Package ‘Rsamtools’

April 5, 2014

Type Package

Title Binary alignment (BAM), variant call (BCF), or tabix file import

Version 1.14.3

Author Martin Morgan, Herv\{}'e Pag\{}`es, Valerie Obenchain

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Description This package provides an interface to the 'samtools','bcftools', and 'tabix' utilities (see 'LICENCE') for manipulating SAM (Sequence Alignment / Map), binary variant call (BCF) and compressed indexed tab-delimited (tabix) files.

URL http://bioconductor.org/packages/release/bioc/html/Rsamtools.html

License Artistic-2.0 | file LICENSE

LazyLoad yes

Depends
methods, IRanges (>= 1.19.11), GenomicRanges (>= 1.13.35),XVector, Biostrings (>= 2.29.7)

Imports utils, BiocGenerics (>= 0.1.3), zlibbioc, bitops

Suggests ShortRead (>= 1.19.10), GenomicFeatures,TxDb.Dmelanogaster.UCSC.dm3.ensGene, KEGG.db,TxDb.Hsapiens.UCSC.hg18.knownGene, RNAseqData.HNRNPC.bam.chr14,BSgenome.Hsapiens.UCSC.hg19, pasillaBamSubset, RUnit, BiocStyle

LinkingTo IRanges, XVector, Biostrings

biocViews DataImport, Sequencing, HighThroughputSequencing

R topics documented:

`Rsamtools-package` *'samtools' aligned sequence utilities interface*

Description

This package provides facilities for parsing samtools BAM (binary) files representing aligned sequences.

Details

See `packageDescription(Rsamtools)` for package details. A useful starting point is the `scanBam` manual page.

Note

This package documents the following classes for purely internal reasons, see help pages in other packages: `bzfile`, `fifo`, `gzfile`, `pipe`, `unz`, `url`.

Author(s)

Author: Martin Morgan

Maintainer: Biocore Team c/o BioC user list <bioconductor@stat.math.ethz.ch>

References

The current source code for samtools and bcftools is from `https://github.com/samtools/samtools`. Additional material is at `http://samtools.sourceforge.net/`.

Examples

```
packageDescription(Rsamtools)
```

`applyPileups` *Create summary pile-up statistics across multiple BAM files.*

Description

`applyPileups` scans one or more BAM files, returning position-specific sequence and quality summaries.

Usage

```
applyPileups(files, FUN, ..., param)
```

Arguments

`files`	A `PileupFiles` instances.
`FUN`	A function of 1 argument, `x`, to be evaluated for each yield (see `yieldSize`, `yieldBy`, `yieldAll`). The argument `x` is a `list`, with elements describing the current pile-up. The elements of the list are determined by the argument `what`, and include: **seqnames:** (Always returned) A named `integer()` representing the seqnames corresponding to each position reported in the pile-up. This is a run-length encoding, where the names of the elements represent the seqnames, and the values the number of successive positions corresponding to that seqname. **pos:** Always returned) A `integer()` representing the genomic coordinate of each pile-up position.

seq: An `array` of dimensions nucleotide x file x position.
The 'nucleotide' dimension is length 5, corresponding to 'A', 'C', 'G', 'T', and 'N' respectively.
Entries in the array represent the number of times the nucleotide occurred in reads in the file overlapping the position.

qual: Like `seq`, but summarizing quality; the first dimension is the Phred-encoded quality score, ranging from '!' (0) to '~' (93).

`...`	Additional arguments, passed to methods.
`param`	An instance of the object returned by `PileupParam`.

Details

Regardless of `param` values, the algorithm follows samtools by excluding reads flagged as unmapped, secondary, duplicate, or failing quality control.

Value

`applyPileups` returns a `list` equal in length to the number of times `FUN` has been called, with each element containing the result of `FUN`.

`PileupParam` returns an object describing the parameters.

Author(s)

Martin Morgan

References

http://samtools.sourceforge.net/

See Also

`PileupParam`.

Examples

```
fl <- system.file("extdata", "ex1.bam", package="Rsamtools",
                  mustWork=TRUE)

fls <- PileupFiles(c(fl, fl))

calcInfo <-
    function(x)
{
    ## information at each pile-up position
    info <- apply(x[["seq"]], 2, function(y) {
        y <- y[c("A", "C", "G", "T"),,drop=FALSE]
        y <- y + 1L                     # continuity
        cvg <- colSums(y)
        p <- y / cvg[col(y)]
        h <- -colSums(p * log(p))
        ifelse(cvg == 4L, NA, h)
    })
    list(seqnames=x[["seqnames"]], pos=x[["pos"]], info=info)
}
which <- GRanges(c("seq1", "seq2"), IRanges(c(1000, 1000), 2000))
param <- PileupParam(which=which, what="seq")
res <- applyPileups(fls, calcInfo, param=param)
str(res)
head(res[[1]][["pos"]]) # positions matching param
head(res[[1]][["info"]]) # inforamtion in each file

## param as part of files
fls1 <- PileupFiles(c(fl, fl), param=param)
res1 <- applyPileups(fls1, calcInfo)
identical(res, res1)

## yield by position, across ranges
param <- PileupParam(which=which, yieldSize=500L, yieldBy="position",
                     what="seq")
res <- applyPileups(fls, calcInfo, param=param)
sapply(res, "[[", "seqnames")
```

`BamFile` *Maintain and use BAM files*

Description

Use `BamFile()` to create a reference to a BAM file (and optionally its index). The reference remains open across calls to methods, avoiding costly index re-loading.

`BamFileList()` provides a convenient way of managing a list of `BamFile` instances.

Usage

```
## Constructors

BamFile(file, index=file, ..., yieldSize=NA_integer_, obeyQname=FALSE,
         asMates=FALSE)
BamFileList(..., yieldSize=NA_integer_, obeyQname=FALSE, asMates=FALSE)

## Opening / closing

## S3 method for class BamFile
open(con, ...)
## S3 method for class BamFile
close(con, ...)
```

```
## accessors; also path(), index(), yieldSize()

## S4 method for signature BamFile
isOpen(con, rw="")
## S4 method for signature BamFile
isIncomplete(con)
## S4 method for signature BamFile
obeyQname(object, ...)
obeyQname(object, ...) <- value
## S4 method for signature BamFile
asMates(object, ...)
asMates(object, ...) <- value

## actions

## S4 method for signature BamFile
scanBamHeader(files, ...)
## S4 method for signature BamFile
seqinfo(x)
## S4 method for signature BamFile
filterBam(file, destination, index=file, ...,
    filter=FilterRules(), indexDestination=TRUE,
    param=ScanBamParam(what=scanBamWhat()))
## S4 method for signature BamFile
indexBam(files, ...)
## S4 method for signature BamFile
sortBam(file, destination, ..., byQname=FALSE, maxMemory=512)
## S4 method for signature BamFileList
mergeBam(files, destination, ...)

## reading

## S4 method for signature BamFile
scanBam(file, index=file, ..., param=ScanBamParam(what=scanBamWhat()))
## S4 method for signature BamFile
readGAlignmentsFromBam(file, index=file, ..., use.names=FALSE, param=NULL,
                       with.which_label=FALSE)
## S4 method for signature BamFile
readGappedReadsFromBam(file, index=file, use.names=FALSE, param=NULL,
                       with.which_label=FALSE)
## S4 method for signature BamFile
readGAlignmentPairsFromBam(file, index=file, use.names=FALSE, param=NULL,
                           with.which_label=FALSE)
## S4 method for signature BamFile
readGAlignmentsListFromBam(file, index=file, ...,
    use.names=FALSE, param=ScanBamParam(), with.which_label=FALSE)
```

```
## counting

## S4 method for signature BamFile
countBam(file, index=file, ..., param=ScanBamParam())
## S4 method for signature BamFileList
countBam(file, index=file, ..., param=ScanBamParam())
## S4 method for signature BamFile
quickCountBam(file, ..., param=ScanBamParam(), mainGroupsOnly=FALSE)
## S4 method for signature BamFile
coverage(x, shift=0L, width=NULL, weight=1L, ..., param = ScanBamParam())
## S4 method for signature GRanges,BamFile
summarizeOverlaps(features, reads, mode, ignore.strand=FALSE, ...,
    inter.feature=TRUE, singleEnd=TRUE, fragments=FALSE, param=ScanBamParam())
## S4 method for signature BamFile,ANY
findSpliceOverlaps(query, subject, ignore.strand=FALSE, ...,
    param=ScanBamParam(), singleEnd=TRUE)
```

Arguments

`...`	Additional arguments. For `BamFileList`, this can either be a single character vector of paths to BAM files, or several instances of `BamFile` objects. When a character vector of paths, a second named argument 'index' can be a `character()` vector of length equal to the first argument specifying the paths to the index files, or `character()` to indicate that no index file is available. See `BamFile`. For `coverage`, the arguments are passed to the `coverage` method for `GAlignments` objects. For `summarizeOverlaps`, providing `count.mapped.reads=TRUE` include additional passes through the BAM file to collect statistics like those from `countBam`.
`con`	An instance of `BamFile`.
`x, object, file, files`	A character vector of BAM file paths (for `BamFile`) or a `BamFile` instance (for other methods).
`index`	character(1); the BAM index file path (for `BamFile`); ignored for all other methods on this page.
`yieldSize`	Number of records to yield each time the file is read from with `scanBam`. See 'Fields' section for details.
`asMates`	Logical indicating if records should be paired as mates. See 'Fields' section for details.
`obeyQname`	Logical indicating if the BAM file is sorted by `qname`. In Bioconductor > 2.12 paired-end files do not need to be sorted by `qname`. Instead use `asMates=TRUE` for reading paired-end data. See 'Fields' section for details.
`value`	Logical value for setting `asMates` and `obeyQname` in a BamFile instance.
`filter`	A `FilterRules` instance. Functions in the `FilterRules` instance should expect a single `DataFrame` argument representing all information specified by `param`.

Each function must return a `logical` vector, usually of length equal to the number of rows of the `DataFrame`. Return values are used to include (when `TRUE`) corresponding records in the filtered BAM file.

`destination`	character(1) file path to write filtered reads to.
`indexDestination`	logical(1) indicating whether the destination file should also be indexed.
`byQname, maxMemory`	See `sortBam`.
`param`	An optional `ScanBamParam` instance to further influence scanning, counting, or filtering.
`use.names`	Construct the names of the returned object from the query template names (QNAME field)? If not (the default), then the returned object has no names.
`with.which_label`	See `?readGAlignmentsFromBam`.
`rw`	Mode of file; ignored.
`ignore.strand`	A logical value indicating if strand should be considered when matching.
`shift, width, weight`	See `coverage`.
`mainGroupsOnly`	See `quickCountBam`.
`features, reads, mode, inter.feature, fragments, singleEnd`	See `summarizeOverlaps`
`query, subject`	See `findSpliceOverlaps`

Objects from the Class

Objects are created by calls of the form `BamFile()`.

Fields

The `BamFile` class inherits fields from the `RsamtoolsFile` class and has fields:

yieldSize: Number of records to yield each time the file is read from using `scanBam`. Only valid when `length(bamWhich(param)) == 0`. Setting `yieldSize` on a `BamFileList` does not alter existing yield sizes set on the individual `BamFile` instances.

asMates: A logical indicating if the records should be returned as mated pairs. When `TRUE` scanBam attempts to `mate` (pair) the records and returns two additional fields of `groupid` and `mates`. `groupid` is an integer vector of unique group ids; `mates` is a logical that is TRUE for records successfully paired by the algorithm.

Mate criteria:

- Bit 0x1 (multiple segments) is 1.
- Bit 0x4 (segment unmapped) is 0.
- Bit 0x8 (next segment unmapped) is 0.
- Bit 0x40 and 0x80 (first/last segment): Segments are a pair of first/last
- Bit 0x100 (secondary alignment): Both segments are secondary OR both not secondary

- Bit 0x2 (properly aligned): Both segments are properly aligned
- `qname` match.
- `tid` match.
- segment1 `mpos` matches segment2 `pos` AND segment2 `mpos` matches segment1 `pos`

Records not passing these criteria are returned with mate status FALSE. Flags, tags and ranges may be specified in the `ScanBamParam` for fine tuning of results.

obeyQname: A logical(0) indicating if the file was sorted by qname. In Bioconductor > 2.12 paired-end files do not need to be sorted by `qname`. Instead set `asMates=TRUE` in the `BamFile` when using `readGAlignmentsListFromBam`.

When counting paired-end data with `summarizeOverlaps`, setting `singleEnd=FALSE` will trigger paired-end reading and counting. It is fine to also set `asMates=TRUE` in the `BamFile` but is not necessary if `singleEnd=FALSE`.

Functions and methods

`BamFileList` inherits methods from `RsamtoolsFileList` and `SimpleList`.

Opening / closing:

open.BamFile Opens the (local or remote) `path` and `index` (if `bamIndex` is not `character(0)`), files. Returns a `BamFile` instance.

close.BamFile Closes the `BamFile` `con`; returning (invisibly) the updated `BamFile`. The instance may be re-opened with `open.BamFile`.

isOpen Tests whether the `BamFile` `con` has been opened for reading.

isIncomplete Tests whether the `BamFile` `con` is niether closed nor at the end of the file.

Accessors:

path Returns a character(1) vector of BAM path names.

index Returns a character(1) vector of BAM index path names.

yieldSize, yieldSize<- Return or set an integer(1) vector indicating yield size.

obeyQname, obeyQname<- Return or set a logical(0) indicating if the file was sorted by qname.

asMates, asMates<- Return or set a logical(0) indicating if the records should be returned as mated pairs.

Methods:

scanBamHeader Visit the path in `path(file)`, returning the information contained in the file header; see `scanBamHeader`.

seqinfo Visit the path in `path(file)`, returning a `Seqinfo` instance containing information on the lengths of each sequence.

scanBam Visit the path in `path(file)`, returning the result of `scanBam` applied to the specified path.

countBam Visit the path(s) in `path(file)`, returning the result of `countBam` applied to the specified path.

filterBam Visit the path in `path(file)`, returning the result of `filterBam` applied to the specified path.

indexBam Visit the path in `path(file)`, returning the result of `indexBam` applied to the specified path.

sortBam Visit the path in `path(file)`, returning the result of `sortBam` applied to the specified path.

mergeBam Merge several BAM files into a single BAM file. See `mergeBam` for details; additional arguments supported by `mergeBam,character-method` are also available for `BamFileList`.

readGAlignmentsFromBam, readGappedReadsFromBam, readGAlignmentPairsFromBam Visit the path in `path(file)`, returning the result of `readGAlignmentsFromBam`, `readGappedReadsFromBam`, or `readGAlignmentPairsFromBam` applied to the specified path. See `readGAlignmentsFromBam`.

readGAlignmentsListFromBam Visit the Bam file in `path(file)`, returning the result of `readGAlignmentsListFromBam` applied to the specified path. See `readGAlignmentsListFromBam`.

show Compactly display the object.

