Package ‘SeqArray’

January 25, 2024

Type Package

Title Data Management of Large-Scale Whole-Genome Sequence Variant Calls

Version 1.42.0

Date 2023-10-20

Depends R (>= 3.5.0), gdsfmt (>= 1.31.1)

Imports methods, parallel, IRanges, GenomicRanges, GenomeInfoDb, Biostrings, S4Vectors

LinkingTo gdsfmt

Suggests Biobase, BiocGenerics, BiocParallel, RUnit, Rcpp, SNPRelate, digest, crayon, knitr, markdown, rmarkdown, Rsamtools, VariantAnnotation

Description Data management of large-scale whole-genome sequencing variant calls with thousands of individuals: genotypic data (e.g., SNVs, indels and structural variation calls) and annotations in SeqArray GDS files are stored in an array-oriented and compressed manner, with efficient data access using the R programming language.

License GPL-3

VignetteBuilder knitr

ByteCompile TRUE

LazyData true

URL https://github.com/zhengxwen/SeqArray

BugReports https://github.com/zhengxwen/SeqArray/issues

biocViews Infrastructure, DataRepresentation, Sequencing, Genetics

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

git_branch RELEASE_3_18

git_last_commit 1100124

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18
Date/Publication 2024-01-24

Author Xiuwen Zheng [aut, cre] (<https://orcid.org/0000-0002-1390-0708>), Stephanie Gogarten [aut], David Levine [ctb], Cathy Laurie [ctb]

Maintainer Xiuwen Zheng <zhengx@u.washington.edu>

R topics documented:

- SeqArray-package .................................................. 3
- KG_P1_SampData ................................................... 6
- seqAddValue .......................................................... 6
- seqAlleleFreq ........................................................ 8
- seqApply .............................................................. 10
- seqAsVCF .............................................................. 13
- seqBED2GDS .......................................................... 14
- seqBlockApply ......................................................... 16
- seqCheck .............................................................. 18
- seqClose-methods ..................................................... 19
- seqDelete ............................................................. 20
- seqDigest ............................................................. 21
- seqEmptyFile ........................................................ 22
- seqExampleFileName ............................................... 23
- seqExport ............................................................. 24
- seqGDS2SNP .......................................................... 25
- seqGDS2VCF .......................................................... 27
- seqGet2bGeno ........................................................ 29
- seqGetData ........................................................... 30
- seqGetFilter .......................................................... 33
- seqMerge .............................................................. 34
- seqMissing ........................................................... 36
- seqNewVarData ........................................................ 38
- seqNumAllele .......................................................... 39
- seqOpen ................................................................. 40
- seqOptimize ........................................................... 41
- seqParallel .......................................................... 42
- seqParallelSetup ..................................................... 45
- seqRecompress ......................................................... 46
- seqResetVariantID .................................................... 48
- seqSetFilter-methods ............................................... 49
- seqSetFilterCond ..................................................... 53
- seqSNP2GDS ........................................................... 54
- seqStorageOption ..................................................... 56
- seqSummary ........................................................... 57
- seqSystem ............................................................. 60
- seqTranspose .......................................................... 61
- seqUnitApply .......................................................... 62
Data management of large-scale whole-genome sequencing variants.

Details

As the cost of DNA sequencing rapidly decreases, whole-genome sequencing (WGS) is generating data at an unprecedented rate. Scientists are being challenged to manage data sets that are terabyte-sized, contain diverse types of data and complex data relationships. Data analyses of WGS requires a general file format for storing genetic variants including single nucleotide variations (SNVs), insertions and deletions (indels) and structural variants. The variant call format (VCF) is a generic and flexible format for storing DNA polymorphisms developed for the 1000 Genomes Project that is the standard WGS format in use today. VCF is a textual format usually stored in compressed files that supports rich annotations and relatively efficient data retrieval. However, VCF files are large and the computational burden associated with all data retrieval from text files can be significant for a large WGS study with thousands of samples.

To provide an efficient alternative to VCF for WGS data, we developed a new data format and accompanying Bioconductor package, “SeqArray”. Key features of SeqArray are efficient storage including multiple high compression options, data retrieval by variant or sample subsets, support for parallel access and computing, and C++ integration in the R programming environment. The SeqArray package provides R functions for efficient block-wise computations, and enables scientists to develop custom R scripts for exploratory data analysis.


Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>
Examples

# the file of VCF
vcf.fn <- seqExampleFileName("vcf")
vcf.fn
# or vcf.fn <- "C:/YourFolder/Your_VCF_File.vcf"

# parse the header
seqVCF_Header(vcf.fn)

# get sample id
seqVCF_SampID(vcf.fn)

# convert
seqVCF2GDS(vcf.fn, "tmp.gds", storage.option="ZIP_RA")
seqSummary("tmp.gds")

# list the structure of GDS variables
f <- seqOpen("tmp.gds")
f
seqClose(f)
unlink("tmp.gds")

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
(samp.id <- seqGetData(f, "sample.id"))
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

# get 'chromosome'
table(seqGetData(f, "chromosome"))

# get 'allele'
head(seqGetData(f, "allele"))
# "T,C" "G,A" "G,A" ...

# set sample and variant filters
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)])
set.seed(100)
seqSetFilter(f, variant.id=sample(variant.id, 10))
# get genotypic data
seqGetData(f, "genotype")

# get annotation/info/DP
seqGetData(f, "annotation/info/DP")

# get annotation/info/AA, a variable-length dataset
seqGetData(f, "annotation/info/AA")
# $length <- indicating the length of each variable-length data
# [1] 1 1 1 1 1 1 ...
# $data <- the data according to $length
# [1] "T" "C" "T" "C" "G" "C" ...

# get annotation/format/DP, a variable-length dataset
seqGetData(f, "annotation/format/DP")
# $length <- indicating the length of each variable-length data
# [1] 1 1 1 1 1 1 ...
# $data <- the data according to $length
# variant
# [1,] 25 25 22 3 4 17 ...

# read multiple variables variant by variant
seqApply(f, c(geno="genotype", phase="phase", qual="annotation/id"),
        FUN=function(x) print(x), as.is="none")

# get the numbers of alleles per variant
head(seqApply(f, "allele",
              FUN=function(x) length(unlist(strsplit(x,""))), as.is="integer"))
# or
head(seqGetData(f, "$num_allele"))

# remove the sample and variant filters
seqResetFilter(f)

# calculate the frequency of reference allele,
# a faster version could be obtained by C coding
af <- seqApply(f, "genotype", FUN=function(x) mean(x==0L, na.rm=TRUE),
               as.is="double")
length(af)
summary(af)

# close the GDS file
seqClose(f)
**KG_P1_SampData**  
*Simulated sample data for 1000 Genomes Phase 1*

**Description**
An AnnotatedDataFrame with columns `sample.id`, `sex`, `age`, and `phenotype`, where the identifiers in `sample.id` match those in the SeqArray file.

**Usage**

```r
KG_P1_SampData
```

**Value**

An AnnotatedDataFrame

---

**seqAddValue**  
*Add values to a GDS File*

**Description**
Add or modify the values in a GDS file with hash code

**Usage**

```r
seqAddValue(gdsfile, varnm, val, desp=character(), replace=FALSE,  
            compress="LZMA_RA", packed=TRUE, packed.idx=TRUE, verbose=TRUE,  
            verbose.attr=TRUE)
```

**Arguments**

- **gdsfile**: character for file name, or a `SeqVarGDSClass` object
- **varnm**: the variable name, e.g., "sample.id", "variant.id", "chromosome", "annotation/info/NEW_VARIABLE"
- **val**: the R value can be integers, real numbers, characters, factor, logical, raw variable, data.frame or a list; a list of vectors is used for variable-length annotation data; or NULL for adding a new folder
- **desp**: variable description
- **replace**: if TRUE, replace the existing variable silently if possible
- **compress**: the compression method can be "" (no compression), see `add.gdsn`
- **packed**: TRUE, pack data if there is any missing value
- **packed.idx**: TRUE, store the index variable using integers with the fewest bits if possible
- **verbose**: if TRUE, show information
- **verbose.attr**: if TRUE, show attribute information in a GDS node
**seqAddValue**

**Value**

Return none.

**Author(s)**

Xiuwen Zheng

**See Also**

seqVCF2GDS, seqNewVarData

**Examples**

```r
# the file of GDS
gds.fn <- seqExampleFileName("gds")
file.copy(gds.fn, "tmp.gds", overwrite=TRUE)

# display
(f <- seqOpen("tmp.gds", readonly=FALSE))
show(index.gdsn(f, "sample.id"))
seqAddValue(f, "sample.id", 1:90, replace=TRUE)
show(index.gdsn(f, "sample.id"))

show(index.gdsn(f, "chromosome"))
v <- seqGetData(f, "chromosome")
seqAddValue(f, "chromosome", paste0("chr", v), replace=TRUE)
show(index.gdsn(f, "chromosome"))
table(seqGetData(f, "chromosome"))

# annotation info
seqAddValue(f, "annotation/info/folder", NULL) # add a new folder
seqAddValue(f, "annotation/info/folder/val", 1:1348, "random number")
seqAddValue(f, "annotation/info/folder/packed", c(rep(2L, 1000), rep(NA, 348)))
seqAddValue(f, "annotation/info/newff",
data.frame(x=1:1348, y=rep("s", 1348), stringsAsFactors=FALSE),
desp=c("integer numbers", "character"))

# variable-length annotation info data
v <- lapply(1:1348, function(x) as.character(x))
v[[1]] <- 1:10
seqAddValue(f, "annotation/info/folder/val1", v)
head(seqGetData(f, "annotation/info/folder/val1", .tolist=TRUE))

# sample annotation
seqAddValue(f, "sample.annotation", data.frame(ii=1:90, y=rep("A", 90)),
replace=TRUE)
seqAddValue(f, "sample.annotation/float", (1:90)/90)

# close the GDS file
seqClose(f)
```
# remove the temporary file
unlink("tmp.gds", force=TRUE)

seqAlleleFreq  
Get Allele Frequencies or Counts

Description

Calculates the allele frequencies or counts for reference or minor alleles.

Usage

```r
seqAlleleFreq(gdsfile, ref.allele=0L, minor=FALSE, parallel=seqGetParallel(), verbose=FALSE)
seqAlleleCount(gdsfile, ref.allele=0L, minor=FALSE, parallel=seqGetParallel(), verbose=FALSE)
seqGetAF_AC_Missing(gdsfile, minor=FALSE, parallel=seqGetParallel(), verbose=FALSE)
```

Arguments

- `gdsfile`: a `SeqVarGDSClass` object
- `ref.allele`: NULL, a single numeric value, a numeric vector or a character vector; see Value
- `minor`: if TRUE, return minor allele frequency/count
- `parallel`: FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in `seqParallel`, see `seqParallel` for more details.
- `verbose`: if TRUE, show progress information

Details

If the gds node 'genotype/data' (integer genotypes) is not available, the node 'annotation/format/DS' (numeric genotype dosages for alternative alleles) will be used to calculate allele frequencies. At a site, it assumes 'annotation/format/DS' stores the dosage of the 1st alternative allele in the 1st column, 2nd alt. allele in the 2nd column if it is multi-allelic, and so on.

