Package: GenomeInfoDb (via r-universe)

November 3, 2026

Title Utilities for manipulating chromosome names, including modifying them to follow a particular naming style

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.

biocViews Genetics, DataRepresentation, Annotation, GenomeAnnotation

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

Video http://youtu.be/wdEjCYSXa7w

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

Version 1.49.2

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.0.0), methods, BiocGenerics (>= 0.59.3), S4Vectors (>= 0.47.6), IRanges (>= 2.41.1), Seqinfo (>= 0.99.2)

Imports stats, utils, UCSC.utils

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

VignetteBuilder knitr

Collate utils.R fetch_table_dump_from_Ensembl_FTP.R list_ftp_dir.R NCBI-utils.R UCSC-utils.R Ensembl-utils.R getChromInfoFromNCBI.R getChromInfoFromUCSC.R getChromInfoFromEnsembl.R loadTaxonomyDb.R mapGenomeBuilds.R seqlevelsStyle.R seqlevels-wrappers.R zzz.R

Config/pak/sysreqs libssl-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-11-03 00:16:43 UTC

RemoteUrl https://github.com/bioc/GenomeInfoDb

RemoteRef HEAD

RemoteSha 1972d871ed24379539fd84f1af6baf183600fc0f

Contents

getChromInfoFromEnsembl 2
getChromInfoFromNCBI 8
getChromInfoFromUCSC 11
loadTaxonomyDb 15
mapGenomeBuilds 16
NCBI-utils 17
seqlevels-wrappers 19
seqlevelsStyle 23

Index **28**

`getChromInfoFromEnsembl`

Get chromosome information for an Ensembl species

Description

`getChromInfoFromEnsembl` returns chromosome information like sequence names, lengths and circularity flags for a given Ensembl species e.g. Human, Cow, Saccharomyces cerevisiae, etc...

Usage

```
getChromInfoFromEnsembl(species,
                        release=NA, division=NA, use.grch37=FALSE,
                        assembled.molecules.only=FALSE,
                        include.non_ref.sequences=FALSE,
                        include.contigs=FALSE,
                        include.clones=FALSE,
                        map.NCBI=FALSE,
                        recache=FALSE,
                        as.Seqinfo=FALSE)
```

Arguments

`species`	A single string specifying the name of an Ensembl species e.g. `"human"`, `"hsapiens"`, or `"Homo sapiens"`. Case is ignored. Alternatively the name of an assembly (e.g. `"GRCh38"`) or a taxonomy id (e.g. 9606) can be supplied.

release — The Ensembl release to query e.g. 89. If set to NA (the default), the current release is used.

division — NA (the default) or one of the EnsemblGenomes marts i.e. "bacteria", "fungi", "metazoa", "plants", or "protists".

use.grch37 — NOT TESTED YET!

TRUE or FALSE (the default).

assembled.molecules.only — NOT IMPLEMENTED YET!

include.non_ref.sequences — TODO: DOCUMENT THIS!

include.contigs — Whether or not sequences for which coord_system is set to "contig" should be included. They are not included by default. Note that the dataset for Human contains more than one hundred thousands *contigs*!

include.clones — Whether or not sequences for which coord_system is set to "clone" should be included. They are not included by default. Note that the dataset for Human contains more than one hundred thousands *clones*!

map.NCBI — TRUE or FALSE (the default).

If TRUE then NCBI chromosome information is bound to the result. This information is retrieved from NCBI by calling getChromInfoFromNCBI on the NCBI assembly that the Ensembl species is based on. Then the data frame returned by getChromInfoFromNCBI ("NCBI chrom info") is *mapped* and bound to the data frame returned by getChromInfoFromEnsembl ("Ensembl chrom info"). This "map and bind" operation is similar to a JOIN in SQL.

Note that not all rows in the "Ensembl chrom info" data frame are necessarily mapped to a row in the "NCBI chrom info" data frame. For the unmapped rows the NCBI columns in the final data frame are filled with NAs (LEFT JOIN in SQL).

The primary use case for using map.NCBI=TRUE is to map Ensembl sequence names to NCBI sequence names.

recache — getChromInfoFromEnsembl uses a cache mechanism so the chromosome information of a given dataset only gets downloaded once during the current R session (note that the caching is done in memory so cached information does NOT persist across sessions). Setting recache to TRUE forces a new download (and recaching) of the chromosome information for the specified dataset.

as.Seqinfo — TRUE or FALSE (the default). If TRUE then a Seqinfo object is returned instead of a data frame. Note that only the name, length, and circular columns of the data frame are used to make the Seqinfo object. All the other columns are ignored (and lost).

Details

COMING SOON...

Value

For `getChromInfoFromEnsembl`: By default, a 7-column data frame with columns:

1. `name`: character.
2. `length`: integer.
3. `coord_system`: factor.
4. `synonyms`: list.
5. `toplevel`: logical.
6. `non_ref`: logical.
7. `circular`: logical.

and with attribute `species_info` which contains details about the species that was used to obtaine the data.

If `map.NCBI` is `TRUE`, then 7 "NCBI columns" are added to the result:

- `NCBI.SequenceName`: character.
- `NCBI.SequenceRole`: factor.
- `NCBI.AssignedMolecule`: factor.
- `NCBI.GenBankAccn`: character.
- `NCBI.Relationship`: factor.
- `NCBI.RefSeqAccn`: character.
- `NCBI.AssemblyUnit`: factor.

