Package 'SummarizedExperiment'

January 24, 2026

Title A container (S4 class) for matrix-like assays

Description The SummarizedExperiment container contains one or more assays, each represented by a matrix-like object of numeric or other mode. The rows typically represent genomic ranges of interest and the columns represent samples.

biocViews Genetics, Infrastructure, Sequencing, Annotation, Coverage, GenomeAnnotation

URL https://bioconductor.org/packages/SummarizedExperiment

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

Version 1.41.0

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.0.0), methods, MatrixGenerics (>= 1.1.3), GenomicRanges (>= 1.61.4), Biobase

Imports utils, stats, tools, Matrix, BiocGenerics (>= 0.51.3), S4Vectors (>= 0.33.7), IRanges (>= 2.23.9), Seqinfo, S4Arrays (>= 1.1.1), DelayedArray (>= 0.31.12)

Suggests GenomeInfoDb (>= 1.45.5), rhdf5, HDF5Array (>= 1.7.5), annotate, AnnotationDbi, GenomicFeatures, SparseArray, SingleCellExperiment, TxDb.Hsapiens.UCSC.hg19.knownGene, hgu95av2.db, airway (>= 1.15.1), BiocStyle, knitr, rmarkdown, RUnit, testthat, digest

VignetteBuilder knitr

Collate Assays-class.R SummarizedExperiment-class.R RangedSummarizedExperiment-class.R intra-range-methods.R inter-range-methods.R coverage-methods.R combine-methods.R findOverlaps-methods.R nearest-methods.R makeSummarizedExperimentFromExpressionSet.R makeSummarizedExperimentFromDataFrame.R makeSummarizedExperimentFromLoom.R zzz.R

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

git_branch devel

git_last_commit 7a9787c

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-01-23

Author Martin Morgan [aut],
Valerie Obenchain [aut],
Jim Hester [aut],
Hervé Pagès [aut, cre]

Maintainer Hervé Pagès <hpages.on.github@gmail.com>

Contents

`Assays-class` *Assays objects*

Description

The Assays virtual class and its methods provide a formal abstraction of the assays slot of SummarizedExperiment objects.

SimpleAssays and ShallowSimpleListAssays are concrete subclasses of Assays with the former being currently the default implementation of Assays objects. Other implementations (e.g. disk-based) could easily be added.

Note that these classes are not meant to be used directly by the end user and the material in this man page is aimed at package developers.

Details

Assays objects have a list-like semantics with elements having matrix- or array-like semantics (e.g., `dim`, `dimnames`).

The Assays API consists of:

- (a) The `Assays()` constructor function.
- (b) Lossless back and forth coercion from/to SimpleList. The coercion method from SimpleList doesn't need (and should not) validate the returned object.
- (c) `length`, `names`, `` `names<-` ``, `getListElement`, `setListElement`, `dim`, `[`, `` `[<-` ``, `rbind`, `cbind`.

An Assays concrete subclass needs to implement (b) (required) plus, optionally any of the methods in (c).

IMPORTANT:

1. Nobody in the Assays hierarchy is allowed to inherit from SimpleList because of the conflicting semantic of `[`.
2. Methods that return a modified Assays object (a.k.a. endomorphisms), that is, `[` as well as replacement methods `names<-`, `setListElement`, and `[<-`, must respect the *copy-on-change contract*. With objects that don't make use of references internally, the developer doesn't need to take any special action for that because it's automatically taken care of by R itself. However, for objects that do make use of references internally (e.g. environments, external pointers, pointer to a file on disk, etc...), the developer needs to be careful to implement endomorphisms with copy-on-change semantics. This can easily be achieved (and is what the default methods for Assays objects do) by performaing a full (deep) copy of the object before modifying it instead of trying to modify it in-place. However note that this full (deep) copy can be very expensive and is actually not necessary in order to achieve copy-on-change semantics: it's enough (and often preferrable for performance reasons) to copy only the parts of the object that need to be modified.

Assays has currently 3 implementations which are formalized by concrete subclasses SimpleAssays, ShallowSimpleListAssays, and AssaysInEnv. SimpleAssays is the default (prior to SummarizedExperiment 1.15.4, ShallowSimpleListAssays was the default). AssaysInEnv is a *broken* alternative to ShallowSimpleListAssays that does NOT respect the *copy-on-change contract*. It is only provided for illustration purposes (see source file Assays-class.R for the details).

A little more detail about ShallowSimpleListAssays: a small reference class hierarchy (not exported from the **GenomicRanges** name space) defines a reference class ShallowData with a single field `data` of type `ANY`, and a derived class ShallowSimpleListAssays that specializes the type of `data` as SimpleList, and `contains=c("ShallowData", "Assays")`. The assays slot of a SummarizedExperiment object contains an instance of ShallowSimpleListAssays.

Author(s)

Martin Morgan and Hervé Pagès

See Also

- SummarizedExperiment objects.
- SimpleList objects in the **S4Vectors** package.

Examples

```
## ---------------------------------------------------------------------
## DIRECT MANIPULATION OF Assays OBJECTS
## ---------------------------------------------------------------------
m1 <- matrix(runif(24), ncol=3)
m2 <- matrix(runif(24), ncol=3)
a <- Assays(SimpleList(m1, m2))
a

as(a, "SimpleList")

length(a)
getListElement(a, 2)
dim(a)

b <- a[-4, 2]
b
length(b)
getListElement(b, 2)
dim(b)

names(a)
names(a) <- c("a1", "a2")
names(a)
getListElement(a, "a2")

rbind(a, a)
cbind(a, a)

## ---------------------------------------------------------------------
## COPY-ON-CHANGE CONTRACT
## ---------------------------------------------------------------------

## ShallowSimpleListAssays objects have copy-on-change semantics but not
## AssaysInEnv objects. For example:
ssla <- as(SimpleList(m1, m2), "ShallowSimpleListAssays")
aie <- as(SimpleList(m1, m2), "AssaysInEnv")

## No names on 'ssla' and 'aie':
names(ssla)
names(aie)

ssla2 <- ssla
aie2 <- aie
names(ssla2) <- names(aie2) <- c("A1", "A2")

names(ssla)  # still NULL (as expected)

names(aie)   # changed! (because the names<-,AssaysInEnv method is not
             # implemented in a way that respects the copy-on-change
             # contract)
```

coverage-methods *Coverage of a RangedSummarizedExperiment object*

Description

This man page documents the coverage method for RangedSummarizedExperiment objects.