Value

If `ref.allele=NULL`, the function returns a list of allele frequencies/count according to all allele per site. If `ref.allele` is a single numeric value (like 0L), it returns a numeric/integer vector for the specified allele (0L for the reference allele, 1L for the first alternative allele, etc). If `ref.allele` is a numeric vector, `ref.allele` specifies each allele per site. If `ref.allele` is a character vector, `ref.allele` specifies the desired allele for each site (e.g. ancestral allele for the derived allele frequency/count).
seqAlleleFreq

seqGetAF_AC_Missing() returns data.frame(af, ac, miss) for allele frequencies, allele counts and missing rates. It is faster than calling seqAlleleFreq(), seqAlleleCount() and seqMissing sequentially.

Author(s)

Xiuwen Zheng

See Also

seqMissing, seqNumAllele, seqParallel, seqGetParallel

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
f <- seqOpen(gds.fn)

# return a list
head(seqAlleleFreq(f, NULL, verbose=TRUE))

# return a numeric vector
summary(seqAlleleFreq(f, 0L, verbose=TRUE))

# return a numeric vector
summary(seqAlleleFreq(f, 0L, minor=TRUE, verbose=TRUE))

# return a numeric vector, AA is ancestral allele
AA <- seqGetData(f, "annotation/info/AA", .padNA=TRUE)
summary(seqAlleleFreq(f, AA))
summary(seqAlleleFreq(f, AA, minor=TRUE))

# allele counts
head(seqAlleleCount(f, NULL, verbose=TRUE))
head(seqAlleleCount(f, 0L, verbose=TRUE))
head(seq AlleleCount(f, 0L, minor=TRUE, verbose=TRUE))
head(seqAlleleCount(f, AA, verbose=TRUE))
head(seqAlleleCount(f, AA, minor=TRUE, verbose=TRUE))

# allele frequencies, allele counts and missing proportions
v <- seqGetAF_AC_Missing(f, minor=TRUE)
head(v)

# close the GDS file
seqClose(f)
seqApply

Apply Functions Over Array Margins

Description

Returns a vector or list of values obtained by applying a function to margins of genotypes and annotations.

Usage

seqApply(gdsfile, var.name, FUN, margin=c("by.variant", "by.sample"),
as.is=c("none", "list", "integer", "double", "character", "logical", "raw"),
var.index=c("none", "relative", "absolute"), parallel=FALSE,
.useraw=FALSE, .progress=FALSE, .list_dup=TRUE, ...)

Arguments

gdsfile a SeqVarGDSClass object
var.name the variable name(s), see details
FUN the function to be applied
margin giving the dimension which the function will be applied over; margin="by.variant" by default
as.is returned value: a list, an integer vector, etc; return nothing by default as.is="none"; as.is can be a connection object, or a GDS node gdsn.class object; if "unlist" is used, produces a vector which contains all the atomic components, via unlist(..., recursive=FALSE)
var.index if "none" (by default), call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1 if margin = "by.variant": "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data
parallel FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details.
.useraw TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW for small numbers instead of INTEGER if possible, it is needed to detect data type (RAW or INTEGER) in the user-defined function; for genotypes, 0xFF is missing value if RAW is used
.progress if TRUE, show progress information
.list_dup internal use only
... optional arguments to FUN
seqApply

Details

The variable name should be "sample.id", "variant.id", "position", "chromosome", "allele", "genotype", "annotation/id", "annotation/qual", "annotation/filter", "annotation/info/VARIABLE_NAME", or "annotation/format/VARIABLE_NAME".

"@genotype", "annotation/info/@VARIABLE_NAME" or "annotation/format/@VARIABLE_NAME" are used to obtain the index associated with these variables.

"$dosage" is also allowed for the dosages of reference allele (integer: 0, 1, 2 and NA for diploid genotypes).

"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles.

"$num_allele" returns an integer vector with the numbers of distinct alleles.

"$ref" returns a character vector of reference alleles

"$alt" returns a character vector of alternative alleles (delimited by comma)

"$chrom_pos" returns characters with the combination of chromosome and position, e.g., "1:1272721".

"$chrom_pos_allele" returns characters with the combination of chromosome, position and alleles, e.g., "1:1272721_A_G" (i.e., chr:position_REF_ALT).

The algorithm is highly optimized by blocking the computations to exploit the high-speed memory instead of disk.

Value

A vector, a list of values or none.

Author(s)

Xiuwen Zheng

See Also

seqBlockApply, seqSetFilter, seqGetData, seqParallel, seqGetParallel

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
(samp.id <- seqGetData(f, "sample.id"))
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))
# set sample and variant filters
set.seed(100)
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)],
    variant.id=sample(variant.id, 10))

# read
seqApply(f, "genotype", FUN=print, margin="by.variant")
seqApply(f, "genotype", FUN=print, margin="by.variant", .useraw=TRUE)

seqApply(f, "genotype", FUN=print, margin="by.sample")
seqApply(f, "genotype", FUN=print, margin="by.sample", .useraw=TRUE)

# read multiple variables variant by variant
seqApply(f, c(geno="genotype", phase="phase", rsid="annotation/id",
            DP="annotation/format/DP"), FUN=print, as.is="none")

# get the numbers of alleles per variant
seqApply(f, "allele",
            FUN=function(x) length(unlist(strsplit(x,""))), as.is="integer")

# output to a file
fl <- file("tmp.txt", "wt")
seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is=fl)
close(fl)
readLines("tmp.txt")

seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is=stdout())
seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is="integer")
# should be identical

################################################################
# with an index of variant

seqApply(f, c(geno="genotype", phase="phase", rsid="annotation/id"),
            FUN=function(index, x) { print(index); print(x); index },
            as.is="integer", var.index="relative")
# it is as the same as
which(seqGetFilter(f)$variant.sel)

################################################################
# reset sample and variant filters
seqResetFilter(f)

# calculate the frequency of reference allele,
# a faster version could be obtained by C coding
af <- seqApply(f, "genotype", FUN=function(x) mean(x==0L, na.rm=TRUE),
            as.is="double")
length(af)
seqAsVCF

summary(af)

# apply the user-defined function sample by sample
# reset sample and variant filters
seqResetFilter(f)
summary(seqApply(f, "genotype", FUN=function(x) { mean(is.na(x)) },
   margin="by.sample", as.is="double")

# set sample and variant filters
set.seed(100)
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)],
   variant.id=sample(variant.id, 10))

seqApply(f, "genotype", FUN=print, margin="by.variant", as.is="none")
seqApply(f, "genotype", FUN=print, margin="by.sample", as.is="none")
seqApply(f, c(sample.id="sample.id", genotype="genotype"), FUN=print,
   margin="by.sample", as.is="none")

# close the GDS file
seqClose(f)

# delete the temporary file
unlink("tmp.txt")

---

seqAsVCF    

**VariantAnnotation objects**

Description

Create a `VCF-class` object

Usage

`seqAsVCF(x, chr.prefix="", info=NULL, geno=NULL)`

Arguments

- `x` a `SeqVarGDSClass` object
- `chr.prefix` prefix to add to seqlevels
- `info` which INFO fields to return
- `geno` which GENO fields to return
seqBED2GDS

Details

Coerces a SeqVarGDSClass object to a VCF-class object. Row names correspond to the variant.id. info and geno specify the 'INFO' and 'GENO' (FORMAT) fields to return, respectively. If not specified, all fields are returned; if 'NA' no fields are returned. Use seqSetFilter prior to calling seqAsVCF to specify samples and variants to return.

The VariantAnnotation package should be loaded to explore this object.

Value

A CollapsedVCF object.

Author(s)

Stephanie Gogarten, Xiuwen Zheng

See Also

VCF-class

Examples

gds <- seqOpen(seqExampleFileName("gds"))

## Not run:
library(VariantAnnotation)
seqAsVCF(gds)

## End(Not run)
seqClose(gds)

seqBED2GDS  Conversion between PLINK BED and SeqArray GDS

Description

Conversion between PLINK BED format and SeqArray GDS format.

Usage

seqBED2GDS(bed.fn, fam.fn, bim.fn, out.gdsfn, compress.geno="LZMA_RA",
compress.annotation="LZMA_RA", chr.conv=TRUE, optimize=TRUE, digest=TRUE,
parallel=FALSE, verbose=TRUE)
seqGDS2BED(gdsfile, out.fn, write.rsid=c("auto", "annot_id", "chr_pos_ref_alt"),
multi.row=FALSE, verbose=TRUE)
Arguments

bed.fn the file name of binary file, genotype information
fam.fn the file name of first six columns of ".ped"
bim.fn the file name of extended MAP file: two extra columns = allele names
gdsfile character (a GDS file name), or a SeqVarGDSClass object
out.gdsfn the file name, output a file of SeqArray format
out.fn the file name of PLINK binary format without extended names
compress.geno the compression method for "genotype"; optional values are defined in the function add.gdsn
compress.annotation the compression method for the GDS variables, except "genotype"; optional values are defined in the function add.gdsn
chr.conv if TRUE, convert numeric chromosome codes 23 to X, 24 to Y, 25 to XY, and 26 to MT
optimize if TRUE, optimize the access efficiency by calling cleanup.gds
digest a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add hash codes to the GDS file if TRUE or a digest algorithm is specified
parallel FALSE (serial processing), TRUE (parallel processing), a numeric value indicating the number of cores, or a cluster object for parallel processing; parallel is passed to the argument cl in seqParallel, see seqParallel for more details
write.rsid "annot_id": use the node "annotation/id" for the variant IDs; "chr_pos_ref_alt": use the format "chrom_position_ref_alt"; "auto": use "annotation/id" for the variant IDs if it is not a blank string or ".", otherwise use "chrom_position_ref_alt"
multi.row if TRUE, a multiallelic site is converted to multiple rows in PLINK bim and bed files
verbose if TRUE, show information

Value

Return the file name of SeqArray file with an absolute path.

