Package 'GenomeInfoDb'

April 14, 2017

Title Utilities for manipulating chromosome and other 'seqname' identifiers

Description Contains data and functions that define and allow translation between different chromosome sequence naming conventions (e.g., ``chr1'' versus ``1''), including a function that attempts to place sequence names in their natural, rather than lexicographic, order.

Version 1.10.3

Encoding UTF-8

Author Sonali Arora, Martin Morgan, Marc Carlson, H. Pagès

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

biocViews Genetics, DataRepresentation, Annotation, GenomeAnnotation

Depends R (>= 3.1), methods, BiocGenerics (>= 0.13.8), S4Vectors (>= 0.9.25), IRanges (>= 1.99.26)

Imports stats, stats4, utils, RCurl

Suggests GenomicRanges, Rsamtools, GenomicAlignments, BSgenome, GenomicFeatures, BSgenome.Scerevisiae.UCSC.sacCer2, BSgenome.Celegans.UCSC.ce2, BSgenome.Hsapiens.NCBI.GRCh38, TxDb.Dmelanogaster.UCSC.dm3.ensGene, RUnit, BiocStyle, knitr

License Artistic-2.0

Collate utils.R rankSeqlevels.R assembly-utils.R available.species.R fetchExtendedChromInfoFromUCSC.R fetchSequenceInfo.R seqinfo.R seqlevelsStyle.R seqlevels-wrappers.R Seqinfo-class.R GenomeDescription-class.R mapGenomeBuilds.R zzz.R

VignetteBuilder knitr

Video http://youtu.be/wdEjCYSXa7w

NeedsCompilation no

R topics documented:

1

---

available.species   *Returns a data.frame that lists the available species strings and their*
                     taxonomy Ids.

Description

NCBI maintains a collection of unique taxonomy Ids and pairs these with associated genus and species designations. This function returns the set of pre-processed values that we use to check that something is a valid Taxonomy ID (or species name)

Usage

```
available.species()
```

Value

A data frame with 1 row per species designation and two columns. The 1st column is the taxonomy Id. The second columns is the species name.

Author(s)

Marc Carlson

Examples

```
## get the data
spec <- available.species()
tail(spec)
## which can then be searched etc.
spec[grepl('yoelii',spec$species),]
```

---

fetchExtendedChromInfoFromUCSC
                     Fetching chromosomes info for some of the UCSC genomes

---

Description

Fetch the chromosomes info for some UCSC genomes. Only supports hg38, hg19, hg18, panTro4, panTro3, panTro2, bosTau8, bosTau7, bosTau6, canFam3, canFam2, canFam1, musFur1, mm10, mm9, mm8, susScr3, susScr2, rn6, rheMac3, rheMac2, galGal4, galGal3, gasAcu1, danRer7, apiMel2, dm6, dm3, ce10, ce6, ce4, ce2, sacCer3, and sacCer2 at the moment.

Usage

```
fetchExtendedChromInfoFromUCSC(genome,
        goldenPath_url="http://hgdownload.cse.ucsc.edu/goldenPath",
        quiet=FALSE)
```

Arguments

genome
: A single string specifying the UCSC genome e.g. "sacCer3".

goldenPath_url
: A single string specifying the URL to the UCSC goldenPath location. This URL is used internally to build the full URL to the 'chromInfo' MySQL dump containing chromosomes information for genome. See Details section below.

quiet
: TRUE or FALSE (the default). If TRUE then some warnings are suppressed. See below for the details.

Details

Chromosomes information (e.g. names and lengths) for any UCSC genome is stored in the UCSC database in the 'chromInfo' table, and is normally available as a MySQL dump at:

```
goldenPath_url/<genome>/database/chromInfo.txt.gz
```

fetchExtendedChromInfoFromUCSC downloads and imports that table into a data frame, keeps only the UCSC_seqlevel and UCSC_seqlength columns (after renaming them), and adds the circular logical column.

Then, if this UCSC genome is based on an NCBI assembly (e.g. hg38 is based on GRCh38), the NCBI seqlevels and GenBank accession numbers are extracted from the NCBI assembly report and the UCSC seqlevels matched to them (using some guided heuristic). Finally the NCBI seqlevels and GenBank accession numbers are added to the returned data frame.

Value

A data frame with 1 row per seqlevel in the UCSC genome, and at least 3 columns:

- UCSC_seqlevel: Character vector with no NAs. This is the chrom field of the UCSC 'chromInfo' table for the genome. See Details section above.

- UCSC_seqlength: Integer vector with no NAs. This is the size field of the UCSC 'chromInfo' table for the genome. See Details section above.

- circular: Logical vector with no NAs. This knowledge is stored in the **GenomeInfoDb** package itself for the supported genomes.

If the UCSC genome is *not* based on an NCBI assembly (e.g. gasAcu1, ce10, sacCer2), there are no additional columns and a warning is emitted (unless quiet is set to TRUE). In this case, the rows are sorted by UCSC seqlevel rank as determined by rankSeqlevels().

If the UCSC genome is based on an NCBI assembly (e.g. sacCer3), the returned data frame has 3 additional columns:

- NCBI_seqlevel: Character vector. This information is obtained from the NCBI assembly report for the genome. Will contain NAs for UCSC seqlevels with no corresponding NCBI seqlevels (e.g. for chrM in hg18 or chrUextra in dm3), in which case fetchExtendedChromInfoFromUCSC emits a warning (unless quiet is set to TRUE).

- SequenceRole: Factor with levels assembled-molecule, alt-scaffold, unlocalized-scaffold, unplaced-scaffold, and pseudo-scaffold. For UCSC seqlevels with corresponding NCBI seqlevels this information is obtained from the NCBI assembly report. Otherwise it is obtained from a base of knowledge included in the **GenomeInfoDb** package. Can contain NAs but no warning is emitted in that case.
- GenBankAccn: Character vector. This information is obtained from the NCBI assembly report for the genome. Can contain NAs but no warning is emitted in that case.