Usage

```
## S4 method for signature 'RangedSummarizedExperiment'
coverage(x, shift=0L, width=NULL, weight=1L,
            method=c("auto", "sort", "hash"))
```

Arguments

x	A RangedSummarizedExperiment object.
shift, width, weight, method	See ?coverage in the **GenomicRanges** package.

Details

This method operates on the rowRanges component of the RangedSummarizedExperiment object, which can be a GenomicRanges or GRangesList object.

More precisely, on RangedSummarizedExperiment object x, coverage(x, ...) is equivalent to coverage(rowRanges(x), ...).

See ?coverage in the **GenomicRanges** package for the details of how coverage operates on a GenomicRanges or GRangesList object.

Value

See ?coverage in the **GenomicRanges** package.

See Also

- RangedSummarizedExperiment objects.
- The coverage man page in the **GenomicRanges** package where the coverage methods for GenomicRanges and GRangesList objects are documented.

Examples

```
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                     IRanges(sample(1000L, 20), width=100),
                     strand=Rle(c("+", "-"), c(12, 8)),
                     seqlengths=c(chr1=1800, chr2=1300))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
```

```
                      row.names=LETTERS[1:6])
rse <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)

cvg <- coverage(rse)
cvg
stopifnot(identical(cvg, coverage(rowRanges(rse))))
```

---

findOverlaps-methods *Finding overlapping ranges in RangedSummarizedExperiment objects*

Description

This man page documents the `findOverlaps` methods for RangedSummarizedExperiment objects.

RangedSummarizedExperiment objects also support `countOverlaps`, `overlapsAny`, and `subsetByOverlaps` thanks to the default methods defined in the **IRanges** package and to the `findOverlaps` methods defined in this package and documented below.

Usage

```
## S4 method for signature 'RangedSummarizedExperiment,Vector'
findOverlaps(query, subject,
    maxgap=-1L, minoverlap=0L,
    type=c("any", "start", "end", "within", "equal"),
    select=c("all", "first", "last", "arbitrary"),
    ignore.strand=FALSE)
## S4 method for signature 'Vector,RangedSummarizedExperiment'
findOverlaps(query, subject,
    maxgap=-1L, minoverlap=0L,
    type=c("any", "start", "end", "within", "equal"),
    select=c("all", "first", "last", "arbitrary"),
    ignore.strand=FALSE)
```

Arguments

`query`, `subject`	One of these two arguments must be a RangedSummarizedExperiment object.
`maxgap`, `minoverlap`, `type`	See ?`findOverlaps` in the **GenomicRanges** package.
`select`, `ignore.strand`	See ?`findOverlaps` in the **GenomicRanges** package.

Details

These methods operate on the `rowRanges` component of the RangedSummarizedExperiment object, which can be a GenomicRanges or GRangesList object.

More precisely, if any of the above functions is passed a RangedSummarizedExperiment object thru the `query` and/or `subject` argument, then it behaves as if `rowRanges(query)` and/or `rowRanges(subject)` had been passed instead.

See `?findOverlaps` in the **GenomicRanges** package for the details of how `findOverlaps` and family operate on GenomicRanges and GRangesList objects.

Value

See `?findOverlaps` in the **GenomicRanges** package.

See Also

- RangedSummarizedExperiment objects.
- The findOverlaps man page in the **GenomicRanges** package where the `findOverlaps` family of methods for GenomicRanges and GRangesList objects is documented.

Examples

```
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                     IRanges(sample(1000L, 20), width=100),
                     strand=Rle(c("+", "-"), c(12, 8)))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
rse0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                             rowRanges=rowRanges, colData=colData)
rse1 <- shift(rse0, 100)

hits <- findOverlaps(rse0, rse1)
hits
stopifnot(identical(hits, findOverlaps(rowRanges(rse0), rowRanges(rse1))))
stopifnot(identical(hits, findOverlaps(rse0, rowRanges(rse1))))
stopifnot(identical(hits, findOverlaps(rowRanges(rse0), rse1)))
```

---

inter-range-methods *Inter range transformations of a RangedSummarizedExperiment object*

Description

This man page documents the *inter range transformations* that are supported on RangedSummarizedExperiment objects.

Usage

```
## S4 method for signature 'RangedSummarizedExperiment'
isDisjoint(x, ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment'
disjointBins(x, ignore.strand=FALSE)
```

Arguments

x	A RangedSummarizedExperiment object.
ignore.strand	See ?isDisjoint in the **GenomicRanges** package.

Details

These transformations operate on the rowRanges component of the RangedSummarizedExperiment object, which can be a GenomicRanges or GRangesList object.

More precisely, any of the above functions performs the following transformation on RangedSummarizedExperiment object x:

```
f(rowRanges(x), ...)
```

where f is the name of the function and ... any additional arguments passed to it.

See ?isDisjoint in the **GenomicRanges** package for the details of how these transformations operate on a GenomicRanges or GRangesList object.

Value

See ?isDisjoint in the **GenomicRanges** package.

See Also

- RangedSummarizedExperiment objects.
- The isDisjoint man page in the **GenomicRanges** package where *inter range transformations* of a GenomicRanges or GRangesList object are documented.

Examples

```
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                     IRanges(sample(1000L, 20), width=100),
                     strand=Rle(c("+", "-"), c(12, 8)))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
rse0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                             rowRanges=rowRanges, colData=colData)
rse1 <- shift(rse0, 99*start(rse0))

isDisjoint(rse0)  # FALSE
```

```
isDisjoint(rse1)  # TRUE

bins0 <- disjointBins(rse0)
bins0
stopifnot(identical(bins0, disjointBins(rowRanges(rse0))))

bins1 <- disjointBins(rse1)
bins1
stopifnot(all(bins1 == bins1[1]))
```

---

intra-range-methods *Intra range transformations of a RangedSummarizedExperiment object*

Description

This man page documents the *intra range transformations* that are supported on RangedSummarizedExperiment objects.