Author(s)

Xiuwen Zheng

See Also

seqSNP2GDS, seqVCF2GDS
Examples

```r
library(SNPRelate)

# PLINK BED files
bed.fn <- system.file("extdata", "plinkhapmap.bed.gz", package="SNPRelate")
fam.fn <- system.file("extdata", "plinkhapmap.fam.gz", package="SNPRelate")
bim.fn <- system.file("extdata", "plinkhapmap.bim.gz", package="SNPRelate")

# convert bed to gds
seqBED2GDS(bed.fn, fam.fn, bim.fn, "tmp.gds")
seqSummary("tmp.gds")

# convert gds to bed
gdsfn <- seqExampleFileName("gds")
seqGDS2BED(gdsfn, "plink")

# remove the temporary file
unlink(c("tmp.gds", "plink.fam", "plink.bim", "plink.bed"), force=TRUE)
```

seqBlockApply  
Apply Functions Over Array Margins via Blocking

Description

Returns a vector or list of values obtained by applying a function to margins of genotypes and annotations via blocking.

Usage

```r
seqBlockApply(gdsfile, var.name, FUN, margin=c("by.variant"),
              as.is=c("none", "list", "unlist"), var.index=c("none", "relative", "absolute"),
              bsize=1024L, parallel=FALSE, .useraw=FALSE, .padNA=TRUE, .tolist=FALSE,
              .progress=FALSE, ...)
```

Arguments

- **gdsfile**: a `SeqVarGDSClass` object
- **var.name**: the variable name(s), see details
- **FUN**: the function to be applied
- **margin**: giving the dimension which the function will be applied over
- **as.is**: returned value: a list, an integer vector, etc; return nothing by default as.is="none"; as.is can be a `connection` object, or a GDS node `gdsn.class` object; if "unlist" is used, produces a vector which contains all the atomic components, via `unlist(...)`, recursive=FALSE)
var.index if "none" (by default), call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1 if margin="by.variant": "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data

bsize block size

parallel FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details.

.useraw TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW instead of INTEGER if possible; for genotypes, 0xFF is missing value if RAW is used

.padNA TRUE, pad a variable-length vector with NA if the number of data points for each variant is not greater than 1

tolist if TRUE, return a list of vectors instead of the structure list(length, data) for variable-length data

.progress if TRUE, show progress information

... optional arguments to FUN

Details

The variable name should be "sample.id", "variant.id", "position", "chromosome", "allele", "genotype", "annotation/id", "annotation/qual", "annotation/filter", "annotation/info/VARIABLE_NAME", or "annotation/format/VARIABLE_NAME".

"@genotype", "annotation/info/@VARIABLE_NAME" or "annotation/format/@VARIABLE_NAME" are used to obtain the index associated with these variables.

"$dosage" is also allowed for the dosages of reference allele (integer: 0, 1, 2 and NA for diploid genotypes).

"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles.

"$num_allele" returns an integer vector with the numbers of distinct alleles.

"$ref" returns a character vector of reference alleles

"$alt" returns a character vector of alternative alleles (delimited by comma)

"$chrom_pos" returns characters with the combination of chromosome and position, e.g., "1:1272721".

"$chrom_pos_allele" returns characters with the combination of chromosome, position and alleles, e.g., "1:1272721_A_G" (i.e., chr:position_REF_ALT).

"$variant_index" returns the indices of selected variants starting from 1, and "$sample_index" returns the indices of selected samples starting from 1.

The algorithm is highly optimized by blocking the computations to exploit the high-speed memory instead of disk.

Value

A vector, a list of values or none.
Author(s)

Xiuwen Zheng

See Also

seqApply, seqSetFilter, seqGetData, seqParallel, seqGetParallel

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
(samp.id <- seqGetData(f, "sample.id"))
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

# set sample and variant filters
set.seed(100)
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)],
variant.id=sample(variant.id, 10))

# read
seqApply(f, "genotype", FUN=print, margin="by.variant")
seqApply(f, "genotype", FUN=print, margin="by.variant", .useraw=TRUE)

seqApply(f, "genotype", FUN=print, margin="by.sample")
seqApply(f, "genotype", FUN=print, margin="by.sample", .useraw=TRUE)

# read in block
seqGetData(f, "$dosage")
seqBlockApply(f, "$dosage", print, bsize=3)
seqBlockApply(f, "$dosage", function(x) x, as.is="list", bsize=3)
seqBlockApply(f, c(dos="$dosage", pos="position"), print, bsize=3)

# close the GDS file
seqClose(f)
**seqClose-methods**

**Close the SeqArray GDS File**

**Description**

Closes a SeqArray GDS file which is open.

**Usage**

```r
## S4 method for signature 'gds.class'
seqClose(object)
## S4 method for signature 'SeqVarGDSClass'
seqClose(object)
```

---

**seqCheck**(gdsfile, verbose=TRUE)

**Description**

Performs data integrity on a SeqArray GDS file.

**Usage**

```r
seqCheck(gdsfile, verbose=TRUE)
```

**Arguments**

- `gdsfile`: a `SeqVarGDSClass` object, or a file name
- `verbose` if TRUE, display information

**Value**

A list of the following components:

- `hash`: a `data.frame` for hash checking, including `algo` for digest algorithms and `ok` for the checking states
- `dimension`: a `data.frame` for checking the dimension of each variable, including `ok` for the checking states and `info` for the error messages

**Author(s)**

Xiuwen Zheng

**Examples**

```r
# the GDS file
(gds.fn <- seqExampleFileName("gds"))

seqCheck(gds.fn)
```
Arguments

object a SeqArray object

Details

If object is

- `gds.class`, close a general GDS file
- `SeqVarGDSClass`, close the sequence GDS file.

Value

None.

Author(s)

Xiuwen Zheng

See Also

seqOpen

Description

Deletes variables in the SeqArray GDS file.

Usage

```r
seqDelete(gdsfile, info.var=character(), fmt.var=character(),
          samp.var=character(), verbose=TRUE)
```

Arguments

- `gdsfile` a `SeqVarGDSClass` object
- `info.var` the variables in the INFO field, i.e., "annotation/info/VARIABLE_NAME"
- `fmt.var` the variables in the FORMAT field, i.e., "annotation/format/VARIABLE_NAME"
- `samp.var` the variables in the sample annotation field, i.e., "sample.annotation/VARIABLE_NAME"
- `verbose` if TRUE, show information

Value

None.
seqDigest

Hash function digests

Description

Create hash function digests for all or a subset of data

Usage

seqDigest(gdsfile, varname, algo=c("md5"), verbose=FALSE)

Arguments

gdsfile a SeqVarGDSClass object
varname the variable name(s), see details
algo the digest hash algorithm: "md5"
verbose if TRUE, show progress information
Details

The variable name should be "sample.id", "variant.id", "position", "chromosome", "allele", "annotation/id", "annotation/qual", "annotation/filter", "annotation/info/VARIABLE_NAME", or "annotation/format/VARIABLE_NAME".

Users can define a subset of data via seqSetFilter and create a hash digest for the subset only.

Value

A hash character.

Author(s)

Xiuwen Zheng

See Also

seqSetFilter, seqApply

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
f <- seqOpen(gds.fn)

seqDigest(f, "genotype")
seqDigest(f, "annotation/filter")
seqDigest(f, "annotation/format/DP")

# close the GDS file
seqClose(f)

Description

Create a new empty GDS file.

Usage

seqEmptyFile(outfn, sample.id=character(), verbose=TRUE)

Arguments

- **outfn**: the output file name for a GDS file
- **sample.id**: a list of sample IDs
- **verbose**: if TRUE, show information
seqExampleFileName

Value
None.

Author(s)
Xiuwen Zheng

See Also
seqVCF2GDS

Examples
seqEmptyFile("tmp.gds")

(f <- seqOpen("tmp.gds"))
seqClose(f)

# remove the temporary file
unlink("tmp.gds", force=TRUE)

<table>
<thead>
<tr>
<th>seqExampleFileName</th>
<th>Example files</th>
</tr>
</thead>
</table>

Description
The example files of VCF and GDS format.

Usage
seqExampleFileName(type=c("gds", "vcf", "KG_Phase1"))

Arguments

type either "gds" (by default) or "vcf"

Value
Return the path of a VCF file in the package if type="vcf", or the path of a GDS file if type="gds". If type="KG_Phase1", return the path of GDS file on Chromosome 22 of the 1000 Genomes Phase 1 project.

Author(s)
Xiuwen Zheng
seqExport

Description
Exports to a GDS file with selected samples and variants, which are defined by seqSetFilter().

Usage
seqExport(gdsfile, out.fn, info.var=NULL, fmt.var=NULL, samp.var=NULL, optimize=TRUE, digest=TRUE, verbose=TRUE)

Arguments
- gdsfile: a SeqVarGDSClass object
- out.fn: the file name of output GDS file
- info.var: characters, the variable name(s) in the INFO field for import; or NULL for all variables
- fmt.var: characters, the variable name(s) in the FORMAT field for import; or NULL for all variables
- samp.var: characters, the variable name(s) in the folder "sample.annotation"
- optimize: if TRUE, optimize the access efficiency by calling cleanup.gds
- digest: a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if TRUE or a digest algorithm is specified
- verbose: if TRUE, show information

Value
Return the file name of GDS format with an absolute path.

Author(s)
Xiuwen Zheng

See Also
seqVCF2GDS
Examples

```r
# open the GDS file
(gds.fn <- seqExampleFileName("gds"))
(f <- seqOpen(gds.fn))

# get 'sample.id'
head(samp.id <- seqGetData(f, "sample.id"))

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

set.seed(100)
# set sample and variant filters
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10,12,14,16)])
seqSetFilter(f, variant.id=sample(variant.id, 100))

# export
seqExport(f, "tmp.gds")
seqExport(f, "tmp.gds", info.var=character())
seqExport(f, "tmp.gds", fmt.var=character())
seqExport(f, "tmp.gds", samp.var=character())

# show file
(f1 <- seqOpen("tmp.gds")); seqClose(f1)

# close
seqClose(f)

# delete the temporary file
unlink("tmp.gds")
```

seqGDS2SNP

Convert to a SNP GDS File

Description

Converts a SeqArray GDS file to a SNP GDS file.

Usage

```r
seqGDS2SNP(gdsfile, out.gdsfn, dosage=FALSE, compress.geno="LZMA_RA",
compress.annotation="LZMA_RA", ds.type=c("packedreal16", "float", "double"),
optimize=TRUE, verbose=TRUE)
```
Arguments

- **gdsfile**: character (a GDS file name), or a `SeqVarGDSClass` object
- **out.gdsfn**: the file name, output a file of VCF format
- **dosage**: a logical value, or characters for the variable name of dosage in the SeqArray file; if FALSE exports genotypes, otherwise exports dosages
- **compress.geno**: the compression method for "genotype"; optional values are defined in the function `add.gdsn`
- **compress.annotation**: the compression method for the GDS variables, except "genotype"; optional values are defined in the function `add.gdsn`
- **ds.type**: applicable when import dosages, the data type for storing dosages; see `add.gdsn`; ds.type="packedreal16" by default
- **optimize**: if TRUE, optimize the access efficiency by calling `cleanup.gds`
- **verbose**: if TRUE, show information

Details

- `seqSetFilter` can be used to define a subset of data for the conversion.