In this case, the rows are sorted first by level in the SequenceRole column, that is, assembled-molecules first, then alt-scaffolds, etc, and NAs last. Then within each group they are sorted by UCSC se-qlevel rank as determined by rankSeqlevels().

Note

fetchExtendedChromInfoFromUCSC queries the UCSC Genome Browser as well as the FTP site at NCBI and thus requires internet access.

Only supports the hg38, hg19, hg18, panTro4, panTro3, panTro2, bosTau8, bosTau7, bosTau6, canFam3, canFam2, canFam1, musFur1, mm10, mm9, mm8, susScr3, susScr2, rn6, rheMac3, rheMac2, galGal4, galGal3, gasAcu1, danRer7, apiMel2, dm6, dm3, ce10, ce6, ce4, ce2, sacCer3, and sacCer2 genomes at the moment. More will come...

Author(s)

H. Pagès

See Also

- The seqlevels getter and setter.
- The rankSeqlevels function for ranking sequence names.
- The seqlevelsStyle getter and setter.
- The getBSgenome utility in the **BSgenome** package for searching the installed BSgenome data packages.

Examples

```
## All the examples below require internet access!

## ---------------------------------------------------------------------
## A. BASIC EXAMPLE
## ---------------------------------------------------------------------

## The sacCer3 UCSC genome is based on an NCBI assembly (RefSeq Assembly
## ID is GCF_000146045.2):
sacCer3_chrominfo <- fetchExtendedChromInfoFromUCSC("sacCer3")
sacCer3_chrominfo

## But the sacCer2 UCSC genome is not:
sacCer2_chrominfo <- fetchExtendedChromInfoFromUCSC("sacCer2")
sacCer2_chrominfo

## ---------------------------------------------------------------------
## B. USING fetchExtendedChromInfoFromUCSC() TO PUT UCSC SEQLEVELS ON
##    THE GRCh38 GENOME
## ---------------------------------------------------------------------
```

```
## Load the BSgenome.Hsapiens.NCBI.GRCh38 package:
library(BSgenome)
genome <- getBSgenome("GRCh38")  # this loads the
                                 # BSgenome.Hsapiens.NCBI.GRCh38 package

## A quick look at the GRCh38 seqlevels:
length(seqlevels(genome))
head(seqlevels(genome), n=30)

## Fetch the extended chromosomes info for the hg38 genome:
hg38_chrominfo <- fetchExtendedChromInfoFromUCSC("hg38")
dim(hg38_chrominfo)
head(hg38_chrominfo, n=30)

## 2 sanity checks:
##   1. Check the NCBI seqlevels:
stopifnot(setequal(hg38_chrominfo$NCBI_seqlevel, seqlevels(genome)))
##   2. Check that the sequence lengths in 'hg38_chrominfo' (which are
##      coming from the same 'chromInfo' table as the UCSC seqlevels)
##      are the same as in 'genome':
stopifnot(
  identical(hg38_chrominfo$UCSC_seqlength,
            unname(seqlengths(genome)[hg38_chrominfo$NCBI_seqlevel]))
)

## Extract the hg38 seqlevels and put the GRCh38 seqlevels on it as
## the names:
hg38_seqlevels <- setNames(hg38_chrominfo$UCSC_seqlevel,
                           hg38_chrominfo$NCBI_seqlevel)

## Set the hg38 seqlevels on 'genome':
seqlevels(genome) <- hg38_seqlevels[seqlevels(genome)]
head(seqlevels(genome), n=30)
```

GenomeDescription-class

GenomeDescription objects

Description

A GenomeDescription object holds the meta information describing a given genome.

Details

In general the user will not need to manipulate directly a GenomeDescription instance but will manipulate instead a higher-level object that belongs to a class that extends the GenomeDescription class. For example, the top-level object defined in any BSgenome data package is a BSgenome object and the BSgenome class contains the GenomeDescription class. Thus a BSgenome object is also a GenomeDescription object and can therefore be treated as such. In other words all the methods described below will work on it.

Accessor methods

In the code snippets below, `object` or `x` is a GenomeDescription object.

- `organism(object)`: Return the scientific name of the organism of the genome e.g. ″Homo sapiens″, ″Mus musculus″, ″Caenorhabditis elegans″, etc...

- `commonName(object)`: Return the common name of the organism of the genome e.g. ″Human″, ″Mouse″, ″Worm″, etc...

- `provider(x)`: Return the provider of this genome e.g. ″UCSC″, ″BDGP″, ″FlyBase″, etc...

- `providerVersion(x)`: Return the provider-side version of this genome. For example UCSC uses versions ″hg18″, ″hg17″, etc... for the different Builds of the Human genome.

- `releaseDate(x)`: Return the release date of this genome e.g. ″Mar. 2006″.

- `releaseName(x)`: Return the release name of this genome, which is generally made of the name of the organization who assembled it plus its Build version. For example, UCSC uses ″hg18″ for the version of the Human genome corresponding to the Build 36.1 from NCBI hence the release name for this genome is ″NCBI Build 36.1″.

- `bsgenomeName(x)`: Uses the meta information stored in x to make the name of the corresponding BSgenome data package (see the [available.genomes](#) function in the **BSgenome** package for details about the naming scheme used for those packages). Of course there is no guarantee that a package with that name actually exists.

- `seqinfo(x)` Gets information about the genome sequences. This information is returned in a [Seqinfo](#) object. Each part of the information can be retrieved separately with seqnames(x), seqlengths(x), and isCircular(x), respectively, as described below.

- `seqnames(x)` Gets the names of the genome sequences. seqnames(x) is equivalent to seqnames(seqinfo(x)).

- `seqlengths(x)` Gets the lengths of the genome sequences. seqlengths(x) is equivalent to seqlengths(seqinfo(x)).

- `isCircular(x)` Returns the circularity flags of the genome sequences. isCircular(x) is equivalent to isCircular(seqinfo(x)).