Usage

```
## S4 method for signature 'RangedSummarizedExperiment'
shift(x, shift=0L, use.names=TRUE)

## S4 method for signature 'RangedSummarizedExperiment'
narrow(x, start=NA, end=NA, width=NA, use.names=TRUE)

## S4 method for signature 'RangedSummarizedExperiment'
resize(x, width, fix="start", use.names=TRUE,
       ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment'
flank(x, width, start=TRUE, both=FALSE,
      use.names=TRUE, ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment'
promoters(x, upstream=2000, downstream=200)
## S4 method for signature 'RangedSummarizedExperiment'
terminators(x, upstream=2000, downstream=200)

## S4 method for signature 'RangedSummarizedExperiment'
restrict(x, start=NA, end=NA, keep.all.ranges=FALSE,
         use.names=TRUE)

## S4 method for signature 'RangedSummarizedExperiment'
trim(x, use.names=TRUE)
```

Arguments

x	A RangedSummarizedExperiment object.
shift, use.names, start, end, width, fix, ignore.strand, both, upstream, downstream, keep.all.ranges	See ?GenomicRanges::shift in the **GenomicRanges** package.

Details

These transformations operate on the rowRanges component of the RangedSummarizedExperiment object, which can be a GenomicRanges or GRangesList object.

More precisely, any of the above functions performs the following transformation on RangedSummarizedExperiment object x:

```
rowRanges(x) <- f(rowRanges(x), ...)
```

where f is the name of the function and ... any additional arguments passed to it.

See ?GenomicRanges::shift in the **GenomicRanges** package for the details of how these transformations operate on a GenomicRanges or GRangesList object.

See Also

- RangedSummarizedExperiment objects.
- The shift man page in the **GenomicRanges** package where *intra range transformations* of a GenomicRanges or GRangesList object are documented.

Examples

```
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                     IRanges(sample(1000L, 20), width=100),
                     strand=Rle(c("+", "-"), c(12, 8)))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
rse0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                             rowRanges=rowRanges, colData=colData)

rse1 <- shift(rse0, 1)
stopifnot(identical(
  rowRanges(rse1),
  shift(rowRanges(rse0), 1)
))

se2 <- narrow(rse0, start=10, end=-15)
stopifnot(identical(
  rowRanges(se2),
  narrow(rowRanges(rse0), start=10, end=-15)
))
```

```
se3 <- resize(rse0, width=75)
stopifnot(identical(
  rowRanges(se3),
  resize(rowRanges(rse0), width=75)
))

se4 <- flank(rse0, width=20)
stopifnot(identical(
  rowRanges(se4),
  flank(rowRanges(rse0), width=20)
))

se5 <- promoters(rse0, upstream=85, downstream=50)
stopifnot(identical(
  rowRanges(se5),
  promoters(rowRanges(rse0), upstream=85, downstream=50)
))

se6 <- terminators(rse0, upstream=85, downstream=50)
stopifnot(identical(
  rowRanges(se6),
  terminators(rowRanges(rse0), upstream=85, downstream=50)
))

se7 <- restrict(rse0, start=200, end=700, keep.all.ranges=TRUE)
stopifnot(identical(
  rowRanges(se7),
  restrict(rowRanges(rse0), start=200, end=700, keep.all.ranges=TRUE)
))
```

`makeSummarizedExperimentFromDataFrame` *Make a RangedSummarizedExperiment from a data.frame or DataFrame*

Description

`makeSummarizedExperimentFromDataFrame` uses `data.frame` or `DataFrame` column names to create a GRanges object for the `rowRanges` of the resulting SummarizedExperiment object. It requires that non-range data columns be coercible into a `numeric` `matrix` for the SummarizedExperiment constructor. All columns that are not part of the row ranges attribute are assumed to be experiment data; thus, keeping metadata columns will not be supported. Note that this function only returns SummarizedExperiment objects with a single assay.

If metadata columns are to be kept, one can first construct the row ranges attribute by using the makeGRangesFromDataFrame function and subsequently creating the SummarizedExperiment.

Usage

```
makeSummarizedExperimentFromDataFrame(df,
```

```
...,
seqinfo = NULL,
starts.in.df.are.0based = FALSE)
```

Arguments

df	A data.frame or DataFrame object. If not, then the function first tries to turn df into a data frame with as.data.frame(df).
...	Additional arguments passed on to makeGRangesFromDataFrame
seqinfo	Either NULL, or a Seqinfo object, or a character vector of seqlevels, or a named numeric vector of sequence lengths. When not NULL, it must be compatible with the genomic ranges in df i.e. it must include at least the sequence levels represented in df.
starts.in.df.are.0based	TRUE or FALSE (the default). If TRUE, then the start positions of the genomic ranges in df are considered to be *0-based* and are converted to *1-based* in the returned GRanges object. This feature is intended to make it more convenient to handle input that contains data obtained from resources using the "0-based start" convention. A notorious example of such resource is the UCSC Table Browser (http://genome.ucsc.edu/cgi-bin/hgTables).

Value

A RangedSummarizedExperiment object with rowRanges and a single assay

Author(s)

M. Ramos

See Also

- makeGRangesFromDataFrame

Examples

```
## ---------------------------------------------------------------------
## BASIC EXAMPLES
## ---------------------------------------------------------------------

# Note that rownames of the data.frame are also rownames of the result
df <- data.frame(chr="chr2", start = 11:15, end = 12:16,
                 strand = c("+", "-", "+", "*", "."), expr0 = 3:7,
                 expr1 = 8:12, expr2 = 12:16,
                 row.names = paste0("GENE", letters[5:1]))
df

exRSE <- makeSummarizedExperimentFromDataFrame(df)

exRSE
```

```
assay(exRSE)

rowRanges(exRSE)
```

`makeSummarizedExperimentFromExpressionSet` *Make a RangedSummarizedExperiment object from an ExpressionSet and vice-versa*

Description

Coercion between RangedSummarizedExperiment and ExpressionSet is supported in both directions.

For going from ExpressionSet to RangedSummarizedExperiment, the `makeSummarizedExperimentFromExpressionSet` function is also provided to let the user control how to map features to ranges.

Usage

```
makeSummarizedExperimentFromExpressionSet(from,
                                          mapFun=naiveRangeMapper,
                                          ...)

## range mapping functions
naiveRangeMapper(from)
probeRangeMapper(from)
geneRangeMapper(txDbPackage, key = "ENTREZID")
```

Arguments

`from`	An ExpressionSet object.
`mapFun`	A function which takes an ExpressionSet object and returns a GRanges, or GRangesList object which corresponds to the genomic ranges used in the ExpressionSet. The rownames of the returned GRanges are used to match the featureNames of the ExpressionSet. The `naiveRangeMapper` function is used by default.
`...`	Additional arguments passed to `mapFun`.
`txDbPackage`	A character string with the Transcript Database to use for the mapping.
`key`	A character string with the Gene key to use for the mapping.

