Rsamtools

March 24, 2012

`BamFile` *Maintain SAM and 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)
BamFileList(...)

## Opening / closing

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

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

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

## actions

## S4 method for signature 'BamFile'
scanBamHeader(files, ...)
## S4 method for signature 'BamFile'
seqinfo(x)
## S4 method for signature 'BamFile'
scanBam(file, index=file, ..., param=ScanBamParam(what=scanBamWhat()))
## 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'
filterBam(file, destination, index=file, ...,
    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 'BamFile'
readBamGappedAlignments(file, index=file, use.names=FALSE, param=NULL)
## S4 method for signature 'BamFile'
readBamGappedReads(file, index=file, use.names=FALSE, param=NULL)

## counting
## S4 method for signature 'GRanges,BamFileList'
summarizeOverlaps(
    features, reads, mode, ignore.strand = FALSE, ..., param = ScanBamParam())
```

Arguments

`...`	Additional arguments. For `BamFileList`, this can either be a single character vector of paths to BAM files, or several instances of `BamFile` objects.
`con`	An instance of `BamFile`.
`x, file, files`	A character vector of BAM file paths (for `BamFile`) or a `BamFile` instance (for other methods).
`index`	A character vector of indices (for `BamFile`); ignored for all other methods on this page.
`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.
`rw`	Mode of file; ignored.
`reads`	A BamFileList that represents the data to be counted by summarizeOverlaps.
`features`	A GRanges or a GRangesList object of genomic regions of interest. When a GRanges is supplied, each row is considered a feature. When a GRangesList is supplied, each higher list-level is considered a feature. This distinction is important when defining an overlap between a read and a feature. See examples for details.
`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.

Objects from the Class

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

Fields

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

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`.

Accessors:

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

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

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.

readBamGappedAlignments, readBamGappedReads Visit the path in `path(file)`, returning the result of `readBamGappedAlignments` or `readBamGappedReads` applied to the specified path. See `readBamGappedAlignments`.

show Compactly display the object.

Author(s)

Martin Morgan and Marc Carlson

See Also

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

Examples

```
fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
bf <- open(BamFile(fl))        # implicit index
bf
identical(scanBam(bf), scanBam(fl))

rng <- GRanges(c("seq1", "seq2"), IRanges(1, c(1575, 1584)))

## repeatedly visit 'bf'
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)


##---------------------------------------------------------------------------##
## How to use summarizeOverlaps with a BamFileList object.
fls = list.files(system.file("extdata",package="GenomicRanges"),
                 recursive=TRUE, pattern="*bam$", full=TRUE)
bfs <- BamFileList(fls)

## "features" will be the argument for an "annotations" object (GRanges
## or GRangesList object.
group_id <- c("A", "B", "C", "C", "D", "D", "E", "F", "G", "H", "H")
features <- GRanges(
    seqnames = Rle(c("chr2L", "chr2R", "chr2L", "chr2R", "chr2L", "chr2R",
        "chr2L", "chr2R", "chr2R", "chr3L", "chr3L")),
    strand = strand(rep("+", length(group_id))),
    ranges = IRanges(
        start=c(1000, 2000, 3000, 3600, 7000, 7500, 4000, 4000, 3000, 5000, 5400),
        width=c(500, 900, 500, 300, 600, 300, 500, 900, 500, 500, 500)),
   DataFrame(group_id)
)
## Then call the method:
summarizeOverlaps(features, bfs, mode = Union, ignore.strand=TRUE)
```

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'
readBamGappedAlignments(file, index=file, use.names=FALSE, param=NULL)

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

## Counting
## S4 method for signature 'GRanges,BamViews'
summarizeOverlaps(
    features, reads, mode, ignore.strand = 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.
reads	A BamFileList that represents the data to be counted by summarizeOverlaps.
features	A GRanges or a GRangesList object of genomic regions of interest. When a GRanges is supplied, each row is considered a feature. When a GRangesList is supplied, each higher list-level is considered a feature. This distinction is important when defining an overlap between a read and a feature. See examples for details.

`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.

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).

readBamGappedAlignments Visit each path in bamPaths(file), returning the result of readBamGappedAlign 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 elementMetadata of the returned SimpleList.

show Compactly display the object.

Author(s)

Martin Morgan

See Also

readBamGappedAlignments. The GenomicRanges package is where the summarizeOverlaps method originates.

Examples

```
fls <- list.files(system.file("extdata", package="Rsamtools"),
                  "\\.bam$", full=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 <- readBamGappedAlignments(bv)
aln
```