Note that the names of the "NCBI columns" are those returned by `getChromInfoFromNCBI` but with the `NCBI.` prefix added to them.

Author(s)

H. Pagès

See Also

- `getChromInfoFromNCBI` and `getChromInfoFromUCSC` for getting chromosome information for an NCBI assembly or UCSC genome.
- Seqinfo objects.

Examples

```
## ---------------------------------------------------------------------
## A. BASIC EXAMPLES
## ---------------------------------------------------------------------

## Internet access required!

## === Worm ===
## https://uswest.ensembl.org/Caenorhabditis_elegans
```

```
celegans <- getChromInfoFromEnsembl("celegans")
attr(celegans, "species_info")

getChromInfoFromEnsembl("celegans", as.Seqinfo=TRUE)

celegans <- getChromInfoFromEnsembl("celegans", map.NCBI=TRUE)

## === Yeast ===
## https://uswest.ensembl.org/Saccharomyces_cerevisiae

scerevisiae <- getChromInfoFromEnsembl("scerevisiae")
attr(scerevisiae, "species_info")

getChromInfoFromEnsembl("scerevisiae", as.Seqinfo=TRUE)

scerevisiae <- getChromInfoFromEnsembl("scerevisiae", map.NCBI=TRUE)

## Arabidopsis thaliana:
athaliana <- getChromInfoFromEnsembl("athaliana", division="plants",
                                     map.NCBI=TRUE)
attr(athaliana, "species_info")

## ------------------------------------------------------------------------
## Temporary stuff that needs to go away...
## ------------------------------------------------------------------------

## TODO: Check all species for which an NCBI assembly is registered!
## Checked so far (with current Ensembl release i.e. 99):
## - celegans       OK
## - scerevisiae    OK
## - athaliana      OK
## - btaurus        OK
## - sscrofa        OK

## Not run:
## WORK IN PROGRESS!!!
library(GenomeInfoDb)

.do_join <- GenomeInfoDb:::.do_join
.map_Ensembl_seqlevels_to_NCBI_seqlevels <-
    GenomeInfoDb:::.map_Ensembl_seqlevels_to_NCBI_seqlevels

.map_Ensembl_seqlevels_to_NCBI_seqlevels(
    paste0("ENS_", 1:26),
    CharacterList(c(list(c(aa="INSDC1", bb="GNBK7"), c("INSDC2", "RefSeq3")),
                    rep(list(NULL), 23), list("NCBI_7"))),
    paste0("NCBI_", 1:10),
    paste0("GNBK", c(1:8, NA, 9)),
    c(paste0("REFSEQ", c(1:7, 1, 1)), NA),
    verbose=TRUE
)
```

```
map_to_NCBI <- function(Ensembl_chrom_info, NCBI_chrom_info,
                        special_mappings=NULL)
{
    .map_Ensembl_seqlevels_to_NCBI_seqlevels(
         Ensembl_chrom_info[ , "name"],
         Ensembl_chrom_info[ , "synonyms"],
         NCBI_chrom_info[ , "SequenceName"],
         NCBI_chrom_info[ , "GenBankAccn"],
         NCBI_chrom_info[ , "RefSeqAccn"],
         special_mappings=special_mappings,
         verbose=TRUE)
}

## ---------------------------------------------------------------------
## Human
## https://uswest.ensembl.org/Homo_sapiens/
## Based on GRCh38.p13 (GCA_000001405.28)

## Return 944 rows
human_chrom_info <- getChromInfoFromEnsembl("hsapiens")
#                 1 id: 131550  <- ref chromosome
# CHR_HSCHR1_1_CTG3 id: 131561  <- non-ref chromosome
#     HSCHR1_1_CTG3 id: 131562  <- scaffold (no scaffold is non_ref)

## Map to NCBI
## Summary:
## - 639/640 NCBI sequences are reverse-mapped.
## - Restricted mapping is one-to-one.
GRCh38.p13 <- getChromInfoFromNCBI("GRCh38.p13")
L2R <- map_to_NCBI(human_chrom_info, GRCh38.p13)
## The only sequence in GRCh38.p13 that cannot be mapped to Ensembl is
## HG2139_PATCH (was introduced in GRCh38.p2)! Why? What's special about
## this patch?
GRCh38.p13$mapped <- tabulate(L2R, nbins=nrow(GRCh38.p13)) != 0L
table(GRCh38.p13$SequenceRole, GRCh38.p13$mapped)
#                       FALSE TRUE
#   assembled-molecule      0   25
#   alt-scaffold            0  261
#   unlocalized-scaffold    0   42
#   unplaced-scaffold       0  127
#   pseudo-scaffold         0    0
#   fix-patch               1  112
#   novel-patch             0   72
human_chrom_info <- .do_join(human_chrom_info, GRCh38.p13, L2R)