Value

`makeSummarizedExperimentFromExpressionSet` takes an ExpressionSet object as input and a *range mapping function* that maps the features to ranges. It then returns a RangedSummarizedExperiment object that corresponds to the input.

The range mapping functions return a GRanges object, with the `rownames` corresponding to the featureNames of the ExpressionSet object.

Author(s)

Jim Hester, james.f.hester@gmail.com

See Also

- RangedSummarizedExperiment objects.
- ExpressionSet objects in the **Biobase** package.
- TxDb objects in the **GenomicFeatures** package.

Examples

```
## ------------------------------------------------------------------------
## GOING FROM ExpressionSet TO SummarizedExperiment
## ------------------------------------------------------------------------

data(sample.ExpressionSet, package="Biobase")

# naive coercion
makeSummarizedExperimentFromExpressionSet(sample.ExpressionSet)
as(sample.ExpressionSet, "RangedSummarizedExperiment")
as(sample.ExpressionSet, "SummarizedExperiment")

# using probe range mapper
makeSummarizedExperimentFromExpressionSet(sample.ExpressionSet, probeRangeMapper)

# using the gene range mapper
se <- makeSummarizedExperimentFromExpressionSet(
    sample.ExpressionSet,
    geneRangeMapper("TxDb.Hsapiens.UCSC.hg19.knownGene")
)
se
rowData(se)  # duplicate row names

## ------------------------------------------------------------------------
## GOING FROM SummarizedExperiment TO ExpressionSet
## ------------------------------------------------------------------------

example(RangedSummarizedExperiment)  # to create 'rse'
rse
as(rse, "ExpressionSet")
```

`makeSummarizedExperimentFromLoom` *Make a SummarizedExperiment from a '.loom' hdf5 file*

Description

makeSummarizedExperimentFromLoom represents a '.loom' file as a SummarizedExperiment. The '/matrix' and '/layers' are represented as HDF5Array objects; row and column attributes are parsed to DataFrame. Optionally, row or column attributes can be specified as row and and column names.

Usage

```
makeSummarizedExperimentFromLoom(file,
                                 rownames_attr = NULL,
                                 colnames_attr = NULL)
```

Arguments

file	The path (as a single character string) to the HDF5 file where the dataset is located.
rownames_attr	The name of the row attribute to be used as row names.
colnames_attr	The name of the column attribute to be used as column names.

Value

A SummarizedExperiment object with row and column data and one or more assays.

Author(s)

Martin Morgan

See Also

http://loompy.org/loompy-docs/format/index.html for a specification of the .loom format.

Examples

```
## ------------------------------------------------------------------------
## BASIC EXAMPLE
## ------------------------------------------------------------------------

file <- system.file(
    package="SummarizedExperiment", "extdata", "example.loom"
)
se <- makeSummarizedExperimentFromLoom(file)
se
assay(se)
metadata(se)
```

nearest-methods *Finding the nearest range neighbor in RangedSummarizedExperiment objects*

Description

This man page documents the `nearest` methods and family (i.e. `precede`, `follow`, `distance`, and `distanceToNearest` methods) for RangedSummarizedExperiment objects.

Usage

```
## S4 method for signature 'RangedSummarizedExperiment,ANY'
precede(x, subject, select=c("arbitrary", "all"),
        ignore.strand=FALSE)
## S4 method for signature 'ANY,RangedSummarizedExperiment'
precede(x, subject, select=c("arbitrary", "all"),
        ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment,ANY'
follow(x, subject, select=c("arbitrary", "all"),
        ignore.strand=FALSE)
## S4 method for signature 'ANY,RangedSummarizedExperiment'
follow(x, subject, select=c("arbitrary", "all"),
        ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment,ANY'
nearest(x, subject, select=c("arbitrary", "all"), ignore.strand=FALSE)
## S4 method for signature 'ANY,RangedSummarizedExperiment'
nearest(x, subject, select=c("arbitrary", "all"), ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment,ANY'
distance(x, y, ignore.strand=FALSE, ...)
## S4 method for signature 'ANY,RangedSummarizedExperiment'
distance(x, y, ignore.strand=FALSE, ...)

## S4 method for signature 'RangedSummarizedExperiment,ANY'
distanceToNearest(x, subject, ignore.strand=FALSE, ...)
## S4 method for signature 'ANY,RangedSummarizedExperiment'
distanceToNearest(x, subject, ignore.strand=FALSE, ...)
```

Arguments

`x`, `subject`	One of these two arguments must be a RangedSummarizedExperiment object.
`select`, `ignore.strand`	See ?`nearest` in the **GenomicRanges** package.
`y`	For the `distance` methods, one of `x` or `y` must be a RangedSummarizedExperiment object.

... Additional arguments for methods.

Details

These methods operate on the rowRanges component of the RangedSummarizedExperiment object, which can be a GenomicRanges or GRangesList object.

More precisely, if any of the above functions is passed a RangedSummarizedExperiment object thru the x, subject, and/or y argument, then it behaves as if rowRanges(x), rowRanges(subject), and/or rowRanges(y) had been passed instead.

See ?nearest in the **GenomicRanges** package for the details of how nearest and family operate on GenomicRanges and GRangesList objects.

Value

See ?nearest in the **GenomicRanges** package.

See Also

- RangedSummarizedExperiment objects.
- The nearest man page in the **GenomicRanges** package where the nearest family of methods for GenomicRanges and GRangesList objects is documented.

