Package 'XVector'

November 29, 2024

Title Foundation of external vector representation and manipulation in Bioconductor

Description Provides memory efficient S4 classes for storing sequences ``externally" (e.g. behind an R external pointer, or on disk).

biocViews Infrastructure, DataRepresentation

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

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

Version 0.46.0

License Artistic-2.0

Encoding UTF-8

Author Hervé Pagès and Patrick Aboyoun

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

Depends R (>= 4.0.0), methods, BiocGenerics (>= 0.37.0), S4Vectors (>= 0.27.12), IRanges (>= 2.23.9)

Imports methods, utils, tools, zlibbioc, BiocGenerics, S4Vectors, IRanges

LinkingTo S4Vectors, IRanges

Suggests Biostrings, drosophila2probe, RUnit

Collate io-utils.R RDS-random-access.R SharedVector-class.R SharedRaw-class.R SharedInteger-class.R SharedDouble-class.R XVector-class.R XRaw-class.R XInteger-class.R XDouble-class.R XVectorList-class.R XRawList-class.R XRawList-comparison.R XIntegerViews-class.R XDoubleViews-class.R OnDiskRaw-class.R RdaCollection-class.R RdsCollection-class.R intra-range-methods.R compact-methods.R reverse-methods.R slice-methods.R view-summarization-methods.R updateObject-methods.R zzz.R

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

git_branch RELEASE_3_20

git_last_commit 1c8f81d

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-11-28

Contents

`compact` *Object compaction*

Description

Compacting an object is modifying its internal representation in order to reduce its size in memory.

Usage

```
compact(x, check=TRUE, ...)

## Internal compact() support function. Not intended to be called
## directly:
xvcopy(x, start=NA, end=NA, width=NA, lkup=NULL, reverse=FALSE)
```

Arguments

`x`	For `compact`, the object to be compacted. `compact` accepts any R object. However, on most of them, `compact` won't do anything and will just return an object identical to `x`. See the Details section below. For `xvcopy`, a SharedVector, XVector, SharedVector_Pool, or XRawList vector.
`check`	After compacting the individual slots of an S4 object, this argument is passed to `` `slot<-` `` when replacing the original value of a slot with the compacted value.
`...`	Arguments to be passed to or from other methods.
`start`, `end`, `width`, `lkup`, `reverse`	For internal use.

Details

The internal reorganization of the object should be transparent to the user i.e. `compact(x)` should "look" the same as `x`, or, more precisely, `x` and `compact(x)` should be interchangeable anywhere in the user's code. However, because they have different internal representations, we generally don't expect `identical(x, compact(x))` to be TRUE, even though most of the times they will, because there are only very few types of objects that `compact` actually knows how to reorganize internally.

`compact` is a generic function.

Here is how the default method works. By default `compact(x)` is obtained by compacting all the "components" in `x`. Only 2 kinds of objects are considered to have "components": lists (the components are the list elements), and S4 objects (the components are the slots). The other objects are not considered to have components, so, by default, `compact` does nothing on them. In particular, it does nothing on environments. Also the attributes of an object (other than the slots of an S4 object) are not considered to be "components" and therefore are not compacted.

Note that, in the absence of specialized `compact` methods that actually know how to reorganize an object internally, the default method would visit the tree of all the components, sub-components, sub-sub-components etc of object `x` without actually modifying anything in `x`. So of course, specialized `compact` methods need to be defined for the objects that can *effectively* be compacted. Otherwise the `compact` function would be equivalent to the `identity` function!

At the moment, 2 specialized `compact` methods are defined (in addition to the default method): one for XVector objects, and one for XVectorList objects.

Value

An object equivalent to `x` but eventually smaller in memory.