Author(s)

H. Pagès

See Also

- The [available.genomes](#) function and the [BSgenome](#) class in the **BSgenome** package.
- The [Seqinfo](#) class.

Examples

```
library(BSgenome.Celegans.UCSC.ce2)
class(Celegans)
is(Celegans, "GenomeDescription")
provider(Celegans)
seqinfo(Celegans)
gendesc <- as(Celegans, "GenomeDescription")
class(gendesc)
gendesc
provider(gendesc)
seqinfo(gendesc)
bsgenomeName(gendesc)
```

---

mapGenomeBuilds | *Mapping between UCSC and Ensembl Genome Builds*

Description

genomeBuilds lists the available genomes for a given species while mapGenomeBuilds maps between UCSC and Ensemble genome builds.

Usage

```
genomeBuilds(organism, style = c("UCSC", "Ensembl"))

mapGenomeBuilds(genome, style = c("UCSC", "Ensembl"))

listOrganisms()
```

Arguments

organism | A character vector of common names or organism
genome | A character vector of genomes equivalent to UCSC version or Ensembl Assemblies
style | A single value equivalent to "UCSC" or "Ensembl" specifying the output genome

Details

genomeBuilds lists the currently available genomes for a given list of organisms. The genomes can be shown as "UCSC" or "Ensembl" IDs determined by style. organism must be specified as a character vector and match common names (i.e "Dog", "Mouse") or organism name (i.e "Homo sapiens", "Mus musculus") . A list of available organisms can be shown using listOrganisms().

mapGenomeBuilds provides a mapping between "UCSC" builds and "Ensembl" builds. genome must be specified as a character vector and match either a "UCSC" ID or an "Ensembl" Id. genomeBuilds can be used to get a list of available build Ids for a given organism. NA's may be present in the output. This would occur when the current genome build removed a previously defined genome for an organism.

In both functions, if style is not specified, "UCSC" is used as default.

Value

A data.frame of builds for a given organism or genome in the specified style. If style == "UCSC", ucscID, ucscDate and ensemblID are given. If style == "Ensembl", ensemblID, ensemblVersion, ensemblDate, and ucscID are given. The opposing ID is given so that it is possible to distinguish between many-to-one mappings.

Author(s)

Valerie Obenchain <Valerie.Obenchain@roswellpark.org> and Lori Shepherd <Lori.Shepherd@roswellpark.org>

References

UCSC genome builds https://genome.ucsc.edu/FAQ/FAQreleases.html Ensembl genome builds http://useast.ensembl.org/info/website/archives/assembly.html

Examples

```
listOrganisms()

genomeBuilds("mouse")
genomeBuilds(c("Mouse", "dog", "human"), style="Ensembl")

mapGenomeBuilds(c("canFam3", "GRCm38", "mm9"))
mapGenomeBuilds(c("canFam3", "GRCm38", "mm9"), style="Ensembl")
```

---

rankSeqlevels   *Assign sequence IDs to sequence names*

Description

rankSeqlevels assigns a unique ID to each unique sequence name in the input vector. The returned IDs span 1:N where N is the number of unique sequence names in the input vector.

orderSeqlevels is similar to rankSeqlevels except that the returned vector contains the order instead of the rank.

Usage

```
rankSeqlevels(seqnames, X.is.sexchrom=NA)
orderSeqlevels(seqnames, X.is.sexchrom=NA)
```

Arguments

seqnames   A character vector or factor containing sequence names.

X.is.sexchrom   A logical indicating whether X refers to the sexual chromosome or to chromo-
                some with Roman Numeral X. If NA, rankSeqlevels does its best to "guess".

Value

An integer vector of the same length as seqnames that tries to reflect the "natural" order of seq-names, e.g.,chr1, chr2, chr3, . . .

The values in the returned vector span 1:N where N is the number of unique sequence names in the input vector.

Author(s)

H. Pagès for rankSeqlevels, orderSeqlevels added by Sonali Arora

See Also

- sortSeqlevels for sorting the sequence levels of an object in "natural" order.

Examples

```
library(BSgenome.Scerevisiae.UCSC.sacCer2)
rankSeqlevels(seqnames(Scerevisiae))
rankSeqlevels(seqnames(Scerevisiae)[c(1:5,5:1)])

newchr <- paste0("chr",c(1:3,6:15,4:5,16:22))
newchr
orderSeqlevels(newchr)
rankSeqlevels(newchr)
```

---

seqinfo   *Accessing/modifying sequence information*

Description

A set of generic functions for getting/setting/modifying the sequence information stored in an object.

Usage

```
seqinfo(x)
seqinfo(x, new2old=NULL, force=FALSE) <- value

seqnames(x)
seqnames(x) <- value

seqlevels(x)
seqlevels(x, force=FALSE) <- value
sortSeqlevels(x, X.is.sexchrom=NA)
seqlevelsInUse(x)
seqlevels0(x)

seqlengths(x)
seqlengths(x) <- value

isCircular(x)
isCircular(x) <- value

genome(x)
genome(x) <- value
```

Arguments

x
: The object from/on which to get/set the sequence information.

new2old
: The new2old argument allows the user to rename, drop, add and/or reorder the "sequence levels" in x.

  new2old can be NULL or an integer vector with one element per row in [Seqinfo](#) object value (i.e. new2old and value must have the same length) describing how the "new" sequence levels should be mapped to the "old" sequence levels, that is, how the rows in value should be mapped to the rows in seqinfo(x). The values in new2old must be >= 1 and <= length(seqinfo(x)). NAs are

allowed and indicate sequence levels that are being added. Old sequence levels
that are not represented in new2old will be dropped, but this will fail if those
levels are in use (e.g. if x is a GRanges object with ranges defined on those
sequence levels) unless force=TRUE is used (see below).

