Package 'GenomicFiles'

October 17, 2025

Title Distributed computing by file or by range

Version 1.45.2

Description This package provides infrastructure for parallel computations distributed 'by file' or 'by range'. User defined MAPPER and REDUCER functions provide added flexibility for data combination and manipulation.

License Artistic-2.0

Depends BiocGenerics, BiocParallel, GenomicRanges, MatrixGenerics, methods, Rsamtools (>= 2.25.1), rtracklayer (>= 1.69.1), SummarizedExperiment (>= 1.39.1)

Imports BiocBaseUtils, GenomeInfoDb (>= 1.45.7), GenomicAlignments (>= 1.45.1), IRanges, S4Vectors, Seqinfo, VariantAnnotation (>= 1.55.1)

Suggests BiocStyle, Biostrings, deepSNV, genefilter, Homo.sapiens, knitr, RNAseqData.HNRNPC.bam.chr14, RUnit, snpStats

VignetteBuilder knitr

biocViews Genetics, Infrastructure, DataImport, Sequencing, Coverage

RoxygenNote 6.1.0

URL https://github.com/Bioconductor/GenomicFiles

BugReports https://github.com/Bioconductor/GenomicFiles/issues

Video https://www.youtube.com/watch?v=3PK_jx44QTs

Collate 'GenomicFiles-class.R' 'VcfStack-class.R' 'reduceByFile-methods.R' 'reduceByRange-methods.R' 'reduceFiles.R' 'reduceRanges.R' 'reduceByYield.R' 'pack-methods.R' 'unpack-methods.R' 'registry.R' 'zzz.R'

git_url https://git.bioconductor.org/packages/GenomicFiles

git_branch devel

git_last_commit 53f71f2

git_last_commit_date 2025-07-21

Repository Bioconductor 3.22

Date/Publication 2025-10-16

Author Bioconductor Package Maintainer [aut, cre],
Valerie Obenchain [aut],
Michael Love [aut],
Lori Shepherd [aut],
Martin Morgan [aut],
Sonali Kumari [ctb] (Converted 'GenomicFiles' vignettes from Sweave to RMarkdown / HTML.)

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Contents

`GenomicFiles` *GenomicFiles objects*

Description

The `GenomicFiles` class is a matrix-like container where rows represent ranges of interest and columns represent files. The class is designed for byFile or byRange queries.

Constructor

```
GenomicFiles(rowRanges, files, colData=DataFrame(), metadata=list(), ...):
```

Details

`GenomicFiles` inherits from the `RangedSummarizedExperiment` class in the `SummarizedExperiment` package. Currently, no use is made of the `elementMetadat` and `assays` slots. This may change in the future.

Accessors

In the code below, `x` is a GenomicFiles object.

rowRanges, rowRanges(x) <- value Get or set the rowRanges on `x`. `value` can be a `GRanges` or `GRangesList` representing ranges or indices defined on the spaces (position) of the files.

files(x), files(x) <- value Get or set the files on `x`. `value` can be a character() of file paths or a List of file objects such as BamFile, BigWigFile, FaFile, etc.

colData, colData(x) <- value Get or set the colData on `x`. `value` must be a `DataFrame` instance describing the files. The number of rows must match the number of files. Row names, if present, become the column names of the `GenomicFiles`.

metadata, metadata(x) <- value Get or set the metadata on `x`. `value` must be a SimpleList of arbitrary content describing the overall experiment.

dimnames, dimnames(x) <- value Get or set the row and column names on `x`.

Methods

In the code below, `x` is a GenomicFiles object.

[Subset the object by `fileRange` or `fileSample`.

show Compactly display the object.

reduceByFile Extract, manipulate and combine data defined in `rowRanges` within the files specified in `files`. See ?`reduceByFile` for details.

reduceByRange Extract, manipulate and combine data defined in `rowRanges` across the files specified in `files`. See ?`reduceByRange` for details.

Author(s)

Martin Morgan and Valerie Obenchain

See Also

- reduceByFile and reduceByRange methods.
- SummarizedExperiment objects in the **SummarizedExperiment** package.