Author(s)

H. Pagès

See Also

XVector-class, XVectorList-class, `subseq`, `object.size`, `save`

Examples

```
## We illustrate the use of compact() on an XInteger vector (XInteger
## is one of the 3 concrete subclasses of the XVector virtual class):
x <- XInteger(500000, sample(500000))

## subseq() does NOT copy the data stored in an XVector object:
y <- subseq(x, start=41, end=60)
x@shared
y@shared  # same address
object.size(x)
object.size(y)  # same size

## compact() copies the data, but only the data actually "used" by 'y':
y0 <- compact(y)
y0@shared  # new address
object.size(y0)  # much smaller now!

## Compaction is particularly relevant when saving an object with
## external references like 'y':
```

```
yfile <- file.path(tempdir(), "y.rda")
save(y, file=yfile)
file.info(yfile)$size

y0file <- file.path(tempdir(), "y0.rda")
save(y0, file=y0file)
file.info(y0file)$size
```

---

intra-range-methods *Intra range transformations of an XVectorList object*

Description

The *intra range transformations* are a set of generic functions defined in the **IRanges** package. Only 2 of them have methods for XVectorList objects: `narrow` and `threebands`. This man page describes those 2 methods only.

See ?`inter-range-methods` for more information.

Usage

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

## S4 method for signature 'XVectorList'
threebands(x, start=NA, end=NA, width=NA)
```

Arguments

x	An XVectorList object.
start, end, width	Vectors of integers, possibly with NAs. See the SEW (Start/End/Width) interface in the **IRanges** package for the details (?`solveUserSEW`).
use.names	TRUE or FALSE. Should names be preserved?

Details

`narrow` is equivalent to `subset` on an XVectorList object.

`threebands` extends the capability of `narrow` by returning the 3 XVectorList objects associated with the narrowing operation. The returned value `y` is a list of 3 XVectorList objects named `"left"`, `"middle"` and `"right"`. The middle component is obtained by calling `narrow` with the same arguments (except that names are dropped). The left and right components are also instances of the same class as `x` and they contain what has been removed on the left and right sides (respectively) of the original ranges during the narrowing.

Author(s)

H. Pagès

See Also

- intra-range-methods in the **IRanges** package for intra range transformations.
- `solveUserSEW` in the **IRanges** package for the SEW (Start/End/Width) interface.
- The XVectorList class.

Examples

```
## ---------------------------------------------------------------------
## narrow()
## ---------------------------------------------------------------------

#TODO: show examples

## ---------------------------------------------------------------------
## threebands()
## ---------------------------------------------------------------------

#TODO: show examples
```

`OnDiskRaw-class` *OnDiskRaw objects*

Description

THIS IS A WORK-IN-PROGRESS!

Author(s)

H. Pagès

`reverse-methods` *Reverse an XVector or XVectorList object*

Description

Methods for reversing an XVector or XVectorList object.

Usage

```
## S4 method for signature 'XVector'
reverse(x, ...)
## S4 method for signature 'XVectorList'
reverse(x, ...)
```

Arguments

`x`	An XVector or XVectorList object.
`...`	Additional arguments. Currently ignored.

Details

On an XVector object, `reverse` and `rev` are equivalent, i.e. they both reverse the order of their elements.

On an XVectorList object, `reverse` reverses each element individually, without modifying the top-level order of the elements. It's equivalent to, but more efficient than, doing `endoapply(x, rev)`.

Value

An object of the same class and length as the original object.

See Also

XVector-class, XVectorList-class, `endoapply`, `rev`

Examples

```
## On an XInteger object:
x <- as(12:-2, "XInteger")
reverse(x)

## On an XIntegerViews object:
v <- successiveViews(x, 1:5)
v
reverse(v)

## On an XVectorList object:
if (require(Biostrings) && require(drosophila2probe)) {
  library(Biostrings)
  library(drosophila2probe)
  probes <- DNAStringSet(drosophila2probe)
  reverse(probes)
}
```

`slice-methods` *Slice an XInteger or XDouble object*

Description

The `slice` methods for XInteger and XDouble objects create views corresponding to the indices where the data are within the specified bounds. The views are returned in a XIntegerViews or XDoubleViews object.

Usage

```
## S4 method for signature 'integer'
slice(x, lower=-.Machine$integer.max, upper=.Machine$integer.max)