table(human_chrom_info$SequenceRole, human_chrom_info$toplevel)
#                      FALSE TRUE
#  assembled-molecule      0   25
#  alt-scaffold          261    0
#  unlocalized-scaffold    0   42
#  unplaced-scaffold       0  127
#  pseudo-scaffold         0    0
#  fix-patch             112    0
#  novel-patch            72    0
```

```
#hsa_seqlevels <- readRDS("hsapiens_gene_ensembl_txdb_seqlevels.rds")

## ------------------------------------------------------------------------
## Mouse
## https://uswest.ensembl.org/Mus_musculus/
## Based on GRCm38.p6 (GCA_000001635.8)

## Return 258 rows
mouse_chrom_info <- getChromInfoFromEnsembl("mmusculus")

## Map to NCBI
## Summary:
## - 139/239 NCBI sequences are reverse-mapped.
## - Restricted mapping is NOT one-to-one: 2 Ensembl sequences (NC_005089.1
##   and MT) are both mapped to NCBI MT.
GRCm38.p6 <- getChromInfoFromNCBI("GRCm38.p6")
L2R <- map_to_NCBI(mouse_chrom_info, GRCm38.p6)
## 100 sequences in GRCm38.p6 are not mapped:
GRCm38.p6$mapped <- tabulate(L2R, nbins=nrow(GRCm38.p6)) != 0L
table(GRCm38.p6$SequenceRole, GRCm38.p6$mapped)
#                        FALSE TRUE
#   assembled-molecule       0   22
#   alt-scaffold            99    0
#   unlocalized-scaffold     0   22
#   unplaced-scaffold        0   22
#   pseudo-scaffold          0    0
#   fix-patch                1   64
#   novel-patch              0    9
## OK so Ensembl doesn't include the alt-scaffolds for Mouse. BUT WHAT
## HAPPENED TO THIS ONE fix-patch SEQUENCE (MG4237_PATCH) THAT IS NOT
## MAPPED? Found it in seq_region_synonym table! It's seq_region_id=100405.
## Hey but that seq_region_id is **NOT** in the seq_region table!!! THIS
## VIOLATES FOREIGN KEY CONSTRAINT!!!!
mouse_chrom_info <- .do_join(mouse_chrom_info, GRCm38.p6, L2R)
## Ensembl does NOT comsider NC_005089.1 (duplicate entry for MT) toplevel:
mouse_chrom_info[mouse_chrom_info$SequenceName
#            name length coord_system                      synonyms toplevel
# 184 NC_005089.1  16299     scaffold                                 FALSE
# 201          MT  16299   chromosome NC_005089.1, chrM, AY172335.1     TRUE
#     SequenceName GenBankAccn  RefSeqAccn
# 184           MT  AY172335.1 NC_005089.1
# 201           MT  AY172335.1 NC_005089.1

## ------------------------------------------------------------------------
## Rat
## https://uswest.ensembl.org/Rattus_norvegicus/
## Based on Rnor_6.0 (GCA_000001895.4)

# Return 1418 rows
rat_chrom_info <- getChromInfoFromEnsembl("rnorvegicus")

## Map to NCBI
```

```
## Summary:
## - 955/955 NCBI sequences are reverse-mapped.
## - Reverse mapping is one-to-many: 2 Ensembl sequences (NC_001665.2 and MT)
##   are mapped to NCBI MT.
Rnor_6.0 <- getChromInfoFromNCBI("Rnor_6.0")
L2R <- map_to_NCBI(rat_chrom_info, Rnor_6.0)
rat_chrom_info <- .do_join(rat_chrom_info, Rnor_6.0, L2R)

## Ensembl does NOT comsider NC_001665.2 (duplicate entry for MT) toplevel:
rat_chrom_info[rat_chrom_info$SequenceName
#             name length coord_system                      synonyms toplevel
# 1417 NC_001665.2  16313     scaffold                                 FALSE
# 1418          MT  16313   chromosome NC_001665.2, AY172581.1, chrM     TRUE
#      SequenceName GenBankAccn  RefSeqAccn
# 1417           MT  AY172581.1 NC_001665.2
# 1418           MT  AY172581.1 NC_001665.2

table(rat_chrom_info$SequenceRole, rat_chrom_info$toplevel)
#                        FALSE TRUE
#   assembled-molecule       1   23
#   alt-scaffold             0    0
#   unlocalized-scaffold     0  354
#   unplaced-scaffold        0  578
#   pseudo-scaffold          0    0
#   fix-patch                0    0
#   novel-patch              0    0

## End(Not run)
```