Examples

```
## ---------------------------------------------------------------------
## Basic Use
## ---------------------------------------------------------------------

if (require(RNAseqData.HNRNPC.bam.chr14)) {
  fl <- RNAseqData.HNRNPC.bam.chr14_BAMFILES
  rd <- GRanges("chr14",
                IRanges(c(62262735, 63121531, 63980327), width=214700))
  cd <- DataFrame(method=rep("RNASeq", length(fl)),
                  format=rep("bam", length(fl)))

  ## Construct an instance of the class:
  gf <- GenomicFiles(files = fl, rowRanges = rd, colData = cd)
  gf

  ## Subset on ranges or files for different experimental runs.
  dim(gf)
  gf_sub <- gf[2, 3:4]
  dim(gf_sub)

  ## When summarize = TRUE and no REDUCE is provided the reduceBy*
  ## functions output a SummarizedExperiment object.
  MAP <- function(range, file, ...) {
      requireNamespace("GenomicFiles", quietly=TRUE) ## for coverage()
      requireNamespace("Rsamtools", quietly=TRUE)     ## for ScanBamParam()
```

```
        param = Rsamtools::ScanBamParam(which=range)
        GenomicFiles::coverage(file, param=param)[range]
    }
    se <- reduceByRange(gf, MAP=MAP, summarize=TRUE)
    se

    ## Data from the rowRanges, colData and metadata slots in the
    ## GenomicFiles are transferred to the SummarizedExperiment.
    colData(se)

    ## Results are in the assays slot.
    assays(se)
}

## ---------------------------------------------------------------------
## Managing cached or remote files with GenomicFiles
## ---------------------------------------------------------------------

## The GenomicFiles class can manage cached or remote files and their
## associated ranges.

## Not run:
## Files from AnnotationHub can be downloaded and cached locally.
library(AnnotationHub)
hub = AnnotationHub()
hublet = query(hub, c("files I'm", "interested in"))
# cache (if need) and return local path to files
fls = cache(hublet)

## An alternative to the local file paths is to use urls to a remote file.
## This approach could be used with something like rtracklayer::bigWig which
## supports remote file queries.
urls = hublet$sourceurls

## Define ranges of interest and use GenomicFiles to manage.
rngs = GRanges("chr10", IRanges(c(100000, 200000), width=1))
gf = GenomicFiles(rngs, fls)

## As an example, one could create a matrix from data extracted
## across multiple BED files.
MAP = function(rng, fl) {
    requireNamespace("rtracklayer", quietly=TRUE)  ## import, BEDFile
    rtracklayer::import(rtracklayer::BEDFile(fl), which=rng)$name
}
REDUCE = unlist
xx = reduceFiles(gf, MAP=MAP, REDUCE=REDUCE)
mcols(rngs) = simplify2array(xx)

## Data and ranges can be stored in a SummarizedExperiment.
SummarizedExperiment(list(my=simplify2array(xx)), rowRanges=rngs)

## End(Not run)
```

GenomicFiles-deprecated

Deprecated functions in package 'GenomicFiles'

Description

These functions are provided for compatibility with older versions of 'GenomicFiles' only, and will be defunct at the next release.

Details

The following functions are deprecated and will be made defunct; use the replacement indicated below:

- getVCFPath(vs, chrtok): `files(vs)[chrtok]`

`pack` *Range transformations of a* `GenomicRanges` *object for optimal file queries.*

Description

Given a `GRanges` object, `pack` produces a `GRangesList` of the same ranges grouped and re-ordered.

Usage

```
## S4 method for signature 'GRanges'
pack(x, ..., range_len = 1e9, inter_range_len = 1e7)
```

Arguments

`x`	A `GRanges` object.
`range_len`	A numeric specifying the max length allowed for ranges in `x`.
`inter_range_len`	A numeric specifying the max length allowed between ranges in `x`.
`...`	Arguments passed to other methods.

Details

Packing ranges: The `pack` method attempts to re-package ranges in optimal form for extracting data from files. Ranges are not modified (made shorter or longer) but re-ordered and / or re-grouped according to the following criteria.

- order: Ranges are ordered by genomic position within chromosomes.
- distance: Ranges separted by a distance greater than the `inter_range_len` are packed in groups around the gap separating the distant ranges.
- length: Ranges longer than `range_len` are packed 'individually' (i.e., retrived from the file as a single range vs grouped with other ranges).

Utilities:

`isPacked(x, ...)`**:** Returns a logical indicating if the ranges in `x` are packed. `x` must be a `GRangesList` object.

Value

A `GRanges` object.

See Also

- unpack for unpacking the result obtained with 'packed' ranges.

Examples

```
## Ranges are ordered by position within chromosome.
gr1 <- GRanges("chr1", IRanges(5:1*5, width = 3))
pack(gr1)

## Ranges separated by > inter_range_len are partitioned
## into groups defined by the endpoints of the gap.
gr2 <- GRanges("chr2", IRanges(c(1:3, 30000:30003), width = 1000))
pack(gr2, inter_range_len = 20000)

## Ranges exceeding 'range_len' are isolated in a single element
## of the GRangesList.
gr3 <- GRanges("chr3", IRanges(c(1:4), width=c(45, 1e8, 45, 45)))
width(gr3)
pack(gr3, range_len = 1e7)
```

---

reduceByFile *Parallel computations by files*

Description

Computations are distributed in parallel by file. Data subsets are extracted and manipulated (MAP) and optionally combined (REDUCE) within a single file.

Usage

```
## S4 method for signature 'GRanges,ANY'
reduceByFile(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)
## S4 method for signature 'GRangesList,ANY'
reduceByFile(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)
## S4 method for signature 'GenomicFiles,missing'
reduceByFile(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)

reduceFiles(ranges, files, MAP, REDUCE, ..., init)
```

Arguments

ranges	A GRanges, GrangesList or GenomicFiles object. A GRangesList implies a grouping of the ranges; MAP is applied to each element of the GRangesList vs each range when ranges is a GRanges. When ranges is a GenomicFiles the files argument is missing; both ranges and files are extracted from the object.
files	A character vector or List of filenames. A List implies a grouping of the files; MAP is applied to each element of the List vs each file individually.