Examples

```
nrows <- 20; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                     IRanges(sample(1000L, 20), width=100),
                     strand=Rle(c("+", "-"), c(12, 8)))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
rse0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                             rowRanges=rowRanges, colData=colData)
rse1 <- shift(rse0, 100)

res <- nearest(rse0, rse1)
res
stopifnot(identical(res, nearest(rowRanges(rse0), rowRanges(rse1))))
stopifnot(identical(res, nearest(rse0, rowRanges(rse1))))
stopifnot(identical(res, nearest(rowRanges(rse0), rse1)))

res <- nearest(rse0)  # missing subject
res
stopifnot(identical(res, nearest(rowRanges(rse0))))

hits <- nearest(rse0, rse1, select="all")
hits
stopifnot(identical(
  hits,
  nearest(rowRanges(rse0), rowRanges(rse1), select="all")
))
```

```
stopifnot(identical(
  hits,
  nearest(rse0, rowRanges(rse1), select="all")
))
stopifnot(identical(
  hits,
  nearest(rowRanges(rse0), rse1, select="all")
))
```

RangedSummarizedExperiment-class
RangedSummarizedExperiment objects

Description

The RangedSummarizedExperiment class is a matrix-like container where rows represent ranges of interest (as a GRanges or GRangesList object) and columns represent samples (with sample data summarized as a DataFrame). A RangedSummarizedExperiment object contains one or more assays, each represented by a matrix-like object of numeric or other mode.

RangedSummarizedExperiment is a subclass of SummarizedExperiment and, as such, all the methods documented in `?SummarizedExperiment` also work on a RangedSummarizedExperiment object or any SummarizedExperiment derivative. The methods documented below are additional methods that are specific to RangedSummarizedExperiment objects.

Usage

```
## Constructor

# See ?SummarizedExperiment for the constructor function.

## Accessors

rowRanges(x, ...)
rowRanges(x, ...) <- value

## Subsetting

## S4 method for signature 'RangedSummarizedExperiment'
subset(x, subset, select, ...)

## rowRanges access
## see 'GRanges compatibility', below
```

Arguments

x	A RangedSummarizedExperiment object or derivative. The `rowRanges` setter will also accept a SummarizedExperiment *instance* and will first coerce it to RangedSummarizedExperiment before it sets `value` on it.

...	Further arguments to be passed to or from other methods.
value	A GRanges or GRangesList object.
subset	An expression which, when evaluated in the context of rowRanges(x), is a logical vector indicating elements or rows to keep: missing values are taken as false.
select	An expression which, when evaluated in the context of colData(x), is a logical vector indicating elements or rows to keep: missing values are taken as false.

Details

The rows of a RangedSummarizedExperiment object represent ranges (in genomic coordinates) of interest. The ranges of interest are described by a GRanges or a GRangesList object, accessible using the rowRanges function, described below. The GRanges and GRangesList classes contains sequence (e.g., chromosome) name, genomic coordinates, and strand information. Each range can be annotated with additional data; this data might be used to describe the range or to summarize results (e.g., statistics of differential abundance) relevant to the range. Rows may or may not have row names; they often will not.

Constructor

RangedSummarizedExperiment instances are constructed using the SummarizedExperiment() function documented in ?SummarizedExperiment.

Accessors

In the code snippets below, x is a RangedSummarizedExperiment object or derivative (e.g. a SingleCellExperiment object).

rowRanges(x), rowRanges(x) <- value: Get or set the row data. value is a GenomicRanges object. Row names of value must be NULL or consistent with the existing row names of x.

GRanges compatibility (rowRanges access)

Many GRanges and GRangesList operations are supported on RangedSummarizedExperiment objects, using rowRanges.

Supported operations include: pcompare, duplicated, end, end<-, granges, is.unsorted, match, mcols, mcols<-, order, ranges, ranges<-, rank, seqinfo, seqinfo<-, seqnames, sort, start, start<-, strand, strand<-, width, width<-.

See also ?shift, ?isDisjoint, ?coverage, ?findOverlaps, and ?nearest for more *GRanges compatibility methods*.

Not all GRanges operations are supported, because they do not make sense for RangedSummarizedExperiment objects (e.g., length, name, as.data.frame, c, splitAsList), involve non-trivial combination or splitting of rows (e.g., disjoin, gaps, reduce, unique), or have not yet been implemented (Ops, map, window, window<-).

Subsetting

In the code snippets below, x is a RangedSummarizedExperiment object or derivative (e.g. a SingleCellExperiment object).

`subset(x, subset, select)`: Create a subset of x using an expression `subset` referring to columns of `rowRanges(x)` (including 'seqnames', 'start', 'end', 'width', 'strand', and `names(rowData(x))`) and / or `select` referring to column names of `colData(x)`.

Extension

RangedSummarizedExperiment is implemented as an S4 class, and can be extended in the usual way, using `contains="RangedSummarizedExperiment"` in the new class definition.

See the SingleCellExperiment class defined in the **SingleCellExperiment** package for an example of such extension.

Author(s)

Martin Morgan, mtmorgan@fhcrc.org

See Also

- SummarizedExperiment for the parent class of RangedSummarizedExperiment and the RangedSummarizedExperiment/SummarizedExperiment constructor function.
- shift, isDisjoint, coverage, findOverlaps, and nearest for more *GRanges compatibility methods*.
- GRanges objects in the **GenomicRanges** package.
- The SingleCellExperiment defined in the **SingleCellExperiment** package, a subclass of RangedSummarizedExperiment specifically designed to represent single-cell sequencing data.

Examples

```
nrows <- 200; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),
                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),
                     strand=sample(c("+", "-"), 200, TRUE),
                     feature_id=sprintf("ID%03d", 1:200))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
rse <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)
rse
dim(rse)
dimnames(rse)
assayNames(rse)
head(assay(rse))
assays(rse) <- endoapply(assays(rse), asinh)
head(assay(rse))
```

```
rowRanges(rse)
rowData(rse)  # same as 'mcols(rowRanges(rse))'
colData(rse)

rse[ , rse$Treatment == "ChIP"]

## cbind() combines objects with the same ranges but different samples:
rse1 <- rse
rse2 <- rse1[ , 1:3]
colnames(rse2) <- letters[1:ncol(rse2)]
cmb1 <- cbind(rse1, rse2)
dim(cmb1)
dimnames(cmb1)

## rbind() combines objects with the same samples but different ranges:
rse1 <- rse
rse2 <- rse1[1:50, ]
rownames(rse2) <- letters[1:nrow(rse2)]
cmb2 <- rbind(rse1, rse2)
dim(cmb2)
dimnames(cmb2)

## Coercion to/from SummarizedExperiment:
se0 <- as(rse, "SummarizedExperiment")
se0

as(se0, "RangedSummarizedExperiment")

## Setting rowRanges on a SummarizedExperiment object turns it into a
## RangedSummarizedExperiment object:
se <- se0
rowRanges(se) <- rowRanges
se  # RangedSummarizedExperiment

## Sanity checks:
stopifnot(identical(assays(se0), assays(rse)))
stopifnot(identical(dim(se0), dim(rse)))
stopifnot(identical(dimnames(se0), dimnames(rse)))
stopifnot(identical(rowData(se0), rowData(rse)))
stopifnot(identical(colData(se0), colData(rse)))
```

`SummarizedExperiment-class` *SummarizedExperiment objects*

Description

The SummarizedExperiment class is a matrix-like container where rows represent features of interest (e.g. genes, transcripts, exons, etc...) and columns represent samples (with sample data

summarized as a DataFrame). A SummarizedExperiment object contains one or more assays, each represented by a matrix-like object of numeric or other mode.

