BSgenome

April 19, 2009

`BSParams-class` *Class "BSParams"*

Description

A parameter class for representing all parameters needed for running the `bsapply` method.

Objects from the Class

Objects can be created by calls of the form `new("BSParams", ...)`.

Slots

`X`: a BSgenome object that contains chromosomes that you wish to apply FUN on

`FUN`: the function to apply to each chromosome in the BSgenome object 'X'

`exclude`: this is a character vector with strings that will be used to filter out chromosomes whose names match these strings.

`simplify`: TRUE/FALSE value to indicate whether or not the function should try to simplify the output for you.

`maskList`: A named logical vector of maskStates preferred when used with a BSGenome object. When using the bsapply function, the masks will be set to the states in this vector.

Methods

`bsapply(p)` Performs the function FUN using the parameters contained within `BSParams`.

Author(s)

Marc Carlson

See Also

`bsapply`

BSgenome-class *The BSgenome class*

Description

A container for the complete genome sequence of a given species.

Accesor methods

In the code snippets below, `x` is a BSgenome object and `name` is the name of a sequence (character-string).

`organism(x)`: Return the target organism for this genome e.g. `"Homo sapiens"`, `"Mus musculus"`, `"Caenorhabditis elegans"`, etc...

`species(x)`: Return the target species for this genome e.g. `"Human"`, `"Mouse"`, `"C. elegans"`, 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"`.

`sourceUrl(x)`: Return the source URL i.e. the permanent URL to the place where the FASTA files used to produce the sequences contained in `x` can be found (and downloaded).

`seqnames(x)`: Return the index of the single sequences contained in `x`. Each single sequence is stored in an XString or MaskedXString object and typically comes from a source file (FASTA) with a single record. The names returned by `seqnames(x)` usually reflect the names of those source files but a common prefix or suffix was eventually removed in order to keep them as short as possible.

`seqlengths(x)`: Return the lengths of the single sequences contained in `x`.

See ¿`length,XString-method`‘ and ¿`length,MaskedXString-method`‘ for the definition of the length of an XString or MaskedXString object. Note that the length of a masked sequence (MaskedXString object) is not affected by the current set of active masks but the `nchar` method for MaskedXString is.

`names(seqlengths(x))` is guaranteed to be identical to `seqnames(x)`.

`mseqnames(x)`: Return the index of the multiple sequences contained in `x`. Each multiple sequence is stored in an XStringSet object and typically comes from a source file (FASTA) with multiple records. The names returned by `mseqnames(x)` usually reflect the names of those source files but a common prefix or suffix was eventually removed in order to keep them as short as possible.

`names(x)`: Return the index of all sequences contained in `x`. This is the same as `c(seqnames(x), mseqnames(x))`.

`length(x)`: Return the length of `x`, i.e., the number of all sequences that it contains. This is the same as `length(names(x))`.

`x[[name]]`: Return sequence (single or multiple) named `name`. No sequence is actually loaded into memory until this is explicitely requested with a call to `x[[name]]` or `x$name`. When loaded, a sequence is kept in a cache. It will be automatically removed from the cache at garbage collection if it's not in use anymore i.e. if there are no reference to it (other than the reference stored in the cache). With `options(verbose=TRUE)`, a message is printed each time a sequence is removed from the cache.

`x$name`: Same as `x[[name]]` but `name` is not evaluated and therefore must be a literal character string or a name (possibly backtick quoted).

`masknames(x)`: The names of the built-in masks that are defined for all the single sequences. There can be up to 4 built-in masks per sequence. These will always be (in this order): (1) the mask of assembly gaps, aka "the AGAPS mask"; (2) the mask of intra-contig ambiguities, aka "the AMB mask"; (3) the mask of repeat regions that were determined by the RepeatMasker software, aka "the RM mask"; (4) the mask of repeat regions that were determined by the Tandem Repeats Finder software (where only repeats with period less than or equal to 12 were kept), aka "the TRF mask". All the single sequences in a given package are guaranteed to have the same collection of built-in masks (same number of masks and in the same order).