MAP — A function executed on each worker. The signature must contain a minimum of two arguments representing the ranges and files. There is no restriction on argument names and additional arguments can be provided.

- MAP = function(range, file, ...)

REDUCE — An optional function that combines output from the MAP step. The signature must contain at least one argument representing the list output from MAP. There is no restriction on argument names and additional arguments can be provided.

- REDUCE = function(mapped, ...)

Reduction combines data from a single worker and is always performed as part of the distributed step. When iterate=TRUE REDUCE is applied after each MAP step; depending on the nature of REDUCE, iterative reduction can substantially decrease the data stored in memory. When iterate=FALSE reduction is applied to the list of MAP output applied to all files / ranges.

When REDUCE is missing, output is a list from MAP.

iterate — A logical indicating if the REDUCE function should be applied iteratively to the output of MAP. When REDUCE is missing iterate is set to FALSE. This argument applies to reduceByFile only (reduceFiles calls MAP a single time on each worker).

Collapsing results iteratively is useful when the number of records to be processed is large (maybe complete files) but the end result is a much reduced representation of all records. Iteratively applying REDUCE reduces the amount of data on each worker at any one time and can substantially reduce the memory footprint.

summarize — A logical indicating if results should be returned as a SummarizedExperiment object instead of a list; data are returned in the assays slot named 'data'. This argument applies to reduceByFile only.

When REDUCE is provided summarize is ignored (i.e., set to FALSE). A SummarizedExperiment requires the number of rows in rowRanges and assays to match. Because REDUCE collapses the data across ranges, the dimension of the result no longer matches that of the original ranges.

init — An optional initial value for REDUCE when iterate=TRUE. init must be an object of the same type as the elements returned from MAP. REDUCE logically adds init to the start (when proceeding left to right) or end of results obtained with MAP.

... — Arguments passed to other methods.

Details

reduceByFile extracts, manipulates and combines multiple ranges within a single file. Each file is sent to a worker where MAP is invoked on each file / range combination. This approach allows multiple ranges extracted from a single file to be kept separate or combined with REDUCE.

In contrast, reduceFiles treats the output of all MAP calls as a group and reduces them together. REDUCE usually plays a minor role by concatenating or unlisting results.

Both MAP and REDUCE are applied in the distributed step ("on the worker"). Results are not combined across workers in the distributed step.

Value

- reduceByFile: When summarize=FALSE the return value is a list or the value from the final invocation of REDUCE. When summarize=TRUE output is a SummarizedExperiment. When

ranges is a GenomicFiles object data from rowRanges, colData and metadata are transferred to the SummarizedExperiment.

- reduceFiles: A list or the value returned by the final invocation of REDUCE.

Author(s)

Martin Morgan and Valerie Obenchain

See Also

- reduceRanges
- reduceByRange
- GenomicFiles-class

Examples

```
if (requireNamespace("RNAseqData.HNRNPC.bam.chr14", quietly=TRUE)) {
  ## ------------------------------------------------------------------------
  ## Count junction reads in BAM files
  ## ------------------------------------------------------------------------
  fls <-                                    ## 8 bam files
      RNAseqData.HNRNPC.bam.chr14::RNAseqData.HNRNPC.bam.chr14_BAMFILES

  ## Ranges of interest.
  gr <- GRanges("chr14", IRanges(c(19100000, 106000000), width=1e7))

  ## MAP outputs a table of junction counts per range.
  MAP <- function(range, file, ...) {
      ## for readGAlignments(), Rsamtools::ScanBamParam()
      requireNamespace("GenomicAlignments", quietly=TRUE)
      param = Rsamtools::ScanBamParam(which=range)
      gal = GenomicAlignments::readGAlignments(file, param=param)
      table(GenomicAlignments::njunc(gal))
  }

  ## ------------------------------------------------------------------------
  ## reduceByFile:

  ## With no REDUCE, counts are computed for each range / file combination.
  counts1 <- reduceByFile(gr, fls, MAP)
  length(counts1)          ## 8 files
  elementNROWS(counts1)    ## 2 ranges each

  ## Tables of counts for each range:
  counts1[[1]]

  ## With a REDUCE, results are combined on the fly. This reducer sums the
  ## number of records in each range with exactly 1 junction.
  REDUCE <- function(mapped, ...)
      sum(sapply(mapped, "[", "1"))

  reduceByFile(gr, fls, MAP, REDUCE)

  ## ------------------------------------------------------------------------
  ## reduceFiles:
```

```
  ## All ranges are treated as a single group:
  counts2 <- reduceFiles(gr, fls, MAP)

  ## Counts are for all ranges grouped:
  counts2[[1]]

  ## Contrast the above with that from reduceByFile() where counts
  ## are for each range separately:
  counts1[[1]]

  ## ---------------------------------------------------------------------
  ## Methods for the GenomicFiles class:

  ## Both reduceByFiles() and reduceFiles() can operate on a GenomicFiles
  ## object.
  colData <- DataFrame(method=rep("RNASeq", length(fls)),
                       format=rep("bam", length(fls)))
  gf <- GenomicFiles(files=fls, rowRanges=gr, colData=colData)
  gf

  ## Subset on ranges or files for different experimental runs.
  dim(gf)
  gf_sub <- gf[2, 3:4]
  dim(gf_sub)

  ## When summarize = TRUE and no REDUCE is given, the output is a
  ## SummarizedExperiment object.
  se <- reduceByFile(gf, MAP=MAP, summarize=TRUE)
  se

  ## Data from the rowRanges, colData and metadata slots in the
  ## GenomicFiles are transferred to the SummarizedExperiment.
  colData(se)

  ## Results are in the assays slot named 'data'.
  assays(se)
}
```

