Package ‘SummarizedExperiment’

April 23, 2016

Title SummarizedExperiment container

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.

Version 1.0.2

Encoding UTF-8

Author Martin Morgan, Valerie Obenchain, Jim Hester, Hervé Pagès

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

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

Depends R (>= 3.2), methods, GenomicRanges (>= 1.22.1), Biobase

Imports BiocGenerics (>= 0.15.3), S4Vectors (>= 0.7.11), IRanges,
GenomeInfoDb

Suggests annotate, AnnotationDbi, GenomicFeatures, BiocStyle, knitr,
rmarkdown, jsonlite, rhdf5

VignetteBuilder knitr

License Artistic-2.0

Collate Assays-class.R SummarizedExperiment0-class.R
RangedSummarizedExperiment-class.R intra-range-methods.R
inter-range-methods.R coverage-methods.R findOverlaps-methods.R
nearest-methods.R makeSummarizedExperimentFromExpressionSet.R
readKallisto.R zzz.R

NeedsCompilation no

R topics documented:

`Assays-class` *Assays objects*

Description

The Assays virtual class and its methods provide a formal abstraction of the assays slot of SummarizedExperiment0 objects.

SimpleListAssays and ShallowSimpleListAssays are concrete subclasses of Assays with the latter 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<-` ``, `[[`, `` `[[<-` ``, `dim`, `[`, `` `[<-` ``, `rbind`, `cbind`.

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

IMPORTANT: Methods that return a modified Assays object (a.k.a. endomorphisms), that is, `[` as well as replacement methods `names<-`, `[[<-`, 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. Note that the full (deep) copy is not always 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 objects that need to be modified.

Assays has currently 3 implementations which are formalized by concrete subclasses SimpleListAssays, ShallowSimpleListAssays, and AssaysInEnv. ShallowSimpleListAssays is 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 SummarizedExperiment0 object contains an instance of ShallowSimpleListAssays.

Author(s)

Martin Morgan, `mtmorgan@fhcrc.org`

See Also

- SummarizedExperiment0 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)
a[[2]]
dim(a)

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

names(a)
names(a) <- c("a1", "a2")
names(a)
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])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)

cvg0 <- coverage(se0)
cvg0
stopifnot(identical(cvg0, coverage(rowRanges(se0))))
```

`findOverlaps-methods` *Finding overlapping ranges in RangedSummarizedExperiment objects*

Description

This man page documents the `findOverlaps` methods and family (i.e. `countOverlaps`, `overlapsAny`, and `subsetByOverlaps` methods) for RangedSummarizedExperiment objects.

Usage

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

## S4 method for signature 'RangedSummarizedExperiment,Vector'
countOverlaps(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    ignore.strand=FALSE)
## S4 method for signature 'Vector,RangedSummarizedExperiment'
countOverlaps(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment,Vector'
overlapsAny(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    ignore.strand=FALSE)
## S4 method for signature 'Vector,RangedSummarizedExperiment'
overlapsAny(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    ignore.strand=FALSE)

## S4 method for signature 'RangedSummarizedExperiment,Vector'
subsetByOverlaps(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    ignore.strand=FALSE)
## S4 method for signature 'Vector,RangedSummarizedExperiment'
subsetByOverlaps(query, subject,
    maxgap=0L, minoverlap=1L,
    type=c("any", "start", "end", "within", "equal"),
    algorithm=c("nclist", "intervaltree"),
    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, algorithm, 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])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)
se1 <- shift(se0, 100)

hits <- findOverlaps(se0, se1)
hits
stopifnot(identical(hits, findOverlaps(rowRanges(se0), rowRanges(se1))))
stopifnot(identical(hits, findOverlaps(se0, rowRanges(se1))))
stopifnot(identical(hits, findOverlaps(rowRanges(se0), se1)))
```

`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])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)
se1 <- shift(se0, 99*start(se0))

isDisjoint(se0)  # FALSE
isDisjoint(se1)  # TRUE

bins0 <- disjointBins(se0)
bins0
stopifnot(identical(bins0, disjointBins(rowRanges(se0))))

bins1 <- disjointBins(se1)
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'
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`	See ?`shift` in the **GenomicRanges** package.
`start, end, width, fix`	See ?`shift` in the **GenomicRanges** package.
`ignore.strand, both`	See ?`shift` in the **GenomicRanges** package.
`upstream, downstream`	See ?`shift` in the **GenomicRanges** package.
`keep.all.ranges`	See ?`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 ?`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])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)

se1 <- shift(se0, 1)
stopifnot(identical(
  rowRanges(se1),
  shift(rowRanges(se0), 1)
))

se2 <- narrow(se0, start=10, end=-15)
stopifnot(identical(
  rowRanges(se2),
  narrow(rowRanges(se0), start=10, end=-15)
))