`getChromInfoFromNCBI` *Get chromosome information for an NCBI assembly*

Description

`getChromInfoFromNCBI` returns chromosome information like sequence names, lengths and circularity flags for a given NCBI assembly e.g. for GRCh38, ARS-UCD1.2, R64, etc...

Note that `getChromInfoFromNCBI` behaves slightly differently depending on whether the assembly is *registered* in the **GenomeInfoDb** package or not. See below for the details.

Use `registered_NCBI_assemblies` to list all the NCBI assemblies currently registered in the **GenomeInfoDb** package.

Usage

```
getChromInfoFromNCBI(assembly,
                     assembled.molecules.only=FALSE,
                     assembly.units=NULL,
                     recache=FALSE,
                     as.Seqinfo=FALSE)

registered_NCBI_assemblies(organism=NA)
```

Arguments

`assembly`	A single string specifying the name of an NCBI assembly (e.g. `"GRCh38"`). Alternatively, an assembly accession (GenBank or RefSeq) can be supplied (e.g. `"GCF_000001405.12"`).
`assembled.molecules.only`	If `FALSE` (the default) then chromosome information is returned for all the sequences in the assembly (unless `assembly.units` is specified, see below), that is, for all the chromosomes, plasmids, and scaffolds. If `TRUE` then chromosome information is returned only for the *assembled molecules*. These are the chromosomes (including the mitochondrial chromosome) and plasmids only. No scaffolds.
`assembly.units`	If `NULL` (the default) then chromosome information is returned for all the sequences in the assembly (unless `assembled.molecules.only` is set to `TRUE`, see above), that is, for all the chromosomes, plasmids, and scaffolds. `assembly.units` can be set to a character vector containing the names of *Assembly Units* (e.g. `"non-nuclear"`) in which case chromosome information is returned only for the sequences that belong to these Assembly Units.
`recache`	`getChromInfoFromNCBI` uses a cache mechanism so the chromosome information of a given assembly only gets downloaded once during the current R session (note that the caching is done in memory so cached information does NOT persist across sessions). Setting `recache` to `TRUE` forces a new download (and recaching) of the chromosome information for the specified assembly.
`as.Seqinfo`	`TRUE` or `FALSE` (the default). If `TRUE` then a Seqinfo object is returned instead of a data frame. Note that only the `SequenceName`, `SequenceLength`, and `circular` columns of the data frame are used to make the Seqinfo object. All the other columns are ignored (and lost).
`organism`	When `organism` is specified, `registered_NCBI_assemblies()` will only return the subset of assemblies that are registered for that organism. `organism` must be specified as a single string and will be used to perform a search (with `grep()`) on the `"organism"` column of the data frame returned by `registered_NCBI_assemblies()`. The search is case-insensitive.

Details

*** *registered* vs *unregistered* NCBI assemblies ***

- All NCBI assemblies can be looked up by assembly accession (GenBank or RefSeq) but only *registered* assemblies can also be looked up by assembly name.
- For *registered* assemblies, the returned circularity flags are guaranteed to be accurate. For *unregistered* assemblies, a heuristic is used to determine the circular sequences.

Please contact the maintainer of the **GenomeInfoDb** package to request registration of additional assemblies.

*** Offline mode ***

The **GenomeInfoDb** package includes a small database that contains chromosome information for the most commonly used NCBI and UCSC genome assemblies e.g. GRCh38.p14, GRCh38.p13,

GRCm39, hg38, mm39, etc... This makes calls like `getChromInfoFromNCBI("GRCh38.p14")` or `getChromInfoFromUCSC("hg38")` work offline. Plus now they are fast and reliable.

Note that calling `getChromInfoFromNCBI()` with `recache=TRUE` will trigger retrieval of the chromosome info from NCBI, and will issue a warning if this info no longer matches the chromosome info stored in the package. Please open an issue on GitHub if you get such warning or want to request adding chromosome information of your favorite NCBI or UCSC genome assembly to **GenomeInfoDb**'s internal database.

Value

For `getChromInfoFromNCBI`: By default, a 10-column data frame with columns:

1. `SequenceName`: character.
2. `SequenceRole`: factor.
3. `AssignedMolecule`: factor.
4. `GenBankAccn`: character.
5. `Relationship`: factor.
6. `RefSeqAccn`: character.
7. `AssemblyUnit`: factor.
8. `SequenceLength`: integer. Note that this column **can** contain NAs! For example this is the case in assembly Amel_HAv3.1 where the length of sequence MT is missing or in assembly Release 5 where the length of sequence Un is missing.
9. `UCSCStyleName`: character.
10. `circular`: logical.

For `registered_NCBI_assemblies`: A data frame summarizing all the NCBI assemblies currently *registered* in the **GenomeInfoDb** package.

Author(s)

H. Pagès

See Also

- `getChromInfoFromUCSC` for getting chromosome information for a UCSC genome.
- `getChromInfoFromEnsembl` for getting chromosome information for an Ensembl species.
- Seqinfo objects.

Examples

```
## All registered NCBI assemblies for Triticum aestivum (bread wheat):
registered_NCBI_assemblies("tri")[1:4]

## All registered NCBI assemblies for Homo sapiens:
registered_NCBI_assemblies("homo")[1:4]

## Internet access required!
```

```
getChromInfoFromNCBI("GRCh37")
getChromInfoFromNCBI("GRCh37", as.Seqinfo=TRUE)
getChromInfoFromNCBI("GRCh37", assembled.molecules.only=TRUE)

## The GRCh38.p14 assembly only adds "patch sequences" to the GRCh38
## assembly:
GRCh38 <- getChromInfoFromNCBI("GRCh38")
table(GRCh38$SequenceRole)
GRCh38.p14 <- getChromInfoFromNCBI("GRCh38.p14")
table(GRCh38.p14$SequenceRole)  # 254 patch sequences (164 fix + 90 novel)

## All registered NCBI assemblies for Arabidopsis thaliana:
registered_NCBI_assemblies("arabi")[1:4]
getChromInfoFromNCBI("TAIR10.1")
getChromInfoFromNCBI("TAIR10.1", assembly.units="non-nuclear")

## Sanity checks:
idx <- match(GRCh38$SequenceName, GRCh38.p14$SequenceName)
stopifnot(!anyNA(idx))
tmp1 <- GRCh38.p14[idx, ]
rownames(tmp1) <- NULL
tmp2 <- GRCh38.p14[-idx, ]
stopifnot(
  identical(tmp1[ , -(5:7)], GRCh38[ , -(5:7)]),
  identical(tmp2, GRCh38.p14[GRCh38.p14$AssemblyUnit == "PATCHES", ])
)
```