Author(s)

Martin Morgan and Marc Carlson

See Also

- `readGAlignmentsFromBam`
- `readGAlignmentPairsFromBam`
- `readGAlignmentsListFromBam`
- `summarizeOverlaps`
- `findSpliceOverlaps`

Examples

```
##
## BamFile options.
##

fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
bf <- BamFile(fl)
bf

## When asMates=TRUE scanBam() reads the data in as
## pairs. See asMates above for details of the pairing
## algorithm.
asMates(bf) <- TRUE

## When yieldSize is set, scanBam() will iterate
## through the file in chunks.
yieldSize(bf) <- 500


##
## Reading Bam files.
```

```
##

fl <- system.file("extdata", "ex1.bam", package="Rsamtools",
                  mustWork=TRUE)
length(scanBam(fl)[[1]][[1]])  # all records

bf <- open(BamFile(fl))        # implicit index
bf
identical(scanBam(bf), scanBam(fl))
close(bf)

## Use yieldSize to iterate through a file in chunks.
bf <- open(BamFile(fl, yieldSize=1000))
while (nrec <- length(scanBam(bf)[[1]][[1]]))
    cat("records:", nrec, "\n")
close(bf)

## Repeatedly visit multiple ranges in the BamFile.
rng <- GRanges(c("seq1", "seq2"), IRanges(1, c(1575, 1584)))
bf <- open(BamFile(fl))
sapply(seq_len(length(rng)), function(i, bamFile, rng) {
    param <- ScanBamParam(which=rng[i], what="seq")
    bam <- scanBam(bamFile, param=param)[[1]]
    alphabetFrequency(bam[["seq"]], baseOnly=TRUE, collapse=TRUE)
}, bf, rng)
close(bf)

## See ?summarizeOverlaps and ?findSpliceOverlaps in the
## GenomicRanges package for examples with these functions.
```

BamInput	*Import, count, index, filter, sort, and merge 'BAM' (binary alignment) files.*

Description

Import binary 'BAM' files into a list structure, with facilities for selecting what fields and which records are imported, and other operations to manipulate BAM files.

Usage

```
scanBam(file, index=file, ..., param=ScanBamParam(what=scanBamWhat()))

countBam(file, index=file, ..., param=ScanBamParam())

scanBamHeader(files, ...)
## S4 method for signature character
scanBamHeader(files, ...)
```

```
asBam(file, destination, ...)
## S4 method for signature character
asBam(file, destination, ...,
    overwrite=FALSE, indexDestination=TRUE)

filterBam(file, destination, index=file, ...)
## S4 method for signature character
filterBam(file, destination, index=file, ...,
    filter=FilterRules(), indexDestination=TRUE,
    param=ScanBamParam(what=scanBamWhat()))

sortBam(file, destination, ...)
## S4 method for signature character
sortBam(file, destination, ..., byQname=FALSE, maxMemory=512)

indexBam(files, ...)
## S4 method for signature character
indexBam(files, ...)

mergeBam(files, destination, ...)
## S4 method for signature character
mergeBam(files, destination, ..., region = RangedData(),
    overwrite = FALSE, header = character(), byQname = FALSE,
    addRG = FALSE, compressLevel1 = FALSE, indexDestination = FALSE)
```

Arguments

`file`	The character(1) file name of the ‘BAM’ (’SAM’ for `asBam`) file to be processed.
`files`	The character() file names of the ‘BAM’ file to be processed. For `mergeBam`, must satisfy `length(files) >=  2`.
`index`	The character(1) name of the index file of the ’BAM’ file being processed; this is given *without* the ’.bai’ extension.
`destination`	The character(1) file name of the location where the sorted, filtered, or merged output file will be created. For `asBam` and `sortBam` this is without the “.bam” file suffix.
`region`	A RangedData() instance with `>= 1` rows, specifying the region of the BAM files to merged.
`...`	Additional arguments, passed to methods.
`overwrite`	A logical(1) indicating whether the destination can be over-written if it already exists.
`filter`	A `FilterRules` instance allowing users to filter BAM files based on arbitrary criteria, as described below.
`indexDestination`	A logical(1) indicating whether the created destination file should also be indexed.

`byQname`	A logical(1) indicating whether the sorted destination file should be sorted by Query-name (TRUE) or by mapping position (FALSE).
`header`	A character(1) file path for the header information to be used in the merged BAM file.
`addRG`	A logical(1) indicating whether the file name should be used as RG (read group) tag in the merged BAM file.
`compressLevel1`	A logical(1) indicating whether the merged BAM file should be compressed to zip level 1.
`maxMemory`	A numerical(1) indicating the maximal amount of memory (in MB) that the function is allowed to use.
`param`	An instance of `ScanBamParam`. This influences what fields and which records are imported.

Details

The `scanBam` function parses binary BAM files; text SAM files can be parsed using R's `scan` function, especially with arguments `what` to control the fields that are parsed.

`countBam` returns a count of records consistent with `param`.

`scanBamHeader` visits the header information in a BAM file, returning for each file a list containing elements `targets` and `text`, as described below. The SAM / BAM specification does not require that the content of the header be consistent with the content of the file, e.g., more targets may be present that are represented by reads in the file.

`asBam` converts 'SAM' files to 'BAM' files, equivalent to the `samtools view -Sb file > destination`. The 'BAM' file is sorted and an index created on the destination (with extension '.bai') when `indexDestination=TRUE`.

`filterBam` parses records in `file`. Records satisfying the `bamWhich` `bamFlag` and `bamSimpleCigar` criteria of `param` are accumulated to a default of `yieldSize = 1000000` records (change this by specifying `yieldSize` when creating a `BamFile` instance; see `BamFile-class`). These records are then parsed to a `DataFrame` and made available for further filtering by user-supplied `FilterRules`. Functions in the `FilterRules` instance should expect a single `DataFrame` argument representing all information specified by `param`. Each function must return a `logical` vector equal to the number of rows of the `DataFrame`. Return values are used to include (when `TRUE`) corresponding records in the filtered BAM file. The BAM file is created at `destination`. An index file is created on the destination when `indexDestination=TRUE`. It is more space- and time-efficient to filter use `bamWHich`, `bamFlag`, and `bamSimpleCigar`, if appropriate, than to supply `FilterRules`.

`sortBam` sorts the BAM file given as its first argument, analogous to the "samtools sort" function.

`indexBam` creates an index for each BAM file specified, analogous to the 'samtools index' function.

`mergeBam` merges 2 or more sorted BAM files. As with samtools, the RG (read group) dictionary in the header of the BAM files is not reconstructed.

Details of the `ScanBamParam` class are provide on its help page; several salient points are reiterated here. `ScanBamParam` can contain a field `what`, specifying the components of the BAM records to be returned. Valid values of `what` are available with `scanBamWhat`. `ScanBamParam` can contain an argument `which` that specifies a subset of reads to return. This requires that the BAM file be indexed, and that the file be named following samtools convention as `<bam_filename>.bai`. `ScanBamParam` can contain an argument `tag` to specify which tags will be extracted.

Value

The `scanBam,character-method` returns a list of lists. The outer list groups results from each `Ranges` list of `bamWhich(param)`; the outer list is of length one when `bamWhich(param)` has length 0. Each inner list contains elements named after `scanBamWhat()`; elements omitted from `bamWhat(param)` are removed. The content of non-null elements are as follows, taken from the description in the samtools API documentation:

- qname: This is the QNAME field in SAM Spec v1.4. The query name, i.e., identifier, associated with the read.
- flag: This is the FLAG field in SAM Spec v1.4. A numeric value summarizing details of the read. See `ScanBamParam` and the `flag` argument, and `scanBamFlag()`.
- rname: This is the RNAME field in SAM Spec v1.4. The name of the reference to which the read is aligned.
- strand: The strand to which the read is aligned.
- pos: This is the POS field in SAM Spec v1.4. The genomic coordinate at the start of the alignment. Coordinates are 'left-most', i.e., at the 3' end of a read on the '-' strand, and 1-based. The position *excludes* clipped nucleotides, even though soft-clipped nucleotides are included in `seq`.
- qwidth: The width of the query, as calculated from the `cigar` encoding; normally equal to the width of the query returned in `seq`.
- mapq: This is the MAPQ field in SAM Spec v1.4. The MAPping Quality.
- cigar: This is the CIGAR field in SAM Spec v1.4. The CIGAR string.
- mrnm: This is the RNEXT field in SAM Spec v1.4. The reference to which the mate (of a paired end or mate pair read) aligns.
- mpos: This is the PNEXT field in SAM Spec v1.4. The position to which the mate aligns.
- isize: This is the TLEN field in SAM Spec v1.4. Inferred insert size for paired end alignments.
- seq: This is the SEQ field in SAM Spec v1.4. The query sequence, in the 5' to 3' orientation. If aligned to the minus strand, it is the reverse complement of the original sequence.
- qual: This is the QUAL field in SAM Spec v1.4. Phred-encoded, phred-scaled base quality score, oriented as `seq`.
- groupid: This is an integer vector of unique group ids returned when `asMates=TRUE` in a BamFile object. `groupid` values are used to create the partitioning for a `GAlignmentsList` object.
- mates: Returned (always) when `asMates=TRUE` in a BamFile object. This is a logical vector indicating mate status of each record.

`scanBamHeader` returns a list, with one element for each file named in `files`. The list contains two element. The `targets` element contains target (reference) sequence lengths. The `text` element is itself a list with each element a list corresponding to tags (e.g., '@SQ') found in the header, and the associated tag values.

`asBam` returns the file name of the BAM file.

`sortBam` returns the file name of the sorted file.

`indexBam` returns the file name of the index file created.

`filterBam` returns the file name of the destination file created.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>. Thomas Unterhiner <thomas.unterthiner@students.jku.at> (`sortBam`).

References

http://samtools.sourceforge.net/

See Also

`ScanBamParam`, `scanBamWhat`, `scanBamFlag`

Examples

```
fl <- system.file("extdata", "ex1.bam", package="Rsamtools",
                  mustWork=TRUE)

##
## scanBam
##

res0 <- scanBam(fl)[[1]] # always list-of-lists
names(res0)
length(res0[["qname"]])
lapply(res0, head, 3)
table(width(res0[["seq"]])) # query widths
table(res0[["qwidth"]], useNA="always") # query widths derived from cigar
table(res0[["cigar"]], useNA="always")
table(res0[["strand"]], useNA="always")
table(res0[["flag"]], useNA="always")

which <- RangesList(seq1=IRanges(1000, 2000),
                    seq2=IRanges(c(100, 1000), c(1000, 2000)))
p1 <- ScanBamParam(which=which, what=scanBamWhat())
res1 <- scanBam(fl, param=p1)
names(res1)
names(res1[[2]])

p2 <- ScanBamParam(what=c("rname", "strand", "pos", "qwidth"))
res2 <- scanBam(fl, param=p2)

p3 <- ScanBamParam(flag=scanBamFlag(isMinusStrand=FALSE))
length(scanBam(fl, param=p3)[[1]])

##
## filterBam
##

param <- ScanBamParam(
   flag=scanBamFlag(isUnmappedQuery=FALSE),
   what="seq")
```

```
dest <- filterBam(fl, tempfile(), param=param)
countBam(dest)  ## 3271 records
filt <- list(MinWidth = function(x) width(x$seq) > 35)
dest <- filterBam(fl, tempfile(), param=param, filter=FilterRules(filt))
countBam(dest)  ## 398 records
res3 <- scanBam(dest, param=ScanBamParam(what="seq"))[[1]]
table(width(res3$seq))

##
## sortBam
##

sorted <- sortBam(fl, tempfile())

## map mcols(gwhich) to output, e.g., of countBam
gwhich <- as(which, "GRanges")[c(2, 1, 3)]
mcols(gwhich)[["OriginalOrder"]] <- 1:3
cnt <- countBam(fl, param=ScanBamParam(which=gwhich))
cntVals <- unlist(split(mcols(gwhich), seqnames(gwhich)))
cbind(cnt, as.data.frame(cntVals))
```

BamSampler   *Sample from a BAM files*

Description

Use `BamSampler()` to create a reference to a BAM file (and optionally its index). Calls to `scanBam` (and many functions that use `scanBam`) draw a random sample from the BAM file.

Usage

```
## Constructors

BamSampler(file, index = file, ..., yieldSize, obeyQname = FALSE, asMates = FALSE)

## S4 method for signature BamSampler
scanBam(file, index=file, ..., param=ScanBamParam(what=scanBamWhat()))
```

Arguments

`file`	character(1); BAM file path for `BamSampler`, or `BamSampler` index for `scanBam` and other functions.
`index`	character(1); the BAM index file path (for `BamFile`); ignored for other methods.
`...`	Additional arguments; see `BamFile`-class.
`yieldSize`	integer(1); number of records to yield each time the file is read from using `scanBam`.

`obeyQname`	logical(1); indicating whether the file is sorted by `qname` and if so, that qnames are not split between yields.
`asMates`	logical(1); indicating whether the records should be returned as mated pairs.
`param`	An optional `ScanBamParam` instance to further influence scanning, counting, or filtering.

Objects from the Class

Objects are created by calls of the form `BamSampler()`.

Fields

The `BamSampler` class inherits fields from the `BamFile` class.

Functions and methods

`BamSampler` inherits methods from `BamFile` and can be used in place of `BamFile` in many functions.

Author(s)

Martin Morgan

Examples

```
fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
samp <- BamSampler(fl, yieldSize=1000)
## two independent samples
head(readGAlignmentsFromBam(samp))
head(readGAlignmentsFromBam(samp))
```

`BamViews` *Views into a set of BAM files*

Description

Use `BamViews()` to reference a set of disk-based BAM files to be processed (e.g., queried using `scanBam`) as a single ‘experiment’.