```
aln[[1]]
aln[colnames(bv)]
elementMetadata(aln)

##------------------------------------------------------------------------------##
## How to use summarizeOverlaps with a BamViews object.
fls = list.files(system.file("extdata",package="GenomicRanges"),
                 recursive=TRUE, pattern="*bam$", full=TRUE)
bfs <- BamViews(fls)

## "features" will be the argument for an "annotations" object (GRanges
## or GRangesList object.
group_id <- c("A", "B", "C", "C", "D", "D", "E", "F", "G", "H", "H")
features <- GRanges(
    seqnames = Rle(c("chr2L", "chr2R", "chr2L", "chr2R", "chr2L", "chr2R",
        "chr2L", "chr2R", "chr2R", "chr3L", "chr3L")),
    strand = strand(rep("+", length(group_id))),
    ranges = IRanges(
        start=c(1000, 2000, 3000, 3600, 7000, 7500, 4000, 4000, 3000, 5000, 5400),
        width=c(500, 900, 500, 300, 600, 300, 500, 900, 500, 500, 500)),
   DataFrame(group_id)
)
## Then call the method:
summarizeOverlaps(features, bfs, mode = Union, ignore.strand=TRUE)
```

`BcfFile` *Manipulate BCF or VCF files.*

Description

Use `BcfFile()` to create a reference to a BCF (and optionally its index) or VCF file. 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 VCF or 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 VCF / 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")
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)
```

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 '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 `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.

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.

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")
fa <- open(FaFile(fl))                    # open
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]))
```

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)
```

`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

http://samtools.sourceforge.net/

Examples

```
packageDescription('Rsamtools')
```

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'
show(object)
```

Arguments

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

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.

Functions and methods

Accessors:

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

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

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, ...)
```

Arguments

`con, object`	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.

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,
    isValidVendorRead = NA, isDuplicate = 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)
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) vector which, when 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.
`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 reads that are primary (FALSE), are not primary (TRUE) or whose primary status does not matter (NA) should be returned. A non-primary read might result when portions of a read aligns to multiple locations, e.g., when spanning splice junctions).

`isValidVendorRead`
: A logical(1) indicating whether invalid (FALSE), valid (TRUE), or any (NA) read should be returned. A 'valid' read is one flagged by the vendor as passing quality control criteria.

`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`).

`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")
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])
```

`ScanBcfParam-class` *Parameters for scanning VCF / BCF files*

Description

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

Usage

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

ScanVcfParam(info=character(), geno=character(), trimEmpty=TRUE,
             which, ...)

## Accessors
bcfInfo(object)
bcfGeno(object)
bcfTrimEmpty(object)
bcfWhich(object)

vcfInfo(object)
vcfGeno(object)
vcfTrimEmpty(object)
vcfWhich(object)
```

Arguments

info	A character() vector of 'INFO' fields (see scanVcfHeader) to be returned. Not currently implemented.
geno	A character() vector of 'GENO' fields (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. Methods defined for ScanBcfParam are available for ScanVcfParam.
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.

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

Functionas and methods

See 'Usage' for details on invocation.

Constructor:

ScanVcfParam, ScanBcfParam: Returns a ScanVcfParam or 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

Examples

```
p0 <- ScanVcfParam()

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

## return only specified GENO field(s)
p2 <- ScanVcfParam(geno="GT", which=which)
```

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="."), ...)
TabixFileList(...)

## Opening / closing

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

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

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

```
## S4 method for signature 'TabixFile,GRanges'
scanTabix(file, ..., param)

## S4 method for signature 'TabixFile'
yieldTabix(file, ..., yieldSize=1000000L)

## S4 method for signature 'TabixFile'
scanVcfHeader(file, ...)
## S4 method for signature 'TabixFile,RangesList'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,RangedData'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,GRanges'
scanVcf(file, ..., param)
## S4 method for signature 'TabixFile,ScanVcfParam'
scanVcf(file, ..., param)
```

Arguments

con	An instance of TabixFile.
file	For TabixFile(), A character(1) vector to the tabix file path; can be remote (http://, ftp://). For others, a TabixFile instance.
index	A character(1) vector of the tabix file index.
param	An instance of GRanges, IRangedData, or RangesList, used to select which records to scan.
yieldSize	integer(1) indicating the maximum number of records to retreieve.
...	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.

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.

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.

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

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

scanVcfHeader see `scanVcfHeader`

scanVcf see `scanVcf`

show Compactly display the object.