Value

Return the file name of VCF file with an absolute path.

Author(s)

Xiuwen Zheng

See Also

- `seqSNP2GDS`, `seqVCF2GDS`, `seqGDS2VCF`

Examples

```r
# the GDS file
gds.fn <- seqExampleFileName("gds")
seqGDS2SNP(gds.fn, "tmp.gds")

# delete the temporary file
unlink("tmp.gds")
```
**seqGDS2VCF**  

*Convert to a VCF File*

**Description**  
Converts a SeqArray GDS file to a Variant Call Format (VCF) file.

**Usage**  
```
seqGDS2VCF(gdsfile, vcf.fn, info.var=NULL, fmt.var=NULL, chr_prefix="", use_Rsamtools=TRUE, verbose=TRUE)
```

**Arguments**
- `gdsfile` a `SeqVarGDSClass` object
- `vcf.fn` the file name, output a file of VCF format; or a `connection` object
- `info.var` a list of variable names in the INFO field, or NULL for using all variables; character(0) for no variable in the INFO field
- `fmt.var` a list of variable names in the FORMAT field, or NULL for using all variables; character(0) for no variable in the FORMAT field
- `chr_prefix` the prefix of chromosome, e.g., "chr"; no prefix by default
- `use_Rsamtools` TRUE for loading the Rsamtools package, see details
- `verbose` if TRUE, show information

**Details**
`seqSetFilter` can be used to define a subset of data for the export.

If the filename extension is "gz", the gzip compression algorithm is used to compress the output data. When the Rsamtools package is installed and `use_Rsamtools=TRUE`, the exported file utilizes the bgzf format (`bgzip`, a variant of gzip format) allowing for fast indexing.

**Value**
Return the file name of VCF file with an absolute path.

**Author(s)**
Xiuwen Zheng

**References**
See Also

seqVCF2GDS

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# output the first 10 samples
samp.id <- seqGetData(f, "sample.id")
seqSetFilter(f, sample.id=samp.id[1:5])

# convert
seqGDS2VCF(f, "tmp.vcf.gz")

# no INFO and FORMAT
seqGDS2VCF(f, "tmp1.vcf.gz", info.var=character(), fmt.var=character())

# output BN,GP,AA,DP,HM2 in INFO (the variables are in this order), no FORMAT
seqGDS2VCF(f, "tmp2.vcf.gz", info.var=c("BN","GP","AA","DP","HM2"),
fmt.var=character())

# read
(txt <- readLines("tmp.vcf.gz", n=20))
(txt <- readLines("tmp1.vcf.gz", n=20))
(txt <- readLines("tmp2.vcf.gz", n=20))

# Users could compare the new VCF file with the original VCF file
# call "diff" in Unix (a command line tool comparing files line by line)
seqResetFilter(f)

# convert
seqGDS2VCF(f, "tmp.vcf.gz")

# file.copy(seqExampleFileName("vcf"), "old.vcf.gz", overwrite=TRUE)
# system("diff <(gunzip -c old.vcf.gz) <(gunzip -c tmp.vcf.gz")

# 1a2,3
# > ##fileDate=20130309
# > # source=SeqArray_RPackage_v1.0
# LOOK GOOD!

# delete temporary files
unlink(c("tmp.vcf.gz", "tmp1.vcf.gz", "tmp2.vcf.gz"))

# close the GDS file
seqClose(f)

---

**seqGet2bGeno**  
*Get packed genotypes*

**Description**

Gets a RAW matrix of genotypes in a packed 2-bit format.

**Usage**

```r
seqGet2bGeno(gdsfile, samp_by_var=TRUE, ext_nbyte=0L, verbose=FALSE)
```

**Arguments**

- `gdsfile`  
a `SeqVarGDSClass` object

- `samp_by_var`  
if TRUE, return a sample-by-variant matrix; otherwise, return a variant-by-sample matrix

- `ext_nbyte`  
additional `ext_nbyte` row(s) with missing genotypes

- `verbose`  
if TRUE, show progress information

**Details**

If `samp_by_var=TRUE`, the function returns a sample-by-variant RAW matrix (nrow = ceiling(# of samples / 4)); otherwise, it returns a variant-by-sample RAW matrix (nrow = ceiling(# of variants / 4)). The RAW matrix consists of a 2-bit array, with 0, 1 and 2 for dosage, and 3 for missing genotype.

**Value**

Return a RAW matrix.

**Author(s)**

Xiuwen Zheng

**See Also**

`seqGetData`
Examples

# open a GDS file
f <- seqOpen(seqExampleFileName("gds"))

str(seqGet2bGeno(f))
str(seqGet2bGeno(f, samp_by_var=FALSE))

# close the GDS file
seqClose(f)

---

**seqGetData**  
**Get Data**

**Description**

Gets data from a SeqArray GDS file.

**Usage**

```
seqGetData(gdsfile, var.name, .useraw=FALSE, .padNA=TRUE, .tolist=FALSE, .envir=NULL)
```

**Arguments**

- `gdsfile`: a `SeqVarGDSClass` object
- `var.name`: a variable name or a character vector, see details
- `.useraw`: TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW for small numbers instead of INTEGER if possible; 0xFF is missing value if RAW is used
- `.padNA`: TRUE, pad a variable-length vector with NA if the number of data points for each variant is not greater than 1
- `.tolist`: if TRUE, return a list of vectors instead of the structure list(length, data) for variable-length data
- `.envir`: NULL, an environment object, a list or a data.frame

**Details**

The variable name should be "sample.id", "variant.id", "position", "chromosome", "allele", "genotype", "annotation/id", "annotation/qual", "annotation/filter", "annotation/info/VARIABLE_NAME", or "annotation/format/VARIABLE_NAME".

"@genotype", "annotation/info/@VARIABLE_NAME" or "annotation/format/@VARIABLE_NAME" are used to obtain the index associated with these variables.

"$dosage" is also allowed for the dosages of reference allele (integer: 0, 1, 2 and NA for diploid genotypes).
"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles.

"$dosage_sp" returns a sparse matrix (dgCMatrix) for the dosages of alternative allele without distinguishing different alternative alleles.

"$num_allele" returns an integer vector with the numbers of distinct alleles.

"$ref" returns a character vector of reference alleles. "$alt" returns a character vector of alternative alleles (delimited by comma).

"$chrom_pos" returns characters with the combination of chromosome and position, e.g., "1:1272721". "$chrom_pos2" is similar to "$chrom_pos", except the suffix "_1" is added to the first duplicate following the variant, "_2" is added to the second duplicate, and so on. "$chrom_pos_allele" returns characters with the combination of chromosome, position and alleles, e.g., "1:1272721_A_G" (i.e., chr:position_REF_ALT).

"$variant_index" returns the indices of selected variants starting from 1, and "$sample_index" returns the indices of selected samples starting from 1.

"$:VAR" return the variable "VAR" from .envir according to the selected variants.

Value

Return vectors, matrices or lists (with length and data components) with a class name SeqVarDataList.

Author(s)

Xiuwen Zheng

See Also

seqSetFilter, seqApply, seqNewVarData, seqListVarData

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
(samp.id <- seqGetData(f, "sample.id"))
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

# get 'chromosome'
table(seqGetData(f, "chromosome"))

# get 'allele'
head(seqGetData(f, "allele"))
# "T,C" "G,A" "G,A" ...
# get '$chrom_pos'
head(seqGetData(f, "$chrom_pos"))

# get '$dosage'
seqGetData(f, "$dosage")[1:6, 1:10]

# get a sparse matrix of dosages
seqGetData(f, "$dosage_sp")[1:6, 1:10]

# get '$num_allele'
head(seqGetData(f, "$num_allele"))

# set sample and variant filters
set.seed(100)
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)])
seqSetFilter(f, variant.id=sample(variant.id, 10))

# get a list
seqGetData(f, c(chr="chromosome", pos="position", allele="allele"))

# get the indices of selected variants/samples
seqGetData(f, "$variant_index")
seqGetData(f, "$sample_index")

# get genotypic data
seqGetData(f, "genotype")

# get annotation/info/DP
seqGetData(f, "annotation/info/DP")

# get annotation/info/AA, a variable-length dataset
seqGetData(f, "annotation/info/AA", .padNA=FALSE)
# $length <- indicating the length of each variable-length data
# [1] 1 1 1 1 1 1 ...  
# $data <- the data according to $length
# [1] "T" "C" "T" "C" "G" "C" ...  

# or return a simplified vector
seqGetData(f, "annotation/info/AA", .padNA=TRUE)

# get annotation/format/DP, a variable-length dataset
seqGetData(f, "annotation/format/DP")
# $length <- indicating the length of each variable-length data
# [1] 1 1 1 1 1 1 ...  
# $data <- the data according to $length
# variant
# [1,] 25 25 22 3 4 17 ...

# get values from R environment
env <- new.env()
env$x <- 1:1348 / 10
env$x[seqGetData(f, "$variant_index")]
seqGetData(f, "$:x", .envir=env)

# close the GDS file
seqClose(f)

---

**seqGetFilter**  
*Get the Filter of GDS File*

**Description**

Gets the filter of samples and variants.

**Usage**

```r
seqGetFilter(gdsfile, .useraw=FALSE)
```

**Arguments**

- `gdsfile`  
a SeqVarGDSClass object
- `.useraw`  
returns logical vectors if FALSE, and returns raw vectors if TRUE

**Value**

Return a list:

- `sample.sel`  
a logical/raw vector indicating selected samples
- `variant.sel`  
a logical/raw vector indicating selected variants

**Author(s)**

Xiuwen Zheng

**See Also**

- `seqSetFilter`

**Examples**

```r
# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
```
(samp.id <- seqGetData(f, "sample.id"))
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

# set sample and variant filters
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8,10)])
set.seed(100)
seqSetFilter(f, variant.id=sample(variant.id, 10))

# get filter
z <- seqGetFilter(f)

# the number of selected samples
sum(z$sample.sel)
# the number of selected variants
sum(z$variant.sel)

z <- seqGetFilter(f, .useraw=TRUE)
head(z$sample.sel)
head(z$variant.sel)

# close the GDS file
seqClose(f)

---

**seqMerge**

*Merge Multiple SeqArray GDS Files*

**Description**

Merges multiple SeqArray GDS files.