`getChromInfoFromUCSC` *Get chromosome information for a UCSC genome*

Description

`getChromInfoFromUCSC` returns chromosome information like sequence names, lengths and circularity flags for a given UCSC genome e.g. for hg19, panTro6, sacCer3, etc...

Note that `getChromInfoFromUCSC` behaves slightly differently depending on whether a genome is *registered* in the **GenomeInfoDb** package or not. See below for the details.

Use `registered_UCSC_genomes` to list all the UCSC genomes currently registered in the **GenomeInfoDb** package.

Usage

```
getChromInfoFromUCSC(genome,
                     assembled.molecules.only=FALSE,
                     map.NCBI=FALSE,
                     add.ensembl.col=FALSE,
                     goldenPath.url=getOption("UCSC.goldenPath.url"),
                     recache=FALSE,
                     as.Seqinfo=FALSE)

registered_UCSC_genomes(organism=NA)
```

Arguments

genome	A single string specifying the name of a UCSC genome e.g. "panTro6", "mm39", "sacCer3", etc...
assembled.molecules.only	If FALSE (the default) then chromosome information is returned for all the sequences in the genome, that is, for all the chromosomes, plasmids, and scaffolds. If TRUE then chromosome information is returned only for the *assembled molecules*. These are the chromosomes (including the mitochondrial chromosome) and plasmids only. No scaffolds. Note that assembled.molecules.only=TRUE is supported only for *registered* genomes. When used on an *unregistered* genome, assembled.molecules.only is ignored with a warning.
map.NCBI	TRUE or FALSE (the default). If TRUE then NCBI chromosome information is bound to the result. This information is retrieved from NCBI by calling getChromInfoFromNCBI on the NCBI assembly that the UCSC genome is based on. Then the data frame returned by getChromInfoFromNCBI ("NCBI chrom info") is *mapped* and bound to the data frame returned by getChromInfoFromUCSC ("UCSC chrom info"). This "map and bind" operation is similar to a JOIN in SQL. Note that not all rows in the "UCSC chrom info" data frame are necessarily mapped to a row in the "NCBI chrom info" data frame. For example chrM in hg19 has no corresponding sequence in the GRCh37 assembly (the mitochondrial chromosome was omitted from GRCh37). For the unmapped rows the NCBI columns in the final data frame are filled with NAs (LEFT JOIN in SQL). The primary use case for using map.NCBI=TRUE is to map UCSC sequence names to NCBI sequence names. This is only supported for *registered* UCSC genomes based on an NCBI assembly!
add.ensembl.col	TRUE or FALSE (the default). Whether or not the Ensembl sequence names should be added to the result (in column ensembl).
goldenPath.url	A single string specifying the URL to the UCSC goldenPath location where the chromosome sizes are expected to be found.
recache	getChromInfoFromUCSC uses a cache mechanism so the chromosome sizes of a given genome only get downloaded once during the current R session (note that the caching is done in memory so cached information does NOT persist across sessions). Setting recache to TRUE forces a new download (and recaching) of the chromosome sizes for the specified genome.
as.Seqinfo	TRUE or FALSE (the default). If TRUE then a Seqinfo object is returned instead of a data frame. Note that only the chrom, size, and circular columns of the data frame are used to make the Seqinfo object. All the other columns are ignored (and lost).
organism	When organism is specified, registered_UCSC_genomes() will only return the subset of genomes that are registered for that organism. organism must be specified as a single string and will be used to perform a search (with grep()) on the "organism" column of the data frame returned by registered_UCSC_genomes(). The search is case-insensitive.

Details

*** *Registered* vs *unregistered* UCSC genomes ***

- For *registered* genomes, the returned data frame contains information about which sequences are assembled molecules and which are not, and the `assembled.molecules.only` argument is supported. For *unregistered* genomes, this information is missing, and the `assembled.molecules.only` argument is ignored with a warning.
- For *registered* genomes, the returned circularity flags are guaranteed to be accurate. For *unregistered* genomes, a heuristic is used to determine the circular sequences.
- For *registered* genomes, special care is taken to make sure that the sequences are returned in a sensible order. For *unregistered* genomes, a heuristic is used to return the sequences in a sensible order.

Please contact the maintainer of the **GenomeInfoDb** package to request registration of additional genomes.

*** Offline mode ***

The **GenomeInfoDb** package includes a small database that contains chromosome information for the most commonly used NCBI and UCSC genome assemblies e.g. GRCh38.p14, GRCh38.p13, GRCm39, hg38, mm39, etc... This makes calls like `getChromInfoFromNCBI("GRCh38.p14")` or `getChromInfoFromUCSC("hg38")` work offline. Plus now they are fast and reliable.