Usage

```
## Constructor
BamViews(bamPaths=character(0),
     bamIndicies=bamPaths,
     bamSamples=DataFrame(row.names=make.unique(basename(bamPaths))),
     bamRanges, bamExperiment = list(), ...)
```

```
## S4 method for signature missing
BamViews(bamPaths=character(0),
     bamIndicies=bamPaths,
     bamSamples=DataFrame(row.names=make.unique(basename(bamPaths))),
     bamRanges, bamExperiment = list(), ..., auto.range=FALSE)
## Accessors
bamPaths(x)
bamSamples(x)
bamSamples(x) <- value
bamRanges(x)
bamRanges(x) <- value
bamExperiment(x)

## S4 method for signature BamViews
names(x)
## S4 replacement method for signature BamViews
names(x) <- value
## S4 method for signature BamViews
dimnames(x)
## S4 replacement method for signature BamViews,ANY
dimnames(x) <- value

bamDirname(x, ...) <- value

## Subset
## S4 method for signature BamViews,ANY,ANY
x[i, j, ..., drop=TRUE]
## S4 method for signature BamViews,ANY,missing
x[i, j, ..., drop=TRUE]
## S4 method for signature BamViews,missing,ANY
x[i, j, ..., drop=TRUE]

## Input
## S4 method for signature BamViews
scanBam(file, index = file, ..., param = ScanBamParam(what=scanBamWhat()))
## S4 method for signature BamViews
countBam(file, index = file, ..., param = ScanBamParam())
## S4 method for signature BamViews
readGAlignmentsFromBam(file, index=file, ..., use.names=FALSE, param=NULL,
                       with.which_label=FALSE)

## Show
## S4 method for signature BamViews
show(object)

## Counting
## S4 method for signature BamViews,missing
summarizeOverlaps(
    features, reads, mode, ignore.strand=FALSE, ..., inter.feature=TRUE,
    singleEnd=TRUE, fragments=FALSE, param=ScanBamParam())
```

Arguments

`bamPaths`	A character() vector of BAM path names.
`bamIndicies`	A character() vector of BAM index file path names, *without* the '.bai' extension.
`bamSamples`	A `DataFrame` instance with as many rows as `length(bamPaths)`, containing sample information associated with each path.
`bamRanges`	A `GRanges`, `RangedData` or missing instance with ranges defined on the spaces of the BAM files. Ranges are *not* validated against the BAM files.
`bamExperiment`	A list() containing additional information about the experiment.
`auto.range`	If `TRUE` and all `bamPaths` exist, populate the ranges with the union of ranges returned in the `target` element of `scanBamHeader`.
`...`	Additional arguments.
`x`	An instance of `BamViews`.
`object`	An instance of `BamViews`.
`value`	An object of appropriate type to replace content.
`i`	During subsetting, a logical or numeric index into `bamRanges`.
`j`	During subsetting, a logical or numeric index into `bamSamples` and `bamPaths`.
`drop`	A logical(1), *ignored* by all `BamViews` subsetting methods.
`file`	An instance of `BamViews`.
`index`	A character vector of indices, corresponding to the `bamPaths(file)`.
`param`	An optional `ScanBamParam` instance to further influence scanning or counting.
`use.names`	Construct the names of the returned object from the query template names (QNAME field)? If not (the default), then the returned object has no names.
`with.which_label`	See `?readGAlignmentsFromBam`.
`reads`	Missing when a `BamViews` is the only argument supplied to `summarizeOverlaps`. `reads` are the files specified in `bamPaths` of the `BamViews` object.
`features`	A `BamFileList`. `features` are extracted from the `bamRanges` of the `BamViews` object. Metadata from `bamPaths` and `bamSamples` are stored in the `colData` slot of the `SummarizedExperiment` object. `bamExperiment` metadata are in the `exptData` slot.
`mode`	A function that defines the method to be used when a read overlaps more than one feature. Pre-defined options are "Union", "IntersectionStrict", or "IntersectionNotEmpty" and are designed after the counting modes available in the HTSeq package by Simon Anders (see references). • "Union" : (Default) Reads that overlap any portion of exactly one feature are counted. Reads that overlap multiple features are discarded.

- "IntersectionStrict" : A read must fall completely "within" the feature to be counted. If a read overlaps multiple features but falls "within" only one, the read is counted for that feature. If the read is "within" multiple features, the read is discarded.
- "IntersectionNotEmpty" : A read must fall in a unique disjoint region of a feature to be counted. When a read overlaps multiple features, the features are partitioned into disjoint intervals. Regions that are shared between the features are discarded leaving only the unique disjoint regions. If the read overlaps one of these remaining regions, it is assigned to the feature the unique disjoint region came from.

`ignore.strand` A logical value indicating if strand should be considered when matching.

`singleEnd` A logical value indicating if the bam files contain single or paired-end reads.

`inter.feature` A logical indicating if the counting `mode` should be aware of overlapping features. When TRUE (default), reads mapping to multiple features are dropped (i.e., not counted). When FALSE, these reads are retained and a count is assigned to each feature they map to.

There are 6 possible combinations of the `mode` and `inter.feature` arguments. When `inter.feature=FALSE` the behavior of modes 'Union' and 'IntersectionNotEmpty' are the same resulting in 5 distinct ways to count.

`fragments` A logical value indicating if singletons, reads with unmapped pairs and other fragments should be included in the counting. When `fragments=FALSE` only reads paired with the algorithm described at ?findMateAlignment are counted. When `fragments=TRUE` (default) all singletons, reads with unmapped pairs and other fragments are counted in addition to the reads paired with the ?findMateAlignment algorithm. This argument applies to paired-end reads only so `singleEnd` must be TRUE.

Objects from the Class

Objects are created by calls of the form `BamViews()`.

Slots

bamPaths A character() vector of BAM path names.

bamIndicies A character() vector of BAM index path names.

bamSamples A `DataFrame` instance with as many rows as `length(bamPaths)`, containing sample information associated with each path.

bamRanges A `GRanges` instance with ranges defined on the spaces of the BAM files. Ranges are *not* validated against the BAM files.

bamExperiment A list() containing additional information about the experiment.

Functions and methods

See 'Usage' for details on invocation.

Constructor:

BamViews: Returns a `BamViews` object.

Accessors:

bamPaths Returns a character() vector of BAM path names.

bamIndicies Returns a character() vector of BAM index path names.

bamSamples Returns a `DataFrame` instance with as many rows as `length(bamPaths)`, containing sample information associated with each path.

bamSamples<- Assign a `DataFrame` instance with as many rows as `length(bamPaths)`, containing sample information associated with each path.

bamRanges Returns a `GRanges` instance with ranges defined on the spaces of the BAM files. Ranges are *not* validated against the BAM files.

bamRanges<- Assign a `GRanges` instance with ranges defined on the spaces of the BAM files. Ranges are *not* validated against the BAM files.

bamExperiment Returns a list() containing additional information about the experiment.

names Return the column names of the `BamViews` instance; same as `names(bamSamples(x))`.

names<- Assign the column names of the `BamViews` instance.

dimnames Return the row and column names of the `BamViews` instance.

dimnames<- Assign the row and column names of the `BamViews` instance.

Methods:

"[" Subset the object by `bamRanges` or `bamSamples`.

scanBam Visit each path in `bamPaths(file)`, returning the result of `scanBam` applied to the specified path. `bamRanges(file)` takes precedence over `bamWhich(param)`.

countBam Visit each path in `bamPaths(file)`, returning the result of `countBam` applied to the specified path. `bamRanges(file)` takes precedence over `bamWhich(param)`.

readGAlignmentsFromBam Visit each path in `bamPaths(file)`, returning the result of `readGAlignmentsFromBam` applied to the specified path. When `index` is missing, it is set equal to `bamIndicies(file)`. Only reads in `bamRanges(file)` are returned (if `param` is supplied, `bamRanges(file)` takes precedence over `bamWhich(param)`). The return value is a `SimpleList`, with elements of the list corresponding to each path. `bamSamples(file)` is available as metadata columns (accessed with `mcols`) of the returned `SimpleList`.

show Compactly display the object.

Author(s)

Martin Morgan

See Also

`readGAlignmentsFromBam`. The `GenomicRanges` package is where the `summarizeOverlaps` method originates.

Examples

```
fls <- system.file("extdata", "ex1.bam", package="Rsamtools",
                   mustWork=TRUE)
rngs <- GRanges(seqnames = Rle(c("chr1", "chr2"), c(9, 9)),
                ranges = c(IRanges(seq(10000, 90000, 10000), width=500),
                           IRanges(seq(100000, 900000, 100000), width=5000)),
                Count = seq_len(18L))
v <- BamViews(fls, bamRanges=rngs)
v
v[1:5,]
bamRanges(v[c(1:5, 11:15),])
bamDirname(v) <- getwd()
v

bv <- BamViews(fls,
               bamSamples=DataFrame(info="test", row.names="ex1"),
               auto.range=TRUE)
aln <- readGAlignmentsFromBam(bv)
aln
aln[[1]]
aln[colnames(bv)]
mcols(aln)

##---------------------------------------------------------------------------
## summarizeOverlaps() with BamViews
##

## bamSamples and bamPaths metadata are included in the colData.
## bamExperiment metadata is put into the exptData slot.
fl <- system.file("extdata", "ex1.bam", package="Rsamtools", mustWork=TRUE)
rngs <- GRanges(c("seq1", "seq2"), IRanges(1, c(1575, 1584)))
samp <- DataFrame(info="test", row.names="ex1")
view <- BamViews(fl, bamSamples=samp, bamRanges=rngs)
se <- summarizeOverlaps(view, mode=Union, ignore.strand=TRUE)
colData(se)
exptData(se)
```

BcfFile *Manipulate BCF files.*

Description

Use `BcfFile()` to create a reference to a BCF (and optionally its index). The reference remains open across calls to methods, avoiding costly index re-loading.

`BcfFileList()` provides a convenient way of managing a list of `BcfFile` instances.

Usage

```
## Constructors

BcfFile(file, index = file,
        mode=ifelse(grepl("\\.bcf$", file), "rb", "r"))
BcfFileList(...)

## Opening / closing

## S3 method for class BcfFile
open(con, ...)
## S3 method for class BcfFile
close(con, ...)

## accessors; also path(), index()

## S4 method for signature BcfFile
isOpen(con, rw="")
bcfMode(object)

## actions

## S4 method for signature BcfFile
scanBcfHeader(file, ...)
## S4 method for signature BcfFile
scanBcf(file, ..., param=ScanBcfParam())
## S4 method for signature BcfFile
indexBcf(file, ...)
```

Arguments

`con, object`	An instance of `BcfFile`.
`file`	A character(1) vector of the BCF file path or, (for indexBcf) an instance of `BcfFile` point to a BCF file.
`index`	A character(1) vector of the BCF index.
`mode`	A character(1) vector; `mode="rb"` indicates a binary (BCF) file, `mode="r"` a text (VCF) file.
`param`	An optional `ScanBcfParam` instance to further influence scanning.
`...`	Additional arguments. For `BcfFileList`, this can either be a single character vector of paths to BCF files, or several instances of `BcfFile` objects.
`rw`	Mode of file; ignored.

Objects from the Class

Objects are created by calls of the form `BcfFile()`.

Fields

The BcfFile class inherits fields from the RsamtoolsFile class.

Functions and methods

BcfFileList inherits methods from RsamtoolsFileList and SimpleList.

Opening / closing:

open.BcfFile Opens the (local or remote) path and index (if bamIndex is not character(0)), files. Returns a BcfFile instance.

close.BcfFile Closes the BcfFile con; returning (invisibly) the updated BcfFile. The instance may be re-opened with open.BcfFile.

Accessors:

path Returns a character(1) vector of the BCF path name.

index Returns a character(1) vector of BCF index name.

bcfMode Returns a character(1) vector BCF mode.

Methods:

scanBcf Visit the path in path(file), returning the result of scanBcf applied to the specified path.

show Compactly display the object.

Author(s)

Martin Morgan

Examples

```
fl <- system.file("extdata", "ex1.bcf", package="Rsamtools",
                  mustWork=TRUE)
bf <- BcfFile(fl)        # implicit index
bf
identical(scanBcf(bf), scanBcf(fl))

rng <- GRanges(c("seq1", "seq2"), IRanges(1, c(1575, 1584)))
param <- ScanBcfParam(which=rng)
bcf <- scanBcf(bf, param=param)  ## all ranges

## ranges one at a time bf
open(bf)
sapply(seq_len(length(rng)), function(i, bcfFile, rng) {
    param <- ScanBcfParam(which=rng)
    bcf <- scanBcf(bcfFile, param=param)[[1]]
    ## do extensive work with bcf
    isOpen(bf)  ## file remains open
}, bf, rng)
```

BcfInput *Operations on 'BCF' files.*

Description

Import, coerce, or index variant call files in text or binary format.

Usage

```
scanBcfHeader(file, ...)
## S4 method for signature character
scanBcfHeader(file, ...)

scanBcf(file, ...)
## S4 method for signature character
scanBcf(file, index = file, ..., param=ScanBcfParam())

asBcf(file, dictionary, destination, ...,
      overwrite=FALSE, indexDestination=TRUE)
## S4 method for signature character
asBcf(file, dictionary, destination, ...,
      overwrite=FALSE, indexDestination=TRUE)

indexBcf(file, ...)
## S4 method for signature character
indexBcf(file, ...)
```

Arguments

`file`	For `scanBcf` and `scanBcfHeader`, the character() file name of the 'BCF' file to be processed, or an instance of class `BcfFile`.
`index`	The character() file name(s) of the 'BCF' index to be processed.
`dictionary`	a character vector of the unique "CHROM" names in the VCF file.
`destination`	The character(1) file name of the location where the BCF output file will be created. For `asBcf` this is without the ".bcf" file suffix.
`param`	A instance of `ScanBcfParam` influencing which records are parsed and the 'INFO' and 'GENO' information returned.
`...`	Additional arguments, e.g., for `scanBcfHeader,character-method`, `mode` of `BcfFile`.
`overwrite`	A logical(1) indicating whether the destination can be over-written if it already exists.
`indexDestination`	A logical(1) indicating whether the created destination file should also be indexed.

Details

bcf* functions are restricted to the GENO fields supported by 'bcftools' (see documentation at the url below). The argument param allows portions of the file to be input, but requires that the file be BCF or bgzip'd and indexed as a TabixFile. For similar functions operating on VCF files see ?scanVcf in the VariantAnnotation package.

Value

scanBcfHeader returns a list, with one element for each file named in file. Each element of the list is itself a list containing three elements. The reference element is a character() vector with names of reference sequences. The sample element is a character() vector of names of samples. The header element is a character() vector of the header lines (preceeded by "##") present in the VCF file.

scanBcf returns a list, with one element per file. Each list has 9 elements, corresponding to the columns of the VCF specification: CHROM, POS, ID, REF, ALTQUAL, FILTER, INFO, FORMAT, GENO.

The GENO element is itself a list, with elements corresponding to fields supported by 'bcftools' (see documentation at the url below).

asBcf creates a binary BCF file from a text VCF file.

indexBcf creates an index into the BCF file.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

References

http://vcftools.sourceforge.net/specs.html outlines the VCF specification.

http://samtools.sourceforge.net/mpileup.shtml contains information on the portion of the specification implemented by bcftools.

http://samtools.sourceforge.net/ provides information on samtools.

See Also

BcfFile, TabixFile

Examples

```
fl <- system.file("extdata", "ex1.bcf", package="Rsamtools",
                  mustWork=TRUE)
scanBcfHeader(fl)
bcf <- scanBcf(fl)
## value: list-of-lists
str(bcf[1:8])
names(bcf[["GENO"]])
str(head(bcf[["GENO"]][["PL"]]))
example(BcfFile)
```

Compression — *File compression for tabix (bgzip) and fasta (razip) files.*

Description

These functions compress files for use in other parts of **Rsamtools**: `bgzip` for tabix files, `razip` for random-access fasta files.

Usage

```
bgzip(file, dest=sprintf("%s.gz", file), overwrite = FALSE)
razip(file, dest=sprintf("%s.rz", file), overwrite = FALSE)
```

Arguments

`file`	A character(1) path to an existing file. This file will be compressed.
`dest`	A character(1) path to a file. This will be the compressed file. If `dest` exists, then it is only over-written when `overwrite=TRUE`.
`overwrite`	A logical(1) indicating whether `dest` should be over-written, if it already exists.

Value

The full path to `dest`.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

References

`http://samtools.sourceforge.net/`

See Also

`TabixFile`, `FaFile`.

Examples

```
from <- system.file("extdata", "ex1.sam", package="Rsamtools",
                    mustWork=TRUE)
to <- tempfile()
zipped <- bgzip(from, to)
```

deprecated *Deprecated functions*

Description

Functions listed on this page are no longer supported.

Details

For `yieldTabix`, use the `yieldSize` argument of `TabixFiles`.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

FaFile *Manipulate indexed fasta files.*

Description

Use `FaFile()` to create a reference to an indexed fasta file. The reference remains open across calls to methods, avoiding costly index re-loading.