Author(s)

Martin Morgan

Examples

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

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

## parse 100 records at a time
tbx <- open(TabixFile(fl))
while(length(res <- yieldTabix(tbx, yieldSize=100L)))
   cat("records read:", length(res), "\n")
close(tbx)
```

---

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.

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")

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"),]
        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")
```

`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")
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")
to <- tempfile()
zipped <- bgzip(from, to)
idx <- indexTabix(zipped, "sam")

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

---

readBamGappedAlignments

Reading GappedAlignments or GappedReads objects from a BAM file

---

Description

Read a BAM file as a GappedAlignments or GappedReads object.

Usage

```
readBamGappedAlignments(file, index=file, use.names=FALSE, param=NULL)
readBamGappedReads(file, index=file, use.names=FALSE, param=NULL)
```

Arguments

file	The character(1) file name of the 'BAM' file to be processed.
index	The character(1) name of the index file of the 'BAM' file being processed; this is given *without* the '.bai' extension.
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, please note that the fields specified thru this ScanBamParam object will be loaded *in addition* to any field required for generating the returned object (GappedAlignments or GappedReads object) and will be stored in its elementMetadata part. By default (i.e. param=NULL), no additional field is loaded and the flag used is scanBamFlag(isUnmappedQuery=FALSE, isDuplicate=FALSE) (i.e. all the records corresponding to mapped reads that are not PCR or optical duplicates are loaded).

Details

See ?GappedAlignments-class for a description of GappedAlignments objects.

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

See ?scanBam for a description of the arguments.

Value

A GappedAlignments object for `readBamGappedAlignments`.

A GappedReads object for `readBamGappedReads`.

Note

BAM records corresponding to unmapped reads or to reads that are PCR or optical duplicates are always ignored.

Author(s)

H. Pages

See Also

GappedAlignments-class, GappedReads-class, `scanBam`, `ScanBamParam`

Examples

```
## Simple use:
bamfile <- system.file("extdata", "ex1.bam", package="Rsamtools")
galn1 <- readBamGappedAlignments(bamfile)
galn1
names(galn1)

## Using the 'use.names' arg:
galn2 <- readBamGappedAlignments(bamfile, use.names=TRUE)
galn2
head(names(galn2))

## Using the 'param' arg to load additional BAM fields:
param <- ScanBamParam(what=c("qual", "flag"))
galn3 <- readBamGappedAlignments(bamfile, param=param)
galn3
elementMetadata(galn3)

## Using the 'param' arg to load reads from particular regions.
## Note that if we weren't 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)
galn4 <- readBamGappedAlignments(bamfile, param=param)
galn4

## Note that a given record is loaded one time for each region it
## belongs to (this is a scanBam() feature, readBamGappedAlignments()
## is based on scanBam()):
which <- IRangesList(seq2=IRanges(c(1563, 1567), width=1))
param <- ScanBamParam(which=which)
galn5 <- readBamGappedAlignments(bamfile, param=param)
galn5

## 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")
galn6 <- readBamGappedAlignments(bamfile, param=param)
elementMetadata(galn6)  # "tag" cols always after "what" cols

## readBamGappedReads():
greads1 <- readBamGappedReads(bamfile)
greads1
names(greads1)
qseq(greads1)
greads2 <- readBamGappedReads(bamfile, use.names=TRUE)
greads2
head(names(greads2))
```

`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")
(res <- readPileup(fl))
xtabs(~referenceBase + consensusBase, elementMetadata(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)
```

BamInput *Import, count, index, and other operations on 'BAM' (binary alignment) files.*

Description

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

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, ...,
    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, ...)
```

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.
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 or filtered output file will be created. For asBam and sortBam this is without the ".bam" file suffix.
...	Additional arguments, passed to methods.
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.
byQname	A logical(1) indicating whether the sorted destination file should be sorted by Query-name (TRUE) or by mapping position (FALSE).
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` satisfying the `bamWhich` of `param`, writing each record satisfying the `bamFlag` and `bamSimpleCigar` criteria of `param` to file `destination`. An index file is created on the destination when `indexDestination=TRUE`.

`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.

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`.