`masknames(x)` gives the names of the masks in this collection. Therefore the value returned by `masknames(x)` is a character vector made of the first N elements of `c("AGAPS", "AMB", "RM", "TRF")`, where N depends only on the BSgenome data package being looked at (0 <= N <= 4). The man page for most BSgenome data packages should provide the exact list and permanent URLs of the source data files that were used to extract the built-in masks. For example, if you've installed the BSgenome.Hsapiens.UCSC.hg18 package, load it and see the Note section in ¿`BSgenome.Hsapiens.UCSC.hg18`‘.

Author(s)

H. Pages

See Also

`available.genomes`, XString-class, MaskedXString-class, XStringSet-class, `injectSNPs`, `subseq`, `getSeq`, `matchPattern`, `rm`, `gc`

Examples

```
## Loading a BSgenome data package doesn't load its sequences
## into memory:
library(BSgenome.Celegans.UCSC.ce2)

## Number of sequences in this genome:
length(Celegans)

## Display a summary of the sequences:
Celegans

## Index of single sequences:
seqnames(Celegans)

## Lengths (i.e. number of nucleotides) of the sequences:
seqlengths(Celegans)

## Load chromosome I from disk to memory (hence takes some time)
## and keep a reference to it:
chrI <- Celegans[["chrI"]]  # equivalent to Celegans$chrI
```

```
chrI

class(chrI)   # a DNAString instance
length(chrI)  # with 15080483 nucleotides

## Multiple sequences:
mseqnames(Celegans)
upstream1000 <- Celegans$upstream1000
upstream1000
class(upstream1000)  # a DNAStringSet instance
## Character vector containing the description lines of the first
## 4 sequences in the original FASTA file:
names(upstream1000)[1:4]

## ---------------------------------------------------------------------
## PASS-BY-ADDRESS SEMANTIC, CACHING AND MEMORY USAGE
## ---------------------------------------------------------------------

## We want a message to be printed each time a sequence is removed
## from the cache:
options(verbose=TRUE)

gc()  # nothing seems to be removed from the cache
rm(chrI, upstream1000)
gc()  # chrI and upstream1000 are removed from the cache (they are
      # not in use anymore)

options(verbose=FALSE)

## Get the current amount of data in memory (in Mb):
mem0 <- gc()["Vcells", "(Mb)"]

system.time(chrV <- Celegans[["chrV"]])  # read from disk

gc()["Vcells", "(Mb)"] - mem0  # chrV occupies 20Mb in memory

system.time(tmp <- Celegans[["chrV"]])  # much faster! (sequence
                                        # is in the cache)

gc()["Vcells", "(Mb)"] - mem0  # we're still using 20Mb (sequences
                               # have a pass-by-address semantic
                               # i.e. the sequence data are not
                               # duplicated)

## subseq() doesn't copy the sequence data either, hence it is very
## fast and memory efficient (but the returned object will hold a
## reference to chrV):
y <- subseq(chrV, 10, 8000000)
gc()["Vcells", "(Mb)"] - mem0

## We must remove all references to chrV before it can be removed from
## the cache (so the 20Mb of memory used by this sequence are freed).
options(verbose=TRUE)
rm(chrV, tmp)
gc()
```

```
## Remember that 'y' holds a reference to chrV too:
rm(y)
gc()

options(verbose=FALSE)
gc()["Vcells", "(Mb)"] - mem0
```

`BSgenomeForge` *The BSgenomeForge functions*

Description

A set of functions for making a BSgenome data package.

Usage

```
## Top-level BSgenomeForge function:

forgeBSgenomeDataPkg(x, seqs_srcdir=".", masks_srcdir=".", destdir=".", verbos

## Low-level BSgenomeForge functions:

forgeSeqlengthsFile(seqnames, prefix="", suffix=".fa",
                    seqs_srcdir=".", seqs_destdir=".", verbose=TRUE)

forgeSeqFiles(seqnames, mseqnames=NULL, prefix="", suffix=".fa",
              seqs_srcdir=".", seqs_destdir=".", verbose=TRUE)

forgeMasksFiles(seqnames, nmask_per_seq,
                seqs_destdir=".", masks_srcdir=".", masks_destdir=".",
                AGAPSfiles_type="gap", AGAPSfiles_name=NA,
                AGAPSfiles_prefix="", AGAPSfiles_suffix="_gap.txt",
                RMfiles_name=NA, RMfiles_prefix="", RMfiles_suffix=".fa.out",
                TRFfiles_name=NA, TRFfiles_prefix="", TRFfiles_suffix=".bed",
                verbose=TRUE)
```