---

reduceByRange *Parallel computations by ranges*

Description

Computations are distributed in parallel by range. Data subsets are extracted and manipulated (MAP) and optionally combined (REDUCE) across all files.

Usage

```
## S4 method for signature 'GRanges,ANY'
reduceByRange(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)
## S4 method for signature 'GRangesList,ANY'
reduceByRange(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)
```

```
## S4 method for signature 'GenomicFiles,missing'
reduceByRange(ranges, files, MAP,
    REDUCE, ..., summarize=FALSE, iterate=TRUE, init)

reduceRanges(ranges, files, MAP, REDUCE, ..., init)
```

Arguments

ranges	A GRanges, GrangesList or GenomicFiles object. A GRangesList implies a grouping of the ranges; MAP is applied to each element of the GRangesList vs each range when ranges is a GRanges. When ranges is a GenomicFiles the files argument is missing; both ranges and files are extracted from the object.
files	A character vector or List of filenames. A List implies a grouping of the files; MAP is applied to each element of the List vs each file individually.
MAP	A function executed on each worker. The signature must contain a minimum of two arguments representing the ranges and files. There is no restriction on argument names and additional arguments can be provided. • MAP = function(range, file, ...)
REDUCE	An optional function that combines output from the MAP step applied across all files. The signature must contain at least one argument representing the list output from MAP. There is no restriction on argument names and additional arguments can be provided. • REDUCE = function(mapped, ...) Reduction combines data from a single worker and is always performed as part of the distributed step. When iterate=TRUE REDUCE is applied after each MAP step; depending on the nature of REDUCE, iterative reduction can substantially decrease the data stored in memory. When iterate=FALSE reduction is applied to the list of MAP outputs for a single range, applied to all files. When REDUCE is missing, output is a list from MAP.
iterate	A logical that, when TRUE, indicates that the REDUCE function should be applied iteratively to the output of MAP. When REDUCE is missing iterate is set to FALSE. This argument applies to reduceByRange only. Collapsing results iteratively is useful when the number of records to be processed is large (maybe complete files) but the end result is a much reduced representation of all records. Iteratively applying REDUCE reduces the amount of data on each worker at any one time and can substantially reduce the memory footprint.
summarize	A logical indicating if results should be returned as a SummarizedExperiment object instead of a list; data are returned in the assays slot named 'data'. This argument applies to reduceByRange only. When REDUCE is provided summarize is ignored (i.e., set to FALSE). A SummarizedExperiment requires the number of rows in colData and the columns in assays to match. Because REDUCE collapses the data across files, the dimension of the result no longer matches that of the original ranges.
init	An optional initial value for REDUCE when iterate=TRUE. init must be an object of the same type as the elements returned from MAP. REDUCE logically adds init to the start (when proceeding left to right) or end of results obtained with MAP.
...	Arguments passed to other methods. Currently not used.

Details

reduceByRange extracts, manipulates and combines ranges across different files. Each element of ranges is sent to a worker; this is a single range when ranges is a GRanges and may be multiple ranges when ranges is a GRangesList. The worker then iterates across all files, applying MAP(range, file, ...) to each. When iterate=FALSE, REDUCE is applied to the list of results from MAP applied to all files. When iterate = TRUE, the argument to REDUCE is always a list of length 2. REDUCE is first invoked after the second file has been processed. The first element of the list to REDUCE is the result of calling MAP on the first file; the second element is the result of calling MAP on the second file. For the nth file, the first element is the result of the call to REDUCE for the n-1th file, and the second element is the result of calling MAP on the nth file.

reduceRanges is essentially equivalent to reduceByRange, but with iterate = FALSE.

Both MAP and REDUCE are applied in the distributed step ("on the worker"). REDUCE provides a way to summarize results for a single range across all files; REDUCE does *not* provide a mechanism to summarize results across ranges.