**Usage**

```r
seqMerge(gds.fn, out.fn, storage.option="LZMA_RA", info.var=NULL, fmt.var=NULL, samp.var=NULL, optimize=TRUE, digest=TRUE, geno.pad=TRUE, verbose=TRUE)
```

**Arguments**

- `gds.fn` the file names of multiple GDS files
- `out.fn` the output file name
- `storage.option` specify the storage and compression option, "ZIP_RA" (**seqStorageOption**("ZIP_RA")); or "LZMA_RA" to use LZMA compression algorithm with higher compression ratio (by default)
info.var characters, the variable name(s) in the INFO field; NULL for all variables, or character() excludes all INFO variables

fmt.var characters, the variable name(s) in the FORMAT field; NULL for all variables, or character() excludes all FORMAT variables

samp.var characters, the variable name(s) in 'sample.annotation'; or NULL for all variables

optimize if TRUE, optimize the access efficiency by calling cleanup.gds

digest a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if TRUE or a digest algorithm is specified

geno.pad TRUE, pad a 2-bit genotype array in bytes to avoid recompressing genotypes if possible

verbose if TRUE, show information

Details

The function merges multiple SeqArray GDS files. Users can specify the compression method and level for the new GDS file. If gds.fn contains one file, users can change the storage type to create a new file.

WARNING: the functionality of seqMerge() is limited.

Value

Return the file name of GDS format with an absolute path.

Author(s)

Xiuwen Zheng

See Also

seqVCF2GDS, seqExport

Examples

# the VCF file
vcf.fn <- seqExampleFileName("vcf")

# the number of variants
total.count <- seqVCF_Header(vcf.fn, getnum=TRUE)$num.variant

split.cnt <- 5
start <- integer(split.cnt)
count <- integer(split.cnt)

s <- (total.count+1) / split.cnt
st <- 1L
for (i in 1:split.cnt)
{
  z <- round(s * i)
start[i] <- st
count[i] <- z - st
st <- z
}

fn <- paste0("tmp", 1:split.cnt, ".gds")

# convert to 5 gds files
for (i in 1:split.cnt)
{
  seqVCF2GDS(vcf.fn, fn[i], storage.option="ZIP_RA",
            start=start[i], count=count[i])
}

# merge different variants
seqMerge(fn, "tmp.gds", storage.option="ZIP_RA")
seqSummary("tmp.gds")

#### merging different samples ####

vcf.fn <- seqExampleFileName("gds")
file.copy(vcf.fn, "test.gds", overwrite=TRUE)

# modify 'sample.id'
seqAddValue("test.gds", "sample.id", paste0("S", 1:90), replace=TRUE)

# merging
seqMerge(c(vcf.fn, "test.gds"), "output.gds", storage.option="ZIP_RA")

# delete the temporary files
unlink(c("tmp.gds", "test.gds", "output.gds"), force=TRUE)
unlink(fn, force=TRUE)

---

**seqMissing**

**Missing genotype percentage**

**Description**

Calculates the missing rates per variant or per sample.

**Usage**

```r
seqMissing(gdsfile, per.variant=TRUE, parallel=seqGetParallel(), verbose=FALSE)
```

**Arguments**

- `gdsfile` a `SeqVarGDSClass` object
seqMissing

- **per.variant**: missing rate per variant if TRUE, missing rate per sample if FALSE, or calculating missing rates for variants and samples if NA
- **parallel**: FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details.
- **verbose**: if TRUE, show progress information

**Details**

If the gds node 'genotype/data' (integer genotypes) is not available, the node 'annotation/format/DS' (numeric genotype dosages for alternative alleles) will be used to calculate allele frequencies. At a site, it assumes 'annotation/format/DS' stores the dosage of the 1st alternative allele in the 1st column, 2nd alt. allele in the 2nd column if it is multi-allelic, and so on.

**Value**

A vector of missing rates, or a list(variant, sample) for both variants and samples.

**Author(s)**

Xiuwen Zheng

**See Also**

seqAlleleFreq, seqNumAllele, seqParallel, seqGetParallel

**Examples**

```r
# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

summary(m1 <- seqMissing(f, TRUE, verbose=TRUE))
summary(m2 <- seqMissing(f, FALSE, verbose=TRUE))

str(m <- seqMissing(f, NA, verbose=TRUE))
identical(m1, m$variant) # should be TRUE
identical(m2, m$sample) # should be TRUE

# close the GDS file
seqClose(f)
```
seqNewVarData

Variable-length data

Description

Gets a variable-length data object.

Usage

seqNewVarData(len, data)
seqListVarData(obj)

Arguments

len a non-negative vector for variable lengths
data a vector of data according to len
obj a SeqVarDataList object

Details

seqNewVarData() creates a SeqVarDataList object for variable-length data, and seqListVarData() converts the SeqVarDataList object to a list. seqGetData() returns a SeqVarDataList object for variable-length data; seqAddValue() can add a SeqVarDataList object to a GDS file.

Value

Return a SeqVarDataList object.

Author(s)

Xiuwen Zheng

See Also

seqGetData, seqAddValue

Examples

obj <- seqNewVarData(c(1,2,1,0,2), c("A", "B", "B", "C", "E", "E"))
obj

seqListVarData(obj)
seqNumAllele

Description

Returns the numbers of alleles for each site.

Usage

seqNumAllele(gdsfile)

Arguments

gdsfile a SeqVarGDSClass object

Value

The numbers of alleles for each site.

Author(s)

Xiuwen Zheng

See Also

seqAlleleFreq, seqMissing

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
f <- seqOpen(gds.fn)

table(seqNumAllele(f))

# close the GDS file
seqClose(f)
seqOpen: Open a SeqArray GDS File

Description

Opens a SeqArray GDS file.

Usage

seqOpen(gds.fn, readonly=TRUE, allow.duplicate=FALSE)

Arguments

gds.fn the file name
readonly whether read-only or not
allow.duplicate if TRUE, it is allowed to open a GDS file with read-only mode when it has been opened in the same R session

Details

It is strongly suggested to call seqOpen instead of openfn.gds, since seqOpen will perform internal checking for data integrality.

Value

Return an object of class SeqVarGDSClass inherited from gds.class.

Author(s)

Xiuwen Zheng

See Also

seqClose, seqGetData, seqApply

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# open the GDS file
gdsfile <- seqOpen(gds.fn)

# display the contents of the GDS file in a hierarchical structure
gdsfile

# close the GDS file
seqClose(gdsfile)
seqOptimize

Optimize the Storage of Data Array

Description

Transpose data array or matrix for possibly higher-speed access.

Usage

seqOptimize(gdsfn, target=c("chromosome", "by.sample"), format.var=TRUE, cleanup=TRUE, verbose=TRUE)

Arguments

gdsfn the file name of GDS

target "chromosome", "by.sample"; see details

format.var a character vector for selected variable names, or TRUE for all variables, according to "annotation/format"

cleanup call link{cleanup.gds} if TRUE

verbose if TRUE, show information

Details

"chromosome": adding or updating two additional nodes '@chrom_rle_val' and '@chrom_rle_len' for faster chromosome indexing, requiring SeqArray>=v1.20.0.

"by.sample": optimizing GDS file for seqApply(..., margin="by.sample"). Warning: optimizing GDS file for reading data by sample may increase file size by up to 2X as genotype data and all format data are duplicated.

Value

None.

Author(s)

Xiuwen Zheng

See Also

seqGetData, seqApply
Examples

# the file name of VCF
(vcf.fn <- seqExampleFileName("vcf"))
# or vcf.fn <- "C:/YourFolder/Your_VCF_File.vcf"

# convert
seqVCF2GDS(vcf.fn, "tmp.gds", storage.option="ZIP_RA")

# prepare data for the SeqVarTools package
seqOptimize("tmp.gds", target="by.sample")

# list the structure of GDS variables
(f <- seqOpen("tmp.gds"))
# close
seqClose(f)

# delete the temporary file
unlink("tmp.gds")

---

seqParallel  
Apply Functions in Parallel

Description

Applies a user-defined function in parallel.

Usage

seqParallel(cl=seqGetParallel(), gdsfile, FUN,
           split=c("by.variant", "by.sample", "none"), .combine="unlist",
           .selection.flag=FALSE, .initialize=NULL, .finalize=NULL, .initparam=NULL,
           .balancing=FALSE, .bl_size=10000L, .bl_progress=FALSE, ...)
seqParApply(cl=seqGetParallel(), x, FUN, load.balancing=TRUE, ...)

Arguments

cl NULL or FALSE: serial processing; TRUE: multicore processing (the maximum number of cores minor one); a numeric value: the number of cores to be used; a cluster object for parallel processing, created by the functions in the package parallel, like makeCluster; a BiocParallelParam object from the BiocParallel package. See details

gdsfile a SeqVarGDSClass object, or NULL
FUN the function to be applied, should be like FUN(gdsfile,...) if gdsfile is given, or FUN(...) if gdsfile=NULL

split split the dataset by variant or sample according to multiple processes, or "none" for no split; split="by.variant" by default
seqParallel

.combine define a function for combining results from different processes; by default, "unlist" is used, to produce a vector which contains all the atomic components, via unlist(..., recursive=FALSE): "list", return a list of results created by child processes; "none", no return; or a function with one or two arguments, like "+"

.selection.flag TRUE -- passes a logical vector of selection to the second argument of FUN(gdsfile, selection, ...)

.initialize a user-defined function for initializing workers, should have two arguments (process_id, param)

.finalize a user-defined function for finalizing workers, should have two arguments (process_id, param)

.initparam parameters passed to .initialize and .initialize

.balancing load balancing if TRUE

.bl_size chuck size, the increment for load balancing, 10000 for variants; only applicable if .balancing=TRUE

.bl_progress if TRUE and .balancing=TRUE, show progress information

.x a vector (atomic or list), passed to FUN

.load.balancing if TRUE, call clusterApplyLB instead of clusterApply

... optional arguments to FUN

Details

When cl is TRUE or a numeric value, forking techniques are used to create a new child process as a copy of the current R process, see ?parallel::mcfork. However, forking is not available on Windows, and makeCluster is called to make a cluster which will be deallocated after calling FUN.

It is strongly suggested to use seqParallel together with seqParallelSetup. seqParallelSetup could work around the problem of forking on Windows, without allocating clusters frequently.

The user-defined function could use two predefined variables SeqArray:::process_count and SeqArray:::process_index to tell the total number of cluster nodes and which cluster node being used.

seqParallel(..., gdsfile=..., FUN=..., split="none") could be used to setup multiple streams of pseudo-random numbers, and see nextRNGStream or nextRNGSubStream in the package parallel.

Value

A vector or list of values.