`FaFileList()` provides a convenient way of managing a list of `FaFile` instances.

Usage

```
## Constructors

FaFile(file, ...)
FaFileList(...)

## Opening / closing

## S3 method for class FaFile
open(con, ...)
## S3 method for class FaFile
close(con, ...)

## accessors; also path(), index()

## S4 method for signature FaFile
isOpen(con, rw="")

## actions
## S4 method for signature FaFile
indexFa(file, ...)

## S4 method for signature FaFile
scanFaIndex(file, ...)
## S4 method for signature FaFileList
scanFaIndex(file, ..., as=c("GRangesList", "GRanges"))

## S4 method for signature FaFile
seqinfo(x)

## S4 method for signature FaFile
countFa(file, ...)

## S4 method for signature FaFile,GRanges
scanFa(file, param, ...)
## S4 method for signature FaFile,RangesList
scanFa(file, param, ...)
## S4 method for signature FaFile,RangedData
scanFa(file, param, ...)
## S4 method for signature FaFile,missing
scanFa(file, param, ...)

## S4 method for signature FaFile
getSeq(x, param, ...)
## S4 method for signature FaFileList
getSeq(x, param, ...)
```

Arguments

`con, x`	An instance of `FaFile` or (for `getSeq`) `FaFileList`.
`file`	A character(1) vector of the fasta file path (for `FaFile`), or an instance of class `FaFile` or `FaFileList` (for `scanFaIndex`, `getSeq`).
`param`	An optional `GRanges`, `RangesList`, or `RangedData` instance to select reads (and sub-sequences) for input. See Methods, below.
`...`	Additional arguments. For `FaFileList`, this can either be a single character vector of paths to BAM files, or several instances of `FaFile` objects.
`rw`	Mode of file; ignored.
`as`	character(1) specifying the return type, selected from specified options. When `GRangesList`, index information from each file is returned as an element of the list. When `GRangesList`, index information is collapsed across files into the unique index elements.

Objects from the Class

Objects are created by calls of the form `FaFile()`.

Fields

The `FaFile` class inherits fields from the `RsamtoolsFile` class.

Functions and methods

`FaFileList` inherits methods from `RsamtoolsFileList` and `SimpleList`.

Opening / closing:

open.FaFile Opens the (local or remote) `path` and `index` files. Returns a `FaFile` instance.

close.FaFile Closes the `FaFile` `con`; returning (invisibly) the updated `FaFile`. The instance may be re-opened with `open.FaFile`.

Accessors:

path Returns a character(1) vector of the fasta path name.

index Returns a character(1) vector of fasta index name (minus the '.fai' extension).

Methods:

indexFa Visit the path in `path(file)` and create an index file (with the extension '.fai').

scanFaIndex Read the sequence names and and widths of recorded in an indexed fasta file, returning the information as a `GRanges` object.

seqinfo Consult the index file for defined sequences (`seqlevels()`) and lengths (`seqlengths()`).

countFa Return the number of records in the fasta file.

scanFa Return the sequences indicated by `param` as a `DNAStringSet` instance. `seqnames(param)` selects the sequences to return; `start(param)` and `end{param}` define the (1-based) region of the sequence to return. Values of `end(param)` greater than the width of the sequence are set to the width of the sequence. When `param` is missing, all records are selected. When `length(param)==0` no records are selected.

getSeq Returns the sequences indicated by `param` from the indexed fasta file(s) of `file`.

For the `FaFile` method, the return type is a `DNAStringSet`. The `getSeq,FaFile` and `scanFa,FaFile,GRanges` methods differ in that `getSeq` will reverse complement sequences selected from the minus strand.

For the `FaFileList` method, the `param` argument must be a `GRangesList` of the same length as `file`, creating a one-to-one mapping between the ith element of `file` and the ith element of `param`; the return type is a `SimpleList` of `DNAStringSet` instances, with elements of the list in the same order as the input elements.

show Compactly display the object.

Author(s)

Martin Morgan

Examples

```
fl <- system.file("extdata", "ce2dict1.fa", package="Rsamtools",
                  mustWork=TRUE)
fa <- open(FaFile(fl))                   # open
countFa(fa)
(idx <- scanFaIndex(fa))
(dna <- scanFa(fa, param=idx[1:2]))
ranges(idx) <- narrow(ranges(idx), -10)  # last 10 nucleotides
(dna <- scanFa(fa, param=idx[1:2]))
```

FaInput *Operations on indexed 'fasta' files.*

Description

Scan indexed fasta (or compressed fasta) files and their indicies.

Usage

```
indexFa(file, ...)
## S4 method for signature character
indexFa(file, ...)

scanFaIndex(file, ...)
## S4 method for signature character
scanFaIndex(file, ...)

countFa(file, ...)
## S4 method for signature character
countFa(file, ...)

scanFa(file, param, ...)
## S4 method for signature character,GRanges
scanFa(file, param, ...)
## S4 method for signature character,RangesList
scanFa(file, param, ...)
## S4 method for signature character,RangedData
scanFa(file, param, ...)
## S4 method for signature character,missing
scanFa(file, param, ...)
```

Arguments

`file`	A character(1) vector containing the fasta file path.
`param`	An optional `GRanges`, `RangesList`, or `RangedData` instance to select reads (and sub-sequences) for input.
`...`	Additional arguments, currently unused.

Value

`indexFa` visits the path in `file` and create an index file at the same location but with extension '.fai').

`scanFaIndex` reads the sequence names and and widths of recorded in an indexed fasta file, returning the information as a `GRanges` object.

`countFa` returns the number of records in the fasta file.

`scanFa` return the sequences indicated by `param` as a `DNAStringSet` instance. `seqnames(param)` selects the sequences to return; `start(param)` and `end{param}` define the (1-based) region of the sequence to return. Values of `end(param)` greater than the width of the sequence are set to the width of the sequence. When `param` is missing, all records are selected. When `param` is `GRanges()`, no records are selected.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

References

`http://samtools.sourceforge.net/` provides information on `samtools`.

Examples

```
fa <- system.file("extdata", "ce2dict1.fa", package="Rsamtools",
                  mustWork=TRUE)
countFa(fa)
(idx <- scanFaIndex(fa))
(dna <- scanFa(fa, idx[1:2]))
ranges(idx) <- narrow(ranges(idx), -10)  # last 10 nucleotides
(dna <- scanFa(fa, idx[1:2]))
```

`findMateAlignment` *Pairing the elements of a GAlignments object*

Description

Utilities for pairing the elements of a GAlignments object.

Usage

```
findMateAlignment(x)
makeGAlignmentPairs(x, use.names=FALSE, use.mcols=FALSE)

## Related low-level utilities:
getDumpedAlignments()
countDumpedAlignments()
flushDumpedAlignments()
```

Arguments

`x`	A named GAlignments object with metadata columns `flag`, `mrnm`, and `mpos`. Typically obtained by loading aligned paired-end reads from a BAM file with: `param <- ScanBamParam(what=c("flag", "mrnm", "mpos"))` `x <- readGAlignmentsFromBam(..., use.names=TRUE, param=param)`
`use.names`	Whether the names on the input object should be propagated to the returned object or not.
`use.mcols`	Names of the metadata columns to propagate to the returned GAlignmentPairs object.

Details

Pairing algorithm used by findMateAlignment: `findMateAlignment` is the power horse used by higher-level functions like `makeGAlignmentPairs` and `readGAlignmentPairsFromBam` for pairing the records loaded from a BAM file containing aligned paired-end reads.

It implements the following pairing algorithm:

- First only records with flag bit 0x1 set to 1, flag bit 0x4 set to 0, and flag bit 0x8 set to 0 are candidates for pairing (see the SAM Spec for a description of flag bits and fields). `findMateAlignment` will ignore any other record. That is, records that correspond to single-end reads, and records that correspond to paired-end reads where one or both ends are unmapped, are discarded.
- Then the algorithm looks at the following fields and flag bits:
  - (A) QNAME
  - (B) RNAME, RNEXT
  - (C) POS, PNEXT
  - (D) Flag bits Ox10 and 0x20
  - (E) Flag bits 0x40 and 0x80
  - (F) Flag bit 0x2
  - (G) Flag bit 0x100

  2 records rec(i) and rec(j) are considered mates iff all the following conditions are satisfied:
  - (A) They have the same QNAME
  - (B) RNEXT(i) == RNAME(j) and RNEXT(j) == RNAME(i)
  - (C) PNEXT(i) == POS(j) and PNEXT(j) == POS(i)

- (D) Flag bit 0x20 of rec(i) == Flag bit 0x10 of rec(j) and Flag bit 0x20 of rec(j) == Flag bit 0x10 of rec(i)
- (E) rec(i) corresponds to the first segment in the template and rec(j) corresponds to the last segment in the template, OR, rec(j) corresponds to the first segment in the template and rec(i) corresponds to the last segment in the template
- (F) rec(i) and rec(j) have same flag bit 0x2
- (G) rec(i) and rec(j) have same flag bit 0x100

Timing and memory requirement of the pairing algorithm: The estimated timings and memory requirements on a modern Linux system are (those numbers may vary depending on your hardware and OS):

```
  nb of alignments |         time | required memory
  -----------------+--------------+----------------
        8 millions |       28 sec |          1.4 GB
       16 millions |       58 sec |          2.8 GB
       32 millions |        2 min |          5.6 GB
       64 millions | 4 min 30 sec |         11.2 GB
```

This is for a GAlignments object coming from a file with an "average nb of records per unique QNAME" of 2.04. A value of 2 (which means the file contains only primary reads) is optimal for the pairing algorithm. A greater value, say > 3, will significantly degrade its performance. An easy way to avoid this degradation is to load only primary alignments by setting the `isNotPrimaryRead` flag to `FALSE` in ScanBamParam(). See examples in `?readGAlignmentPairsFromBam` for how to do this.

Ambiguous pairing: The above algorithm will find almost all pairs unambiguously, even when the same pair of reads maps to several places in the genome. Note that, when a given pair maps to a single place in the genome, looking at (A) is enough to pair the 2 corresponding records. The additional conditions (B), (C), (D), (E), (F), and (G), are only here to help in the situation where more than 2 records share the same QNAME. And that works most of the times. Unfortunately there are still situations where this is not enough to solve the pairing problem unambiguously.

For example, here are 4 records (loaded in a GAlignments object) that cannot be paired with the above algorithm:

Showing the 4 records as a GAlignments object of length 4:

```
GAlignments with 4 alignments and 2 metadata columns:
                    seqnames strand       cigar    qwidth     start    end
                       <Rle>  <Rle> <character> <integer> <integer> <integer>
  SRR031714.2658602    chr2R      +   21M384N16M        37   6983850 6984270
  SRR031714.2658602    chr2R      +   21M384N16M        37   6983850 6984270
  SRR031714.2658602    chr2R      -   13M372N24M        37   6983858 6984266
  SRR031714.2658602    chr2R      -   13M378N24M        37   6983858 6984272
                        width      ngap |     mrnm      mpos
                    <integer> <integer> | <factor> <integer>
  SRR031714.2658602       421         1 |    chr2R   6983858
  SRR031714.2658602       421         1 |    chr2R   6983858
  SRR031714.2658602       409         1 |    chr2R   6983850
  SRR031714.2658602       415         1 |    chr2R   6983850
```

Note that the BAM fields show up in the following columns:

- QNAME: the names of the GAlignments object (unnamed col)
- RNAME: the seqnames col
- POS: the start col
- RNEXT: the mrnm col
- PNEXT: the mpos col

As you can see, the aligner has aligned the same pair to the same location twice! The only difference between the 2 aligned pairs is in the CIGAR i.e. one end of the pair is aligned twice to the same location with exactly the same CIGAR while the other end of the pair is aligned twice to the same location but with slightly different CIGARs.

Now showing the corresponding flag bits:

```
     isPaired isProperPair isUnmappedQuery hasUnmappedMate isMinusStrand
[1,]        1            1               0               0             0
[2,]        1            1               0               0             0
[3,]        1            1               0               0             1
[4,]        1            1               0               0             1
     isMateMinusStrand isFirstMateRead isSecondMateRead isNotPrimaryRead
[1,]                 1               0                1                0
[2,]                 1               0                1                0
[3,]                 0               1                0                0
[4,]                 0               1                0                0
     isNotPassingQualityControls isDuplicate
[1,]                           0           0
[2,]                           0           0
[3,]                           0           0
[4,]                           0           0
```

As you can see, rec(1) and rec(2) are second mates, rec(3) and rec(4) are both first mates. But looking at (A), (B), (C), (D), (E), (F), and (G), the pairs could be rec(1) <-> rec(3) and rec(2) <-> rec(4), or they could be rec(1) <-> rec(4) and rec(2) <-> rec(3). There is no way to disambiguate!

So `findMateAlignment` is just ignoring (with a warning) those alignments with ambiguous pairing, and dumping them in a place from which they can be retrieved later (i.e. after `findMateAlignment` has returned) for further examination (see "Dumped alignments" subsection below for the details). In other words, alignments that cannot be paired unambiguously are not paired at all. Concretely, this means that `readGAlignmentPairs` is guaranteed to return a GAlignmentPairs object where every pair was formed in an non-ambiguous way. Note that, in practice, this approach doesn't seem to leave aside a lot of records because ambiguous pairing events seem pretty rare.

Dumped alignments: Alignments with ambiguous pairing are dumped in a place ("the dump environment") from which they can be retrieved with `getDumpedAlignments()` after `findMateAlignment` has returned.

Two additional utilities are provided for manipulation of the dumped alignments: `countDumpedAlignments` for counting them (a fast equivalent to `length(getDumpedAlignments())`), and `flushDumpedAlignments` to flush "the dump environment". Note that "the dump environment" is automatically flushed at the beginning of a call to `findMateAlignment`.

Value

For `findMateAlignment`: An integer vector of the same length as `x`, containing only positive or NA values, where the i-th element is interpreted as follow:

- An NA value means that no mate or more than 1 mate was found for `x[i]`.
- A non-NA value j gives the index in `x` of `x[i]`'s mate.

For `makeGAlignmentPairs`: A GAlignmentPairs object where the pairs are formed internally by calling `findMateAlignment` on `x`.

For `getDumpedAlignments`: `NULL` or a GAlignments object containing the dumped alignments. See "Dumped alignments" subsection in the "Details" section above for the details.

For `countDumpedAlignments`: The number of dumped alignments.

Nothing for `flushDumpedAlignments`.

Author(s)

H. Pages

See Also

GAlignments-class, GAlignmentPairs-class, `readGAlignmentsFromBam`, `readGAlignmentPairsFromBam`

Examples

```
bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools",
                       mustWork=TRUE)
param <- ScanBamParam(what=c("flag", "mrnm", "mpos"))
x <- readGAlignmentsFromBam(bamfile, use.names=TRUE, param=param)
mate <- findMateAlignment(x)
head(mate)
table(is.na(mate))
galp0 <- makeGAlignmentPairs(x)
galp <- makeGAlignmentPairs(x, use.name=TRUE, use.mcols="flag")
galp
colnames(mcols(galp))
colnames(mcols(first(galp)))
colnames(mcols(last(galp)))
```

`headerTabix` *Retrieve sequence names defined in a tabix file.*

Description

This function queries a tabix file, returning the names of the ‘sequences’ used as a key when creating the file.