If new2old=NULL, then sequence levels can only be added to the existing ones,
that is, value must have at least as many rows as seqinfo(x) (i.e. length(values) >= length(seq
and also seqlevels(values)[seq_len(length(seqlevels(x)))] must be iden-
tical to seqlevels(x).

force
: Force dropping sequence levels currently in use. This is achieved by drop-
ping the elements in x where those levels are used (hence typically reducing
the length of x).

: Note that if x is a list-like object (e.g. GRangesList, GAlignmentPairs, or
GAlignmentsList), then any list element in x where at least one of the sequence
levels to drop is used is *fully* dropped. In other words, the seqlevels setter
always keeps or drops *full list elements* and never tries to change their content.
This guarantees that the geometry of the list elements is preserved, which is a
desirable property when they represent compound features (e.g. exons grouped
by transcript or paired-end reads). See below for an example.

value
: Typically a Seqinfo object for the seqinfo setter.

: Either a named or unnamed character vector for the seqlevels setter.

: A vector containing the sequence information to store for the other setters.

X.is.sexchrom
: A logical indicating whether X refers to the sexual chromosome or to chromo-
some with Roman Numeral X. If NA, sortSeqlevels does its best to "guess".

Details

The Seqinfo class plays a central role for the functions described in this man page because:

- All these functions (except seqinfo, seqlevelsInUse, and seqlevels0) work on a Seqinfo
  object.

- For classes that implement it, the seqinfo getter should return a Seqinfo object.

- Default seqlevels, seqlengths, isCircular, and genome getters and setters are provided.
  By default, seqlevels(x) does seqlevels(seqinfo(x)), seqlengths(x) does seqlengths(seqinfo(x)),
  isCircular(x) does isCircular(seqinfo(x)), and genome(x) does genome(seqinfo(x)).
  So any class with a seqinfo getter will have all the above getters work out-of-the-box. If, in
  addition, the class defines a seqinfo setter, then all the corresponding setters will also work
  out-of-the-box.

  Examples of containers that have a seqinfo getter and setter: the GRanges and GRanges-
  List classes in the **GenomicRanges** package; the SummarizedExperiment class in the **Sum-
  marizedExperiment** package; the GAlignments, GAlignmentPairs, and GAlignmentsList
  classes in the **GenomicAlignments** package; the TxDb class in the **GenomicFeatures** pack-
  age; the BSgenome class in the **BSgenome** package; etc...

The **GenomicRanges** package defines seqinfo and seqinfo<- methods for these low-level data
types: List, RangesList and RangedData. Those objects do not have the means to formally store
sequence information. Thus, the wrappers simply store the Seqinfo object within metadata(x).
Initially, the metadata is empty, so there is some effort to generate a reasonable default Seqinfo.
The names of any List are taken as the seqnames, and the universe of RangesList or RangedData
is taken as the genome.

Note

The full list of methods defined for a given generic can be seen with e.g. showMethods("seqinfo")
or showMethods("seqnames") (for the getters), and showMethods("seqinfo<-") or showMethods("seqnames<-")
(for the setters aka *replacement methods*). Please be aware that this shows only methods defined in
packages that are currently attached.

Author(s)

H. Pagès

See Also

- The seqlevelsStyle generic getter and setter.
- Seqinfo objects.
- GRanges and GRangesList objects in the **GenomicRanges** package.
- SummarizedExperiment objects in the **SummarizedExperiment** package.
- GAlignments, GAlignmentPairs, and GAlignmentsList objects in the **GenomicAlignments**
  package.
- TxDb objects in the **GenomicFeatures** package.
- BSgenome objects in the **BSgenome** package.
- seqlevels-wrappers for convenience wrappers to the seqlevels getter and setter.
- rankSeqlevels, on which sortSeqlevels is based.

Examples

```
## ---------------------------------------------------------------------
## A. MODIFY THE SEQLEVELS OF AN OBJECT
## ---------------------------------------------------------------------
## Overlap and matching operations between objects require matching
## seqlevels. Often the seqlevels in one must be modified to match
## the other. The seqlevels() function can rename, drop, add and reorder
## seqlevels of an object. Examples below are shown on TxDb
## and GRanges but the approach is the same for all objects that have
## a 'Seqinfo' class.

library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
seqlevels(txdb)

## Rename:
seqlevels(txdb) <- sub("chr", "", seqlevels(txdb))
seqlevels(txdb)

seqlevels(txdb) <- paste0("CH", seqlevels(txdb))
seqlevels(txdb)

seqlevels(txdb)[seqlevels(txdb) == "CHM"] <- "M"
seqlevels(txdb)

## Restore original seqlevels:
seqlevels(txdb) <- seqlevels0(txdb)
seqlevels(txdb)
```

```
gr <- GRanges(rep(c("chr2", "chr3", "chrM"), 2), IRanges(1:6, 10))

## Add:
seqlevels(gr) <- c("chr1", seqlevels(gr), "chr4")
seqlevels(gr)
seqlevelsInUse(gr)

## Reorder:
seqlevels(gr) <- rev(seqlevels(gr))
seqlevels(gr)

## Drop all unused seqlevels:
seqlevels(gr) <- seqlevelsInUse(gr)

## Drop some seqlevels in use:
seqlevels(gr, force=TRUE) <- setdiff(seqlevels(gr), "chr3")
gr

## Rename/Add/Reorder:
seqlevels(gr) <- c("chr1", chr2="chr2", chrM="Mitochondrion")
seqlevels(gr)

## ----------------------------------------------------------------------
## B. DROP SEQLEVELS OF A LIST-LIKE OBJECT
## ----------------------------------------------------------------------

grl0 <- GRangesList(GRanges("chr2", IRanges(3:2, 5)),
                    GRanges("chr5", IRanges(11, 18)),
                    GRanges(c("chr4", "chr2"), IRanges(7:6, 15)))
grl0

grl1 <- grl0
seqlevels(grl1, force=TRUE) <- c("chr2", "chr5")
grl1  # grl0[[3]] was fully dropped!

## If what is desired is to drop the first range in grl0[[3]] only, or,
## more generally speaking, to drop the ranges within each list element
## that are located on one of the seqlevels to drop, then do:
grl2 <- grl0[seqnames(grl0) %in% c("chr2", "chr5")]
grl2

## Note that the above subsetting doesn't drop any seqlevel:
seqlevels(grl2)

## To drop them (no need to use 'force=TRUE' anymore):
seqlevels(grl2) <- c("chr2", "chr5")
seqlevels(grl2)

## ----------------------------------------------------------------------
## C. SORT SEQLEVELS IN "NATURAL" ORDER
## ----------------------------------------------------------------------

sortSeqlevels(c("11", "Y", "1", "10", "9", "M", "2"))

seqlevels <- c("chrXI", "chrY", "chrI", "chrX", "chrIX", "chrM", "chrII")
sortSeqlevels(seqlevels)
```

```
sortSeqlevels(seqlevels, X.is.sexchrom=TRUE)
sortSeqlevels(seqlevels, X.is.sexchrom=FALSE)

seqlevels <- c("chr2RHet", "chr4", "chrUextra", "chrYHet",
               "chrM", "chrXHet", "chr2LHet", "chrU",
               "chr3L", "chr3R", "chr2R", "chrX")
sortSeqlevels(seqlevels)

gr <- GRanges()
seqlevels(gr) <- seqlevels
sortSeqlevels(gr)

## ---------------------------------------------------------------------
## D. SUBSET OBJECTS BY SEQLEVELS
## ---------------------------------------------------------------------

tx <- transcripts(txdb)
seqlevels(tx)

## Drop 'M', keep all others.
seqlevels(tx, force=TRUE) <- seqlevels(tx)[seqlevels(tx) != "M"]
seqlevels(tx)

## Drop all except 'ch3L' and 'ch3R'.
seqlevels(tx, force=TRUE) <- c("ch3L", "ch3R")
seqlevels(tx)

## ---------------------------------------------------------------------
## E. FINDING METHODS
## ---------------------------------------------------------------------

showMethods("seqinfo")
showMethods("seqinfo<-")

showMethods("seqnames")
showMethods("seqnames<-")

showMethods("seqlevels")
showMethods("seqlevels<-")

if (interactive()) {
  library(GenomicRanges)
  ?`GRanges-class`
}
```