Note that calling `getChromInfoFromUCSC()` with `recache=TRUE` will trigger retrieval of the chromosome info from UCSC, and will issue a warning if this info no longer matches the chromosome info stored in the package. Please open an issue on GitHub if you get such warning or want to request adding chromosome information of your favorite NCBI or UCSC genome assembly to **GenomeInfoDb**'s internal database.

Value

For `getChromInfoFromUCSC`: By default, a 4-column data frame with columns:

1. `chrom`: character.
2. `size`: integer.
3. `assembled`: logical.
4. `circular`: logical.

If `map.NCBI` is `TRUE`, then 7 "NCBI columns" are added to the result:

- `NCBI.SequenceName`: character.
- `NCBI.SequenceRole`: factor.
- `NCBI.AssignedMolecule`: factor.
- `NCBI.GenBankAccn`: character.
- `NCBI.Relationship`: factor.
- `NCBI.RefSeqAccn`: character.
- `NCBI.AssemblyUnit`: factor.

Note that the names of the "NCBI columns" are those returned by `getChromInfoFromNCBI` but with the `NCBI.` prefix added to them.

If `add.ensembl.col` is `TRUE`, the column `ensembl` is added to the result.

For `registered_UCSC_genomes`: A data frame summarizing all the UCSC genomes currently *registered* in the **GenomeInfoDb** package.

Author(s)

H. Pagès

See Also

- `getChromInfoFromNCBI` for getting chromosome information for an NCBI assembly.
- `getChromInfoFromEnsembl` for getting chromosome information for an Ensembl species.
- Seqinfo objects.
- The `getBSgenome` convenience utility in the **BSgenome** package for getting a BSgenome object from an installed BSgenome data package.

Examples

```
## ---------------------------------------------------------------------
## A. BASIC EXAMPLES
## ---------------------------------------------------------------------

getChromInfoFromUCSC("hg19")

getChromInfoFromUCSC("hg19", as.Seqinfo=TRUE)

## Map the hg38 sequences to their corresponding sequences in
## the GRCh38.p13 assembly:
getChromInfoFromUCSC("hg38", map.NCBI=TRUE)[c(1, 5)]

## Note that some NCBI-based UCSC genomes contain sequences that
## are not mapped. For example this is the case for chrM in hg19:
hg19 <- getChromInfoFromUCSC("hg19", map.NCBI=TRUE)
hg19[is.na(hg19$NCBI.SequenceName), ]

## Map the hg19 sequences to the Ensembl sequence names:
getChromInfoFromUCSC("hg19", add.ensembl.col=TRUE)

## --- List of UCSC genomes currently registered in the package ---

registered_UCSC_genomes()

## All registered UCSC genomes for Felis catus (domestic cat):
registered_UCSC_genomes(organism = "Felis catus")

## All registered UCSC genomes for Homo sapiens:
registered_UCSC_genomes("homo")
```

```
## ---------------------------------------------------------------------
## B. USING getChromInfoFromUCSC() TO SET UCSC SEQUENCE NAMES ON THE
##    GRCh38 GENOME
## ---------------------------------------------------------------------

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

## Get the chromosome info for the hg38 genome:
hg38_chrom_info <- getChromInfoFromUCSC("hg38", map.NCBI=TRUE)
ncbi2ucsc <- setNames(hg38_chrom_info$chrom,
                      hg38_chrom_info$NCBI.SequenceName)

## Set the UCSC sequence names on 'genome':
seqlevels(genome) <- ncbi2ucsc[seqlevels(genome)]
genome

## Sanity check: check that the sequence lengths in 'genome' are the same
## as in 'hg38_chrom_info':
m <- match(seqlevels(genome), hg38_chrom_info$chrom)
stopifnot(identical(unname(seqlengths(genome)), hg38_chrom_info$size[m]))
```

`loadTaxonomyDb` *Return a data.frame that lists the known taxonomy IDs and their corresponding organisms.*

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 organism).

Requires the **GenomeInfoDbData** package.

Usage

```
loadTaxonomyDb()
```

Value

A data frame with 1 row per genus/species designation and three columns. The 1st column is the taxonomy ID. The second columns is the genus and the third is the species name.

Author(s)

Marc Carlson

Examples

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

---

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")
```

NCBI-utils *Utility functions to access NCBI resources*

Description

Low-level utility functions to access NCBI resources. Not intended to be used directly by the end user.

Usage