Usage

```
headerTabix(file, ...)
## S4 method for signature character
headerTabix(file, ...)
```

Arguments

`file`	A `character(1)` file path or `TabixFile` instance pointing to a 'tabix' file.
`...`	Additional arguments, currently ignored.

Value

A `list(4)` of the sequence names, column indicies used to sort the file, the number of lines skipped while indexing, and the comment character used while indexing.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

Examples

```
fl <- system.file("extdata", "example.gtf.gz", package="Rsamtools",
                  mustWork=TRUE)
headerTabix(fl)
```

`indexTabix` *Compress and index tabix-compatible files.*

Description

Index (with `indexTabix`) files that have been sorted into ascending sequence, start and end position ordering.

Usage

```
indexTabix(file,
           format=c("gff", "bed", "sam", "vcf", "vcf4", "psltbl"),
           seq=integer(), start=integer(), end=integer(),
           skip=0L, comment="#", zeroBased=FALSE, ...)
```

Arguments

`file`	A characater(1) path to a sorted, bgzip-compressed file.
`format`	The format of the data in the compressed file. A characater(1) matching one of the types named in the function signature.
`seq`	If `format` is missing, then `seq` indicates the column in which the 'sequence' identifier (e.g., chrq) is to be found.
`start`	If `format` is missing, `start` indicates the column containing the start coordinate of the feature to be indexed.

`end`	If `format` is missing, end indicates the column containing the ending coordinate of the feature to be indexed.
`skip`	The number of lines to be skipped at the beginning of the file.
`comment`	A single character which, when present as the first character in a line, indicates that the line is to be omitted. from indexing.
`zeroBased`	A logical(1) indicating whether coordinats in the file are zero-based.
`...`	Additional arguments.

Value

The return value of `indexTabix` is an updated instance of `file` reflecting the newly-created index file.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

References

`http://samtools.sourceforge.net/tabix.shtml`

Examples

```
from <- system.file("extdata", "ex1.sam", package="Rsamtools",
                    mustWork=TRUE)
to <- tempfile()
zipped <- bgzip(from, to)
idx <- indexTabix(zipped, "sam")

tab <- TabixFile(zipped, idx)
```

`PileupFiles` *Represent BAM files for pileup summaries.*

Description

Use `PileupFiles()` to create a reference to a BAM files (and their indicies), to be used for calculating pile-up summaries.

Usage

```
## Constructors
PileupFiles(files, ..., param=PileupParam())
## S4 method for signature character
PileupFiles(files, ..., param=PileupParam())
## S4 method for signature list
```

```
PileupFiles(files, ..., param=PileupParam())

## opening / closing
## S3 method for class PileupFiles
open(con, ...)
## S3 method for class PileupFiles
close(con, ...)

## accessors; also path()
## S4 method for signature PileupFiles
isOpen(con, rw="")
plpFiles(object)
plpParam(object)

## actions
## S4 method for signature PileupFiles,missing
applyPileups(files, FUN, ..., param)
## S4 method for signature PileupFiles,PileupParam
applyPileups(files, FUN, ..., param)

## display
## S4 method for signature PileupFiles
show(object)
```

Arguments

`files`	For `PileupFiles`, a `character()` or `list` of `BamFile` instances representing files to be included in the pileup. Using a `list` of `BamFile` allows indicies to be specified when these are in non-standard format. All elements of `...` must be the same type. For `applyPileups,PileupFiles-method`, a `PileupFiles` instance.
`...`	Additional arguments, currently ignored.
`con, object`	An instance of `PileupFiles`.
`FUN`	A function of one argument; see `applyPileups`.
`param`	An instance of `PileupParam`, to select which records to include in the pileup, and which summary information to return.
`rw`	character() indicating mode of file; not used for `TabixFile`.

Objects from the Class

Objects are created by calls of the form `PileupFiles()`.

Fields

The `PileupFiles` class is implemented as an S4 reference class. It has the following fields:

files A list of `BamFile` instances.

param An instance of `PileupParam`.

Functions and methods

Opening / closing:

open.PileupFiles Opens the (local or remote) `path` and `index` of each file in the `PileupFiles` instance. Returns a `PileupFiles` instance.

close.PileupFiles Closes each file in the `PileupFiles` instance; returning (invisibly) the updated `PileupFiles`. The instance may be re-opened with `open.PileupFiles`.

Accessors:

plpFiles Returns the `list` of the files in the `PileupFiles` instance.

plpParam Returns the `PileupParam` content of the `PileupFiles` instance.

Methods:

applyPileups Calculate the pileup across all files in `files` according to criteria in `param` (or `plpParam(files)` if `param` is missing), invoking `FUN` on each range or collection of positions. See `applyPileups`.

show Compactly display the object.

Author(s)

Martin Morgan

Examples

```
example(applyPileups)
```

`PileupParam` *Parameters for creating pileups from BAM files*

Description

Use `PileupParam()` to create a parameter object influencing what fields and which records are used to calculate pile-ups, and to influence the values returned.

Usage

```
# Constructor
PileupParam(flag = scanBamFlag(),
    minBaseQuality = 13L, minMapQuality = 0L,
    minDepth = 0L, maxDepth = 250L,
    yieldSize = 1L, yieldBy = c("range", "position"), yieldAll = FALSE,
    which = GRanges(), what = c("seq", "qual"))
```

```
# Accessors
plpFlag(object)
plpFlag(object) <- value
plpMaxDepth(object)
plpMaxDepth(object) <- value
plpMinBaseQuality(object)
plpMinBaseQuality(object) <- value
plpMinDepth(object)
plpMinDepth(object) <- value
plpMinMapQuality(object)
plpMinMapQuality(object) <- value
plpWhat(object)
plpWhat(object) <- value
plpWhich(object)
plpWhich(object) <- value
plpYieldAll(object)
plpYieldAll(object) <- value
plpYieldBy(object)
plpYieldBy(object) <- value
plpYieldSize(object)
plpYieldSize(object) <- value

## S4 method for signature PileupParam
show(object)
```

Arguments

`flag`	An instance of the object returned by `scanBamFlag`, restricting various aspects of reads to be included or excluded.
`minBaseQuality`	The minimum read base quality below which the base is ignored when summarizing pileup information.
`minMapQuality`	The minimum mapping quality below which the entire read is ignored.
`minDepth`	The minimum depth of the pile-up below which the position is ignored.
`maxDepth`	The maximum depth of reads considered at any position; this can be used to limit memory consumption.
`yieldSize`	The number of records to include in each call to `FUN`.
`yieldBy`	How records are to be counted. By range (in which case `yieldSize` must equal 1) means that `FUN` is invoked once for each range in `which`. By position means that `FUN` is invoked whenever pile-ups have been accumulated for `yieldSize` positions, regardless of ranges in `which`.
`yieldAll`	Whether to report all positions (`yieldAll=TRUE`), or just those passing the filtering criteria of `flag`, `minBaseQuality`, etc. When `yieldAll=TRUE`, positions not passing filter criteria have ‘0’ entries in `seq` or `qual`.
`which`	A `GRanges` or `RangesList` instance restricting pileup calculations to the corresponding genomic locations.

`what`	A `character()` instance indicating what values are to be returned. One or more of `c("seq", "qual")`.
`object`	An instace of class `PileupParam`.
`value`	An instance to be assigned to the corresponding slot of the `PileupParam` instance.

Objects from the Class

Objects are created by calls of the form `PileupParam()`.

Slots

Slot interpretation is as described in the 'Arguments' section.

`flag` Object of class `integer` encoding flags to be kept when they have their '0' (`keep0`) or '1' (`keep1`) bit set.

`minBaseQuality` An `integer(1)`.

`minMapQuality` An `integer(1)`.

`minDepth` An `integer(1)`.

`maxDepth` An `integer(1)`.

`yieldSize` An `integer(1)`.

`yieldBy` An `character(1)`.

`yieldAll` A `logical(1)`.

`which` A `GRanges` or `RangesList` instance.

`what` A `character()`.

Functions and methods

See 'Usage' for details on invocation.

Constructor:

PileupParam: Returns a `PileupParam` object.

Accessors: get or set corresponding slot values; for setters, `value` is coerced to the type of the corresponding slot.

plpFlag, plpFlag<- Returns or sets the named `integer` vector of flags; see `scanBamFlag`.

plpMinBaseQuality, plpMinBaseQuality<- Returns or sets an `integer(1)` vector of miminum base qualities.

plpMinMapQuality, plpMinMapQuality<- Returns or sets an `integer(1)` vector of miminum map qualities.

plpMinDepth, plpMinDepth<- Returns or sets an `integer(1)` vector of miminum pileup depth.

plpMaxDepth, plpMaxDepth<- Returns or sets an `integer(1)` vector of the maximum depth to which pileups are calculated.

plpYieldSize, plpYieldSize<- Returns or sets an `integer(1)` vector of yield size.

plpYieldBy, plpYieldBy<- Returns or sets an `character(1)` vector determining how pileups will be returned.

plpYieldAll, plpYieldAll<- Returns or sets an `logical(1)` vector indicating whether all positions, or just those satisfying pileup positions, are to be returned.

plpWhich, plpWhich<- Returns or sets the object influencing which locations pileups are calculated over.

plpWhat, plpWhat<- Returns or sets the `character` vector describing what summaries are returned by pileup.

Methods:

show Compactly display the object.

Author(s)

Martin Morgan

See Also

applyPileups.

Examples

```
example(applyPileups)
```

`quickCountBam` *Group the records of a BAM file based on their flag bits and count the number of records in each group*

Description

`quickCountBam` groups the records of a BAM file based on their flag bits and counts the number of records in each group.

Usage

```
quickCountBam(file, ..., param=ScanBamParam(), mainGroupsOnly=FALSE)

## S4 method for signature character
quickCountBam(file, index=file, ..., param=ScanBamParam(),
    mainGroupsOnly=FALSE)

## S4 method for signature list
quickCountBam(file, ..., param=ScanBamParam(), mainGroupsOnly=FALSE)
```

Arguments

`file, index`	For the character method, the path to the BAM file to read, and to the index file of the BAM file to read, respectively. For the list() method, `file` is a named list with elements “qname” and “flag” with content as from `scanBam`.
`...`	Additional arguments, perhaps used by methods.
`param`	An instance of `ScanBamParam`. This determines which records are considered in the counting.
`mainGroupsOnly`	If `TRUE`, then the counting is performed for the main groups only.

Value

Nothing is returned. A summary of the counts is printed to the console unless redirected by `sink`.

Author(s)

H. Pages

References

`http://samtools.sourceforge.net/`

See Also

`scanBam`, `ScanBamParam`.

`BamFile` for a method for that class.

Examples

```
bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools",
                       mustWork=TRUE)
quickCountBam(bamfile)
```

`readGAlignmentsFromBam` *Reading a GAlignments, GappedReads, GAlignmentPairs, or GAlignmentsList object from a BAM file*

Description

Read a GAlignments, GappedReads, GAlignmentPairs, or GAlignmentsList object from a BAM file.

Usage

```
readGAlignmentsFromBam(file, index=file, ..., use.names=FALSE, param=NULL,
                       with.which_label=FALSE)

readGappedReadsFromBam(file, index=file, use.names=FALSE, param=NULL,
                       with.which_label=FALSE)

readGAlignmentPairsFromBam(file, index=file, use.names=FALSE, param=NULL,
                           with.which_label=FALSE)

readGAlignmentsListFromBam(file, index=file, ..., use.names=FALSE,
                           param=ScanBamParam(), with.which_label=FALSE)
```

Arguments

`file, index`	The path to the BAM file to read, and to the index file of the BAM file to read, respectively. The latter is given *without* the '.bai' extension. See `scanBam` for more information.
`...`	Arguments passed to other methods.
`use.names`	Use the query template names (QNAME field) as the names of the returned object? If not (the default), then the returned object has no names.
`param`	`NULL` or an instance of `ScanBamParam`. Like for `scanBam`, this influences what fields and which records are imported. However, note that the fields specified thru this `ScanBamParam` object will be loaded *in addition* to any field required for generating the returned object (GAlignments, GappedReads, or GAlignment-Pairs object), but only the fields requested by the user will actually be kept as metadata columns of the object. By default (i.e. `param=NULL` or `param=ScanBamParam()`), no additional field is loaded. The flag used is `scanBamFlag(isUnmappedQuery=FALSE)` for `readGAlignmentsFromBam`, `readGappedReadsFromBam` and `readGAlignmentsListFromBam` (i.e. only records corresponding to mapped reads are loaded), and `scanBamFlag(isUnmappedQuery=FALSE, isPaired=TR` for `readGAlignmentPairsFromBam` (i.e. only records corresponding to paired-end reads with both ends mapped are loaded).
`with.which_label`	`TRUE` or `FALSE` (the default). If `TRUE` and if `param` has a `which` component, a `"which_label"` metadata column is added to the returned GAlignments or GappedReads object, or to the `first` and `last` components of the returned GAlignmentPairs object. In the case of `readGAlignmentsListFromBam`, it's added as an *inner* metadata column, that is, the metadata column is placed on the GAlignments object obtained by unlisting the returned GAlignmentsList object. The purpose of this metadata column is to unambiguously identify the range in `which` where each element in the returned object originates from. The labels used to identify the ranges are normally of the form `"seq1:12250-246500"`, that is, they're the same as the names found on the outer list that `scanBam` would return if called with the same `param` argument. If some ranges are duplicated,

then the labels are made unique by appending a unique suffix to all of them. The `"which_label"` metadata column is represented as a factor-Rle.

Details

See ?`GAlignments-class` for a description of GAlignments objects.

See ?`GappedReads-class` for a description of GappedReads objects.

`readGAlignmentPairsFromBam` proceeds in 2 steps:

1. Load the BAM file into a GAlignments object with `readGAlignmentsFromBam`;
2. Turn this GAlignments object into a GAlignmentPairs object by pairing its elements.

See ?`GAlignmentPairs-class` for a description of GAlignmentPairs objects, and ?`findMateAlignment` for a description of the pairing algorithm (including timing and memory requirement).

`readGAlignmentsListFromBam` pairs records into 'mates' acording to the criteria below. A `GAlignmentsList` is returned with a 'mates' metadata column which indicates mate status. The mates are returned first followed by non-mates. When the 'file' argument is a BamFile, 'asMates=TRUE' must be set, otherwise the data are treated as single-end reads. See the 'asMates' section of ?`BamFile` for details.

Mate criteria:

- Bit 0x1 (multiple segments) is 1.
- Bit 0x4 (segment unmapped) is 0.
- Bit 0x8 (next segment unmapped) is 0.
- Bit 0x40 and 0x80 (first/last segment): Segments are a pair of first/last OR neither segment is marked first/last.
- Bit 0x100 (secondary alignment): Both segments are secondary OR both not secondary
- Bit 0x2 (properly aligned): Both segments are properly aligned
- 'qname' match.
- 'tid' match.
- segment1 'mpos' matches segment2 'pos' AND segment2 'mpos' matches segment1 'pos'

Records not passing these criteria are returned with mate status FALSE. Flags, tags and ranges may be specified in the `ScanBamParam` for fine tuning of results.