Seqinfo-class   *Seqinfo objects*

Description

A Seqinfo object is a table-like object that contains basic information about a set of genomic sequences. The table has 1 row per sequence and 1 column per sequence attribute. Currently the only attributes are the length, circularity flag, and genome provenance (e.g. hg19) of the sequence, but more attributes might be added in the future as the need arises.

Details

> Typically Seqinfo objects are not used directly but are part of higher level objects. Those higher
> level objects will generally provide a `seqinfo` accessor for getting/setting their Seqinfo component.

Constructor

> `Seqinfo(seqnames, seqlengths=NA, isCircular=NA, genome=NA)`: Creates a Seqinfo object.

Accessor methods

> In the code snippets below, `x` is a Seqinfo object.
>
> `length(x)`: Return the number of sequences in `x`.
>
> `seqnames(x)`, `seqnames(x) <- value`: Get/set the names of the sequences in `x`. Those names
> must be non-NA, non-empty and unique. They are also called the *sequence levels* or the *keys*
> of the Seqinfo object.
>
> Note that, in general, the end-user should not try to alter the sequence levels with `seqnames(x) <- value`.
> The recommended way to do this is with `seqlevels(x) <- value` as described below.
>
> `names(x)`, `names(x) <- value`: Same as `seqnames(x)` and `seqnames(x) <- value`.
>
> `seqlevels(x)`: Same as `seqnames(x)`.
>
> `seqlevels(x) <- value`: Can be used to rename, drop, add and/or reorder the sequence levels.
> `value` must be either a named or unnamed character vector. When `value` has names, the
> names only serve the purpose of mapping the new sequence levels to the old ones. Otherwise
> (i.e. when `value` is unnamed) this mapping is implicitly inferred from the following rules:
>
> (1) If the number of new and old levels are the same, and if the positional mapping between
> the new and old levels shows that some or all of the levels are being renamed, and if the levels
> that are being renamed are renamed with levels that didn't exist before (i.e. are not present in
> the old levels), then `seqlevels(x) <- value` will just rename the sequence levels. Note that
> in that case the result is the same as with `seqnames(x) <- value` but it's still recommended
> to use `seqlevels(x) <- value` as it is safer.
>
> (2) Otherwise (i.e. if the conditions for (1) are not satisfied) `seqlevels(x) <- value` will
> consider that the sequence levels are not being renamed and will just perform `x <- x[value]`.
> See below for some examples.
>
> `seqlengths(x)`, `seqlengths(x) <- value`: Get/set the length for each sequence in `x`.
>
> `isCircular(x)`, `isCircular(x) <- value`: Get/set the circularity flag for each sequence in `x`.
>
> `genome(x)`, `genome(x) <- value`: Get/set the genome identifier or assembly name for each
> sequence in `x`.

Subsetting

> In the code snippets below, `x` is a Seqinfo object.
>
> `x[i]`: A Seqinfo object can be subsetted only by name i.e. `i` must be a character vector. This is a
> convenient way to drop/add/reorder the rows (aka the sequence levels) of a Seqinfo object.
> See below for some examples.

Coercion

> In the code snippets below, `x` is a Seqinfo object.
>
> `as.data.frame(x)`: Turns `x` into a data frame.

Combining Seqinfo objects

There are no c or rbind method for Seqinfo objects. Both would be expected to just append the rows in y to the rows in x resulting in an object of length length(x) + length(y). But that would tend to break the constraint that the seqnames of a Seqinfo object must be unique keys.

So instead, a merge method is provided.