Author(s)

Xiuwen Zheng

See Also

seqSetFilter, seqGetData, seqApply, seqParallelSetup, seqGetParallel
Examples

```r
library(parallel)

# choose an appropriate cluster size or number of cores
seqParallelSetup(2)

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(gdsfile <- seqOpen(gds.fn))

# the uniprocessor version
afreq1 <- seqParallel(, gdsfile, FUN = function(f) {
  seqApply(f, "genotype", as.is="double",
            FUN=function(x) mean(x==0, na.rm=TRUE))
}, split="by.variant")
length(afreq1)
summary(afreq1)

# run in parallel
afreq2 <- seqParallel(, gdsfile, FUN = function(f) {
  seqApply(f, "genotype", as.is="double",
            FUN=function(x) mean(x==0, na.rm=TRUE))
}, split="by.variant")
length(afreq2)
summary(afreq2)

# check
length(afreq1) # 1348
all(afreq1 == afreq2)

# check -- variant splits

# seqParallel(NULL, FUN = function() {
  paste(SeqArray:::process_index, SeqArray:::process_count, sep=" / ")
}, split="none")
```
seqParallelSetup

seqParallel(, NULL, FUN = function() {
    SeqArray:::process_index
}, split="none", .combine=function(i) print(i))

seqParallel(, NULL, FUN = function() {
    SeqArray:::process_index
}, split="none", .combine="+")

#########################################################################

# close the GDS file
seqClose(gdsfile)

# clear the parallel cluster
seqParallelSetup(FALSE)

---

seqParallelSetup  Setup/Get a Parallel Environment

Description

Sets up a parallel environment in R for the current session.

Usage

seqParallelSetup(cluster=TRUE, verbose=TRUE)
seqGetParallel()
seqMulticoreSetup(num, type=c("psock", "fork"), verbose=TRUE)

Arguments

cluster  NULL or FALSE: serial processing; TRUE: parallel processing with the maximum number of cores minor one; a numeric value: the number of cores to be used; a cluster object for parallel processing, created by the functions in the package parallel, like makeCluster. See details
num  the maximum number of cores used for the user-defined multicore setting; FALSE, NA or any value less than 2, to disable the multicore cluster
type  either PSOCK or Fork cluster setup for the multicore setting, the resulting parallel cluster will be used if ‘parallel’ is a number greater than one in associated functions
verbose  if TRUE, show information
Details
When cl is TRUE or a numeric value, forking techniques are used to create a new child process as a copy of the current R process, see ?parallel::mcfork. However, forking is not available on Windows, so multiple processes created by makeCluster are used instead. The R environment option seqarray.parallel will be set according to the value of cluster. Using seqParallelSetup(FALSE) removes the registered cluster, as does stopping the registered cluster.

Value
seqParallelSetup() has no return, and seqGetParallel() returnsgetOption("seqarray.parallel", FALSE).

Author(s)
Xiuwen Zheng

See Also
seqParallel, seqApply

Examples
library(parallel)
seqParallelSetup(2L)

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

# run in parallel
summary(seqMissing(f))

# close the GDS file
seqClose(f)

seqParallelSetup(FALSE)

seqRecompress

Recompress the GDS file

Description
Recompress the SeqArray GDS file.
seqRecompress

Usage

seqRecompress(gds.fn, compress=c("ZIP", "LZ4", "LZMA", "Ultra", "UltraMax", "none"), exclude=character(), optimize=TRUE, verbose=TRUE)

Arguments

gds.fn the file name of SeqArray file
compress the compression method, compress="ZIP" by default
exclude a list of GDS nodes to be excluded, see details
optimize if TRUE, optimize the access efficiency by calling cleanup.gds
verbose if TRUE, show information

Details

This function requires gdsfmt (>= v1.17.2). seqVCF2GDS usually takes lots of memory when the compression method "LZMA_RA.max", "Ultra" or "UltraMax" is specified. So users could call seqVCF2GDS(storge.option="ZIP_RA") first, and then recompress the GDS file with a higher compression option, e.g., "UltraMax". seqRecompress() takes much less memory than seqVCF2GDS(), since it recompresses data in a GDS node each time.

"UltraMax" might be not better than "Ultra", and its behavior is similar to xz -9 --extreme: use a slower variant of the selected compression preset level (-9) to hopefully get a little bit better compression ratio, but with bad luck this can also make it worse.

ls.gdsn(gdsfile, include.hidden=TRUE, recursive=TRUE) returns a list of GDS nodes to be re-compressed, and users can specify the excluded nodes in the argument exclude.

Value

None.

Author(s)

Xiuwen Zheng

See Also

seqVCF2GDS, seqStorageOption

Examples

gds.fn <- seqExampleFileName("gds")
file.copy(gds.fn, "tmp.gds")
seqRecompress("tmp.gds", "LZMA")
unlink("tmp.gds")
seqResetVariantID  Reset Variant ID in SeqArray GDS Files

Description

Resets the variant IDs in multiple SeqArray GDS files.

Usage

seqResetVariantID(gds.fn, set=NULL, digest=TRUE, optimize=TRUE, verbose=TRUE)

Arguments

gds.fn a character vector of multiple GDS file names
set NULL or a logical vector; NULL for resetting all files, or TRUE for resetting variant.id for that GDS file
digest a logical value, if TRUE, add a md5 hash code
optimize if TRUE, optimize the access efficiency by calling cleanup.gds
verbose if TRUE, show information

Details

The variant IDs will be replaced by the numbers in sequential order and adjacent to each file. The variant ID starts from 1 in the first GDS file.

Value

None.

Author(s)

Xiuwen Zheng

See Also

seqVCF2GDS

Examples

fn <- seqExampleFileName("gds")

file.copy(fn, "tmp1.gds", overwrite=TRUE)
file.copy(fn, "tmp2.gds", overwrite=TRUE)

gds.fn <- c("tmp1.gds", "tmp2.gds")
seqResetVariantID(gds.fn)
seqSetFilter-methods

Sets a filter to sample and/or variant.

Usage

```r
## S4 method for signature 'SeqVarGDSClass,ANY'
seqSetFilter(object, variant.sel, 
  sample.sel=NULL, variant.id=NULL, sample.id=NULL, 
  action=c("set", "intersect", "push", "push+set", "push+intersect", "pop"), 
  ret.idx=FALSE, warn=TRUE, verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,GRanges'
seqSetFilter(object, variant.sel, 
  rm.txt="chr", intersect=FALSE, verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,GRangesList'
seqSetFilter(object, variant.sel, 
  rm.txt="chr", intersect=FALSE, verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,IRanges'
seqSetFilter(object, variant.sel, 
  chr, intersect=FALSE, verbose=TRUE)
seqResetFilter(object, sample=TRUE, variant=TRUE, verbose=TRUE)
seqSetFilterChrom(object, include=NULL, is.num=NA, from.bp=NULL, to.bp=NULL, 
  intersect=FALSE, verbose=TRUE)
seqSetFilterPos(object, chr, pos, ref=NULL, alt=NULL, intersect=FALSE, 
  multi.pos=TRUE, ret.idx=FALSE, verbose=TRUE)
seqSetFilterAnnotID(object, id, ret.idx=FALSE, verbose=TRUE)
seqFilterPush(object) # store the current filter
seqFilterPop(object) # restore the last filter
```

Arguments

- `object` a `SeqVarGDSClass` object
variant.sel a logical/raw/index vector indicating the selected variants; \texttt{GRanges}, a GRanges object for the genomic locations; \texttt{GRangesList}, a GRangesList object for storing a collection of GRanges objects; \texttt{IRanges}, a IRanges object for storing a collection of range objects

sample.sel a logical/raw/index vector indicating the selected samples

variant.id ID of selected variants

sample.id ID of selected samples

action "set" – set the current filter via \texttt{sample.id}, \texttt{variant.id}, \texttt{samp.sel} or \texttt{variant.sel}; "intersect" – set the current filter to the intersection of selected samples and/or variants; "push" – push the current filter to the stack, and it could be recovered by "pop" later, no change on the current filter; "push+set" – push the current filter to the stack, and changes the current filter via \texttt{sample.id}, \texttt{variant.id}, \texttt{samp.sel} or \texttt{variant.sel}; "push+intersect" – push the current filter to the stack, and set the current filter to the intersection of selected samples and/or variants; "pop" – pop up the last filter

ret.idx if TRUE, return the index in the output array according to the order of 'sample.id', 'sample.sel', 'variant.id' or 'variant.sel'

rm.txt a character, the characters will be removed from seqnames(\texttt{variant.sel})

chr a vector of character for chromosome coding

pos a vector of numeric values for genome coordinate

sample logical, if TRUE, include all samples

variant logical, if TRUE, include all variants

include NULL, or a vector of characters for specified chromosome(s)

is.num a logical variable: TRUE, chromosome code is numeric; FALSE, chromosome is not numeric; \texttt{is.num=TRUE} is usually used to exclude non-autosomes

from.bp NULL, no limit; a numeric vector, the lower bound of position

to.bp NULL, no limit; a numeric vector, the upper bound of position

intersect if FALSE, the candidate samples/variants for selection are all samples/variants (by default); if TRUE, the candidate samples/variants are from the selected samples/variants defined via the previous call

ref the reference alleles

alt the alternative alleles

multi.pos FALSE, use the first matched position; TRUE, allow multiple variants at the same position

id a character vector for RS IDs (stored in "annotation/id")

warn if TRUE, show a warning when the input \texttt{sample.sel} or \texttt{variant.sel} is not ordered as the GDS file or there is any duplicate

verbose if TRUE, show information
**seqSetFilter-methods**

**Details**

`seqResetFilter(file)` is equivalent to `seqSetFilter(file)`, where the selection arguments in `seqSetFilter` are `NULL`.

If `from.bp` and `to.bp` has values, they should be equal-size as `include`. A trio of `include`, `from.bp` and `to.bp` indicates a region on human genomes. NA in `from.bp` is treated as 0, and NA in `to.bp` is treated as the maximum of integer ($2^{31} - 1$).

**Value**

If `ret.idx=TRUE`, `seqSetFilter()` returns a list with two components `sample_idx` and `variant_idx` to indicate the indices of the output array according to the input 'sample.id', 'sample.sel', 'variant.id' or 'variant.sel'; if `ret.idx=TRUE`, `seqSetFilterAnnotID()` return an index vector; otherwise no return.