`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")

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]])

sorted <- sortBam(fl, tempfile())
```

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

VcfInput *Operations on 'VCF' or 'BCF' (variant call) 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, ...)

scanVcfHeader(file, ...)
## S4 method for signature 'character'
scanVcfHeader(file, ...)

scanVcf(file, ..., param)
## S4 method for signature 'character,ANY'
scanVcf(file, ..., param)
## S4 method for signature 'character,missing'
scanVcf(file, ..., param)
## S4 method for signature 'connection,missing'
scanVcf(file, ..., param)

unpackVcf(x, hdr, ..., info=TRUE, geno=TRUE)
## S4 method for signature 'list,missing'
unpackVcf(x, hdr, ..., info=TRUE, geno=TRUE)
```

```
## S4 method for signature 'list,character'
unpackVcf(x, hdr, ..., info=TRUE, geno=TRUE)
## S4 method for signature 'list,TabixFile'
unpackVcf(x, hdr, ..., info=TRUE, geno=TRUE)
```

Arguments

file	For scanBcf and scanBcfHeader, the character() file name of the 'VCF' or 'BCF' file to be processed, or an instance of class BcfFile. For scanVcf and scanVcfHeader, the character() file name, TabixFile, or class connection (file() or bgzip()) of the 'VCF' file to be processed.
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 or ScanVcfParam 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.
x	A list() resulting from scanVcf.
hdr	A character(1) or TabixFile instance from which scanBamHeader can extract information on the structure of INFO and FORMAT specifications.
info, geno	For non-"missing" methods of unpackVcf, a logical(1) indicating whether the 'INFO' or 'GENO' fields of x should be expanded. If TRUE, then scanVcfHeader(hdr) is consulted for the description of INFO and / or FORMAT fields. For the "missing" method of unpackVcf, a logical(1) (in which case the corresponding field is not unpacked, regardless of value) or DataFrame or data.frame with row names corresponding to field elements, and with columns Number and Type as defined in the VCF specification at the URL below. Usually, these are obtained from scanVcfHeader on the same file as used to parse the data passed as argument x.

Details

Most users will use the vcf* functions; bcf* 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.

scanVcf with param="missing" and file="character" or file="connection" scan the entire file. With file="connection", an argument n indicates the number of lines of the VCF file to input; a connection open at the beginning of the call is open and incremented by n lines at the end of the call, providing a convenient way to stream through large VCF files.

The INFO field of the scanned VCF file is returned as a single 'packed' vector, as in the VCF file. The GENO field is returned as a list of matricies, each matrix corresponds to a field as defined in the FORMAT field of the VCF header. Each matrix has as many rows as scanned in the VCF file,

and as many columns as there are samples. As with the INFO field, the elements of the matrix are 'packed'. The reason that INFO and GENO are returned packed is to facilitate manipulation, e.g., selecting particular rows or samples in a consistent manner across elements.

`unpackVcf` processes the INFO and / or GENO fields, typically using the information encoded in the header and extracted by consulting `scanVcfHeader`. When the INFO or FORMAT specification includes a field Number. When this is an integer value, the corresponding INFO or GENO is unpacked as a matrix or array. For fields with variable numbers of elements ('A', 'G', '.'), the unpacked data is a list of vectors (for INFO) or list of list of vectors (for GENO), with the outer list corresponding to rows in the scanned VCF, the inner list of GENO corresponding to samples, and the inner vector corresponding to sub-elements of the element.

Value

`scanVcfHeader` / `scanBcfHeader` returns a list, with one element for each file named in `file`. Each element of the list is itself a list containing three element. 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.

`scanVcf` / `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 those defined in the VCF file header. For `scanVcf`, elements of GENO are returned as a matrix of records x samples; if the description of the element in the file header indicated multiplicity other than 1 (e.g., variable number for "A", "G", or "."), then each entry in the matrix is a character string with sub-entries comma-delimited.

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

`indexBcf` creates an index into the BCF file.

`unpackVcf` returns a list of the same form as `scanVcf`, but with INFO and / or GENO elements unpacked to matrix or list elements as appropriate.

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")
scanBcfHeader(fl)
bcf <- scanBcf(fl)
```

```
## value: list-of-lists
str(bcf[1:8])
names(bcf[["GENO"]])
str(head(bcf[["GENO"]][["PL"]]))
example(BcfFile)
```

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")
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]))
```

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.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

References

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

Examples

```
example(TabixFile)
```

---

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")
seqnamesTabix(fl)
```

`yieldTabix` *Yield records from a stream of records contained in a tabix file.*

Description

Return the next set of records from an already opened file.

Usage

```
yieldTabix(file, ..., yieldSize=1000000L)
```

Arguments

`file`	An opened instance of a type for which a `yield` method exists.
`yieldSize`	The number of records to return on each invocation.
`...`	Additional arguments, currently ignored.

Value

`yield` returns the next `yieldSize` records from `file`, in a format defined by the method.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>.

Examples

```
showMethods(yieldTabix)
example(TabixFile)
```

`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")
to <- tempfile()
zipped <- bgzip(from, to)
```

Index