In the code snippet below, x and y are Seqinfo objects.

merge(x, y): Merge x and y into a single Seqinfo object where the keys (aka the seqnames) are union(seqnames(x), seqnames(y)). If a row in y has the same key as a row in x, and if the 2 rows contain compatible information (NA values are compatible with anything), then they are merged into a single row in the result. If they cannot be merged (because they contain different seqlengths, and/or circularity flags, and/or genome identifiers), then an error is raised. In addition to check for incompatible sequence information, merge(x, y) also compares seqnames(x) with seqnames(y) and issues a warning if each of them has names not in the other. The purpose of these checks is to try to detect situations where the user might be combining or comparing objects based on different reference genomes.

intersect(x, y): Finds the intersection between two Seqinfo objects by merging them and subsetting for the intersection of their sequence names. This makes it easy to avoid warnings about the objects not being subsets of each other during overlap operations.

Author(s)

H. Pagès

See Also

- seqinfo
- The fetchExtendedChromInfoFromUCSC utility function that is used behind the scene to make a Seqinfo object for a supported genome (see examples below).

Examples

```
## ---------------------------------------------------------------------
## A. MAKING A Seqinfo OBJECT FOR A SUPPORTED GENOME
## ---------------------------------------------------------------------

if (interactive()) {
  ## This uses fetchExtendedChromInfoFromUCSC() behind the scene and
  ## thus requires internet access. See ?fetchExtendedChromInfoFromUCSC
  ## for the list of UCSC genomes that are currently supported.
  Seqinfo(genome="hg38")
  Seqinfo(genome="bosTau8")
  Seqinfo(genome="canFam3")
  Seqinfo(genome="musFur1")
  Seqinfo(genome="mm10")
  Seqinfo(genome="rn6")
  Seqinfo(genome="galGal4")
  Seqinfo(genome="dm6")
  Seqinfo(genome="sacCer3")
}

## ---------------------------------------------------------------------
## B. BASIC MANIPULATION OF A Seqinfo OBJECT
```

```
## ---------------------------------------------------------------------

## Note that all the arguments (except 'genome') must have the
## same length. 'genome' can be of length 1, whatever the lengths
## of the other arguments are.
x <- Seqinfo(seqnames=c("chr1", "chr2", "chr3", "chrM"),
             seqlengths=c(100, 200, NA, 15),
             isCircular=c(NA, FALSE, FALSE, TRUE),
             genome="toy")
x

## Accessors:
length(x)
seqnames(x)
names(x)
seqlevels(x)
seqlengths(x)
isCircular(x)
genome(x)

## Get a compact summary:
summary(x)

## Subset by names:
x[c("chrY", "chr3", "chr1")]

## Rename, drop, add and/or reorder the sequence levels:
xx <- x
seqlevels(xx) <- sub("chr", "ch", seqlevels(xx))  # rename
xx
seqlevels(xx) <- rev(seqlevels(xx))  # reorder
xx
seqlevels(xx) <- c("ch1", "ch2", "chY")  # drop/add/reorder
xx
seqlevels(xx) <- c(chY="Y", ch1="1", "22")  # rename/reorder/drop/add
xx

## ---------------------------------------------------------------------
## C. MERGING 2 Seqinfo OBJECTS
## ---------------------------------------------------------------------

y <- Seqinfo(seqnames=c("chr3", "chr4", "chrM"),
             seqlengths=c(300, NA, 15))
y

## This issues a warning:
merge(x, y)  # rows for chr3 and chrM are merged

## To get rid of the above warning, either use suppressWarnings() or
## set the genome on 'y':
suppressWarnings(merge(x, y))
genome(y) <- genome(x)
merge(x, y)

## Note that, strictly speaking, merging 2 Seqinfo objects is not
## a commutative operation, i.e., in general 'z1 <- merge(x, y)'
## is not identical to 'z2 <- merge(y, x)'. However 'z1' and 'z2'
```

```
## are guaranteed to contain the same information (i.e. the same
## rows, but typically not in the same order):
merge(y, x)

## This contradicts what 'x' says about circularity of chr3 and chrM:
isCircular(y)[c("chr3", "chrM")] <- c(TRUE, FALSE)
y
if (interactive()) {
  merge(x, y)  # raises an error
}

## Sanity checks:
stopifnot(identical(x, merge(x, Seqinfo())))
stopifnot(identical(x, merge(Seqinfo(), x)))
stopifnot(identical(x, merge(x, x)))
```

---

seqlevels-wrappers   *Convenience wrappers to the seqlevels() getter and setter*

Description

Keep, drop or rename seqlevels in objects with a [Seqinfo](#) class.

Usage

```
keepSeqlevels(x, value)
dropSeqlevels(x, value)
renameSeqlevels(x, value)
restoreSeqlevels(x)
keepStandardChromosomes(x, species=NULL)
```

Arguments

x	Any object having a [Seqinfo](#) class in which the seqlevels will be kept, dropped or renamed.
value	A named or unnamed character vector.
	Names are ignored by keepSeqlevels and dropSeqlevels. Only the values in the character vector dictate which seqlevels to keep or drop.
	In the case of renameSeqlevels, the names are used to map new sequence levels to the old (names correspond to the old levels). When value is unnamed, the replacement vector must the same length and in the same order as the original seqlevels(x).
species	The genus and species of the organism. Supported species can be seen with names(genomeStyles()).

Details

Matching and overlap operations on range objects often require that the seqlevels match before a comparison can be made (e.g., findOverlaps). keepSeqlevels, dropSeqlevels and renameSeqlevels are high-level convenience functions that wrap the low-level seqlevels setter.

- keepSeqlevels, dropSeqlevels: Subsetting operations that modify the size of x. keepSeqlevels keeps only the seqlevels in value and removes all others. dropSeqlevels drops the levels in value and retains all others. If value does not match any seqlevels in x an empty object is returned.

- renameSeqlevels: Rename the seqlevels in x to those in value. If value is a named character vector, the names are used to map the new seqlevels to the old. When value is unnamed, the replacement vector must be the same length and in the same order as the original seqlevels(x).