## S4 method for signature 'XInteger'
slice(x, lower=-.Machine$integer.max, upper=.Machine$integer.max)

## S4 method for signature 'numeric'
slice(x, lower=-Inf, upper=Inf,
```

```
      includeLower=TRUE, includeUpper=TRUE, rangesOnly=FALSE)

## S4 method for signature 'XDouble'
slice(x, lower=-.Machine$double.xmax, upper=.Machine$double.xmax,
      includeLower=TRUE, includeUpper=TRUE, rangesOnly=FALSE)
```

Arguments

x	An XInteger or XDouble object. Alternatively, it can also be an integer or numeric vector.
lower, upper	The lower and upper bounds for the slice.
includeLower, includeUpper	Logical indicating whether or not the specified boundary is open or closed.
rangesOnly	A logical indicating whether or not to drop the original data from the output.

Value

An XIntegerViews or XDoubleViews object if `rangesOnly=FALSE`.

An IRanges object if `rangesOnly=TRUE`.

Author(s)

P. Aboyoun

See Also

- view-summarization-methods for summarizing the views returned by `slice`.
- slice-methods in the IRanges package for more `slice` methods.
- The XInteger, XIntegerViews, XDouble, and XDoubleViews classes.

Examples

```
vec <- as.integer(c(19, 5, 0, 8, 5))
slice(vec, lower=5, upper=8)

set.seed(0)
vec <- sample(24)
slice(vec, lower=4, upper=16)
```

`updateObject-methods` *Update an object of a class defined in the XVector package to its current class definition*

Description

The XVector package provides an extensive collection of `updateObject` methods for updating almost any instance of a class defined in the package.

Usage

```
## Showing usage of method defined for XVector objects only (usage
## is the same for all methods).

## S4 method for signature 'XVector'
updateObject(object, ..., verbose=FALSE)
```

Arguments

object	Object to be updated. Many (but not all) XVector classes are supported. If no specific method is available for the object, then the default method (defined in the BiocGenerics package) is used. See ?updateObject for a description of the default method.
..., verbose	See ?updateObject.

Value

Returns a valid instance of object.

See Also

updateObject

---

view-summarization-methods

Summarize views on an XInteger or XDouble object

Description

The viewMins, viewMaxs, viewSums, and viewMeans methods described here calculate respectively the minima, maxima, sums, and means of the views in an XIntegerViews or XDoubleViews object.

Usage

```
## "viewMins" methods:
## -------------------

## S4 method for signature 'XIntegerViews'
viewMins(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
viewMins(x, na.rm=FALSE)


## "viewMaxs" methods:
## -------------------

## S4 method for signature 'XIntegerViews'
viewMaxs(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
```

```
viewMaxs(x, na.rm=FALSE)


## "viewSums" methods:
## -------------------

## S4 method for signature 'XIntegerViews'
viewSums(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
viewSums(x, na.rm=FALSE)


## "viewMeans" methods:
## --------------------

## S4 method for signature 'XIntegerViews'
viewMeans(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
viewMeans(x, na.rm=FALSE)


## "viewWhichMins" methods:
## ------------------------

## S4 method for signature 'XIntegerViews'
viewWhichMins(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
viewWhichMins(x, na.rm=FALSE)


## "viewWhichMaxs" methods:
## ------------------------

## S4 method for signature 'XIntegerViews'
viewWhichMaxs(x, na.rm=FALSE)

## S4 method for signature 'XDoubleViews'
viewWhichMaxs(x, na.rm=FALSE)
```

Arguments

x	An XIntegerViews or XDoubleViews object.
na.rm	Logical indicating whether or not to include missing values in the results.

Value

A numeric vector of the length of x.

Note

For convenience, methods for `min`, `max`, `sum`, `mean`, `which.min` and `which.max` are provided as wrappers around the corresponding `view*` functions (which might be deprecated at some point).

Author(s)

P. Aboyoun

See Also

- slice-methods for slicing an XInteger or XDouble object.
- view-summarization-methods in the IRanges package for the view summarization generics.
- The XIntegerViews and XDoubleViews classes.

Examples

