

Package ‘BamScale’

June 18, 2026

Type Package

Title Bioconductor-Friendly Multithreaded BAM Processing

Version 0.99.10

Description Multithreaded sequential BAM processing built on top of the ompBAM C++ engine. BamScale provides user-friendly BAM read and scan interfaces designed for compatibility with existing Bioconductor workflows.

License MIT + file LICENSE

biocViews Software, DataImport, Sequencing, Coverage, Alignment, QualityControl

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports BiocGenerics, BiocParallel, Biostrings, GenomicAlignments, GenomicRanges, IRanges, methods, ompBAM, Rcpp, Rsamtools, S4Vectors

LinkingTo Rcpp, ompBAM

Suggests BiocStyle, chipseqDBData, dplyr, ExperimentHub, GenomeInfoDb, ggplot2, knitr, readr, rmarkdown, scales, tibble, testthat (>= 3.0.0), tidy,

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements C++17, OpenMP

NeedsCompilation yes

URL <https://cparsania.github.io/BamScale/>

BugReports <https://github.com/cparsania/BamScale/issues>

git_url <https://git.bioconductor.org/packages/BamScale>

git_branch devel

git_last_commit 026ff90

git_last_commit_date 2026-05-26

Repository Bioconductor 3.24

Date/Publication 2026-06-18

Author Chirag Parsania [aut, cre]

Maintainer Chirag Parsania <chirag.parsania@gmail.com>

Contents

BamScale-package	2
bam_count	2
bam_read	3
decode_compact_qual	6
decode_compact_seq	7
decode_seqqual_compact	7
Index	9

BamScale-package	<i>BamScale: Bioconductor-Friendly Multithreaded BAM Processing</i>
------------------	---

Description

Multithreaded sequential BAM processing built on top of the ompBAM C++ engine. BamScale provides user-friendly BAM read and scan interfaces designed for compatibility with existing Bioconductor workflows.

Author(s)

Maintainer: Chirag Parsania <chirag.parsania@gmail.com>

See Also

Useful links:

- <https://cparsania.github.io/BamScale/>
- Report bugs at <https://github.com/cparsania/BamScale/issues>

bam_count	<i>Count BAM records with Bioconductor-compatible filtering</i>
-----------	---

Description

bam_count() provides a fast chromosome-level count summary, honoring key filtering fields from ScanBamParam (mapqFilter, flag, and which).

Usage

```
bam_count(
  file,
  param = NULL,
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE,
  include_unmapped = TRUE
)
```

Arguments

file	BAM input (character, BamFile, or BamFileList).
param	Optional Rsamtools::ScanBamParam (or compatible list).
threads	Requested number of OpenMP threads. May be capped when auto_threads = TRUE.
BPPARAM	BiocParallel parameter for multi-file operation. Defaults to BiocParallel::bpparam(). Set to NULL to force serial file processing.
auto_threads	Logical; when TRUE and BPPARAM has multiple workers, BamScale adaptively avoids oversubscription by preserving higher per-file OpenMP thread counts when possible and reducing the number of concurrently active file workers before shrinking per-file threads.
include_unmapped	Whether to include an extra * row for unmapped records.

Details

Parallelism behavior matches bam_read(): BPPARAM distributes work across BAM files, while threads controls OpenMP work within each file. If auto_threads = TRUE and BPPARAM has multiple workers, BamScale first limits the number of concurrently active workers to preserve the requested per-file thread count within the detected core budget, then caps per-file OpenMP threads only if a single file would still oversubscribe the machine.

Value

For one file: a data.frame with columns seqname, seqlength, count. For multiple files: named list of such data.frames.

Examples

```
bam <- ompBAM::example_BAM("Unsorted")
bam_count(bam, threads = 2)
```

bam_read

Fast BAM reading with Bioconductor-compatible arguments

Description

bam_read() is a multithreaded sequential BAM reader built on top of ompBAM. The interface is designed to be familiar to users of Rsamtools::scanBam(), GenomicAlignments::readGAlignments(), and GenomicAlignments::readGAlignmentPairs().

