)
- colData, [24](#)
- correct\_strand, [10](#)
- DataFrame, [17](#)
- filter\_clustered\_variants, [11, 13](#)
- filter\_multiallelic, [12, 12, 13](#)
- filter\_splice\_variants, [12, 13, 13](#)
- FilterParam(pileup\_sites), [23](#)
- FilterParam(), [5, 9, 22, 24](#)
- find\_de\_sites, [14](#)
- find\_mispriming\_sites, [15](#)
- find\_scde\_sites, [17](#)
- GenomicRanges::findOverlaps(), [3](#)
- GenomicRanges::GPos, [18](#)
- get\_overlapping\_snps, [18](#)
- get\_overlapping\_snps(), [5](#)
- get\_scAEI\_sites(calc\_scAEI), [8](#)
- get\_scAEI\_sites(), [9](#)
- get\_splice\_sites, [18](#)
- GPos, [5](#)
- GRanges, [3, 5, 21, 24](#)
- make\_de\_object, [19](#)
- make\_de\_object(), [14](#)
- mock\_rse, [20](#)
- pileup\_cells, [21, 26](#)
- pileup\_cells(), [9](#)
- pileup\_sites, [22, 23](#)
- pileup\_sites(), [7, 8](#)
- raer, [27](#)
- raer-package (raer), [27](#)
- raer\_example, [27](#)
- RangedSummarizedExperiment, [7, 8, 14, 19, 20, 23–25](#)
- read\_spararray, [28](#)
- read\_spararray(), [22](#)
- rowData, [17](#)
- rowRanges(), [22, 25](#)
- Rsamtools::BamFile(), [16](#)
- Rsamtools::scanBamFlag(), [24](#)
- S4Vectors::Rle(), [3, 4](#)
- scrان::combineMarkers, [17](#)
- SingleCellExperiment, [17, 22](#)
- SNPlocs, [5](#)
- SummarizedExperiment, [3](#)
- TxDb, [5](#)