Value

- reduceByRange: When summarize=FALSE the return value is a list or the value from the final invocation of REDUCE. When summarize=TRUE output is a SummarizedExperiment. When ranges is a GenomicFiles object data from rowRanges, colData and metadata are transferred to the SummarizedExperiment.
- reduceRanges: A list or the value returned by the final invocation of REDUCE.

Author(s)

Martin Morgan and Valerie Obenchain

See Also

- reduceFiles
- reduceByFile
- GenomicFiles-class

Examples

```
if (all(requireNamespace("RNAseqData.HNRNPC.bam.chr14", quietly=TRUE) &&
        require(GenomicAlignments))) {
  ## ------------------------------------------------------------------------
  ## Compute coverage across BAM files.
  ## ------------------------------------------------------------------------
  fls <-                                ## 8 bam files
      RNAseqData.HNRNPC.bam.chr14::RNAseqData.HNRNPC.bam.chr14_BAMFILES

  ## Regions of interest.
  gr <- GRanges("chr14", IRanges(c(62262735, 63121531, 63980327),
                width=214700))

  ## The MAP computes the coverage ...
  MAP <- function(range, file, ...) {
      requireNamespace("GenomicFiles", quietly=TRUE)
      ## for coverage(), Rsamtools::ScanBamParam()
      param = Rsamtools::ScanBamParam(which=range)
      GenomicFiles::coverage(file, param=param)[range]
```

```
}
## and REDUCE adds the last and current results.
REDUCE <- function(mapped, ...)
    Reduce("+", mapped)

## ---------------------------------------------------------------------
## reduceByRange:

## With no REDUCE, coverage is computed for each range / file combination.
cov1 <- reduceByRange(gr, fls, MAP)
cov1[[1]]

## Each call to coverage() produces an RleList which accumulate on the
## workers. We can use a reducer to combine these lists either iteratively
## or non-iteratively. When iterate = TRUE the current result
## is collapsed with the last resulting in a maximum of 2 RleLists on
## a worker at any given time.
cov2 <- reduceByRange(gr, fls, MAP, REDUCE, iterate=TRUE)
cov2[[1]]

## If memory use is not a concern (or if MAP output is not large) the
## REDUCE function can be applied non-iteratively.
cov3 <- reduceByRange(gr, fls, MAP, REDUCE, iterate=FALSE)

## Results match those obtained with the iterative REDUCE.
cov3[[1]]

## When 'ranges' is a GRangesList, the list elements are sent to the
## workers instead of a single range as in the case of a GRanges.
grl <- GRangesList(gr[1], gr[2:3])
grl

cov4 <- reduceByRange(grl, fls, MAP)
length(cov4)          ## length of GRangesList
elementNROWS(cov4)  ## number of files

## ---------------------------------------------------------------------
## reduceRanges:

## This function passes the character vector of all file names to MAP.
## MAP must handle each file separately or invoke a method that operates
## on a list of files.

## TODO: example
}
```

`reduceByYield` *Iterate through a BAM (or other) file, reducing output to a single result.*

Description

Rsamtools files can be created with a 'yieldSize' argument that influences the number of records (chunk size) input at one time (see, e.g,. `BamFile`). `reduceByYield` iterates through the file, processing each chunk and reducing it with previously input chunks. This is a memory efficient way to process large data files, especially when the final result fits in memory.

Usage

```
reduceByYield(X, YIELD,  MAP = identity, REDUCE = `+`,
              DONE = function(x) is.null(x) || length(x) == 0L,
              ..., parallel = FALSE, iterate = TRUE, init)

REDUCEsampler(sampleSize=1000000, verbose=FALSE)
```

Arguments

X	A BamFile instance (or other class for which isOpen, open, close methods are defined, and which support extraction of sequential chunks).
YIELD	A function name or user-supplied function that operates on X to produce a VALUE that is passed to DONE and MAP. Generally YIELD will be a data extractor such as readGAlignments, scanBam, yield, etc. and VALUE is a chunk of data. • YIELD(X)
MAP	A function of one or more arguments that operates on the chunk of data from YIELD. • MAP(VALUE, ...)
REDUCE	A function of one (iterate=FALSE or two (iterate=TRUE) arguments, returning the reduction (e.g., sum, mean, concatenate) of the arguments. • REDUCE(mapped, ...) ## iterate=FALSE • REDUCE(x, y, ...) ## iterate=TRUE
DONE	A function of one argument, the VALUE output of the most recent call to YIELD(X, ...). If missing, DONE is function(VALUE) length(VALUE) == 0.
...	Additional arguments, passed to MAP.
iterate	logical(1) determines whether the call to REDUCE is iterative (iterate=TRUE) or cumulative (iterate=FALSE).
parallel	logical(1) determines if the MAP step is run in parallel. bpiterate is used under the hood and is currently supported for Unix/Mac only. For Windows machines, parallel is ignored.
init	(Optional) Initial value used for REDUCE when iterate=TRUE.
sampleSize	Initial value used for REDUCEsampler.
verbose	logical(1) determines if total records sampled are reported at each iteration. Applicable to REDUCEsampler only.