See ?`GAlignmentsList-class` for a description of GAlignmentsList objects.

Value

A GAlignments object for `readGAlignmentsFromBam`.

A GappedReads object for `readGappedReadsFromBam`.

A GAlignmentPairs object for `readGAlignmentPairsFromBam`. Note that a BAM (or SAM) file can in theory contain a mix of single-end and paired-end reads, but in practise it seems that single-end and paired-end are not mixed. In other words, the value of flag bit 0x1 (`isPaired`) is the same for all the records in a file. So if `readGAlignmentPairsFromBam` returns a GAlignmentPairs object of length zero, this almost certainly means that the BAM (or SAM) file contains alignments for single-end reads (although it could also mean that the user-supplied `ScanBamParam` is filtering out everything, or that the file is empty, or that all the records in the file correspond to unmapped reads).

A GAlignmentsList object for `readGAlignmentsListFromBam`.

Note

BAM records corresponding to unmapped reads are always ignored.

Starting with Rsamtools 1.7.1 (BioC 2.10), PCR or optical duplicates are loaded by default (use `scanBamFlag(isDuplicate=FALSE)` to drop them).

Author(s)

H. Pages <hpages@fhcrc.org> and Valerie Obenchain <vobencha@fhcrc.org>

See Also

GAlignments-class, GAlignmentsList-class, GappedReads-class, GAlignmentPairs-class, `findMateAlignment`, `scanBam`, `ScanBamParam`

Examples

```
## ---------------------------------------------------------------------
## A. readGAlignmentsFromBam()
## ---------------------------------------------------------------------

## Simple use:
bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools",
                       mustWork=TRUE)
gal1 <- readGAlignmentsFromBam(bamfile)
gal1
names(gal1)

## Using the use.names arg:
gal2 <- readGAlignmentsFromBam(bamfile, use.names=TRUE)
gal2
head(names(gal2))

## Using the param arg to drop PCR or optical duplicates as well as
## secondary alignments, and to load additional BAM fields:
param <- ScanBamParam(flag=scanBamFlag(isDuplicate=FALSE,
                                       isNotPrimaryRead=FALSE),
                      what=c("qual", "flag"))
gal3 <- readGAlignmentsFromBam(bamfile, param=param)
gal3
mcols(gal3)

## Using the param arg to load reads from particular regions.
## Note that if we werent providing a what argument here, all the
## BAM fields would be loaded:
which <- RangesList(seq1=IRanges(1000, 2000),
                    seq2=IRanges(c(100, 1000), c(1000, 2000)))
param <- ScanBamParam(which=which)
gal4 <- readGAlignmentsFromBam(bamfile, param=param)
gal4

## Note that a given record is loaded one time for each region it
## belongs to (this is a scanBam() feature, readGAlignmentsFromBam()
## is based on scanBam()):
which <- IRangesList(seq2=IRanges(c(1563, 1567), width=1))
param <- ScanBamParam(which=which)
gal5 <- readGAlignmentsFromBam(bamfile, param=param)
gal5

## Use with.which_label=TRUE to identify the range in which
## where each element in gal5 originates from.
gal5 <- readGAlignmentsFromBam(bamfile, param=param,
                               with.which_label=TRUE)
gal5

## Using the param arg to load tags. Except for MF and Aq, the tags
## specified below are predefined tags (see the SAM Spec for the list
## of predefined tags and their meaning).
param <- ScanBamParam(tag=c("MF", "Aq", "NM", "UQ", "H0", "H1"),
                      what="isize")
gal6 <- readGAlignmentsFromBam(bamfile, param=param)
mcols(gal6)  # "tag" cols always after "what" cols

## ---------------------------------------------------------------------
## B. readGappedReadsFromBam()
## ---------------------------------------------------------------------
greads1 <- readGappedReadsFromBam(bamfile)
greads1
names(greads1)
qseq(greads1)
greads2 <- readGappedReadsFromBam(bamfile, use.names=TRUE)
head(greads2)
head(names(greads2))

## ---------------------------------------------------------------------
## C. readGAlignmentPairsFromBam()
## ---------------------------------------------------------------------
galp1 <- readGAlignmentPairsFromBam(bamfile)
head(galp1)
names(galp1)
## Using the param arg to drop PCR or optical duplicates as well as
## secondary alignments (dropping secondary alignments can help make the
## pairing algorithm run significantly faster, see ?findMateAlignment):
param <- ScanBamParam(flag=scanBamFlag(isDuplicate=FALSE,
                                       isNotPrimaryRead=FALSE))
galp2 <- readGAlignmentPairsFromBam(bamfile, use.names=TRUE, param=param)
galp2
head(galp2)
head(names(galp2))

## ---------------------------------------------------------------------
## D. readGAlignmentsListFromBam()
## ---------------------------------------------------------------------

library(pasillaBamSubset)
```

```
## file as character.
fl <- untreated3_chr4()
galist1 <- readGAlignmentsListFromBam(fl)
galist1[1:3]
length(galist1)
table(elementLengths(galist1))

## When file is a BamFile, asMates must be TRUE. If FALSE,
## the data are treated as single-end and each list element of the
## GAlignmentsList will be of length 1. For single-end data
## use readGAlignments().
bf <- BamFile(fl, yieldSize=3, asMates=TRUE)
readGAlignmentsList(bf)

## Use a param to fine tune the results.
param <- ScanBamParam(flag=scanBamFlag(isProperPair=TRUE))
galist2 <- readGAlignmentsListFromBam(fl, param=param)
length(galist2)
```

`readPileup` *Import samtools 'pileup' files.*

Description

Import files created by evaluation of samtools' `pileup -cv` command.

Usage

```
readPileup(file, ...)
## S4 method for signature connection
readPileup(file, ..., variant=c("SNP", "indel", "all"))
```

Arguments

`file`	The file name, or `connection`, of the pileup output file to be parsed.
`...`	Additional arguments, passed to methods. For instance, specify `variant` for the readPileup,character-method.
`variant`	Type of variant to parse; select one.

Value

`readPileup` returns a `GRanges` object.

The value returned by `variant="SNP"` or `variant="all"` contains:

space: The chromosome names (fastq ids) of the reference sequence

position: The nucleotide position (base 1) of the variant.

referenceBase: The nucleotide in the reference sequence.

consensusBase; The consensus nucleotide, as determined by samtools pileup.

consensusQuality: The phred-scaled consensus quality.

snpQuality: The phred-scaled SNP quality (probability of the consensus being identical to the reference).

maxMappingQuality: The root mean square mapping quality of reads overlapping the site.

coverage: The number of reads covering the site.

The value returned by `variant="indel"` contains space, position, reference, consensus, consensusQuality, snpQuality, maxMappingQuality, and coverage fields, and:

alleleOne, alleleTwo The first (typically, in the reference sequence) and second allelic variants.

alleleOneSupport, alleleTwoSupport The number of reads supporting each allele.

additionalIndels The number of additional indels present.

Author(s)

Sean Davis

References

http://samtools.sourceforge.net/

Examples

```
fl <- system.file("extdata", "pileup.txt", package="Rsamtools",
                  mustWork=TRUE)
(res <- readPileup(fl))
xtabs(~referenceBase + consensusBase, mcols(res))[DNA_BASES,]

## Not run: ## uses a pipe, and arguments passed to read.table
## three successive piles of 100 records each
cmd <- "samtools pileup -cvf human_b36_female.fa.gz na19240_3M.bam"
p <- pipe(cmd, "r")
snp <- readPileup(p, nrow=100)  # variant="SNP"
indel <- readPileup(p, nrow=100, variant="indel")
all <- readPileup(p, nrow=100, variant="all")

## End(Not run)
```

RsamtoolsFile	*A base class for managing file references in Rsamtools*

Description

RsamtoolsFile is a base class for managing file references in **Rsamtools**; it is not intended for direct use by users – see, e.g., BamFile.

Usage

```
## accessors
index(object)
## S4 method for signature RsamtoolsFile
path(object, ...)
## S4 method for signature RsamtoolsFile
isOpen(con, rw="")
## S4 method for signature RsamtoolsFile
yieldSize(object, ...)
yieldSize(object, ...) <- value
## S4 method for signature RsamtoolsFile
show(object)
```

Arguments

con, object	An instance of a class derived from RsamtoolsFile.
rw	Mode of file; ignored.
...	Additional arguments, unused.
value	Replacement value.

Objects from the Class

Users do not directly create instances of this class; see, e.g., BamFile-class.

Fields

The RsamtoolsFile class is implemented as an S4 reference class. It has the following fields:

.extptr An externalptr initialized to an internal structure with opened bam file and bam index pointers.

path A character(1) vector of the file name.

index A character(1) vector of the index file name.

yieldSize An integer(1) vector of the number of records to yield.

Functions and methods

Accessors:

path Returns a character(1) vector of path names.

index Returns a character(1) vector of index path names.

yieldSize, yieldSize<- Return or set an integer(1) vector indicating yield size.

Methods:

isOpen Report whether the file is currently open.

show Compactly display the object.

Author(s)

Martin Morgan

`RsamtoolsFileList` *A base class for managing lists of Rsamtools file references*

Description

`RsamtoolsFileList` is a base class for managing lists of file references in **Rsamtools**; it is not intended for direct use – see, e.g., `BamFileList`.

Usage

```
## S4 method for signature RsamtoolsFileList
path(object, ...)
## S4 method for signature RsamtoolsFileList
isOpen(con, rw="")
## S3 method for class RsamtoolsFileList
open(con, ...)
## S3 method for class RsamtoolsFileList
close(con, ...)
## S4 method for signature RsamtoolsFileList
names(x)
## S4 method for signature RsamtoolsFileList
yieldSize(object, ...)
```

Arguments

`con, object, x`	An instance of a class derived from `RsamtoolsFileList`.
`rw`	Mode of file; ignored.
`...`	Additional arguments.

Objects from the Class

Users do not directly create instances of this class; see, e.g., `BamFileList-class`.

Functions and methods

This class inherits functions and methods for subseting, updating, and display from the `SimpleList` class.

Methods:

isOpen: Report whether each file in the list is currently open.

open: Attempt to open each file in the list.

close: Attempt to close each file in the list.

names: Names of each element of the list or, if names are `NULL`, the basename of the path of each element.

Author(s)

Martin Morgan

ScanBamParam *Parameters for scanning BAM files*

Description

Use `ScanBamParam()` to create a parameter object influencing what fields and which records are imported from a (binary) BAM file. Use of `which` requires that a BAM index file (`<filename>.bai`) exists.

Usage

```
# Constructor
ScanBamParam(flag = scanBamFlag(), simpleCigar = FALSE,
    reverseComplement = FALSE, tag = character(0),
    what = character(0), which)

# Constructor helpers
scanBamFlag(isPaired = NA, isProperPair = NA, isUnmappedQuery = NA,
    hasUnmappedMate = NA, isMinusStrand = NA, isMateMinusStrand = NA,
    isFirstMateRead = NA, isSecondMateRead = NA, isNotPrimaryRead = NA,
    isNotPassingQualityControls = NA, isDuplicate = NA,
    isValidVendorRead = NA)

scanBamWhat()

# Accessors
```

```
bamFlag(object, asInteger=FALSE)
bamFlag(object) <- value
bamReverseComplement(object)
bamReverseComplement(object) <- value
bamSimpleCigar(object)
bamSimpleCigar(object) <- value
bamTag(object)
bamTag(object) <- value
bamWhat(object)
bamWhat(object) <- value
bamWhich(object)
bamWhich(object) <- value

## S4 method for signature ScanBamParam
show(object)

# Flag utils
bamFlagAsBitMatrix(flag, bitnames=FLAG_BITNAMES)
bamFlagAND(flag1, flag2)
bamFlagTest(flag, value)
```

Arguments

`flag`	For `ScanBamParam`, an integer(2) vector used to filter reads based on their 'flag' entry. This is most easily created with the `scanBamFlag()` helper function. For `bamFlagAsBitMatrix`, `bamFlagTest` an integer vector where each element represents a 'flag' entry.
`simpleCigar`	A logical(1) vector which, when TRUE, returns only those reads for which the cigar (run-length encoded representation of the alignment) is missing or contains only matches / mismatches (`M`).
`reverseComplement`	A logical(1) vectors. BAM files store reads mapping to the minus strand as though they are on the plus strand. Rsamtools obeys this convention by default (`reverseComplement=FALSE`), but when this value is set to TRUE returns the sequence and quality scores of reads mapped to the minus strand in the reverse complement (sequence) and reverse (quality) of the read as stored in the BAM file. This might be useful if wishing to recover read and quality scores as represented in fastq files, but is NOT appropriate for variant calling or other alignment-based operations.
`tag`	A character vector naming tags to be extracted. A tag is an optional field, with arbitrary information, stored with each record. Tags are identified by two-letter codes, so all elements of `tag` must have exactly 2 characters.
`what`	A character vector naming the fields to return `scanBamWhat()` returns a vector of available fields. Fields are described on the `scanBam` help page.
`which`	A `GRanges`, `RangesList`, `RangedData`, or missing object, from which a `IRangesList` instance will be constructed. Names of the `IRangesList` correspond to reference sequences, and ranges to the regions on that reference sequence for which

matches are desired. Because data types are coerced to `IRangesList`, which does *not* include strand information (use the `flag` argument instead). Only records with a read overlapping the specified ranges are returned. All ranges must have ends less than or equal to 536870912.

`isPaired` A logical(1) indicating whether unpaired (FALSE), paired (TRUE), or any (NA) read should be returned.

`isProperPair` A logical(1) indicating whether improperly paired (FALSE), properly paired (TRUE), or any (NA) read should be returned. A properly paired read is defined by the alignment algorithm and might, e.g., represent reads aligning to identical reference sequences and with a specified distance.

`isUnmappedQuery`
A logical(1) indicating whether unmapped (TRUE), mapped (FALSE), or any (NA) read should be returned.

`hasUnmappedMate`
A logical(1) indicating whether reads with mapped (FALSE), unmapped (TRUE), or any (NA) mate should be returned.

`isMinusStrand` A logical(1) indicating whether reads aligned to the plus (FALSE), minus (TRUE), or any (NA) strand should be returned.

`isMateMinusStrand`
A logical(1) indicating whether mate reads aligned to the plus (FALSE), minus (TRUE), or any (NA) strand should be returned.

`isFirstMateRead`
A logical(1) indicating whether the first mate read should be returned (TRUE) or not (FALSE), or whether mate read number should be ignored (NA).

`isSecondMateRead`
A logical(1) indicating whether the second mate read should be returned (TRUE) or not (FALSE), or whether mate read number should be ignored (NA).

`isNotPrimaryRead`
A logical(1) indicating whether alignments that are primary (FALSE), are not primary (TRUE) or whose primary status does not matter (NA) should be returned. A non-primary alignment (“secondary alignment” in the SAM specification) might result when a read aligns to multiple locations. One alignment is designated as primary and has this flag set to FALSE; the remainder, for which this flag is TRUE, are designated by the aligner as secondary.

`isNotPassingQualityControls`
A logical(1) indicating whether reads passing quality controls (FALSE), reads not passing quality controls (TRUE), or any (NA) read should be returned.