Arguments

`x`	A BSgenomeDataPkgSeed object or the name of a BSgenome data package seed file. See the BSgenomeForge vignette in this package for more information.
`seqs_srcdir, masks_srcdir`	Single strings indicating the path to the source directories i.e. to the directories containing the source data files. Only read access to these directories is needed. See the BSgenomeForge vignette in this package for more information.
`destdir`	A single string indicating the path to the directory where the source tree of the target package should be created. This directory must already exist. See the BSgenomeForge vignette in this package for more information.
`verbose`	`TRUE` or `FALSE`.
`seqnames, mseqnames`	A character vector containing the names of the single (for `seqnames`) and multiple (for `mseqnames`) sequences to forge. See the BSgenomeForge vignette in this package for more information.

`prefix, suffix`
: See the BSgenomeForge vignette in this package for more information, in particular the description of the `seqfiles_prefix` and `seqfiles_suffix` fields of a BSgenome data package seed file.

`seqs_destdir, masks_destdir`
: During the forging process the source data files are converted into serialized Biostrings objects. `seqs_destdir` and `masks_destdir` must be single strings indicating the path to the directories where these serialized objects should be saved. These directories must already exist.

  `forgeSeqlengthsFile` will produce a single .rda file. Both `forgeSeqFiles` and `forgeMasksFiles` will produce one .rda file per sequence.

`nmask_per_seq`
: A single integer indicating the desired number of masks per sequence. See the BSgenomeForge vignette in this package for more information.

`AGAPSfiles_type, AGAPSfiles_name, AGAPSfiles_prefix, AGAPSfiles_suffix, RMfiles_`
: These arguments are named accordingly to the corresponding fields of a BSgenome data package seed file. See the BSgenomeForge vignette in this package for more information.

Details

These functions are intended for Bioconductor users who want to make a new BSgenome data package, not for regular users of these packages. See the BSgenomeForge vignette in this package (`vignette("BSgenomeForge")`) for an extensive coverage of this topic.

Author(s)

H. Pages

Examples

```
forgeSeqFiles("chrM", prefix="ce2", suffix=".fa",
              seqs_srcdir=system.file("extdata", package="BSgenome"),
              seqs_destdir=tempdir())
load(file.path(tempdir(), "chrM.rda"))
chrM
```

`available.genomes` *available.genomes*

Description

Get the list of BSgenome data packages that are currently available on the Bioconductor repositories for your version of R/Bioconductor.

Usage

```
available.genomes(type=getOption("pkgType"))
```

Arguments

`type`	Character string indicating the type of package (`"source"`, `"mac.binary"` or `"win.binary"`) to look for.

Details

A BSgenome data package contains the full genome for a given organism. Its name has 4 parts separated by a dot (e.g. BSgenome.Celegans.UCSC.ce2). The 1st part is always BSgenome, the 2nd part is the name of the organism (abbreviated), the 3rd part is the name of the organisation who assembled the genome and the 4th part is the release string or number used by this organisation for this genome. A BSgenome data package contains a single BSgenome object named like the second part of the package name (e.g. Celegans in the case of BSgenome.Celegans.UCSC.ce2) where all the sequences for this genome are stored.

Value

A character vector containing the names of the BSgenome data packages currently available.

Author(s)

H. Pages

See Also

BSgenome-class, `available.packages`

Examples

```
# Get the list of BSgenome data packages currently available:
available.genomes()

# Make your choice and install with:
source("http://bioconductor.org/biocLite.R")
biocLite("BSgenome.Celegans.UCSC.ce2")

# Have a coffee ;-)

# Load the package and display the index of sequences for this genome:
library(BSgenome.Celegans.UCSC.ce2)
Celegans
```

`bsapply` *bsapply*

Description

Apply a function to each chromosome in a genome.