Note that SummarizedExperiment is the parent of the RangedSummarizedExperiment class which means that all the methods documented below also work on a RangedSummarizedExperiment object.

Usage

```
## Constructor for RangedSummarizedExperiment/SummarizedExperiment objects

SummarizedExperiment(assays=SimpleList(),
                     rowData=NULL, rowRanges=NULL,
                     colData=DataFrame(),
                     metadata=list(),
                     checkDimnames=TRUE)

## Accessors

assayNames(x, ...)
assayNames(x, ...) <- value
assays(x, withDimnames=TRUE, ...)
assays(x, withDimnames=TRUE, ...) <- value
assay(x, i, withDimnames=TRUE, ...)
assay(x, i, withDimnames=TRUE, ...) <- value
rowData(x, use.names=TRUE, ...)
rowData(x, ...) <- value
colData(x, ...)
colData(x, ...) <- value
#dim(x)
#dimnames(x)
#dimnames(x) <- value

## Quick colData access

## S4 method for signature 'SummarizedExperiment'
x$name
## S4 replacement method for signature 'SummarizedExperiment'
x$name <- value
## S4 method for signature 'SummarizedExperiment,ANY,missing'
x[[i, j, ...]]
## S4 replacement method for signature 'SummarizedExperiment,ANY,missing'
x[[i, j, ...]] <- value

## Subsetting

## S4 method for signature 'SummarizedExperiment'
x[i, j, ..., drop=TRUE]
## S4 replacement method for signature 'SummarizedExperiment,ANY,ANY,SummarizedExperiment'
```

```
x[i, j] <- value
## S4 method for signature 'SummarizedExperiment'
subset(x, subset, select, ...)

## Combining

## S4 method for signature 'SummarizedExperiment'
rbind(..., deparse.level=1)
## S4 method for signature 'SummarizedExperiment'
cbind(..., deparse.level=1)
## S4 method for signature 'SummarizedExperiment'
combineRows(x, ..., delayed=TRUE, fill=NA, use.names=TRUE)
## S4 method for signature 'SummarizedExperiment'
combineCols(x, ..., delayed=TRUE, fill=NA, use.names=TRUE)

## On-disk realization

## S4 method for signature 'SummarizedExperiment'
realize(x, BACKEND=getAutoRealizationBackend())
```

Arguments

assays	A list or SimpleList of matrix-like elements, or a matrix-like object (e.g. an ordinary matrix, a data frame, a DataFrame object from the **S4Vectors** package, a SparseMatrix derivative from the **SparseArray** package, a sparseMatrix derivative from the **Matrix** package, a DelayedMatrix object from the **DelayedArray** package, etc...). All elements of the list must have the same dimensions, and dimension names (if present) must be consistent across elements and with the row names of rowRanges and colData.
rowData	NULL (the default) or a DataFrame object describing the rows. Row names, if present, become the row names of the constructed SummarizedExperiment object. The number of rows of the DataFrame must equal the number of rows of the matrices in assays.
rowRanges	NULL (the default), or a GRanges or GRangesList object describing the ranges of interest. If NULL, a SummarizedExperiment *instance* is returned. Otherwise, a RangedSummarizedExperiment *instance* is returned. Names on rowRanges, if present, become the row names of the returned object. The length of the GRanges or GRangesList must equal the number of rows of the matrices in assays.
colData	An optional DataFrame describing the samples. Row names on colData, if present, become the column names of the returned object.
metadata	An optional list of arbitrary content describing the overall experiment.
checkDimnames	By default the rownames and colnames of the supplied assay(s) are checked for consistency with those of the SummarizedExperiment or RangedSummarizedExperiment object to construct. More precisely, the rownames and colnames of each assay must be NULL or identical to those of the object. Use checkDimnames=FALSE to skip this check.

`x` A SummarizedExperiment object or derivative.

`...` For `assay`, arguments in `...` are forwarded to `assays`.

For `rbind`, `cbind`, `...` contains SummarizedExperiment objects (or derivatives) to be combined.

For other accessors, ignored.

`value` An object of a class specified in the S4 method signature or as outlined in 'Details'.

`i, j` For `assay`, `assay<-`, `i` is an integer or numeric scalar; see 'Details' for additional constraints.

For `[,SummarizedExperiment`, `[,SummarizedExperiment<-`, `i`, `j` are subscripts that can act to subset the rows and columns of `x`, that is the `matrix` elements of `assays`.

For `[[,SummarizedExperiment`, `[[<-,SummarizedExperiment`, `i` is a scalar index (e.g., `character(1)` or `integer(1)`) into a column of `colData`.

`name` A symbol representing the name of a column of `colData`.

`withDimnames` A `logical(1)`, indicating whether the dimnames of the SummarizedExperiment object (or derivative) should be applied (i.e. copied) to the extracted assays. More precisely, setting `withDimnames=FALSE` in the *getter* returns the assays *as-is* whereas setting `withDimnames=TRUE` return them with possibly modified dimnames.

See "Top-level dimnames vs assay-level dimnames" section in the SummarizedExperiment vignette for more information about this and some examples.

Setting `withDimnames=FALSE` in the *setter* (`assays<-`) is required when the dimnames on the supplied assays are not identical to the dimnames on the SummarizedExperiment object; it does not influence actual assignment of dimnames to assays (they're always stored as-is).

Note that

```
assays(x, withDimnames=FALSE) <- assays(x, withDimnames=FALSE)
```

is guaranteed to always work and be a no-op. This is not the case if `withDimnames=TRUE` is used or if `withDimnames` is not specified.