- restoreSeqlevels: Perform seqlevels(txdb) <- seqlevels0(txdb), that is, restore the seqlevels in x back to the original values. Applicable only when x is a TxDb object.

- keepStandardChromosomes: Subsetting operation that returns only the 'standard' Chromosomes. We define 'standard chromosomes' as those which represent sequences in the assembly that are not scaffolds. Also referred to as 'assembly molecule' on NCBI. The function attempts to detect the seqlevel style and if more than one style is matched, e.g., 'UCSC' and 'Ensembl', the first is chosen.

  x must have a Seqinfo object. species can be specified as a character string; supported species are listed with names(genomeStyles()).

  When x contains seqlevels from multiple organisms all those considered standard will be kept. For example, if seqlevels are "chr1" and "chr3R" from human and fly both will be kept. If species="Homo sapiens" is specified then only "chr1" is kept.

Value

The x object with seqlevels removed or renamed. If x has no seqlevels (empty object) or no replacement values match the current seqlevels in x the unchanged x is returned.

Author(s)

Valerie Obenchain, Sonali Arora

See Also

- seqinfo ## Accessing sequence information
- Seqinfo ## The Seqinfo class

Examples

```
## ---------------------------------------------------------------------
## keepSeqlevels / dropSeqlevels
## ---------------------------------------------------------------------

## GRanges / GAlignments:

library(GenomicRanges)
gr <- GRanges(c("chr1", "chr1", "chr2", "chr3"), IRanges(1:4, width=3))
seqlevels(gr)
## Keep only 'chr1'
chr1 <- keepSeqlevels(gr, "chr1")
## Drop 'chr1'. Both 'chr2' and 'chr3' are kept.
chr2 <- dropSeqlevels(gr, "chr1")

library(Rsamtools)  # for the ex1.bam file
```

```
library(GenomicAlignments)  # for readGAlignments()

fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
gal <- readGAlignments(fl)
## If 'value' is named, the names are ignored.
seq2 <- keepSeqlevels(gal, c(foo="seq2"))
seqlevels(seq2)

## GRangesList / GAlignmentsList:

grl <- split(gr, as.character(seqnames(gr)))
dropSeqlevels(grl, c("chr1", "chr2"))
galist <- split(gal, as.character(seqnames(gal)))
keepSeqlevels(galist, "seq2")

## TxDb:

## A TxDb cannot be directly subset with 'keepSeqlevels'
## and 'dropSeqlevels'.
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
seqlevels(txdb)
## Not run:
keepSeqlevels(txdb, "chr2L") ## fails

## End(Not run)

## GRanges or GRangesLists extracted from the TxDb can be subset.
txbygene <- transcriptsBy(txdb, "gene")
seqlevels(txbygene)
chr2L <- keepSeqlevels(txbygene, "chr2L")
seqlevels(chr2L)

## ---------------------------------------------------------------------
## renameSeqlevels
## ---------------------------------------------------------------------

## GAlignments:

seqlevels(gal)
## Rename 'seq2' to 'chr2' with a named vector.
gal2a <- renameSeqlevels(gal, c(seq2="chr2"))
## Rename 'seq2' to 'chr2' with an unnamed vector that includes all
## seqlevels as they appear in the object.
gal2b <- renameSeqlevels(gal, c("seq1", "chr2"))
## Names that do not match existing seqlevels are ignored.
## This attempt at renaming does nothing.
gal3 <- renameSeqlevels(gal, c(foo="chr2"))
identical(seqlevels(gal), seqlevels(gal3))

## TxDb:

seqlevels(txdb)
## When the seqlevels of a TxDb are renamed, all future
## extractions reflect the modified seqlevels.
renameSeqlevels(txdb, sub("chr", "CH", seqlevels(txdb)))
renameSeqlevels(txdb, c(CHM="M"))
```

```
seqlevels(txdb)

transcripts <- transcripts(txdb)
identical(seqlevels(txdb), seqlevels(transcripts))

## ---------------------------------------------------------------------
## restoreSeqlevels
## ---------------------------------------------------------------------

## Restore seqlevels in a TxDb to original values.
txdb <- restoreSeqlevels(txdb)
seqlevels(txdb)

## ---------------------------------------------------------------------
## keepStandardChromosomes
## ---------------------------------------------------------------------

gr <- GRanges(c(paste0("chr",c(1:3)), "chr1_gl000191_random",
            "chr1_gl000192_random"), IRanges(1:5, width=3))
gr

## GRanges
keepStandardChromosomes(gr)

## GRangesList
grl <- split(gr,seqnames(gr))
keepStandardChromosomes(grl)

## NCBI
worm <- GRanges(c("I", "II", "wrong", "X", "MT"), IRanges(1:5, width=5))
keepStandardChromosomes(worm)

## UCSC
seqlevelsStyle(worm) <- "UCSC"
keepStandardChromosomes(worm)

## Ensembl
seqlevelsStyle(worm) <- "Ensembl"
keepStandardChromosomes(worm)
```

---

seqlevelsStyle	*Conveniently rename the seqlevels of an object according to a given style*

Description

The `seqlevelsStyle` getter and setter can be used to get the current seqlevels style of an object and to rename its seqlevels according to a given style.

Usage

```
seqlevelsStyle(x)
seqlevelsStyle(x) <- value
```

```
## Related low-level utilities:
genomeStyles(species)
extractSeqlevels(species, style)
extractSeqlevelsByGroup(species, style, group)
mapSeqlevels(seqnames, style, best.only=TRUE, drop=TRUE)
seqlevelsInGroup(seqnames, group, species, style)
```

Arguments

x	The object from/on which to get/set the seqlevels style. x must have a `seqlevels` method or be a `character` vector.
value	A single character string that sets the seqlevels style for x.
species	The genus and species of the organism in question separated by a single space. Don't forget to capitalize the genus.
style	a character vector with a single element to specify the style.
group	Group can be 'auto' for autosomes, 'sex' for sex chromosomes/allosomes, 'circular' for circular chromosomes. The default is 'all' which returns all the chromosomes.
best.only	if TRUE (the default), then only the "best" sequence renaming maps (i.e. the rows with less NAs) are returned.
drop	if TRUE (the default), then a vector is returned instead of a matrix when the matrix has only 1 row.
seqnames	a character vector containing the labels attached to the chromosomes in a given genome for a given style. For example : For *Homo sapiens*, NCBI style - they are "1","2","3",...,"X","Y","MT"

Details

`seqlevelsStyle(x)`, `seqlevelsStyle(x) <- value`: Get the current seqlevels style of an object, or rename its seqlevels according to the supplied style.