`isValidVendorRead`
Deprecated; use `isNotPassingQualityControls`.

`isDuplicate` A logical(1) indicating that un-duplicated (FALSE), duplicated (TRUE), or any (NA) reads should be returned. 'Duplicated' reads may represent PCR or optical duplicates.

`object` An instance of class `ScanBamParam`.

`value` An instance of the corresponding slot, to be assigned to `object` or, for `bamFlagTest`, a `character(1)` name of the flag to test, e.g., “isUnmappedQuery”, from the arguments to `scanBamFlag`.

`asInteger`	logical(1) indicating whether 'flag' should be returned as an encoded integer vector (`TRUE`) or human-readable form (`FALSE`).
`bitnames`	Names of the flag bits to extract. Will be the colnames of the returned matrix.
`flag1, flag2`	Integer vectors containing 'flag' entries.

Objects from the Class

Objects are created by calls of the form `ScanBamParam()`.

Slots

- `flag` Object of class `integer` encoding flags to be kept when they have their '0' (`keep0`) or '1' (`keep1`) bit set.
- `simpleCigar` Object of class `logical` indicating, when TRUE, that only 'simple' cigars (empty or 'M') are returned.
- `reverseComplement` Object of class `logical` indicating, when TRUE, that reads on the minus strand are to be reverse complemented (sequence) and reversed (quality).
- `tag` Object of class `character` indicating what tags are to be returned.
- `what` Object of class `character` indicating what fields are to be returned.
- `which` Object of class `RangesList` indicating which reference sequence and coordinate reads must overlap.

Functions and methods

See 'Usage' for details on invocation.

Constructor:

ScanBamParam: Returns a `ScanBamParam` object. The `which` argument to the constructor can be one of several different types, as documented above.

Accessors:

bamTag, bamTag<- Returns or sets a `character` vector of tags to be extracted.

bamWhat, bamWhat<- Returns or sets a `character` vector of fields to be extracted.

bamWhich, bamWhich<- Returns or sets a `RangesList` of bounds on reads to be extracted. A length 0 `RangesList` represents all reads.

bamFlag, bamFlag<- Returns or sets an `integer(2)` representation of reads flagged to be kept or excluded.

bamSimpleCigar, bamSimpleCigar<- Returns or sets a `logical(1)` vector indicating whether reads without indels or clipping be kept.

bamReverseComplement, bamReverseComplement<- Returns or sets a `logical(1)` vector indicating whether reads on the minus strand will be returned with sequence reverse complemented and quality reversed.

Methods:

show Compactly display the object.

Author(s)

Martin Morgan

See Also

scanBam

Examples

```
## defaults
p0 <- ScanBamParam()

## subset of reads based on genomic coordinates
which <- RangesList(seq1=IRanges(1000, 2000),
                    seq2=IRanges(c(100, 1000), c(1000, 2000)))
p1 <- ScanBamParam(which=which)

## subset of reads based on flag value
p2 <- ScanBamParam(flag=scanBamFlag(isMinusStrand=FALSE))

## subset of fields
p3 <- ScanBamParam(what=c("rname", "strand", "pos", "qwidth"))

## use
fl <- system.file("extdata", "ex1.bam", package="Rsamtools",
                  mustWork=TRUE)
res <- scanBam(fl, param=p2)[[1]]
lapply(res, head)

## tags; NM: edit distance; H1: 1-difference hits
p4 <- ScanBamParam(tag=c("NM", "H1"), what="flag")
bam4 <- scanBam(fl, param=p4)
str(bam4[[1]][["tag"]])

## flag utils
flag <- scanBamFlag(isUnmappedQuery=FALSE, isMinusStrand=TRUE)
flag
bamFlagAsBitMatrix(flag)
flag4 <- bam4[[1]][["flag"]]
bamFlagAsBitMatrix(flag4[1:9], bitnames=c("isUnmappedQuery", "isMinusStrand"))
```

ScanBcfParam-class *Parameters for scanning BCF files*

Description

Use ScanBcfParam() to create a parameter object influencing the 'INFO' and 'GENO' fields parsed, and which sample records are imported from a BCF file. Use of which requires that a BCF index file (<filename>.bci) exists.

Usage

```
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)

## S4 method for signature missing
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)
## S4 method for signature RangesList
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)
## S4 method for signature RangedData
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)
## S4 method for signature GRanges
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)
## S4 method for signature GRangesList
ScanBcfParam(fixed=character(), info=character(), geno=character(),
             samples=character(), trimEmpty=TRUE, which, ...)

## Accessors
bcfFixed(object)
bcfInfo(object)
bcfGeno(object)
bcfSamples(object)
bcfTrimEmpty(object)
bcfWhich(object)
```

Arguments

`fixed`	A logical(1) for use with `ScanVcfParam` only.
`info`	A character() vector of 'INFO' fields (see scanVcfHeader) to be returned.
`geno`	A character() vector of 'GENO' fields (see scanVcfHeader) to be returned. `character(0)` returns all fields, `NA_character_` returns none.
`samples`	A character() vector of sample names (see scanVcfHeader) to be returned. `character(0)` returns all fields, `NA_character_` returns none.
`trimEmpty`	A logical(1) indicating whether 'GENO' fields with no values should be returned.
`which`	An object, for which a method is defined (see usage, above), describing the sequences and ranges to be queried. Variants whose `POS` lies in the interval(s) `[start, end)` are returned.
`object`	An instance of class `ScanBcfParam`.
`...`	Arguments used internally.

Objects from the Class

Objects can be created by calls of the form `ScanBcfParam()`.

Slots

`which`**:** Object of class `"RangesList"` indicating which reference sequence and coordinate variants must overlap.

`info`**:** Object of class `"character"` indicating portions of 'INFO' to be returned.

`geno`**:** Object of class `"character"` indicating portions of 'GENO' to be returned.

`samples`**:** Object of class `"character"` indicating the samples to be returned.

`trimEmpty`**:** Object of class `"logical"` indicating whether empty 'GENO' fields are to be returned.

`fixed`**:** Object of class `"character"`. For use with `ScanVcfParam` only.

Functions and methods

See 'Usage' for details on invocation.

Constructor:

ScanBcfParam: Returns a `ScanBcfParam` object. The `which` argument to the constructor can be one of several types, as documented above.

Accessors:

bcfInfo, bcfGeno, bcfTrimEmpty, bcfWhich: Return the corresponding field from `object`.

Methods:

show Compactly display the object.

Author(s)

Martin Morgan `mtmorgan@fhcrc.org`

See Also

scanVcf ScanVcfParam

Examples

```
## see ?ScanVcfParam examples
```

seqnamesTabix *Retrieve sequence names defined in a tabix file.*

Description

This function queries a tabix file, returning the names of the 'sequences' used as a key when creating the file.

Usage

```
seqnamesTabix(file, ...)
## S4 method for signature character
seqnamesTabix(file, ...)
```

Arguments

`file`	A `character(1)` file path or `TabixFile` instance pointing to a 'tabix' file.
`...`	Additional arguments, currently ignored.

Value

A `character()` vector of sequence names present in the file.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

Examples

```
fl <- system.file("extdata", "example.gtf.gz", package="Rsamtools",
                  mustWork=TRUE)
seqnamesTabix(fl)
```

sequenceLayer *Lay read sequences alongside the reference space, using their CIGARs*

Description

`sequenceLayer` can lay strings that belong to a given space (e.g. the `"query"` space) alongside another space (e.g. the `"reference"` space) by removing/injecting substrings from/into them, using the supplied CIGARs.

Its primary use case is to lay the read sequences stored in a BAM file (which are considered to belong to the `"query"` space) alongside the `"reference"` space. It can also be used to remove the parts of the read sequences that correspond to soft-clipping. More generally it can lay strings that belong to any supported space alongside any other supported space. See the Details section below for the list of supported spaces.

Usage

```
sequenceLayer(x, cigar, from="query", to="reference",
              D.letter="-", N.letter="-",
              I.letter="-", S.letter="+", H.letter="+")
```

Arguments

`x`	An XStringSet object containing strings that belong to a given space.
`cigar`	A character vector or factor of the same length as `x` containing the extended CIGAR strings (one per element in `x`).
`from, to`	A single string specifying one of the 8 supported spaces listed in the Details section below. `from` must be the current space (i.e. the space the strings in `x` belong to) and `to` is the space alonside which to lay the strings in `x`.
`D.letter, N.letter, I.letter, S.letter, H.letter`	A single letter used as a filler for injections. More on this in the Details section below.

Details

The 8 supported spaces are: `"reference"`, `"reference-N-regions-removed"`, `"query"`, `"query-before-hard-clipping` `"query-after-soft-clipping"`, `"pairwise"`, `"pairwise-N-regions-removed"`, and `"pairwise-dense"`.

Each space can be characterized by the extended CIGAR operations that are *visible* in it. A CIGAR operation is said to be *visible* in a given space if it "runs along it", that is, if it's associated with a block of contiguous positions in that space (the size of the block being the length of the operation). For example, the M/=/X operations are *visible* in all spaces, the D/N operations are *visible* in the `"reference"` space but not in the `"query"` space, the S operation is *visible* in the `"query"` space but not in the `"reference"` or in the `"query-after-soft-clipping"` space, etc...

Here are the extended CIGAR operations that are *visible* in each space:

1. reference: M, D, N, =, X
2. reference-N-regions-removed: M, D, =, X
3. query: M, I, S, =, X
4. query-before-hard-clipping: M, I, S, H, =, X
5. query-after-soft-clipping: M, I, =, X
6. pairwise: M, I, D, N, =, X
7. pairwise-N-regions-removed: M, I, D, =, X
8. pairwise-dense: M, =, X

`sequenceLayer` lays a string that belongs to one space alongside another by (1) removing the substrings associated with operations that are not *visible* anymore in the new space, and (2) injecting substrings associated with operations that become *visible* in the new space. Each injected substring has the length of the operation associated with it, and its content is controlled via the corresponding `*.letter` argument.

For example, when going from the `"query"` space to the `"reference"` space (the default), the I- and S-substrings (i.e. the substrings associated with I/S operations) are removed, and substrings associated with D/N operations are injected. More precisely, the D-substrings are filled with the letter specified in `D.letter`, and the N-substrings with the letter specified in `N.letter`. The other `*.letter` arguments are ignored in that case.

Value

An XStringSet object of the same class and length as `x`.

Author(s)

H. Pages

See Also

- The `stackStringsFromBam` function for stacking the read sequences (or their quality strings) stored in a BAM file on a region of interest.
- The `readGAlignmentsFromBam` function for loading read sequences from a BAM file (via a GAlignments object).
- The `extractAt` and `replaceAt` functions in the **Biostrings** package for extracting/replacing arbitrary substrings from/in a string or set of strings.
- cigar-utils in the **GenomicRanges** package for the CIGAR utility functions used internally by `sequenceLayer`.