```
set.seed(0)
vec <- sample(24)
vec_views <- slice(vec, lower=4, upper=16)
vec_views

viewApply(vec_views, function(x) diff(as.integer(x)))

viewMins(vec_views)
viewMaxs(vec_views)

viewSums(vec_views)
viewMeans(vec_views)

viewWhichMins(vec_views)
viewWhichMaxs(vec_views)
```

XDoubleViews-class *The XDoubleViews class*

Description

The XDoubleViews class is the basic container for storing a set of views (start/end locations) on the same XDouble object.

Details

An XDoubleViews object contains a set of views (start/end locations) on the same XDouble object called "the subject numeric vector" or simply "the subject". Each view is defined by its start and end locations: both are integers such that start <= end. An XDoubleViews object is in fact a particular case of a Views object (the XDoubleViews class contains the Views class) so it can be manipulated in a similar manner: see ?`Views` for more information. Note that two views can overlap and that a view can be "out of limits" i.e. it can start before the first element of the subject or/and end after its last element.

Other methods

In the code snippets below, `x`, `object`, `e1` and `e2` are XDoubleViews objects, and `i` can be a numeric or logical vector.

`x[[i]]`**:** Extract a view as an XDouble object. `i` must be a single numeric value (a numeric vector of length 1). Can't be used for extracting a view that is "out of limits" (raise an error). The returned object has the same XDouble subtype as `subject(x)`.

`e1 == e2`**:** A vector of logicals indicating the result of the view by view comparison. The views in the shorter of the two XDoubleViews object being compared are recycled as necessary.

`e1 != e2`**:** Equivalent to `!(e1 == e2)`.

Author(s)

P. Aboyoun for the `XIntegerViews*` code, which was adapted to work over `XDouble`'s by S. Lianoglou

See Also

view-summarization-methods, Views-class, XDouble-class, XIntegerViews-class

Examples

```
  ## One standard way to create an XDoubleViews object is to use
  ## the Views() constructor:
  subject <- as(rnorm(6), "XDouble")
  v4 <- Views(subject, start=3:0, end=5:8)
  v4
  subject(v4)
  length(v4)
  start(v4)
  end(v4)
  width(v4)

  ## Attach a comment to views #3 and #4:
  names(v4)[3:4] <- "out of limits"
  names(v4)

  ## A more programatical way to "tag" the "out of limits" views:
  idx <- start(v4) < 1 | end(v4) > length(subject(v4))
  names(v4)[idx] <- "out of limits"

  ## Extract a view as an XDouble object:
  v4[[2]]

  ## It is an error to try to extract an "out of limits" view:
## Not run:
  v4[[3]] # Error!

## End(Not run)

  ## Here the first view doesn't even overlap with the subject:
  subject <- as(c(97, 97, 97, 45, 45, 98), "XDouble")
  Views(subject, start=-3:4, end=-3:4 + c(3:6, 6:3))

  ## Some fast view* functionalities:
```

```
x <- rnorm(55)
bounds <- IRanges(c(1, 11, 35, 20), width=c(5, 10, 15, 28))
v <- Views(x, bounds)
val <- viewMins(v)
expect <- sapply(1:length(bounds), function(i) {
  min(x[start(bounds)[i]:end(bounds[i])])
})
stopifnot(all(val == expect))
```

`XIntegerViews-class` *The XIntegerViews class*

Description

The XIntegerViews class is the basic container for storing a set of views (start/end locations) on the same XInteger object.

Details

An XIntegerViews object contains a set of views (start/end locations) on the same XInteger object called "the subject integer vector" or simply "the subject". Each view is defined by its start and end locations: both are integers such that start <= end. An XIntegerViews object is in fact a particular case of a Views object (the XIntegerViews class contains the Views class) so it can be manipulated in a similar manner: see ?`Views` for more information. Note that two views can overlap and that a view can be "out of limits" i.e. it can start before the first element of the subject or/and end after its last element.

Other methods

In the code snippets below, `x`, `object`, `e1` and `e2` are XIntegerViews objects, and `i` can be a numeric or logical vector.

`x[[i]]`**:** Extract a view as an XInteger object. `i` must be a single numeric value (a numeric vector of length 1). Can't be used for extracting a view that is "out of limits" (raise an error). The returned object has the same XInteger subtype as `subject(x)`.

`e1 == e2`**:** A vector of logicals indicating the result of the view by view comparison. The views in the shorter of the two XIntegerViews object being compared are recycled as necessary.

`e1 != e2`**:** Equivalent to `!(e1 == e2)`.

Author(s)

P. Aboyoun

See Also

view-summarization-methods, Views-class, XInteger-class, XDoubleViews-class

Examples