```
find_NCBI_assembly_ftp_dir(assembly_accession, assembly_name=NA)

fetch_assembly_report(assembly_accession, assembly_name=NA,
                      AssemblyUnits=NULL)
```

Arguments

`assembly_accession`

A single string containing either a GenBank assembly accession (e.g. `"GCA_000001405.15"`) or a RefSeq assembly accession (e.g. `"GCF_000001405.26"`).
Alternatively, for `fetch_assembly_report()`, the `assembly_accession` argument can be set to the URL to the *assembly report* (a.k.a. "Full sequence report").

`assembly_name` A single string or `NA`.

`AssemblyUnits` By default, all the *assembly units* are included in the data frame returned by `fetch_assembly_report()`. To include only a subset of assembly units, pass a character vector containing the names of the assembly units to include to the `AssemblyUnits` argument.

Value

For `find_NCBI_assembly_ftp_dir()`: A length-2 character vector:

- The 1st element in the vector is the URL to the FTP dir, without the trailing slash.
- The 2nd element in the vector is the prefix used in the names of most of the files in the FTP dir.

For `fetch_assembly_report()`: A data frame with 1 row per sequence in the assembly and 10 columns:

1. SequenceName
2. SequenceRole
3. AssignedMolecule
4. AssignedMoleculeLocationOrType
5. GenBankAccn
6. Relationship
7. RefSeqAccn
8. AssemblyUnit
9. SequenceLength
10. UCSCStyleName

Note

`fetch_assembly_report` is the workhorse behind higher-level and more user-friendly `getChromInfoFromNCBI`.

Author(s)

H. Pagès

See Also

`getChromInfoFromNCBI` for a higher-level and more user-friendly version of `fetch_assembly_report`.

Examples

```
ftp_dir <- find_NCBI_assembly_ftp_dir("GCA_000001405.15")
ftp_dir

url <- ftp_dir[[1]]     # URL to the FTP dir
prefix <- ftp_dir[[2]]  # prefix used in names of most files
```

```
list_ftp_dir(url)

assembly_report_url <- paste0(url, "/", prefix, "_assembly_report.txt")

## To fetch the assembly report for assembly GCA_000001405.15, you can
## call fetch_assembly_report() on the assembly accession or directly
## on the URL to the assembly report:
assembly_report <- fetch_assembly_report("GCA_000001405.15")
dim(assembly_report)
head(assembly_report)

## Sanity check:
assembly_report2 <- fetch_assembly_report(assembly_report_url)
stopifnot(identical(assembly_report, assembly_report2))
```

`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, pruning.mode=c("error", "coarse", "fine", "tidy"))
dropSeqlevels(x, value, pruning.mode=c("error", "coarse", "fine", "tidy"))
renameSeqlevels(x, value)
standardChromosomes(x, species=NULL)
keepStandardChromosomes(x, species=NULL,
                        pruning.mode=c("error", "coarse", "fine", "tidy"))
```

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)`.
`pruning.mode`	See `?seqinfo` for a description of the pruning modes.
`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.

  When `x` is a GRangesList it is possible to have 'mixed' list elements that have ranges from different chromosomes. `keepSeqlevels` will not keep 'mixed' list elements

- `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)`.

- `standardChromosomes`: Lists the 'standard' chromosomes defined as sequences in the assembly that are not scaffolds; also referred to as an 'assembly molecule' in NCBI. `standardChromosomes` 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.

- `keepStandardChromosomes`: Subsetting operation that returns only the 'standard' chromosomes.

  `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'
gr1 <- keepSeqlevels(gr, "chr1", pruning.mode="coarse")
## Drop 'chr1'. Both 'chr2' and 'chr3' are kept.
gr2 <- dropSeqlevels(gr, "chr1", pruning.mode="coarse")

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"), pruning.mode="coarse")
seqlevels(seq2)

##
## List-like objects:
##

grl0 <- GRangesList(A=GRanges("chr2", IRanges(3:2, 5)),
                    B=GRanges(c("chr2", "chrMT"), IRanges(7:6, 15)),
                    C=GRanges(c("chrY", "chrMT"), IRanges(17:16, 25)),
                    D=GRanges())
## See ?seqinfo for a description of the pruning modes.
keepSeqlevels(grl0, "chr2", pruning.mode="coarse")
keepSeqlevels(grl0, "chr2", pruning.mode="fine")
keepSeqlevels(grl0, "chr2", pruning.mode="tidy")

library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
txdb <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
## Pruning mode "coarse" is particularly well suited on a GRangesList
## object that contains exons grouped by transcript:
ex_by_tx <- exonsBy(txdb, by="tx")
seqlevels(ex_by_tx)
ex_by_tx2 <- keepSeqlevels(ex_by_tx, "chr2L", pruning.mode="coarse")
seqlevels(ex_by_tx2)
## Pruning mode "tidy" is particularly well suited on a GRangesList
## object that contains transcripts grouped by gene:
tx_by_gene <- transcriptsBy(txdb, by="gene")
seqlevels(tx_by_gene)
tx_by_gene2 <- keepSeqlevels(tx_by_gene, "chr2L", pruning.mode="tidy")
```

```
seqlevels(tx_by_gene2)

## ---------------------------------------------------------------------
## 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"))
stopifnot(identical(gal, 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))

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

##
## GRanges:
##
gr <- GRanges(c(paste0("chr",c(1:3)), "chr1_gl000191_random",
              "chr1_gl000192_random"), IRanges(1:5, width=3))
gr
keepStandardChromosomes(gr, pruning.mode="coarse")

##
## List-like objects:
##

grl <- GRangesList(GRanges("chr1", IRanges(1:2, 5)),
                   GRanges(c("chr1_GL383519v1_alt", "chr1"), IRanges(5:6, 5)))
```

```
## Use pruning.mode="coarse" to drop list elements with mixed seqlevels:
keepStandardChromosomes(grl, pruning.mode="coarse")
## Use pruning.mode="tidy" to keep all list elements with ranges on
## standard chromosomes:
keepStandardChromosomes(grl, pruning.mode="tidy")