`genomeStyles`: Different organizations have different naming conventions for how they name the biologically defined sequence elements (usually chromosomes) for each organism they support. The Seqnames package contains a database that defines these different conventions.

genomeStyles() returns the list of all supported seqname mappings, one per supported organism. Each mapping is represented as a data frame with 1 column per seqname style and 1 row per chromosome name (not all chromosomes of a given organism necessarily belong to the mapping).

genomeStyles(species) returns a data.frame only for the given organism with all its supported seqname mappings.

`extractSeqlevels`: Returns a character vector of the seqnames for a single style and species.

`extractSeqlevelsByGroup`: Returns a character vector of the seqnames for a single style and species by group. Group can be 'auto' for autosomes, 'sex' for sex chromosomes/ allosomes, 'circular' for circular chromosomes. The default is 'all' which returns all the chromosomes.

`mapSeqlevels`: Returns a matrix with 1 column per supplied sequence name and 1 row per sequence renaming map compatible with the specified style. If `best.only` is `TRUE` (the default), only the "best" renaming maps (i.e. the rows with less NAs) are returned.

`seqlevelsInGroup`: It takes a character vector along with a group and optional style and species.If group is not specified , it returns "all" or standard/top level seqnames. Returns a character vector of seqnames after subsetting for the group specified by the user. See examples for more details.

Value

For seqlevelsStyle: A single string containing the style of the seqlevels in x, or a character vector containing the styles of the seqlevels in x if the current style cannot be determined unambiguously. Note that this information is not stored in x but inferred by looking up a seqlevels style database stored inside the **GenomeInfoDb** package.

For extractSeqlevels , extractSeqlevelsByGroup and seqlevelsInGroup: A character vector of seqlevels for given supported species and group.

For mapSeqlevels: A matrix with 1 column per supplied sequence name and 1 row per sequence renaming map compatible with the specified style.

For genomeStyle: If species is specified returns a data.frame containg the seqlevels style and its mapping for a given organism. If species is not specified, a list is returned with one list per species containing the seqlevels style with the corresponding mappings.

Author(s)

Sonali Arora, Martin Morgan, Marc Carlson, H. Pagès

Examples

```
## ---------------------------------------------------------------------
## seqlevelsStyle() getter and setter
## ---------------------------------------------------------------------

## character vectors:
x <- paste0("chr", 1:5)
seqlevelsStyle(x)
seqlevelsStyle(x) <- "NCBI"
x

## GRanges:
library(GenomicRanges)
gr <- GRanges(rep(c("chr2", "chr3", "chrM"), 2), IRanges(1:6, 10))

seqlevelsStyle(gr)
seqlevelsStyle(gr) <- "NCBI"
gr

seqlevelsStyle(gr)
seqlevelsStyle(gr) <- "dbSNP"
gr

seqlevelsStyle(gr)
seqlevelsStyle(gr) <- "UCSC"
gr

## ---------------------------------------------------------------------
## Related low-level utilities
## ---------------------------------------------------------------------

## Genome styles:
names(genomeStyles())
genomeStyles("Homo_sapiens")
"UCSC" %in% names(genomeStyles("Homo_sapiens"))
```

```
## Extract seqlevels based on species, style and group:
## The 'group' argument can be 'sex', 'auto', 'circular' or 'all'.

## All:
extractSeqlevels(species="Drosophila_melanogaster", style="Ensembl")

## Sex chromosomes:
extractSeqlevelsByGroup(species="Homo_sapiens", style="UCSC", group="sex")

## Autosomes:
extractSeqlevelsByGroup(species="Homo_sapiens", style="UCSC", group="auto")


## Identify which seqnames belong to a particular 'group':
newchr <- paste0("chr",c(1:22,"X","Y","M","1_gl000192_random","4_ctg9"))
seqlevelsInGroup(newchr, group="sex")

newchr <- as.character(c(1:22,"X","Y","MT"))
seqlevelsInGroup(newchr, group="all","Homo_sapiens","NCBI")

## Identify which seqnames belong to a species and style:
seqnames <- c("chr1","chr9", "chr2", "chr3", "chr10")
all(seqnames %in% extractSeqlevels("Homo_sapiens", "UCSC"))

## Find mapped seqlevelsStyles for exsiting seqnames:
mapSeqlevels(c("chrII", "chrIII", "chrM"), "NCBI")
mapSeqlevels(c("chrII", "chrIII", "chrM"), "Ensembl")
```

---

speciesMap	*Species Mapper object and/or list of valid taxonomy Ids*

Description

A data frame containing a mapping between NCBI Taxonomy IDs and species OR a list of valid taxomony IDs (pre-subsetted and cleaned up from that same data.frame).

The reason why is because sometimes we need the entire frame, but other times we only need the list of integers, (which is much smaller and loads much more quickly)

The specData object is an even more complex data.frame that returns genus and species separately.

Usage

```
data(speciesMap)
data(validTaxIds)
```

Format

A data frame with 1416382 observations on the following 2 variables.

taxon a numeric vector

species a character vector

Author(s)

Marc Carlson

Examples

```
data(speciesMap)
sapply(speciesMap, class)    #       taxon     species
                             #     "integer" "character"
subset(speciesMap, species=="Homo sapiens")$taxon # [1] 9606
```

Index