```
## One standard way to create an XIntegerViews object is to use
## the Views() constructor:
subject <- as(c(45, 67, 84, 67, 45, 78), "XInteger")
v4 <- Views(subject, start=3:0, end=5:8)
v4
subject(v4)
length(v4)
start(v4)
end(v4)
width(v4)

## Attach a comment to views #3 and #4:
names(v4)[3:4] <- "out of limits"
names(v4)

## A more programatical way to "tag" the "out of limits" views:
idx <- start(v4) < 1 | end(v4) > length(subject(v4))
names(v4)[idx] <- "out of limits"

## Extract a view as an XInteger object:
v4[[2]]

## It is an error to try to extract an "out of limits" view:
## Not run:
v4[[3]] # Error!

## End(Not run)

## Here the first view doesn't even overlap with the subject:
subject <- as(c(97, 97, 97, 45, 45, 98), "XInteger")
Views(subject, start=-3:4, end=-3:4 + c(3:6, 6:3))

## Views on a big XInteger subject:
subject <- XInteger(99999, sample(99, 99999, replace=TRUE) - 50)
v5 <- Views(subject, start=1:99*1000, end=1:99*1001)
v5
v5[-1]
v5[[5]]

## 31 adjacent views:
successiveViews(subject, 40:10)
```

`XRawList-class` *XRawList objects*

Description

THIS IS A WORK-IN-PROGRESS!

An XRawList object is *conceptually* a list of XRaw objects.

Author(s)

H. Pagès

See Also

XRaw-class, XVectorList-class

XRawList-comparison *Comparing and ordering the list elements of XRawList objects*

Description

Methods for comparing and ordering the elements in one or more XRawList objects.

Usage

```
## Element-wise (aka "parallel") comparison of 2 XRawList objects
## ---------------------------------------------------------------

## S4 method for signature 'XRawList,XRawList'
e1 == e2

## S4 method for signature 'XRawList,XRawList'
e1 <= e2

## duplicated()
## ------------

## S4 method for signature 'XRawList'
duplicated(x, incomparables=FALSE, ...)

## match()
## -------

## S4 method for signature 'XRawList,XRawList'
match(x, table, nomatch=NA_integer_, incomparables=NULL)

## order() and related methods
## ---------------------------

## S4 method for signature 'XRawList'
is.unsorted(x, na.rm=FALSE, strictly=FALSE)

## S4 method for signature 'XRawList'
order(..., na.last=TRUE, decreasing=FALSE,
          method=c("auto", "shell", "radix"))

## S4 method for signature 'XRawList'
rank(x, na.last=TRUE,
     ties.method=c("average", "first", "random", "max", "min"))

## Generalized element-wise (aka "parallel") comparison of 2 XRawList objects
## --------------------------------------------------------------------------

## S4 method for signature 'XRawList,XRawList'
pcompare(x, y)
```

Arguments

e1, e2, x, table, y	XRawList objects.
incomparables	Not supported.
...	For duplicated: currently no additional arguments are allowed. For order: additional XRawList objects used for breaking ties.
nomatch	The value to be returned in the case when no match is found. It is coerced to an integer.
na.rm, na.last, method	Ignored.
strictly	TRUE or FALSE. Should the check be for *strictly* increasing values?
decreasing	TRUE or FALSE.
ties.method	A character string specifying how ties are treated. Only "first" and "min" are supported for now.

Details

[TODO]

Author(s)

H. Pagès

See Also

- The XRawList class.
- Ranges-comparison in the IRanges package for comparing and ordering ranges.
- ==, duplicated, unique, match, %in%, order, sort, rank for general information about those functions.

Examples