**Author(s)**

Xiuwen Zheng

**See Also**

`seqSetFilterCond`, `seqGetFilter`, `seqGetData`, `seqApply`

**Examples**

# the GDS file
gds.fn <- seqExampleFileName("gds")

# display
(f <- seqOpen(gds.fn))

# get 'sample.id'
samp.id <- seqGetData(f, "sample.id")
# "NA06984" "NA06985" "NA06986" ...

# get 'variant.id'
head(variant.id <- seqGetData(f, "variant.id"))

# get 'chromosome'
table(seqGetData(f, "chromosome"))

# get 'allele'
head(seqGetData(f, "allele"))
# "T,C" "G,A" "G,A" ...

# set sample filters
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8)])
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8)], ret.idx=TRUE)

(v <- seqSetFilter(f, sample.id=samp.id[c(8,2,6,4)], ret.idx=TRUE))
all(seqGetData(f, "sample.id")[v$sample_idx] == samp.id[c(8,2,6,4)])
# set variant filters
seqSetFilter(f, variant.id=variant.id[c(2,4,6,8,10,12)], ret.idx=TRUE)
(v <- seqSetFilter(f, variant.id=variant.id[c(12,4,6,10,8,12)], ret.idx=TRUE))
all(variant.id[c(12,4,6,10,8,12)] == seqGetData(f, "variant.id")[v$variant_idx])

set.seed(100)
seqSetFilter(f, variant.id=sample(variant.id, 5))

# get genotypic data
seqGetData(f, "genotype")

## OR
# set sample and variant filters
seqSetFilter(f, sample.sel=c(2,4,6,8))
set.seed(100)
seqSetFilter(f, variant.sel=sample.int(length(variant.id), 5))

# get genotypic data
seqGetData(f, "genotype")

## set the intersection
seqResetFilter(f)
seqSetFilterChrom(f, 10L)
seqSummary(f, "genotype", check="none")

AF <- seqAlleleFreq(f)
table(AF <= 0.9)
seqSetFilter(f, variant.sel=(AF<=0.9), action="intersect")
seqSummary(f, "genotype", check="none")

## chromosome
seqResetFilter(f)

seqSetFilterChrom(f, is.num=TRUE)
seqSummary(f, "genotype", check="none")

seqSetFilterChrom(f, is.num=FALSE)
seqSummary(f, "genotype", check="none")

seqSetFilterChrom(f, 1:4)
seqSummary(f, "genotype", check="none")
table(seqGetData(f, "chromosome"))

# HLA region
seqSetFilterChrom(f, 6, from.bp=29719561, to.bp=32883508)
seqSummary(f, "genotype", check="none")

# two regions
seqSetFilterChrom(f, c(1, 6), from.bp=c(1000000, 29719561),
    to.bp=c(90000000, 32883508))
seqSummary(f, "genotype", check="none")
seqGetData(f, "chromosome")

## intersection option

seqResetFilter(f)
seqSetFilterChrom(f, 6, from.bp=29719561, to.bp=32883508) # MHC
seqSetFilterChrom(f, include=6) # chromosome 6

seqResetFilter(f)
seqSetFilterChrom(f, 6, from.bp=29719561, to.bp=32883508) # MHC
seqSetFilterChrom(f, include=6, intersect=TRUE) # MHC region only

# close the GDS file
seqClose(f)

---

### seqSetFilterCond

**Set a Filter to Variant with Allele Count/Freq**

**Description**

Sets a filter to variant with specified allele count/frequency and missing rate.

**Usage**

```r
gdsfile, maf=NaN, mac=1L, missing.rate=NaN,
    parallel=seqGetParallel(), .progress=FALSE, verbose=TRUE)
```

**Arguments**

- `gdsfile`: a `SeqVarGDSClass` object
- `maf`: minimum minor reference allele frequency, or a range of MAF `maf[1] <= ... < maf[2]`
- `mac`: minimum minor reference allele count, or a range of MAC `mac[1] <= ... < mac[2]`
- `missing.rate`: maximum missing genotype rate
- `.progress`: if TRUE, show progress information
parallel FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details.

verbose if TRUE, show information

Value None.

Author(s) Xiuwen Zheng

See Also

seqSetFilter, seqSetFilterChrom, seqAlleleFreq, seqAlleleCount, seqMissing

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

# display
(f <- seqOpen(gds.fn))

seqSetFilterChrom(f, c(1, 6))
seqSetFilterCond(f, maf=0.05, .progress=TRUE)

seqSetFilterChrom(f, c(1, 6))
seqSetFilterCond(f, maf=c(0.01, 0.05), .progress=TRUE)

# close the GDS file
seqClose(f)
seqSNP2GDS

Arguments

- `gds.fn` the file name of SNP format
- `out.fn` the file name, output a file of SeqArray format
- `storage.option` specify the storage and compression options, "LZMA_RA" to use LZMA compression algorithm with higher compression ratio compared to "ZIP_RA"
- `major.ref` if TRUE, use the major allele as a reference allele; otherwise, use A allele in SNP GDS file as a reference allele
- `ds.type` applicable when import dosages, the data type for storing dosages; see `add.gdsn`; `ds.type="packedreal16"` by default
- `optimize` if TRUE, optimize the access efficiency by calling `cleanup.gds`
- `digest` a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add hash codes to the GDS file if TRUE or a digest algorithm is specified
- `verbose` if TRUE, show information

Value

Return the file name of SeqArray file with an absolute path. If the input file is genotype dosage, the dosage matrix is stored in the node `annotation/format/DS` with the estimated dosage of alternative alleles. Any value less than 0 or greater than 2 will be replaced by NaN.

Author(s)

Xiuwen Zheng

See Also

`seqGDS2SNP`, `seqVCF2GDS`, `seqGDS2VCF`, `seqBED2GDS`

Examples

```r
library(SNPRelate)

# the GDS file
gds.fn <- snpgdsExampleFileName()
seqSNP2GDS(gds.fn, "tmp.gds")
seqSummary("tmp.gds")

# remove the temporary file
unlink("tmp.gds", force=TRUE)
```
seqStorageOption

Description

Storage and compression options for GDS import and merging.

Usage


Arguments

compression the default compression level ("ZIP_RA"), see add.gdsn for the description of compression methods
mode a character vector, specifying storage type for corresponding variable, e.g., c(‘annotation/info/HM’="int16", ‘annotation/format/PL’="int")
float.mode specify the storage mode for read numbers, e.g., "float32", "float64", "packedreal16"; the additional parameters can follow by colon, like "packedreal16:scale=0.0001"
geno.compress NULL for the default value, or the compression method for genotypic data
info.compress NULL for the default value, or the compression method for data sets stored in the INFO field (i.e., "annotation/info")
format.compress NULL for the default value, or the compression method for data sets stored in the FORMAT field (i.e., "annotation/format")
index.compress NULL for the default value, or the compression method for data index variables (e.g., "annotation/info/@HM")
...
other specified storage compression for corresponding variable, e.g., ‘annotation/info/HM’="ZIP_MAX"

Details

The compression modes "Ultra" and "UltraMax" attempt to maximize the compression ratio using gigabyte-sized or even terabyte-sized virtual memory, according to "LZMA_RA.ultra" and "LZMA_RA.ultra_max" in compression.gdsn. These features require gdsfmt (>=v1.16.0). "Ultra" and "UltraMax" may not increase the compression ratio much compared with "LZMA_RA.max", and these options are designed for the users who want to exhaust the computational resources.

Value

Return a list with a class name "SeqGDSStorageClass", contains the compression algorithm for each data type.
seqSummary

Author(s)
Xiuwen Zheng

See Also
seqVCF2GDS, seqRecompress, seqMerge

Examples

# the file of VCF
(vcf.fn <- seqExampleFileName("vcf"))

# convert
seqVCF2GDS(vcf.fn, "tmp1.gds", storage.option=seqStorageOption())
(f1 <- seqOpen("tmp1.gds"))

# convert (maximize the compression ratio)
seqVCF2GDS(vcf.fn, "tmp2.gds", storage.option=seqStorageOption("ZIP_RA.max"))
(f2 <- seqOpen("tmp2.gds"))

# does not compress the genotypic data
seqVCF2GDS(vcf.fn, "tmp3.gds", storage.option=
    seqStorageOption("ZIP_RA", geno.compress=""))
(f3 <- seqOpen("tmp3.gds"))

# compress with LZ4
seqVCF2GDS(vcf.fn, "tmp4.gds", storage.option=seqStorageOption("LZ4_RA"))
(f4 <- seqOpen("tmp4.gds"))

# close and remove the files
seqClose(f1)
seqClose(f2)
seqClose(f3)
seqClose(f4)

unlink(c("tmp1.gds", "tmp2.gds", "tmp3.gds", "tmp4.gds"))

seqSummary

Summarize a SeqArray GDS File

Description

Gets the summary of SeqArray GDS file.

Usage

seqSummary(gdsfile, varname=NULL, check=c("default", "none", "full"),
    verbose=TRUE)
Arguments

gdsfile a \texttt{SeqVarGDSClass} object, or a file name
varname if \texttt{NULL}, check the whole GDS file; or a character specifying variable name, and return a description of that variable. See details
check should be one of "default", "none", "full"; check="default" by default
verbose if \texttt{TRUE}, display information

Details

If check="default", the function performs regular checking, like variable dimensions. If check="full", it performs more checking, e.g., unique sample id, unique variant id, whether genotypic data are in a valid range or not.

Value

If varname=NULL, the function returns a list:

- filename the file name
- version the version of SeqArray format
- reference genome reference, a character vector (0-length for undefined)
- ploidy the number of sets of chromosomes
- num.sample the total number of samples
- num.variant the total number of variants
- allele allele information, see seqSummary(gdsfile, "allele")
- annot.qual the total number of "annotation/qual" if check="none", or a summary object including min, max, median, mean
- filter filter information, see seqSummary(gdsfile, "annotation/filter")
- info a \texttt{data.frame} of INFO field: ID, Number, Type, Description, Source and Version
- format a \texttt{data.frame} of FORMAT field: ID, Number, Type and Description
- sample.annot a \texttt{data.frame} of sample annotation with ID, Type and Description

— seqSummary(gdsfile, "genotype", check="none", verbose=FALSE) returns a list with components:

- dim an integer vector: ploidy, # of samples, # of variants
- seldim an integer vector: ploidy, # of selected samples, # of selected variants

— seqSummary(gdsfile, "allele") returns a \texttt{data.frame} with ID and descriptions (check="none"), or a list with components:

- value a \texttt{data.frame} with ID and Description
- table cross tabulation for the number of alleles per site
— seqSummary(gdsfile, "$alt") returns a data.frame with ID and Description for describing the alternative alleles.

— seqSummary(gdsfile, "annotation/filter") or seqSummary(gdsfile, "$filter") returns a data.frame with ID and description (check="none"), or a list with components: value (a data.frame with ID and Description), table (cross tabulation for the variable 'filter').

— seqSummary(gdsfile, "annotation/info") or seqSummary(gdsfile, "$info") returns a data.frame describing the variables in the folder "annotation/info" with ID, Number, Type, Description, Source and Version.

— seqSummary(gdsfile, "annotation/format") returns a data.frame describing the variables in the folder "annotation/format" with ID, Number, Type and Description.

— seqSummary(gdsfile, "sample.annotation") returns a data.frame describing sample annotation with ID, Type and Description.

— seqSummary(gdsfile, "$reference") returns the genome reference if it is defined (a 0-length character vector if undefined).

— seqSummary(gdsfile, "$contig") returns the contig information, a data.frame including ID.

— seqSummary(gdsfile, "$format") returns a data.frame describing VCF FORMAT header with ID, Number, Type and Description. The first row is used for genotypes.

— seqSummary(gdsfile, "$digest") returns a data.frame with the full names of GDS variables, digest codes and validation (FALSE/TRUE).