`use.names` For `rowData`: Like `mcols(x)`, by default `rowData(x)` propagates the rownames of `x` to the returned DataFrame object (note that for a SummarizedExperiment object or derivative, the rownames are also the names i.e. `rownames(x)` is always the same as `names(x)`). Setting `use.names=FALSE` suppresses this propagation i.e. it returns a DataFrame object with no rownames. Use this when `rowData(x)` fails, which can happen when the rownames contain NAs (because the rownames of a SummarizedExperiment object or derivative can contain NAs, but the rownames of a DataFrame object cannot).

For `combineRows` and `combineCols`: See Combining section below.

`drop` A `logical(1)`, ignored by these methods.

`deparse.level` See `?base::cbind` for a description of this argument.

`subset` An expression which, when evaluated in the context of `rowData(x)`, is a logical vector indicating elements or rows to keep: missing values are taken as false.

select	An expression which, when evaluated in the context of colData(x), is a logical vector indicating elements or rows to keep: missing values are taken as false.
delayed, fill	See combineRows and combineCols in Combining section below.
BACKEND	NULL (the default), or a single string specifying the name of the backend. When the backend is set to NULL, each element of assays(x) is realized in memory as an ordinary array by just calling as.array on it.

Details

The SummarizedExperiment class is meant for numeric and other data types derived from a sequencing experiment. The structure is rectangular like a matrix, but with additional annotations on the rows and columns, and with the possibility to manage several assays simultaneously so long as they be of the same dimensions.

The rows of a SummarizedExperiment object represent features of interest. Information about these features is stored in a DataFrame object, accessible using the function rowData. The DataFrame must have as many rows as there are rows in the SummarizedExperiment object, with each row of the DataFrame providing information on the feature in the corresponding row of the SummarizedExperiment object. Columns of the DataFrame represent different attributes of the features of interest, e.g., gene or transcript IDs, etc.

Each column of a SummarizedExperiment object represents a sample. Information about the samples are stored in a DataFrame, accessible using the function colData, described below. The DataFrame must have as many rows as there are columns in the SummarizedExperiment object, with each row of the DataFrame providing information on the sample in the corresponding column of the SummarizedExperiment object. Columns of the DataFrame represent different sample attributes, e.g., tissue of origin, etc. Columns of the DataFrame can themselves be annotated (via the mcols function). Column names typically provide a short identifier unique to each sample.

A SummarizedExperiment object can also contain information about the overall experiment, for instance the lab in which it was conducted, the publications with which it is associated, etc. This information is stored as a list object, accessible using the metadata function. The form of the data associated with the experiment is left to the discretion of the user.

The SummarizedExperiment container is appropriate for matrix-like data. The data are accessed using the assays function, described below. This returns a SimpleList object. Each element of the list must itself be a matrix (of any mode) and must have dimensions that are the same as the dimensions of the SummarizedExperiment in which they are stored. Row and column names of each matrix must either be NULL or match those of the SummarizedExperiment during construction. It is convenient for the elements of SimpleList of assays to be named.

Accessors

In the code snippets below, x is a SummarizedExperiment object or derivative (e.g. a RangedSummarizedExperiment object).

assays(x), assays(x) <- value: Get or set the assays. value is a list or SimpleList, each element of which is a matrix with the same dimensions as x.

assay(x, i), assay(x, i) <- value: A convenient alternative (to assays(x)[[i]], assays(x)[[i]] <- value) to get or set the ith (default first) assay element. value must be a matrix of the same dimension as x, and with dimension names NULL or consistent with those of x.

`assayNames(x)`, `assayNames(x) <- value`: Get or set the names of `assay()` elements.

`rowData(x, use.names=TRUE)`, `rowData(x) <- value`: Get or set the row data. `value` is a DataFrame object.

`colData(x)`, `colData(x) <- value`: Get or set the column data. `value` is a DataFrame object. Row names of `value` must be `NULL` or consistent with the existing column names of `x`.

`metadata(x)`, `metadata(x) <- value`: Get or set the experiment data. `value` is a `list` with arbitrary content.

`dim(x)`: Get the dimensions (features of interest x samples) of the SummarizedExperiment object.

`dimnames(x)`, `dimnames(x) <- value`: Get or set the dimension names. `value` is usually a list of length 2, containing elements that are either `NULL` or vectors of appropriate length for the corresponding dimension. `value` can be `NULL`, which removes dimension names. This method implies that `rownames`, `rownames<-`, `colnames`, and `colnames<-` are all available.

Subsetting

In the code snippets below, `x` is a SummarizedExperiment object or derivative (e.g. a RangedSummarizedExperiment object).

`x[i,j]`, `x[i,j] <- value`: Create or replace a subset of `x`. `i`, `j` can be `numeric`, `logical`, `character`, or `missing`. `value` must be a SummarizedExperiment object with dimensions, dimension names, and assay elements consistent with the subset `x[i,j]` being replaced.

`subset(x, subset, select)`: Create a subset of `x` using an expression `subset` referring to columns of `rowData(x)` and / or `select` referring to column names of `colData(x)`.

Additional subsetting accessors provide convenient access to `colData` columns

`x$name`, `x$name <- value` Access or replace column `name` in `x`.

`x[[i, ...]]`, `x[[i, ...]] <- value` Access or replace column `i` in `x`.

Combining

In the code snippets below, `x`, `y` and `...` are SummarizedExperiment objects (or derivatives) to be combined.

`rbind(...)`: `rbind` combines objects with the same samples but different features of interest (rows in `assays`). The colnames in `rowData(SummarizedExperiment)` must match or an error is thrown. Duplicate columns of `colData(SummarizedExperiment)` must contain the same data.

Data in `assays` are combined by name matching; if all assay names are `NULL` matching is by position. A mixture of names and `NULL` throws an error.

`metadata` from all objects are combined into a `list` with no name checking.

`cbind(...)`: `cbind` combines objects with the same features of interest but different samples (columns in `assays`). The colnames in `colData(SummarizedExperiment)` must match or an error is thrown. Duplicate columns of `rowData(SummarizedExperiment)` must contain the same data.

Data in `assays` are combined by name matching; if all assay names are `NULL` matching is by position. A mixture of names and `NULL` throws an error.

`metadata` from all objects are combined into a `list` with no name checking.

`combineRows(x, ..., use.names=TRUE, delayed=TRUE, fill=NA)`: `combineRows` acts like more flexible `rbind`, returning a SummarizedExperiment with features equal to the concatenation of features across all input objects. Unlike `rbind`, it permits differences in the number and identity of the columns, differences in the available `rowData` fields, and even differences in the available `assays` among the objects being combined.