Details

reduceByYield: When iterate=TRUE, REDUCE requires 2 arguments and is invoked with init and the output from the first call to MAP. If init is missing, it operates on the first two outputs from MAP.

When iterate=FALSE, REDUCE requires 1 argument and is is invoked with a list containing a list containing all results from MAP.

REDUCEsampler: REDUCEsampler creates a function that can be used as the REDUCE argument to reduceByYield.

Invoking REDUCEsampler with sampleSize returns a function (call it myfun) that takes two arguments, x and y. As with any iterative REDUCE function, x represents records that have been yield'ed and y is the new chunk of records. myfun samples records from consecutive chunks returned by the YIELD function. (Re)sampling takes into consideration the total number of records yield'ed, the sampleSize, and the size of the new chunk.

Value

The value returned by the final invocation of REDUCE, or `init` if provided and no data were yield'ed, or `list()` if `init` is missing and no data were yield'ed.

Author(s)

Martin Morgan and Valerie Obenchain

See Also

- BamFile and TabixFile for examples of 'X'.
- reduceByFile and reduceByRange

Examples

```
if (all(require(RNAseqData.HNRNPC.bam.chr14) &&
        require(GenomicAlignments))) {

  ## ---------------------------------------------------------------------
  ## Nucleotide frequency of mapped reads
  ## ---------------------------------------------------------------------

  ## In this example nucleotide frequency of mapped reads is computed
  ## for a single file. The MAP step is run in parallel and REDUCE
  ## is iterative.

  ## Create a BamFile and set a 'yieldSize'.
  fl <- system.file(package="Rsamtools", "extdata", "ex1.bam")
  bf <- BamFile(fl, yieldSize=500)

  ## Define 'YIELD', 'MAP' and 'REDUCE' functions.
  YIELD <- function(X, ...) {
      flag = scanBamFlag(isUnmappedQuery=FALSE)
      param = ScanBamParam(flag=flag, what="seq")
      scanBam(X, param=param, ...)[[1]][['seq']]
  }
  MAP <- function(value, ...) {
      requireNamespace("Biostrings", quietly=TRUE)  ## for alphabetFrequency()
      Biostrings::alphabetFrequency(value, collapse=TRUE)
  }
  REDUCE <- `+`        # add successive alphabetFrequency matrices

  ## 'parallel=TRUE' runs the MAP step in parallel and is currently
  ## implemented for Unix/Mac only.
  register(MulticoreParam(3))
  reduceByYield(bf, YIELD, MAP, REDUCE, parallel=TRUE)

  ## ---------------------------------------------------------------------
  ## Coverage
  ## ---------------------------------------------------------------------

  ## If sufficient resources are available coverage can be computed
  ## across several large BAM files by combining reduceByYield() with
  ## bplapply().

  ## Create a BamFileList with a few sample files and a Snow cluster
```

```
## with the same number of workers as files.
bfl <- BamFileList(RNAseqData.HNRNPC.bam.chr14_BAMFILES[1:3])
bpparam <- SnowParam(length(bfl))

## 'FUN' is run on each worker. Because these are Snow workers each
## variable used in 'FUN' must be explicitly passed. (This is not the case
## when using Multicore.)
FUN <- function(bf, YIELD, MAP, REDUCE, parallel, ...) {
  requireNamespace("GenomicFiles", quietly=TRUE)      ## for reduceByYield()
  GenomicFiles::reduceByYield(bf, YIELD, MAP, REDUCE, parallel=parallel)
}

## Passing parallel=FALSE to reduceByYield() runs the MAP step in serial on
## each worker. In this example, parallel dispatch is at the file-level
## only (bplapply()).
YIELD <- `readGAlignments`
MAP <- function(value, ...) {
    requireNamespace("GenomicAlignments", quietly=TRUE)
    GenomicAlignments::coverage(value)[["chr14"]]
}
bplapply(bfl, FUN, YIELD=YIELD, MAP=MAP, REDUCE=`+`,
         parallel=FALSE, BPPARAM = bpparam)


## ----------------------------------------------------------------------
## Sample records from a Bam file
## ----------------------------------------------------------------------

fl <- system.file(package="Rsamtools", "extdata", "ex1.bam")
bf <- BamFile(fl, yieldSize=1000)

yield <- function(x)
    readGAlignments(x, param=ScanBamParam(what=c( "qwidth", "mapq" )))
map <- identity

## Samples records from successive chunks of aligned reads.
reduceByYield(bf, yield, map, REDUCEsampler(1000, TRUE))
}
```

`registry-utils` *Functions for creating and searching a registry of file types.*

Description

Functions for creating and searching a registry of file types based on file extension.