Usage

```
bsapply(BSParams, ...)
```

Arguments

`BSParams`	a BSParams object that holds the various parameters needed to configure the bsapply function
`...`	optional arguments to 'FUN'.

Details

By default the exclude parameter is set to not exclude anything. A popular option will probably be to set this to "rand" so that random bits of unassigned contigs are filtered out.

Value

A named list is returned where each element is whatever was returned by the function FUN. The names of the list correspond to the names of the chromosomes as they were labeled by the BSgenome obect in the slot accessed by seqnames().

Author(s)

Marc Carlson

See Also

BSgenome-class

Examples

```
## Load the Worm genome:
library("BSgenome.Celegans.UCSC.ce2")

## Count the alphabet frequencies for every chromosome but exclude
## mitochrondrial ones:
params <- new("BSParams", X = Celegans, FUN = alphabetFrequency,
exclude = "M")
bsapply(params)

## Or we can do this same function with the simplif option:
params <- new("BSParams", X = Celegans, FUN = alphabetFrequency,
exclude = "M", simplify = TRUE)
bsapply(params)

## Examples to show how we might look for a string (in this case an
## ebox motif) across the whole genome.
Ebox <- DNAStringSet("CACGTG")
pdict0 <- PDict(Ebox)

params <- new("BSParams", X = Celegans, FUN = countPDict, simplify = TRUE)
bsapply(params, pdict = pdict0)

params@FUN <- matchPDict
bsapply(params, pdict = pdict0)

## And since its really overkill to use matchPDict to find a single pattern:
params@FUN <- matchPattern
bsapply(params, pattern = "CACGTG")

## Examples on how to use the masks
library("BSgenome.Hsapiens.UCSC.hg18")
## I can make things verbose if I want to see the chromosomes getting processed.
options(verbose=TRUE)
## For the 1st example, lets use default masks
params <- new("BSParams", X = Hsapiens, FUN = alphabetFrequency,
exclude = c(1:8,"random","hap"), simplify = TRUE)
```

```
bsapply(params)

##Enable all masks
params@maskList <- c("RM"=TRUE,"TRF"=TRUE)
bsapply(params)

##Disable all masks
params@maskList <- c("AGAPS"=FALSE,"AMB"=FALSE)
bsapply(params)
```

`getSeq` *getSeq*

Description

A convenience function for extracting a set of sequences (or subsequences) from a BSgenome object.

Usage

```
getSeq(bsgenome, names, start=NA, end=NA, width=NA, as.character=TRUE)
```

Arguments

`bsgenome`	A BSgenome object. See the `available.genomes` function for how to install a genome.
`names`	The names of the sequences to extract from `bsgenome`. If missing, then `seqnames(bsgenome)` is used. See `?seqnames` and `?mseqnames` to get the list of single sequences and multiple sequences (respectively) contained in `bsgenome`. Here is how the lookup between the names passed to the `names` argument and the sequences in `bsgenome` is performed. For each `name` in `names`: (1) if `bsgenome` contains a single sequence with that name then this sequence is returned; (2) otherwise the names of all the elements in all the multiple sequences are searched: `name` is treated as a regular expression and `grep` is used for this search. If exactly one sequence is found, then it's returned, otherwise an error is raised.
`start, end, width`	Specify these arguments only if you don't want to extract the entire sequences. Then the subsequences specified by `start`, `end` and `width` (single integers or NAs) will be extracted by a call to `subseq` before they are returned by `getSeq`.
`as.character`	`TRUE` or `FALSE`. Should the extracted sequences be returned in a standard character vector?

Value

A standard character vector when `as.character=TRUE`. Note that when `as.character=TRUE`, then the masks that are defined on top of the sequences to extract are ignored if any (see ¿`MaskedXString-class`` for more information about masked sequences).

A DNAString or MaskedDNAString object when `as.character=FALSE`. Note that `as.character=FALSE` is not supported when more than one sequence name is supplied.

Note

Be aware that using `as.character=TRUE` can be very inefficient when the returned character vector contains very long strings (> 1 million letters) or is itself a long vector (> 10000 strings).

`getSeq` is much more efficient when used with `as.character=FALSE` but this works only for extracting one sequence at a time for now.