```
## TODO
```

XVector internals *XVector internals*

Description

Objects, classes and methods defined in the XVector package that are not intended to be used directly.

XVector-class *XVector objects*

Description

The XVector virtual class is a general container for storing an "external vector". It inherits from the Vector class, which has a rich interface.

The following classes derive directly from the XVector class:

The XRaw class is a container for storing an "external raw vector" i.e. an external sequence of bytes (stored as char values at the C level).

The XInteger class is a container for storing an "external integer vector" i.e. an external sequence of integer values (stored as int values at the C level).

The XDouble class is a container for storing an "external double vector" i.e. an external sequence of numeric values (stored as double values at the C level).

Also the XString class defined in the Biostrings package.

The purpose of the X* containers is to provide a "pass by address" semantic and also to avoid the overhead of copying the sequence data when a linear subsequence needs to be extracted.

Additional Subsetting operations on XVector objects

In the code snippets below, x is an XVector object.

`subseq(x, start=NA, end=NA, width=NA)`**:** Extract the subsequence from `x` specified by `start`, `end` and `width`. The supplied start/end/width values are solved by a call to `solveUserSEW(length(x), start=start, end=end, width=width)` and therefore must be compliant with the rules of the SEW (Start/End/Width) interface (see `?solveUserSEW` for the details).

A note about performance: `subseq` does NOT copy the sequence data of an XVector object. Hence it's very efficient and is therefore the recommended way to extract a linear subsequence (i.e. a set of consecutive elements) from an XVector object. For example, extracting a 100Mb subsequence from Human chromosome 1 (a 250Mb DNAString object) with `subseq` is (almost) instantaneous and has (almost) no memory footprint (the cost in time and memory does not depend on the length of the original sequence or on the length of the subsequence to extract).

`subseq(x, start=NA, end=NA, width=NA) <- value`**:** Replace the subsequence specified on the left (i.e. the subsequence in `x` specified by `start`, `end` and `width`) by `value`. `value` must belong to the same class as `x`, or to one of its subclasses, or must be `NULL`. This replacement method can modify the length of `x`, depending on how the length of the left subsequence compares to the length of `value`. It can be used for inserting elements in `x` (specify an empty left subsequence for this) or deleting elements from `x` (use a `NULL` right value for this). Unlike the extraction method above, this replacement method always copies the sequence data of `x` (even for XVector objects). NOTE: Only works for XRaw (and derived) objects for now.

Author(s)

H. Pagès

See Also

Vector-class, DNAString-class, XVectorList-class, Views-class, `solveUserSEW`, `compact`

Examples

```
## ---------------------------------------------------------------------
## A. XRaw OBJECTS
## ---------------------------------------------------------------------

x1 <- XRaw(4)  # values are not initialized
x1
x2 <- as(c(255, 255, 199), "XRaw")
x2
y <- c(x1, x2, NULL, x1)  # NULLs are ignored
y
subseq(y, start=-4)
subseq(y, start=-4) <- x2
y

## ---------------------------------------------------------------------
## B. XInteger OBJECTS
## ---------------------------------------------------------------------

x3 <- XInteger(12, val=c(-1:10))
x3
length(x3)

## Subsetting
x4 <- XInteger(99999, val=sample(99, 99999, replace=TRUE) - 50)
x4
subseq(x4, start=10)
subseq(x4, start=-10)
subseq(x4, start=-20, end=-10)
subseq(x4, start=10, width=5)
subseq(x4, end=10, width=5)
subseq(x4, end=10, width=0)

x3[length(x3):1]
x3[length(x3):1, drop=FALSE]
```

`XVectorList-class` *XVectorList objects*

Description

THIS IS A WORK-IN-PROGRESS!!

An XVectorList object is *conceptually* a list of XVector objects.

Author(s)

H. Pagès

See Also

XVector-class, XRawList-class, `compact`

Index