Usage

```
registerFileType(type, package, regex)
findTypeRegistry(fnames)
makeFileType(fnames, ..., regex=findTypeRegistry(fnames))
```

Arguments

type	The List class the file is associated with such as BamFileList, BigWigFileList, FaFileList.
package	The package where the List class (type) is defined.
regex	A regular expression that uniquely identifies the file extension.
fnames	A character vector of file names.
...	Additional arguments passed to the List-class constructor (e.g., yieldSize for BamFileList).

Details

registerFileType: The registerFileType function adds entries to the file type register created at load time. The point of the register is for discovery of file type (class) by file extension. These are List-type classes (e.g., BamFileList) that occupy the fileList slot of a GenomicFiles class.

Each List class entry in the register is associated with (1) a regular expression that identifies the file extension, (2) a class and (3) the package where the class is defined. At load time the register is populated with classes known to GenomicFiles. New classes / file types can be added to the register with registerFileType by providing these three pieces of information.

findTypeRegistry: Searches the registry for a match to the extension of fname. Internal use only.

makeFileType: Performs a look-up in the file registry based on the supplied regular expression; returns an object of the associated class. Internal use only.

Value

registerFileType: NULL

findTypeRegistry: The regular expression associated with the file.

makeFileType: A List-type object defined in the registry.

Examples

```
## At load time the registry is populated with file types
## known to GenomicFiles.
sapply(as.list(.fileTypeRegistry), "[", "type")

## Add a new class to the file register.
## Not run: registerFileType(NewClassList, NewPackage, "\.NewExtension$")
```

---

unpack *Un-pack results obtained with a pack()ed group of ranges*

Description

unpack returns results obtained with pack()ed ranges to the geometry of the original, unpacked ranges.

Usage

```
## S4 method for signature 'list,GRangesList'
unpack(flesh, skeleton, ...)
## S4 method for signature 'List,GRangesList'
unpack(flesh, skeleton, ...)
```

Arguments

flesh	A List object to be unpacked; the result from querying a file with skeleton.
skeleton	The GRangesList created with ‘pack(x)‘.
...	Arguments passed to other methods.

Details

unpack returns a List obtained with packed ranges to the geometry and order of the original, unpacked ranges.

Value

A unpacked form of flesh.

See Also

- pack for packing ranges.

Examples

```
fl <- system.file("extdata", "ex1.bam", package = "Rsamtools")
gr <- GRanges(c(rep("seq2", 3), "seq1"),
              IRanges(c(75, 1, 100, 1), width = 2))

## Ranges are packed by order within chromosome and grouped
## around gaps greater than 'inter_range_len'. See ?pack for details.
pk <- pack(gr, inter_range_len = 25)

## FUN computes coverage for the range passed as 'rng'.
FUN <- function(rng, fl, param) {
    requireNamespace("GenomicAlignments") ## for bamWhich() and coverage()
    Rsamtools::bamWhich(param) <- rng
    GenomicAlignments::coverage(Rsamtools::BamFile(fl), param=param)[rng]
}

## Compute coverage on the packed ranges.
dat <- bplapply(as.list(pk), FUN, fl = fl, param = ScanBamParam())

## The result list contains RleLists of coverage.
lapply(dat, class)

## unpack() transforms the results back to the order of
## the original ranges (i.e., unpacked 'gr').
unpack(dat, pk)
```

VcfStack *VcfStack and RangedVcfStack Objects*

Description

The VcfStack class is a vector of related VCF files, for instance each file representing a separate chromosome. The class helps manage these files as a group. The RangedVcfStack class extends VcfStack by associating genomic ranges of interest to the collection of VCF files.

Constructor

VcfStack(files=NULL, seqinfo=NULL, colData=NULL, index=TRUE, check=TRUE): Creates a VcfStack object.

files A VcfFilelist object. If a VcfFile or character vector is given a VcfFileList will be coerced. The character vector should be files paths pointing to VCF files. The character vector must be named, with names correspond to seqnames in each VCF file.

seqinfo A Seqinfo object describing the levels genome and circularity of each sequence.

colData An optional DataFrame describing each sample in the VcfStack. When present, row names must correspond to sample names in the VCF file.

index A logical indicating if the vcf index files should be created.

check A logical indicating if the check across samples should be performed

RangedVcfStack(vs=NULL, rowRanges=NULL): Creates a RangedVcfStack object.

vs A VcfStack object.

rowRanges An optional GRanges object associating the genomic ranges of interest to the collection of VCF files. The seqnames of rowRanges are a subset of seqnames(vs). If missing, a default is created from the seqinfo object of the provided VcfStack.

Accessors

In the code below, x is a VcfStack or RangedVcfStack object.

dim(x) Get the number of files and samples in the VcfStack object.

colnames(x, do.NULL=TRUE, prefix="col") Get the sample names in the VcfStack.

rownames(x), do.NULL=TRUE, prefix="row") Get the names of the files in VcfStack.

dimnames(x)) Get the names of samples and the names of files in VcfStack.

files(x, ...), files(x, ..., check=TRUE) <- value Get or set the files on x. value can be a named character() of file paths or a VcfFileList. The return value will be a VcfFileList.

seqinfo(x), seqinfo(x, new2old = NULL, pruning.mode = c("error", "coarse", "fine", "tidy")) <- value Get or set the seqinfo on x. See seqinfo<- for details on new2old and pruning.mode.

seqlevelsStyle(x) <- value Set the seqlevels according to the supplied style. File names and rowRanges will also be updated if applicable. See seqlevelsStyle<- for more details.

colData(x), colData(x, ...) <- value Get or set the colData on x. value is a DataFrame.

rowRanges(x), rowRanges(x, ...) <- value Get or set the rowRanges on x. x has to be a RangedVcfStack object. value is a GRanges.