Examples

```
## ---------------------------------------------------------------------
## A. FROM "query" TO "reference" SPACE
## ---------------------------------------------------------------------

## Load read sequences from a BAM file (they will be returned in a
## GAlignments object):
bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools")
param <- ScanBamParam(what="seq")
gal <- readGAlignmentsFromBam(bamfile, param=param)
qseq <- mcols(gal)$seq  # the read sequences (aka query sequences)

## Lay the query sequences alongside the reference space. This will
## remove the substrings associated with insertions to the reference
## (I operations) and soft clipping (S operations), and will inject new
## substrings (filled with "-") where deletions from the reference (D
## operations) and skipped regions from the reference (N operations)
## occurred during the alignment process:
qseq_on_ref <- sequenceLayer(qseq, cigar(gal))

## A typical use case for doing the above is to compute 1 consensus
## sequence per chromosome. The code below shows how this can be done
## in 2 extra steps.

## Step 1: Compute one consensus matrix per chromosome.
qseq_on_ref_by_chrom <- splitAsList(qseq_on_ref, seqnames(gal))
pos_by_chrom <- splitAsList(start(gal), seqnames(gal))
```

```
cm_by_chrom <- lapply(names(pos_by_chrom),
    function(seqname)
        consensusMatrix(qseq_on_ref_by_chrom[[seqname]],
                        as.prob=TRUE,
                        shift=pos_by_chrom[[seqname]]-1,
                        width=seqlengths(gal)[[seqname]]))
names(cm_by_chrom) <- names(pos_by_chrom)

## cm_by_chrom is a list of consensus matrices. Each matrix has 17
## rows (1 per letter in the DNA alphabet) and 1 column per chromosome
## position.

## Step 2: Compute the consensus string from each consensus matrix.
## Well put "+" in the strings wherever there is no coverage for that
## position, and "N" where there is coverage but no consensus.
cs_by_chrom <- lapply(cm_by_chrom,
    function(cm) {
        ## Because consensusString() doesnt like consensus matrices
        ## with columns that contain only zeroes (and you will have
        ## columns like that for chromosome positions that dont
        ## receive any coverage), we need to "fix" cm first.
        idx <- colSums(cm) == 0
        cm["+", idx] <- 1
        DNAString(consensusString(cm, ambiguityMap="N"))
    })

## consensusString() provides some flexibility to let you extract
## the consensus in different ways. See ?consensusString in the
## Biostrings package for the details.

## Finally, note that the read quality strings can also be used as
## input for sequenceLayer():
param <- ScanBamParam(what="qual")
gal <- readGAlignmentsFromBam(bamfile, param=param)
qual <- mcols(gal)$qual  # the read quality strings
qual_on_ref <- sequenceLayer(qual, cigar(gal))
## Note that since the "-" letter is a valid quality code, there is
## no way to distinguish it from the "-" letters inserted by
## sequenceLayer().

## ---------------------------------------------------------------------
## B. FROM "query" TO "query-after-soft-clipping" SPACE
## ---------------------------------------------------------------------

## Going from "query" to "query-after-soft-clipping" simply removes
## the substrings associated with soft clipping (S operations):
qseq <- DNAStringSet(c("AAAGTTCGAA", "TTACGATTAN", "GGATAATTTT"))
cigar <- c("3H10M", "2S7M1S2H", "2M1I1M3D2M4S")
clipped_qseq <- sequenceLayer(qseq, cigar,
                              from="query", to="query-after-soft-clipping")

sequenceLayer(clipped_qseq, cigar,
              from="query-after-soft-clipping", to="query")

sequenceLayer(clipped_qseq, cigar,
              from="query-after-soft-clipping", to="query",
              S.letter="-")

## ---------------------------------------------------------------------
## C. BRING QUERY AND REFERENCE SEQUENCES TO THE "pairwise" or
##    "pairwise-dense" SPACE
## ---------------------------------------------------------------------

## Load read sequences from a BAM file:
library(RNAseqData.HNRNPC.bam.chr14)
bamfile <- RNAseqData.HNRNPC.bam.chr14_BAMFILES[1]
param <- ScanBamParam(what="seq",
                      which=GRanges("chr14", IRanges(1, 25000000)))
gal <- readGAlignmentsFromBam(bamfile, param=param)
qseq <- mcols(gal)$seq  # the read sequences (aka query sequences)

## Load the corresponding reference sequences from the appropriate
## BSgenome package (the reads in RNAseqData.HNRNPC.bam.chr14 were
## aligned to hg19):
library(BSgenome.Hsapiens.UCSC.hg19)
rseq <- getSeq(Hsapiens, as(gal, "GRanges"))  # the reference sequences

## Bring qseq and rseq to the "pairwise" space.
## For qseq, this will remove the substrings associated with soft
## clipping (S operations) and inject substrings (filled with "-")
## associated with deletions from the reference (D operations) and
## skipped regions from the reference (N operations). For rseq, this
## will inject substrings (filled with "-") associated with insertions
## to the reference (I operations).
qseq2 <- sequenceLayer(qseq, cigar(gal),
                       from="query", to="pairwise")
rseq2 <- sequenceLayer(rseq, cigar(gal),
                       from="reference", to="pairwise")

## Sanity check: qseq2 and rseq2 should have the same shape.
stopifnot(identical(elementLengths(qseq2), elementLengths(rseq2)))

## A closer look at reads with insertions and deletions:
cigar_op_table <- cigarOpTable(cigar(gal))
head(cigar_op_table)

I_idx <- which(cigar_op_table[ , "I"] >= 2)  # at least 2 insertions
qseq2[I_idx]
rseq2[I_idx]

D_idx <- which(cigar_op_table[ , "D"] >= 2)  # at least 2 deletions
qseq2[D_idx]
rseq2[D_idx]

## A closer look at reads with skipped regions:
```

```
N_idx <- which(cigar_op_table[ , "N"] != 0)
qseq2[N_idx]
rseq2[N_idx]

## A variant of the "pairwise" space is the "pairwise-dense" space.
## In that space, all indels and skipped regions are removed from qseq
## and rseq.
qseq3 <- sequenceLayer(qseq, cigar(gal),
                       from="query", to="pairwise-dense")
rseq3 <- sequenceLayer(rseq, cigar(gal),
                       from="reference", to="pairwise-dense")

## Sanity check: qseq3 and rseq3 should have the same shape.
stopifnot(identical(elementLengths(qseq3), elementLengths(rseq3)))

## Insertions were removed:
qseq3[I_idx]
rseq3[I_idx]

## Deletions were removed:
qseq3[D_idx]
rseq3[D_idx]

## Skipped regions were removed:
qseq3[N_idx]
rseq3[N_idx]

## ---------------------------------------------------------------------
## D. SANITY CHECKS
## ---------------------------------------------------------------------
SPACES <- c("reference",
            "reference-N-regions-removed",
            "query",
            "query-before-hard-clipping",
            "query-after-soft-clipping",
            "pairwise",
            "pairwise-N-regions-removed",
            "pairwise-dense")

cigarWidth <- list(
    function(cigar) cigarWidthAlongReferenceSpace(cigar),
    function(cigar) cigarWidthAlongReferenceSpace(cigar,
                                                  N.regions.removed=TRUE),
    function(cigar) cigarWidthAlongQuerySpace(cigar),
    function(cigar) cigarWidthAlongQuerySpace(cigar,
                                              before.hard.clipping=TRUE),
    function(cigar) cigarWidthAlongQuerySpace(cigar,
                                              after.soft.clipping=TRUE),
    function(cigar) cigarWidthAlongPairwiseSpace(cigar),
    function(cigar) cigarWidthAlongPairwiseSpace(cigar,
                                                 N.regions.removed=TRUE),
    function(cigar) cigarWidthAlongPairwiseSpace(cigar, dense=TRUE)
)
```

```
cigar <- c("3H2S4M1D2M2I1M5N3M6H", "5M1I3M2D4M2S")

seq <- list(
    BStringSet(c(A="AAAA-BBC+++++DDD", B="AAAAABBB--CCCC")),
    BStringSet(c(A="AAAA-BBCDDD", B="AAAAABBB--CCCC")),
    BStringSet(c(A="..AAAABBiiCDDD", B="AAAAAiBBBCCCC..")),
    BStringSet(c(A=".....AAAABBiiCDDD......", B="AAAAAiBBBCCCC..")),
    BStringSet(c(A="AAAABBiiCDDD", B="AAAAAiBBBCCCC")),
    BStringSet(c(A="AAAA-BBiiC+++++DDD", B="AAAAAiBBB--CCCC")),
    BStringSet(c(A="AAAA-BBiiCDDD", B="AAAAAiBBB--CCCC")),
    BStringSet(c(A="AAAABBCDDD", B="AAAAABBBCCCC"))
)

stopifnot(all(sapply(1:8,
    function(i) identical(width(seq[[i]]), cigarWidth[[i]](cigar))
)))

sequenceLayer2 <- function(x, cigar, from, to)
    sequenceLayer(x, cigar, from=from, to=to, D.letter="-", N.letter= "+",
                  I.letter="i", S.letter=".", H.letter=".")

identical_XStringSet <- function(target, current)
{
    ok1 <- identical(class(target), class(current))
    ok2 <- identical(names(target), names(current))
    ok3 <- all(target == current)
    ok1 && ok2 && ok3
}

res <- sapply(1:8, function(i) {
           sapply(1:8, function(j) {
               target <- seq[[j]]
               current <- sequenceLayer2(seq[[i]], cigar,
                                         from=SPACES[i], to=SPACES[j])
               identical_XStringSet(target, current)
           })
       })
stopifnot(all(res))
```

`stackStringsFromBam` *Stack the read sequences stored in a BAM file on a region of interest*

Description

`stackStringsFromBam` lays the read sequences (or their quality strings) stored in a BAM file along-side the reference space, and stacks them on the specified region.

Usage

```
stackStringsFromBam(file, index=file, param,
                    what="seq", use.names=FALSE,
                    D.letter="-", N.letter="-",
                    Lpadding.letter="+", Rpadding.letter="+")
```

Arguments

`file, index`	The path to the BAM file to read, and to the index file of the BAM file to read, respectively. The latter is given *without* the '.bai' extension. See `scanBam` for more information.
`param`	A ScanBamParam object containing exactly 1 genomic region (i.e. `unlist(bamWhich(param))` must have length 1). Alternatively, `param` can be a GRanges or RangesList object containing exactly 1 genomic region, or a character string specifying a single genomic region (in the `"chr14:5201-5300"` format).
`what`	A single string. Either `"seq"` or `"qual"`. If `"seq"` (the default), the read sequences will be stacked. If `"qual"`, the read quality strings will be stacked.
`use.names`	Use the query template names (QNAME field) as the names of the returned object? If not (the default), then the returned object has no names.
`D.letter, N.letter`	A single letter used as a filler for injections. The 2 arguments are passed down to the `sequenceLayer` function. See `?sequenceLayer` for more details.
`Lpadding.letter, Rpadding.letter`	A single letter to use for padding the sequences on the left, and another one to use for padding on the right. The 2 arguments are passed down to the `stackStrings` function defined in the **Biostrings** package. See `?stackStrings` in the **Biostrings** package for more details.

Details

`stackStringsFromBam` performs the 3 following steps:

1. Load the read sequences (or their quality strings) from the BAM file. Only the read sequences that overlap with the specified region are loaded. This is done with the `readGAlignmentsFromBam` function. Note that if the file contains paired-end reads, the pairing is ignored.
2. Lay the sequences alongside the reference space, using their CIGARs. This is done with the `sequenceLayer` function.
3. Stack them on the specified region. This is done with the `stackStrings` function defined in the **Biostrings** package.

Value

A rectangular (i.e. constant-width) DNAStringSet object (if `what` is `"seq"`) or BStringSet object (if `what` is `"qual"`).

Note

TWO IMPORTANT CAVEATS:

Specifying a big genomic region, say >= 100000 bp, can require a lot of memory (especially with high coverage reads) and is not recommended.

Paired-end reads are treated as single-end reads (i.e. they're not paired).

Author(s)

H. Pages

See Also

- The `readGAlignmentsFromBam` function for loading read sequences (or their quality strings) from a BAM file (via a GAlignments object).
- The `sequenceLayer` function for laying read sequences alongside the reference space, using their CIGARs.
- The `stackStrings` function in the **Biostrings** package for stacking an arbitrary XStringSet object.
- The SAMtools mpileup command available at `http://samtools.sourceforge.net/` as part of the SAMtools project.

Examples

```
## ---------------------------------------------------------------------
## A. EXAMPLE WITH TOY DATA
## ---------------------------------------------------------------------

bamfile <- BamFile(system.file("extdata", "ex1.bam", package="Rsamtools"))

stackStringsFromBam(bamfile, param=GRanges("seq1", IRanges(1, 60)))

options(showHeadLines=18)
options(showTailLines=6)
stackStringsFromBam(bamfile, param=GRanges("seq1", IRanges(61, 120)))

stacked_qseq <- stackStringsFromBam(bamfile, param="seq2:1509-1519")
stacked_qseq  # deletion in read 13

stackStringsFromBam(bamfile, param="seq2:1509-1519", what="qual")
consensusMatrix(stacked_qseq)

## ---------------------------------------------------------------------
## B. EXAMPLE WITH REAL DATA
## ---------------------------------------------------------------------

library(RNAseqData.HNRNPC.bam.chr14)
bamfile <- BamFile(RNAseqData.HNRNPC.bam.chr14_BAMFILES[1])

## My Region Of Interest:
```

```
my_ROI <- GRanges("chr14", IRanges(19650095, 19650159))

readGAlignments(bamfile, param=ScanBamParam(which=my_ROI))
stackStringsFromBam(bamfile, param=my_ROI)
```

`TabixFile` *Manipulate tabix indexed tab-delimited files.*

Description

Use `TabixFile()` to create a reference to a Tabix file (and its index). Once opened, the reference remains open across calls to methods, avoiding costly index re-loading.

`TabixFileList()` provides a convenient way of managing a list of `TabixFile` instances.

Usage

```
## Constructors

TabixFile(file, index = paste(file, "tbi", sep="."), ...,
          yieldSize=NA_integer_)
TabixFileList(..., yieldSize=NA_integer_)

## Opening / closing

## S3 method for class TabixFile
open(con, ...)
## S3 method for class TabixFile
close(con, ...)

## accessors; also path(), index(), yieldSize()

## S4 method for signature TabixFile
isOpen(con, rw="")

## actions

## S4 method for signature TabixFile
seqnamesTabix(file, ...)
## S4 method for signature TabixFile
headerTabix(file, ...)
## S4 method for signature TabixFile,GRanges
scanTabix(file, ..., param)
## S4 method for signature TabixFile,RangesList
scanTabix(file, ..., param)
## S4 method for signature TabixFile,RangedData
scanTabix(file, ..., param)
```

```
## S4 method for signature TabixFile,missing
scanTabix(file, ..., param)
## S4 method for signature character,ANY
scanTabix(file, ..., param)
## S4 method for signature character,missing
scanTabix(file, ..., param)

countTabix(file, ...)
```

Arguments

`con`	An instance of `TabixFile`.
`file`	For TabixFile(), A character(1) vector to the tabix file path; can be remote (http://, ftp://). For `countTabix`, a character(1) or `TabixFile` instance. For others, a `TabixFile` instance.
`index`	A character(1) vector of the tabix file index.
`yieldSize`	Number of records to yield each time the file is read from using `scanTabix`. Only valid when `param` is unspecified. `yieldSize` does not alter existing yield sizes, include `NA`, when creating a `TabixFileList` from `TabixFile` instances.
`param`	An instance of GRanges, IRangedData, or RangesList, used to select which records to scan.
`...`	Additional arguments. For `TabixFileList`, this can either be a single character vector of paths to tabix files, or several instances of `TabixFile` objects.
`rw`	character() indicating mode of file; not used for `TabixFile`.

Objects from the Class

Objects are created by calls of the form `TabixFile()`.

Fields

The `TabixFile` class inherits fields from the `RsamtoolsFile` class.

Functions and methods

`TabixFileList` inherits methods from `RsamtoolsFileList` and `SimpleList`.

Opening / closing:

open.TabixFile Opens the (local or remote) `path` and `index`. Returns a `TabixFile` instance. `yieldSize` determines the number of records parsed during each call to `scanTabix`; `NA` indicates that all records are to be parsed.

close.TabixFile Closes the `TabixFile` `con`; returning (invisibly) the updated `TabixFile`. The instance may be re-opened with `open.TabixFile`.

Accessors:

path Returns a character(1) vector of the tabix path name.

index Returns a character(1) vector of tabix index name.

yieldSize, yieldSize<- Return or set an integer(1) vector indicating yield size.

Methods:

seqnamesTabix Visit the path in `path(file)`, returning the sequence names present in the file.

headerTabix Visit the path in `path(file)`, returning the sequence names, column indicies used to sort the file, the number of lines skipped while indexing, the comment character used while indexing, and the header (preceeded by comment character, at start of file) lines.

countTabix Return the number of records in each range of `param`, or the count of all records in the file (when `param` is missing).

scanTabix For `signature(file="TabixFile")`, Visit the path in `path(file)`, returning the result of `scanTabix` applied to the specified path. For `signature(file="character")`, call the corresponding method after coercing `file` to `TabixFile`.

indexTabix This method operates on file paths, rather than `TabixFile` objects, to index tab-separated files. See `indexTabix`.

show Compactly display the object.

Author(s)

Martin Morgan

Examples

```
fl <- system.file("extdata", "example.gtf.gz", package="Rsamtools",
                  mustWork=TRUE)
tbx <- TabixFile(fl)

param <- GRanges(c("chr1", "chr2"), IRanges(c(1, 1), width=100000))
countTabix(tbx)
countTabix(tbx, param=param)
res <- scanTabix(tbx, param=param)
sapply(res, length)
res[["chr1:1-100000"]][1:2]

## parse to list of data.frames
dff <- Map(function(elt) {
    read.csv(textConnection(elt), sep="\t", header=FALSE)
}, res)
dff[["chr1:1-100000"]][1:5,1:8]

## parse 100 records at a time
length(scanTabix(tbx)[[1]]) # total number of records
tbx <- open(TabixFile(fl, yieldSize=100))
while(length(res <- scanTabix(tbx)[[1]]))
   cat("records read:", length(res), "\n")
close(tbx)
```

TabixInput *Operations on 'tabix' (indexed, tab-delimited) files.*

Description

Scan compressed, sorted, tabix-indexed, tab-delimited files.

Usage

```
scanTabix(file, ..., param)
## S4 method for signature character,RangesList
scanTabix(file, ..., param)
## S4 method for signature character,RangedData
scanTabix(file, ..., param)
## S4 method for signature character,GRanges
scanTabix(file, ..., param)
```

Arguments

`file`	The character() file name(s) of the tabix file be processed, or more flexibly an instance of class `TabixFile`.
`param`	A instance of `GRanges`, `RangedData`, or `RangesList` provide the sequence names and regions to be parsed.
`...`	Additional arguments, currently ignored.

Value

`scanTabix` returns a list, with one element per region. Each element of the list is a character vector representing records in the region.

Error

`scanTabix` signals errors using `signalCondition`. The following errors are signaled:

`scanTabix_param` `yieldSize(file)` must be NA when more than one range is specified.

`scanTabix_io` A read error occured while inputing the tabix file. This might be because the file is corrupt, or of incorrect format (e.g., when `path` points to a plain text file but `index` is present, implying that `path` should be a `bgzip`ed file.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

References

http://samtools.sourceforge.net/tabix.shtml

Examples

```
example(TabixFile)
```

Index