Usage

```
bam_read(
  file,
  param = NULL,
  what = NULL,
  tag = NULL,
  as = c("DataFrame", "data.frame", "GAlignments", "GAlignmentPairs", "scanBam"),
  seqqual_mode = c("compatible", "compact"),
```

```

threads = 1L,
BPPARAM = BiocParallel::bpparam(),
auto_threads = FALSE,
use.names = FALSE,
with.which_label = FALSE,
include_unmapped = TRUE
)

```

Arguments

file	A BAM input. Supported values are: <ul style="list-style-type: none"> a single BAM path (<code>character(1)</code>) or multiple BAM paths, a <code>Rsamtools::BamFile</code>, a <code>Rsamtools::BamFileList</code>.
param	Optional <code>Rsamtools::ScanBamParam</code> (or a compatible list for lightweight use). The following fields are honored: <code>mapqFilter</code> , <code>flag</code> , <code>which</code> , <code>what</code> , and <code>tag</code> .
what	Character vector of fields to return, similar to <code>scanBam(what=...)</code> . Supported fields are <code>qname</code> , <code>flag</code> , <code>rname</code> , <code>strand</code> , <code>pos</code> , <code>qwidth</code> , <code>mapq</code> , <code>cigar</code> , <code>mrnm</code> , <code>mpos</code> , <code>isize</code> , <code>seq</code> , <code>qual</code> .
tag	Character vector of 2-letter tag names to extract.
as	Output format: <ul style="list-style-type: none"> "<code>DataFrame</code>": returns <code>S4Vectors::DataFrame</code> (default), "<code>data.frame</code>": returns base <code>data.frame</code>, "<code>GAlignments</code>": returns <code>GenomicAlignments::GAlignments</code>, "<code>GAlignmentPairs</code>": returns <code>GenomicAlignments::GAlignmentPairs</code>, "<code>scanBam</code>": returns a <code>scanBam()</code>-shaped list-of-lists.
seqqual_mode	Controls representation of <code>seq/qual</code> when those fields are requested: <ul style="list-style-type: none"> "<code>compatible</code>" (default): return character vectors matching <code>scanBam</code>-style expectations, "<code>compact</code>": return lower-level raw list-columns for faster/lower-overhead extraction. In compact mode, <code>seq</code> is returned as one raw vector per read containing BAM-native packed sequence bytes (two bases per byte), and <code>qual</code> is returned as one raw vector per read containing per-base Phred bytes. These are not plain character strings. <code>qwidth</code> is needed to decode compact <code>seq</code> back to base letters, and <code>qual</code> values of 255 correspond to missing quality values. This mode is currently supported for <code>as = "data.frame"</code> or <code>as = "DataFrame"</code>.
threads	Requested number of OpenMP threads used for reading/decompression. May be capped when <code>auto_threads = TRUE</code> .
BPPARAM	<code>BiocParallel</code> parameter used when file contains more than one BAM. Defaults to <code>BiocParallel::bpparam()</code> . Set to <code>NULL</code> to force serial file processing.
auto_threads	Logical; when <code>TRUE</code> and <code>BPPARAM</code> has multiple workers, <code>BamScale</code> adaptively avoids oversubscription by preserving higher per-file OpenMP thread counts when possible and reducing the number of concurrently active file workers before shrinking per-file threads.
use.names	Passed to alignment object conversion. When <code>TRUE</code> , read names (<code>qname</code>) are used as object names.

<code>with.which_label</code>	Logical; if TRUE and <code>param</code> includes <code>which</code> , an extra <code>which_label</code> column is returned.
<code>include_unmapped</code>	Logical; whether unmapped records are retained (subject to <code>param\$flag</code> constraints).

Details

`bam_read()` is intentionally column-compatible with common BAM fields used by Bioconductor workflows and can be used as a fast drop-in reader before conversion to downstream classes.

Parallelism model:

- BPPARAM parallelizes across files (one file per `BiocParallel` worker).
- `threads` parallelizes within each file via OpenMP.
- Effective total concurrency is approximately `min(length(file), BiocParallel::bpnworkers(BPPARAM)) * threads`.
- If `auto_threads = TRUE` and BPPARAM has multiple workers, `BamScale` first limits the number of concurrently active workers to preserve the requested per-file thread count within the detected core budget, then caps per-file OpenMP threads only if a single file would still oversubscribe the machine.

Compatibility notes:

- Region filtering via `param$which` is supported as a sequential filter (not index-jump random access).
- Flag filtering uses `ScanBamFlag` semantics by converting logical flag requirements into required-set and required-unset bit masks.
- Tag values are returned as character columns. Scalar tags are scalar strings; B tags are comma-separated vectors.
- `seqqual_mode = "compact"` is optimized for throughput-oriented benchmarking and returns raw list-columns for `seq/qual`, not ordinary sequence or quality strings. In this representation, `seq` contains BAM-packed nucleotide bytes and `qual` contains raw Phred bytes. Compact output is intended for users who want to defer or avoid full string-materialization costs; use `decode_compact_seq()`, `decode_compact_qual()`, or `decode_seqqual_compact()` to decode compact output back to standard string form when needed.
- "GAlignments" and "GAlignmentPairs" output exclude unmapped records.
- `as = "scanBam"` returns a strict scan-like list-of-lists: without `param$which`, it returns one unnamed batch; with `param$which`, it returns one batch per range label (including empty ranges), with requested what fields and tag values under `$tag`. In this output mode, `seq` and `qual` are returned as `Biostrings::DNAStringSet` and `Biostrings::PhredQuality` for closer `scanBam()` compatibility.