Methods

In the code below, x is a VcfStack or RangedVcfStack object. i is a GRanges object, character() vector of seqnames, numeric() vector, logical() vector, or can be missing. For a RangedVcfStack object, assay and readVcfStack will use the associated rowRanges object for i.

vcfFields(x) Returns a CharacterList of all available VCF fields, with names of fixed, info, geno and samples indicating the four categories. Each element is a character() vector of available VCF field names within each category.

assay(x, i, ..., BPPARAM=bpparam()) Get matrix of genotype calls from the VCF files. See genotypeToSnpMatrix. Argument i specifies which files to read. BPPARAM is the argument to the bpmapply.

readVcfStack(x, i, j=colnames(x), param=ScanVcfParam()) Get content of VCF files in the VcfStack. i indicates which files to read. j can be missing or a character() vector of sample names (see samples) present in the VCF files. param is a ScanVcfParam object. If param is used i and j are ignored.

show(object) Display abbreviated information about VcfStack or RangedVcfStack object.

Subsetting

In the code below, x is a VcfStack or RangedVcfStack object.

x[i, j] Get elements from ranges i and samples j as a VcfStack or RangedVcfStack object. Note: for a RangedVcfStack, the rowRanges object will also be subset.

i can be missing, a character() vector of seqnames, numeric() vector of indexes, logical() or GRanges object. When i is a GRanges object, seqnames(i) is then used to subset the files in the VcfStack.

j can be missing, a character() vector of sample names, a numeric(), logical() vector.

Helpers

getVCFPath(vs, chrtok) Deprecated. Use files(vs)[chrtok] instead.

paths1kg(chrtoks) Translate seqnames chrtoks to 1000 genomes genotype VCF urls.

Author(s)

Lori Shepherd mailto:Lori.Shepherd@RoswellPark.org and Martin Morgan mailto:Martin.Morgan@RoswellPark.org

See Also

VcfFile, VcfFileList.

Examples

```
## ---------------------------------------------------------------------
## CONSTRUCTION
## ---------------------------------------------------------------------
## point to VCF files and add names corresponding to the sequence
## present in the file
extdata <- system.file(package="GenomicFiles", "extdata")
files <- dir(extdata, pattern="^CEUtrio.*bgz$", full=TRUE)
names(files) <- sub(".*_([0-9XY]+).*", "\\1", basename(files))
```

```
## input data.frame describing the length of each sequence, coerce to
## 'Seqinfo' object
seqinfo <- as(readRDS(file.path(extdata, "seqinfo.rds")), "Seqinfo")

stack <- VcfStack(files, seqinfo)
stack

## Use seqinfo from VCF files instead of explict value
stack2 <- VcfStack(files)

rstack <- RangedVcfStack(stack)
gr <- GRanges(c("7:1-159138000", "X:1-155270560"))
rstack2 <- RangedVcfStack(stack, gr)
rstack2

## ---------------------------------------------------------------------
## ACCESSORS
## ---------------------------------------------------------------------
dim(stack)
colnames(stack)
rownames(stack)
dimnames(stack)
head(files(stack))
seqinfo(stack)
colData(stack)

## ---------------------------------------------------------------------
## METHODS
## ---------------------------------------------------------------------
readVcfStack(stack, i=GRanges("20:862167-62858306"))
i <- GRanges(c("20:862167-62858306", "7:1-159138000"))
readVcfStack(stack, i=i, j="NA12891")

head(assay(stack, gr))
head(assay(rstack2))

seqlevels(stack2)
rownames(stack2)

library(GenomeInfoDb)  # for seqlevelsStyle()
seqlevelsStyle(stack2)
seqlevelsStyle(stack2) <- "UCSC"
seqlevelsStyle(stack2)
seqlevels(stack2)
rownames(stack2)
vcfFields(stack2)

## ---------------------------------------------------------------------
## SUBSETTING
## ---------------------------------------------------------------------
## select rows 4, 5, 6 and samples 1, 2
stack[4:6, 1:2]
## select rownames "7", "11" and sample "NA12891"
stack[c("7", "11"), "NA12891"]
stack[c("7", "11", "X"), 2:3]
## subset with GRanges
stack[GRanges("20:862167-62858306")]
```

```
rstack2[]
rstack2[,1]

## ------------------------------------------------------------------------
## HELPERS
## ------------------------------------------------------------------------
paths1kg(1:3)
```

Index