##
## The set of standard chromosomes should not be affected by the
## particular seqlevel style currently in use:
##

## NCBI
worm <- GRanges(c("I", "II", "foo", "X", "MT"), IRanges(1:5, width=5))
keepStandardChromosomes(worm, pruning.mode="coarse")

## UCSC
seqlevelsStyle(worm) <- "UCSC"
keepStandardChromosomes(worm, pruning.mode="coarse")

## Ensembl
seqlevelsStyle(worm) <- "Ensembl"
keepStandardChromosomes(worm, pruning.mode="coarse")
```

`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 from its seqlevels using a heuristic helped by a seqlevels style database stored in the **GenomeInfoDb** package. If the underlying genome is known (i.e. if unique(genome(x)) is not NA), the name of the genome or assembly (e.g. ce11 or WBcel235) is also used by the heuristic.

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
## ---------------------------------------------------------------------

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

## On a GRanges object:
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

## In general the seqlevelsStyle() setter doesn't know how to rename
## scaffolds. However, if the genome is specified, it's very likely
## that seqlevelsStyle() will be able to take advantage of that:
gr <- GRanges(rep(c("2", "Y", "Hs6_111610_36"), 2), IRanges(1:6, 10))
genome(gr) <- "NCBI36"
seqlevelsStyle(gr) <- "UCSC"
gr

## On a Seqinfo object:
si <- si0 <- Seqinfo(genome="apiMel2")
si
```

```
seqlevelsStyle(si) <- "NCBI"
si
seqlevelsStyle(si) <- "RefSeq"
si
seqlevelsStyle(si) <- "UCSC"
stopifnot(identical(si0, si))

si <- si0 <- Seqinfo(genome="WBcel235")
si
seqlevelsStyle(si) <- "UCSC"
si
seqlevelsStyle(si) <- "RefSeq"
si
seqlevelsStyle(si) <- "NCBI"
stopifnot(identical(si0, si))

si <- Seqinfo(genome="macFas5")
si
seqlevelsStyle(si) <- "NCBI"
si

## ---------------------------------------------------------------------
## 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")
```

Index

∗ **manip**
    getChromInfoFromEnsembl, 2
    getChromInfoFromNCBI, 8
    getChromInfoFromUCSC, 11
    loadTaxonomyDb, 15
    NCBI-utils, 17
∗ **methods**
    seqlevels-wrappers, 19
∗ **utilities**
    seqlevels-wrappers, 19

BSgenome, *14*

dropSeqlevels *(*seqlevels-wrappers*)*, 19

extractSeqlevels *(*seqlevelsStyle*)*, 23
extractSeqlevelsByGroup
        *(*seqlevelsStyle*)*, 23

fetch_assembly_report *(*NCBI-utils*)*, 17
find_NCBI_assembly_ftp_dir
        *(*NCBI-utils*)*, 17

genomeBuilds *(*mapGenomeBuilds*)*, 16
genomeStyles *(*seqlevelsStyle*)*, 23
get_and_fix_chrom_info_from_UCSC
        *(*getChromInfoFromUCSC*)*, 11
getBSgenome, *14*
getChromInfoFromEnsembl, 2, *10*, *14*
getChromInfoFromNCBI, *3, 4*, 8, *12*, *14*, *18*
getChromInfoFromUCSC, *4*, *10*, 11

keepSeqlevels *(*seqlevels-wrappers*)*, 19
keepStandardChromosomes
        *(*seqlevels-wrappers*)*, 19

listOrganisms *(*mapGenomeBuilds*)*, 16
loadTaxonomyDb, 15

mapGenomeBuilds, 16
mapSeqlevels *(*seqlevelsStyle*)*, 23

NCBI-utils, 17

registered_NCBI_assemblies
        *(*getChromInfoFromNCBI*)*, 8
registered_UCSC_genomes
        *(*getChromInfoFromUCSC*)*, 11
renameSeqlevels *(*seqlevels-wrappers*)*, 19

saveChromInfoFromNCBI
        *(*getChromInfoFromNCBI*)*, 8
saveChromInfoFromUCSC
        *(*getChromInfoFromUCSC*)*, 11
Seqinfo, *3, 4*, *9, 10*, *12*, *14*, *19, 20*
seqinfo, *20*
seqlevels-wrappers, 19
seqlevelsInGroup *(*seqlevelsStyle*)*, 23
seqlevelsStyle, 23
seqlevelsStyle,ANY-method
        *(*seqlevelsStyle*)*, 23
seqlevelsStyle,character-method
        *(*seqlevelsStyle*)*, 23
seqlevelsStyle,Seqinfo-method
        *(*seqlevelsStyle*)*, 23
seqlevelsStyle<- *(*seqlevelsStyle*)*, 23
seqlevelsStyle<-,ANY-method
        *(*seqlevelsStyle*)*, 23
seqlevelsStyle<-,character-method
        *(*seqlevelsStyle*)*, 23
seqlevelsStyle<-,Seqinfo-method
        *(*seqlevelsStyle*)*, 23
standardChromosomes
        *(*seqlevels-wrappers*)*, 19