If `use.names=TRUE`, each input object must have non-`NULL`, non-duplicated column names. These names do not have to be the same, or even shared, across the input objects. The column names of the returned `SummarizedExperiment` will be a union of the column names across all input objects. If a column is not present in an input, the corresponding assay and `colData` entries will be filled with `fill` and `NA`s, respectively, in the combined SummarizedExperiment.

If `use.names=FALSE`, all objects must have the same number of columns. The column names of the returned object is set to `colnames(x)`. Any differences in the column names between input objects are ignored.

Data in `assays` are combined by matching the names of the assays. If one input object does not contain a named assay present in other input objects, the corresponding assay entries in the returned object will be set to `fill`. If all assay names are `NULL`, matching is done by position. A mixture of named and unnamed assays will throw an error.

If `delayed=TRUE`, assay matrices are wrapped in `DelayedArray`s to avoid any extra memory allocation during the matrix `rbind`ing. Otherwise, the matrices are combined as-is; note that this may still return `DelayedMatrix`s if the inputs were also `DelayedMatrix` objects.

If any input is a `RangedSummarizedExperiment`, the returned object will also be a `RangedSummarizedExperiment`. The `rowRanges` of the returned object is set to the concatenation of the `rowRanges` of all inputs. If any input is a `SummarizedExperiment`, the returned `rowRanges` is converted into a `GRangesList` and the entries corresponding to the rows of the `SummarizedExperiment` are set to zero-length `GRanges`. If all inputs are `SummarizedExperiment` objects, a `SummarizedExperiment` is also returned.

`rowData` are combined using `combineRows` for `DataFrame` objects. It is not necessary for all input objects to have the same fields in their `rowData`; missing fields are filled with `NA`s for the corresponding rows in the returned object.

`metadata` from all objects are combined into a `list` with no name checking.

`combineCols(x, ..., use.names=TRUE, delayed=TRUE, fill=NA)`: `combineCols` acts like more flexible `cbind`, returning a SummarizedExperiment with columns equal to the concatenation of columns across all input objects. Unlike `cbind`, it permits differences in the number and identity of the rows, differences in the available `colData` fields, and even differences in the available `assays` among the objects being combined.

If `use.names=TRUE`, each input object must have non-`NULL`, non-duplicated row names. These names do not have to be the same, or even shared, across the input objects. The row names of the returned `SummarizedExperiment` will be a union of the row names across all input objects. If a row is not present in an input, the corresponding assay and `rowData` entries will be filled with `fill` and `NA`s, respectively, in the combined SummarizedExperiment.

If `use.names=FALSE`, all objects must have the same number of rows. The row names of the returned object is set to `rownames(x)`. Any differences in the row names between input objects are ignored.

Data in `assays` are combined by matching the names of the assays. If one input object does not contain a named assay present in other input objects, the corresponding assay entries in the returned object will be set to `fill`. If all assay names are `NULL`, matching is done by position. A mixture of named and unnamed assays will throw an error.

If delayed=TRUE, assay matrices are wrapped in DelayedArrays to avoid any extra memory allocation during the matrix rbinding. Otherwise, the matrices are combined as-is; note that this may still return DelayedMatrixs if the inputs were also DelayedMatrix objects.

If any input is a RangedSummarizedExperiment, the returned object will also be a RangedSummarizedExperiment. The rowRanges of the returned object is set to a merge of the rowRanges of all inputs, where the coordinates for each row are taken from the input object that contains that row. Any conflicting ranges for shared rows will raise a warning and all rowRanges information from the offending RangedSummarizedExperiment will be ignored. If any input is a SummarizedExperiment, the returned rowRanges is converted into a GRangesList and the entries corresponding to the unique rows of the SummarizedExperiment are set to zero-length GRanges. If all inputs are SummarizedExperiment objects, a SummarizedExperiment is also returned.

colData are combined using combineRows for DataFrame objects. It is not necessary for all input objects to have the same fields in their colData; missing fields are filled with NAs for the corresponding columns in the returned object.

metadata from all objects are combined into a list with no name checking.

Implementation and Extension

This section contains advanced material meant for package developers.

SummarizedExperiment is implemented as an S4 class, and can be extended in the usual way, using contains="SummarizedExperiment" in the new class definition.

In addition, the representation of the assays slot of SummarizedExperiment is as a virtual class Assays. This allows derived classes (contains="Assays") to implement alternative requirements for the assays, e.g., backed by file-based storage like NetCDF or the ff package, while re-using the existing SummarizedExperiment class without modification. See Assays for more information.

Author(s)

Martin Morgan; combineRows and combineCols by Aaron Lun

See Also

- RangedSummarizedExperiment objects.
- DataFrame, SimpleList, and Annotated objects in the **S4Vectors** package.
- The metadata and mcols accessors in the **S4Vectors** package.
- saveHDF5SummarizedExperiment and loadHDF5SummarizedExperiment in the **HDF5Array** package for saving/loading an HDF5-based SummarizedExperiment object to/from disk.
- The realize generic function in the **DelayedArray** package for more information about on-disk realization of objects carrying delayed operations.

Examples

```
nrows <- 200; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
```

```
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            colData=colData)
se0
dim(se0)
dimnames(se0)
assayNames(se0)
head(assay(se0))
assays(se0) <- endoapply(assays(se0), asinh)
head(assay(se0))

rowData(se0)
colData(se0)

se0[, se0$Treatment == "ChIP"]
subset(se0, select = Treatment == "ChIP")

## rbind() combines objects with the same samples but different
## features of interest:
se1 <- se0
se2 <- se1[1:50,]
rownames(se2) <- letters[seq_len(nrow(se2))]
cmb2 <- rbind(se1, se2)
dim(cmb2)
dimnames(cmb2)

## cbind() combines objects with the same features of interest
## but different samples:
se1 <- se0
se2 <- se1[,1:3]
colnames(se2) <- letters[seq_len(ncol(se2))]
cmb1 <- cbind(se1, se2)
dim(cmb1)
dimnames(cmb1)

## ---------------------------------------------------------------------
## ON-DISK REALIZATION
## ---------------------------------------------------------------------
library(DelayedArray)
setAutoRealizationBackend("HDF5Array")
cmb3 <- realize(cmb2)
assay(cmb3, withDimnames=FALSE)  # an HDF5Matrix object
```

Index