se3 <- resize(se0, width=75)
stopifnot(identical(
  rowRanges(se3),
  resize(rowRanges(se0), width=75)
))

se4 <- flank(se0, width=20)
stopifnot(identical(
  rowRanges(se4),
  flank(rowRanges(se0), width=20)
))

se5 <- restrict(se0, start=200, end=700, keep.all.ranges=TRUE)
stopifnot(identical(
  rowRanges(se5),
  restrict(rowRanges(se0), start=200, end=700, keep.all.ranges=TRUE)
))
```

`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 RangedSummarizedExperiment
## ----------------------------------------------------------------------

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

# 2 equivalent ways of doing the naive coercion
makeSummarizedExperimentFromExpressionSet(sample.ExpressionSet)
as(sample.ExpressionSet, "RangedSummarizedExperiment")

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

# using the gene range mapper
makeSummarizedExperimentFromExpressionSet(sample.ExpressionSet,
                                   geneRangeMapper("TxDb.Hsapiens.UCSC.hg19.knownGene"))
```

```
## ---------------------------------------------------------------------
## GOING FROM RangedSummarizedExperiment TO ExpressionSet
## ---------------------------------------------------------------------

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

---

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

Arguments

`x, subject`	One of these two arguments must be a RangedSummarizedExperiment object.
`select, ignore.strand, algorithm`	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])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            rowRanges=rowRanges, colData=colData)
se1 <- shift(se0, 100)

res <- nearest(se0, se1)
res
```

```
stopifnot(identical(res, nearest(rowRanges(se0), rowRanges(se1))))
stopifnot(identical(res, nearest(se0, rowRanges(se1))))
stopifnot(identical(res, nearest(rowRanges(se0), se1)))

res <- nearest(se0)  # missing subject
res
stopifnot(identical(res, nearest(rowRanges(se0))))

hits <- nearest(se0, se1, select="all")
hits
stopifnot(identical(
  hits,
  nearest(rowRanges(se0), rowRanges(se1), select="all")
))
stopifnot(identical(
  hits,
  nearest(se0, rowRanges(se1), select="all")
))
stopifnot(identical(
  hits,
  nearest(rowRanges(se0), se1, 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 contains one or more assays, each represented by a matrix-like object of numeric or other mode.

RangedSummarizedExperiment is a subclass of SummarizedExperiment0 and, as such, all the methods documented in ?`SummarizedExperiment0` also work on a RangedSummarizedExperiment object. The methods documented below are additional methods that are specific to RangedSummarizedExperiment objects.

Usage

```
## Constructor

SummarizedExperiment(assays, ...)
## S4 method for signature 'SimpleList'
SummarizedExperiment(assays, rowRanges=GRangesList(),
    colData=DataFrame(), metadata=list(), exptData=SimpleList())
## S4 method for signature 'missing'
```

```
SummarizedExperiment(assays, ...)
## S4 method for signature 'list'
SummarizedExperiment(assays, ...)
## S4 method for signature 'matrix'
SummarizedExperiment(assays, ...)

## Accessors

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

## Subsetting

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

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

Arguments

`assays`	A `list` or `SimpleList` of matrix elements, or a `matrix`. 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`.
`rowRanges`	A GRanges or GRangesList object describing the ranges of interest. Names, if present, become the row names of the RangedSummarizedExperiment. The length of the GRanges or the GRangesList must equal the number of rows of the matrices in `assays`. If `rowRanges` is missing, a SummarizedExperiment0 instance is returned.
`colData`	An optional DataFrame describing the samples. Row names, if present, become the column names of the RangedSummarizedExperiment.
`metadata`	An optional `list` of arbitrary content describing the overall experiment.
`exptData`	Deprecated. Please use the `metadata` argument instead (see above).
`...`	For `SummarizedExperiment`, S4 methods `list` and `matrix`, arguments identical to those of the `SimpleList` method. For `rowRanges`, ignored.
`x`	A RangedSummarizedExperiment object. The `rowRanges` setter will also accept a SummarizedExperiment0 object and will first coerce it to RangedSummarizedExperiment before it sets `value` on it.
`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 with arguments outlined above.

Accessors

In the following code snippets, `x` is a RangedSummarizedExperiment 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: `compare`, `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.

`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(mcols(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.

Author(s)

Martin Morgan, mtmorgan@fhcrc.org

See Also

- SummarizedExperiment0 objects.
- shift, isDisjoint, coverage, findOverlaps, and nearest for more *GRanges compatibility methods*.
- GRanges objects in the **GenomicRanges** package.