Author(s)

H. Pages; improvements suggested by Matt Settles

See Also

`available.genomes`, BSgenome-class, `seqnames`, `mseqnames`, `grep`, `subseq`, `DNAString`, `MaskedDNAString`, `[[,BSgenome-method`

Examples

```
# Load the Caenorhabditis elegans genome (UCSC Release ce2):
library(BSgenome.Celegans.UCSC.ce2)

# Look at the index of sequences:
Celegans

# Get chromosome V as a DNAString object:
getSeq(Celegans, "chrV", as.character=FALSE)
# which is in fact the same as doing:
Celegans$chrV

# Never try this:
#getSeq(Celegans, "chrV")
# or this (even worse):
#getSeq(Celegans)

# Get the first 20 bases of each chromosome:
getSeq(Celegans, end=20)

# Get the last 20 bases of each chromosome:
getSeq(Celegans, start=-20)

# Get the "NM_058280_up_1000" sequence (belongs to the upstream1000
# multiple sequence) as a character string:
s1 <- getSeq(Celegans, "NM_058280_up_1000")
# or a DNAString object (more efficient):
s2 <- getSeq(Celegans, "NM_058280_up_1000", as.character=FALSE)

getSeq(Celegans, "NM_058280_up_5000", start=-1000) == s1  # TRUE

getSeq(Celegans, "NM_058280_up_5000",
       start=-1000, as.character=FALSE) == s2  # TRUE
```

`injectSNPs` *SNP injection*

Description

Inject SNPs from a SNPlocs data package into a genome.

Usage

```
available.SNPs(type=getOption("pkgType"))
injectSNPs(x, SNPlocs_pkgname)

## Related utilities
SNPlocs_pkgname(x)
SNPcount(x)
SNPlocs(x, seqname)
```

Arguments

`type`	Character string indicating the type of package (`"source"`, `"mac.binary"` or `"win.binary"`) to look for.
`x`	A BSgenome object.
`SNPlocs_pkgname`	The name of a SNPlocs data package containing SNP information for the single sequences contained in `x`. This package must be already installed (`injectSNPs` won't try to install it).
`seqname`	The name of a single sequence in `x`.

Value

`available.SNPs` returns a character vector containing the names of the SNPlocs data packages that are currently available on the Bioconductor repositories for your version of R/Bioconductor. A SNPlocs data package contains basic SNP information (location and alleles) for a given organism.

`injectSNPs` returns a copy of the original genome `x` where some or all of the single sequences were altered by injecting the SNPs defined in the `SNPlocs_pkgname` package.

`SNPlocs_pkgname`, `SNPcount` and `SNPlocs` return `NULL` if no SNPs were injected in `x` (i.e. if `x` is not a BSgenome object returned by a previous call to `injectSNPs`). Otherwise `SNPlocs_pkgname` returns the name of the package from which the SNPs were injected, `SNPcount` the number of SNPs for each altered sequence in `x`, and `SNPlocs` their locations in the sequence whose name is specified by `seqname`.

Note

`injectSNPs`, `SNPlocs_pkgname`, `SNPcount` and `SNPlocs` have the side effect to try to load the SNPlocs data package if it's not already loaded.

Author(s)

H. Pages

See Also

BSgenome-class, `.inplaceReplaceLetterAt`

Examples

```
## Get the list of SNPlocs data packages currently available:
available.SNPs()

if (interactive()) {
  ## Make your choice and install with:
  source("http://bioconductor.org/biocLite.R")
  biocLite("SNPlocs.Hsapiens.dbSNP.20071016")
}

## Inject SNPs from dbSNP into the Human genome:
library(BSgenome.Hsapiens.UCSC.hg18)
Hsapiens
SNPlocs_pkgname(Hsapiens)

HsWithSNPs <- injectSNPs(Hsapiens, "SNPlocs.Hsapiens.dbSNP.20071016")
HsWithSNPs  # note the extra "with SNPs injected from ..." line
SNPlocs_pkgname(HsWithSNPs)
SNPcount(HsWithSNPs)
SNPlocs(HsWithSNPs, "chr1")

alphabetFrequency(Hsapiens$chr1)
alphabetFrequency(HsWithSNPs$chr1)
```

Index