Value

If `file` is length 1: one object in the format specified by `as`. If `file` has length > 1 (or is a `BamFileList`): a named list of outputs, one per BAM file.

Examples

```

bam <- ompBAM::example_BAM("Unsorted")

# Familiar scanBam-like field selection
x <- bam_read(bam, what = c("qname", "flag", "rname", "pos", "cigar"))

# Include sequence + quality
y <- bam_read(bam, what = c("qname", "seq", "qual"), threads = 2)

# scanBam-shaped output
z <- bam_read(bam, what = c("qname", "flag"), tag = "NM", as = "scanBam")

```

decode_compact_qual *Decode compact BamScale quality output*

Description

Decodes qual values returned by `bam_read(..., seqqual_mode = "compact")` back to ASCII Phred-quality strings.

Usage

```
decode_compact_qual(qual)
```

Arguments

`qual` A list (or list-column) of raw vectors produced by compact BamScale quality extraction.

Value

A character vector containing decoded quality strings. Entries with all-missing quality bytes are returned as "*", matching BamScale's compatibility mode.

See Also

[decode_compact_seq\(\)](#), [decode_seqqual_compact\(\)](#), [bam_read\(\)](#)

Examples

```

decode_compact_qual(
  qual = list(as.raw(c(0L, 1L, 2L, 3L)))
)

```

decode_compact_seq *Decode compact BamScale sequence output*

Description

Decodes seq values returned by `bam_read(..., seqqual_mode = "compact")` back to ordinary character strings.

Usage

```
decode_compact_seq(seq, qwidth)
```

Arguments

seq	A list (or list-column) of raw vectors produced by compact BamScale sequence extraction.
qwidth	Integer vector of read widths. This is required because compact sequence bytes use BAM's 4-bit packed encoding (two bases per byte).

Value

A character vector containing decoded sequence strings.

See Also

[decode_compact_qual\(\)](#), [decode_seqqual_compact\(\)](#), [bam_read\(\)](#)

Examples

```
decode_compact_seq(  
  seq = list(as.raw(c(0x12, 0x48))),  
  qwidth = 4L  
)
```

decode_seqqual_compact

Decode compact seq and qual columns in BamScale output

Description

Convenience wrapper for converting a compact `bam_read()` result back to ordinary sequence and quality strings.

Usage

```
decode_seqqual_compact(  
  x,  
  seq_col = "seq",  
  qual_col = "qual",  
  qwidth_col = "qwidth"  
)
```

Arguments

<code>x</code>	A <code>data.frame</code> , <code>S4Vectors::DataFrame</code> , or list-like object containing compact BamScale seq and/or qual columns.
<code>seq_col</code>	Name of the compact sequence column.
<code>qual_col</code>	Name of the compact quality column.
<code>qwidth_col</code>	Name of the read-width column used to decode compact sequence bytes.

Value

`x` with compact seq and/or qual columns replaced by decoded character vectors. The input class is preserved.

See Also

[decode_compact_seq\(\)](#), [decode_compact_qual\(\)](#), [bam_read\(\)](#)

Examples

```
x <- data.frame(qwidth = 4L)
x$seq <- I(list(as.raw(c(0x12, 0x48))))
x$qual <- I(list(as.raw(c(0L, 1L, 2L, 3L))))
decode_seqqual_compact(x)
```

Index

* **internal**

BamScale-package, [2](#)

[bam_count](#), [2](#)

[bam_read](#), [3](#)

[bam_read\(\)](#), [6–8](#)

[BamScale](#) (BamScale-package), [2](#)

BamScale-package, [2](#)

[decode_compact_qual](#), [6](#)

[decode_compact_qual\(\)](#), [7, 8](#)

[decode_compact_seq](#), [7](#)

[decode_compact_seq\(\)](#), [6, 8](#)

[decode_seqqual_compact](#), [7](#)

[decode_seqqual_compact\(\)](#), [6, 7](#)