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)
mcols(rse)  # same as mcols(rowRanges(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 SummarizedExperiment0:
se0 <- as(rse, "SummarizedExperiment0")
se0

as(se0, "RangedSummarizedExperiment")

## Setting rowRanges on a SummarizedExperiment0 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(mcols(se0), mcols(rse)))
stopifnot(identical(colData(se0), colData(rse)))
```

`readKallisto` *Input kallisto or kallisto bootstrap results.*

Description

`readKallisto` inputs several kallisto output files into a single `SummarizedExperiment` instance, with rows corresponding to estimated transcript abundance and columns to samples. `readKallistoBootstrap` inputs kallisto bootstrap replicates of a single sample into a matrix of transcript x bootstrap abundance estimates.

Usage

```
readKallisto(files,
    json = file.path(dirname(files), "run_info.json"),
    h5 = any(grepl("\\.h5$", files)), what = KALLISTO_ASSAYS,
    as = c("SummarizedExperiment", "list", "matrix"))

readKallistoBootstrap(file, i, j)
```

Arguments

`files`	character() paths to kallisto 'abundance.tsv' output files. The assumption is that files are organized in the way implied by kallisto, with each sample in a distinct directory, and the directory containing files abundance.tsv, run_info.json, and perhaps abundance.h5.
`json`	character() vector of the same length as `files` specifying the location of JSON files produced by kallisto and containing information on the run. The default assumes that json files are in the same directory as the corresponding abundance file.

`h5`	character() vector of the same length as `files` specifying the location of HDF5 files produced by kallisto and containing bootstrap estimates. The default assumes that HDF5 files are in the same directory as the corresponding abundance file.
`what`	character() vector of kallisto per-sample outputs to be input. See KALLISTO_ASSAYS for available values.
`as`	character(1) specifying the output format. See `Value` for additional detail.
`file`	character(1) path to a single HDF5 output file.
`i, j`	integer() vector of row (`i`) and column (`j`) indexes to input.

Value

A `SummarizedExperiment`, `list`, or `matrix`, depending on the value of argument `as`; by default a `SummarizedExperiment`. The `as="SummarizedExperiment"` `mcols(rowRanges(se))` the length of each transcript; `colData(se)` includes summary information on each sample, including the number of targets and bootstraps, the kallisto and index version, the start time and operating system call used to create the file. `assays()` contains one or more transcript x sample matrices of parameters estimated by kallisto (see `KALLISTO_ASSAYS`).

`as="list"` return value contains information simillar to `SummarizedExperiment` with row, column and assay data as elements of the list without coordination of row and column annotations into an integrated data container. `as="matrix"` returns the specified assay as a simple *R* matrix.

Author(s)

Martin Morgan `martin.morgan@roswellpark.org`

References

`http://pachterlab.github.io/kallisto` software for quantifying transcript abundance.

Examples

```
outputs <- system.file(package="SummarizedExperiment", "extdata",
    "kallisto")
files <- dir(outputs, pattern="abundance.tsv", full=TRUE, recursive=TRUE)
stopifnot(all(file.exists(files)))

## default: input 'est_counts'
(se <- readKallisto(files, as="SummarizedExperiment"))
str(readKallisto(files, as="list"))
str(readKallisto(files, as="matrix"))

## available assays
KALLISTO_ASSAYS
## one or more assay
readKallisto(files, what=c("tpm", "eff_length"))

## alternatively: read hdf5 files
files <- sub(".tsv", ".h5", files, fixed=TRUE)
```

```
readKallisto(files)

## input all bootstraps
xx <- readKallistoBootstrap(files[1])
ridx <- head(which(rowSums(xx) != 0), 3)
cidx <- c(1:5, 96:100)
xx[ridx, cidx]

## selective input of rows (transcripts) and/or bootstraps
readKallistoBootstrap(files[1], i=c(ridx, rev(ridx)), j=cidx)
```

SummarizedExperiment0-class *SummarizedExperiment0 objects*

Description

The SummarizedExperiment0 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 SummarizedExperiment0 object contains one or more assays, each represented by a matrix-like object of numeric or other mode.

Note that SummarizedExperiment0 is the parent of the RangedSummarizedExperiment class which means that all the methods documented below also work on a RangedSummarizedExperiment object.

Usage

```
## Constructor

# See ?RangedSummarizedExperiment for the constructor function.

## Accessors

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

## Quick colData access
## S4 method for signature 'SummarizedExperiment0'
x$name
## S4 replacement method for signature 'SummarizedExperiment0'
x$name <- value
## S4 method for signature 'SummarizedExperiment0,ANY,missing'
x[[i, j, ...]]
## S4 replacement method for signature 'SummarizedExperiment0,ANY,missing'
x[[i, j, ...]] <- value

## Subsetting

## S4 method for signature 'SummarizedExperiment0'
x[i, j, ..., drop=TRUE]
## S4 replacement method for signature
## 'SummarizedExperiment0,ANY,ANY,SummarizedExperiment0'
x[i, j] <- value

## Combining

## S4 method for signature 'SummarizedExperiment0'
cbind(..., deparse.level=1)
## S4 method for signature 'SummarizedExperiment0'
rbind(..., deparse.level=1)
```

Arguments

`x`	A SummarizedExperiment0 object.
`...`	For `assay`, `...` may contain `withDimnames`, which is forwarded to `assays`. For `cbind`, `rbind`, `...` contains SummarizedExperiment0 objects to be combined. For other accessors, ignored.
`i, j`	For `assay`, `assay<-`, `i` is an integer or numeric scalar; see 'Details' for additional constraints. For `[,SummarizedExperiment0`, `[,SummarizedExperiment0<-`, `i`, `j` are subscripts that can act to subset the rows and columns of `x`, that is the `matrix` elements of `assays`. For `[[,SummarizedExperiment0`, `[[<-,SummarizedExperiment0`, `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 dimnames should be applied to extracted assay elements. Setting `withDimnames=FALSE` increases the speed and memory efficiency with which assays are extracted. `withDimnames=TRUE` in the getter `assays<-` allows efficient complex assignments (e.g., updating names of assays, `names(assays(x, withDimnames=FALSE)) = ...` is more efficient than `names(assays(x)) = ...`); it does not influence actual assignment of dimnames to assays.

`drop`	A `logical(1)`, ignored by these methods.
`value`	An object of a class specified in the S4 method signature or as outlined in 'Details'.
`deparse.level`	See `?base::cbind` for a description of this argument.

Details

The SummarizedExperiment0 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.

The rows of a SummarizedExperiment0 object represent features of interest. Information about these features is stored in a DataFrame object, accessible using the function `mcols`. The DataFrame must have as many rows as there are rows in the SummarizedExperiment0 object, with each row of the DataFrame providing information on the feature in the corresponding row of the SummarizedExperiment0 object. Columns of the DataFrame represent different attributes of the features of interest, e.g., gene or transcript IDs, etc.

Each column of a SummarizedExperiment0 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 SummarizedExperiment0 object, with each row of the DataFrame providing information on the sample in the corresponding column of the SummarizedExperiment0 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 SummarizedExperiment0 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 SummarizedExperiment0 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 SummarizedExperiment0 in which they are stored. Row and column names of each matrix must either be `NULL` or match those of the SummarizedExperiment0 during construction. It is convenient for the elements of SimpleList of assays to be named.

Constructor

SummarizedExperiment0 instances are constructed using the `SummarizedExperiment` function documented in `?RangedSummarizedExperiment`.

Accessors

In the following code snippets, `x` is a SummarizedExperiment0 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 `i`th (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.

`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 SummarizedExperiment0.

`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 SummarizedExperiment0 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 SummarizedExperiment0 object with dimensions, dimension names, and assay elements consistent with the subset `x[i,j]` being replaced.

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, `...` are SummarizedExperiment0 objects to be combined.

`cbind(...)`: `cbind` combines objects with the same features of interest but different samples (columns in `assays`). The colnames in `colData(SummarizedExperiment0)` must match or an error is thrown. Duplicate columns of `mcols(SummarizedExperiment0)` 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.

`rbind(...)`: `rbind` combines objects with the same samples but different features of interest (rows in `assays`). The colnames in `mcols(SummarizedExperiment0)` must match or an error is thrown. Duplicate columns of `colData(SummarizedExperiment0)` 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.

Implementation and Extension

This section contains advanced material meant for package developers.

SummarizedExperiment0 is implemented as an S4 class, and can be extended in the usual way, using `contains="SummarizedExperiment0"` in the new class definition.

In addition, the representation of the `assays` slot of SummarizedExperiment0 is as a virtual class Assays. This allows derived classes (`contains="Assays"`) to easily implement alternative requirements for the assays, e.g., backed by file-based storage like NetCDF or the `ff` package, while re-using the existing SummarizedExperiment0 class without modification. See Assays for more information.

The current `assays` slot is implemented as a reference class that has copy-on-change semantics. This means that modifying non-assay slots does not copy the (large) assay data, and at the same time the user is not surprised by reference-based semantics. Updates to non-assay slots are very fast; updating the assays slot itself can be 5x or more faster than with an S4 instance in the slot. One useful technique when working with `assay` or `assays` function is use of the `withDimnames=FALSE` argument, which benefits speed and memory use by not copying dimnames from the row- and colData elements to each assay.

Author(s)

Martin Morgan, `mtmorgan@fhcrc.org`

See Also

- RangedSummarizedExperiment objects.
- DataFrame, SimpleList, and Annotated objects in the **S4Vectors** package.

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))

se0[, se0$Treatment == "ChIP"]

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

```
cmb1 <- cbind(se1, se2)
dim(cmb1)
dimnames(cmb1)

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

Index