Author(s)
Xiuwen Zheng

See Also
seqGetData, seqApply

Examples

# the GDS file
(gds.fn <- seqExampleFileName("gds"))

seqSummary(gds.fn)

ans <- seqSummary(gds.fn, check="full")
ans

seqSummary(gds.fn, "genotype")
seqSummary(gds.fn, "allele")
seqSummary(gds.fn, "annotation/filter")
seqSummary(gds.fn, "annotation/info")
seqSummary(gds.fn, "annotation/format")
seqSummary(gds.fn, "sample.annotation")

seqSummary(gds.fn, "$reference")
seqSummary(gds.fn, "$contig")
# seqSystem

Get the parameters in the GDS system

**Description**

Get a list of parameters in the GDS system

**Usage**

seqSystem()

**Value**

A list including

- `num.logical.core`  
  the number of logical cores
- `compiler.flag`  
  SIMD instructions supported by the compiler
- `options`  
  list all options associated with SeqArray GDS format or packages

**Author(s)**

Xiuwen Zheng
**seqTranspose**

**Examples**

seqSystem()

---

**seqTranspose**

*Transpose Data Array*

**Description**

Transpose data array or matrix for possibly higher-speed access.

**Usage**

seqTranspose(gdsfile, var.name, compress=NULL, digest=TRUE, verbose=TRUE)

**Arguments**

- **gdsfile**
  - a `SeqVarGDSClass` object
- **var.name**
  - the variable name with ‘/’ as a separator
- **compress**
  - the compression option used in `add.gdsn`; or determine automatically if `NULL`
- **digest**
  - a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if TRUE or a digest algorithm is specified
- **verbose**
  - if TRUE, show information

**Details**

It is designed for possibly higher-speed access. More details will be provided in the future version.

**Value**

None.

**Author(s)**

Xiuwen Zheng

**See Also**

`seqGetData`, `seqApply`
seqUnitApply

Apply Function Over Variant Units

Description

Applies a user-defined function to each variant unit.

Usage

seqUnitApply(gdsfile, units, var.name, FUN, as.is=c("none", "list", "unlist"),
parallel=FALSE, ..., .bl_size=256L, .progress=FALSE, .useraw=FALSE,
.padNA=TRUE, .tolist=FALSE, .envir=NULL)

Arguments

gdsfile a SeqVarGDSClass object
units a list of units of selected variants, with S3 class SeqUnitListClass
var.name the variable name(s), see details
FUN the function to be applied
as.is returned value: a list, an integer vector, etc; return nothing by default as.is="none";
as.is can be a connection object, or a GDS node gdsn.class object; if "unlist" is used, produces a vector which contains all the atomic components, via unlist(..., recursive=FALSE)
**seqUnitApply**

<table>
<thead>
<tr>
<th>parallel</th>
<th>FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details.</th>
</tr>
</thead>
<tbody>
<tr>
<td>.bl_size</td>
<td>chuck size, the increment for load balancing, 256 for units</td>
</tr>
<tr>
<td>.progress</td>
<td>if TRUE, show progress information</td>
</tr>
<tr>
<td>.useraw</td>
<td>TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW instead of INTEGER if possible; for genotypes, 0xFF is missing value if RAW is used</td>
</tr>
<tr>
<td>.padNA</td>
<td>TRUE, pad a variable-length vector with NA if the number of data points for each variant is not greater than 1</td>
</tr>
<tr>
<td>.tolist</td>
<td>if TRUE, return a list of vectors instead of the structure list(length, data) for variable-length data</td>
</tr>
<tr>
<td>.envir</td>
<td>NULL, an environment object, or a list/data.frame</td>
</tr>
<tr>
<td>...</td>
<td>optional arguments to FUN</td>
</tr>
</tbody>
</table>

**Details**

The variable name should be "sample.id", "variant.id", "position", "chromosome", "allele", "genotype", "annotation/id", "annotation/qual", "annotation/filter", "annotation/info/VARIABLE_NAME", or "annotation/format/VARIABLE_NAME".

"@genotype", "annotation/info/@VARIABLE_NAME" or "annotation/format/@VARIABLE_NAME" are used to obtain the index associated with these variables.

"$dosage" is also allowed for the dosages of reference allele (integer: 0, 1, 2 and NA for diploid genotypes).

"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles.

"$num_allele" returns an integer vector with the numbers of distinct alleles.

"$ref" returns a character vector of reference alleles

"$alt" returns a character vector of alternative alleles (delimited by comma)

"$chrom_pos" returns characters with the combination of chromosome and position, e.g., "1:1272721".

"$chrom_pos_allele" returns characters with the combination of chromosome, position and alleles, e.g., "1:1272721_A_G" (i.e., chr:position_REF_ALT).

"$variant_index" returns the indices of selected variants starting from 1, and "$sample_index" returns the indices of selected samples starting from 1.

**Value**

A vector, a list of values or none.

**Author(s)**

Xiuwen Zheng
See Also

seqUnitSlidingWindows, seqUnitFilterCond

Examples

# open the GDS file
gdsfile <- seqOpen(seqExampleFileName("gds"))

# variant units via sliding windows
units <- seqUnitSlidingWindows(gdsfile)

v1 <- seqUnitApply(gdsfile, units, "genotype", function(x) dim(x)[3L],
                   as.is="unlist", .progress=TRUE)
v2 <- seqUnitApply(gdsfile, units, "genotype", function(x) dim(x)[3L],
                   as.is="unlist", parallel=2, .progress=TRUE)

all(v1 == lengths(units$index))
all(v1 == v2)

# call with an external R variable
ext <- list(x=1:1348/10)
v3 <- seqUnitApply(gdsfile, units, ":x", function(x) x,
                   as.is="list", .progress=TRUE, .envir=ext)
head(units$index)
head(v3)

table(sapply(seq_along(units$index), function(i) all(units$index[[i]] == v3[[i]]*10)))
# all TRUE

# close the GDS file
seqClose(gdsfile)

seqUnitCreate Subset and merge the units

Description

Subset and merge the variant unit(s).

Usage

seqUnitCreate(idx, desp=NULL)
seqUnitSubset(units, i)
seqUnitMerge(ut1, ut2)
seqUnitFilterCond

Arguments

- `idx`: a list of numeric indexing vectors for specifying variants
- `desp`: a data.frame for annotating the variant sets
- `units`: a list of units of selected variants, with S3 class SeqUnitListClass
- `ut1`: a list of units of selected variants, with S3 class SeqUnitListClass
- `ut2`: a list of units of selected variants, with S3 class SeqUnitListClass
- `i`: a numeric or logical vector for indices specifying elements

Value

The variant unit of SeqUnitListClass.

Author(s)

Xiuwen Zheng

See Also

seqUnitSlidingWindows, seqUnitFilterCond

Examples

```r
# open the GDS file
gdsfile <- seqOpen(seqExampleFileName("gds"))

# variant units via sliding windows
units <- seqUnitSlidingWindows(gdsfile)
(u1 <- seqUnitSubset(units, 1:10))
(u2 <- seqUnitSubset(units, 30:39))
seqUnitMerge(u1, u2)
seqUnitCreate(list(1:10, 20:30), data.frame(gene=c("g1", "g2")))

# close the GDS file
seqClose(gdsfile)
```

Description

Filters out the unit variants according to MAF, MAC and missing rates.
Usage

seqUnitFilterCond(gdsfile, units, maf=NaN, mac=1L, missing.rate=NaN,
                 msize=1L, parallel=seqGetParallel(), verbose=TRUE)

Arguments

gdsfile        a SeqVarGDSClass object
units          a list of units of selected variants, with S3 class SeqUnitListClass
maf             minimum minor reference allele frequency, or a range of MAF maf[1] <= ...
                 < maf[2]
mac             minimum minor reference allele count, or a range of MAC mac[1] <= ... <
                 mac[2]
missing.rate   maximum missing genotype rate
minsize        the minimum of unit size
parallel       FALSE (serial processing), TRUE (multicore processing), numeric value or other
                 value; parallel is passed to the argument cl in seqParallel, see seqParallel
                 for more details.
verbose        if TRUE, show information

Value

A S3 object with the class name "SeqUnitListClass" and two components (desp and index): the
first is a data.frame with columns "chr", "start" and "end", and the second is list of integer vectors
(the variant indices).

Author(s)

Xiuwen Zheng

See Also

seqUnitApply, seqUnitCreate, seqUnitSubset, seqUnitMerge

Examples

# open the GDS file
gdsfile <- seqOpen(seqExampleFileName("gds"))

unit1 <- seqUnitSlidingWindows(gdsfile)
unit1 # "desp" "index"

# only rare variants
newunit <- seqUnitFilterCond(gdsfile, unit1, maf=c(0, 0.01))
newunit

# excluded variants
exvar <- setdiff(unique(unlist(unit1$index)), unique(unlist(newunit$index)))
seqSetFilter(gdsfile, variant.sel = exvar)
maf <- seqAlleleFreq(gdsfile, minor = TRUE)
table(maf > 0)
summary(maf[maf > 0])  # > 0.01

# close the GDS file
seqClose(gdsfile)

---

### seqUnitSlidingWindows

**Sliding units of selected variants**

**Description**

Generates units of selected variants via sliding windows.

**Usage**

```r
seqUnitSlidingWindows(gdsfile, win.size=5000L, win.shift=2500L, win.start=0L,
                      dup.rm=TRUE, verbose=TRUE)
```

**Arguments**

- `gdsfile`: a `SeqVarGDSClass` object
- `win.size`: window size in basepair
- `win.shift`: the shift of sliding window in basepair
- `win.start`: the start position in basepair
- `dup.rm`: if `TRUE`, remove duplicate and zero-length windows
- `verbose`: if `TRUE`, display information

**Value**

A S3 object with the class name "SeqUnitListClass" and two components (desp and index): the first is a data.frame with columns "chr", "start" and "end", and the second is list of integer vectors (the variant indices).

**Author(s)**

Xiuwen Zheng

**See Also**

`seqUnitApply, seqUnitFilterCond`
seqVarioGDSClass

Examples

```r
# open the GDS file
gdsfile <- seqOpen(seqExampleFileName("gds"))

v <- seqUnitSlidingWindows(gdsfile)
v  # "desp" "index"

# close the GDS file
seqClose(gdsfile)
```

Description

A seqVarioGDSClass object provides access to a GDS file containing Variant Call Format (VCF) data. It extends `gds.class`.

Details

A SeqArray GDS file is created from a VCF file with `seqVCF2GDS`. This file can be opened with `seqOpen` to create a seqVarioGDSClass object.

Accessors

In the following code snippets `x` is a seqVarioGDSClass object.

- `granges(x)` Returns the chromosome and position of variants as a GRanges object. Names correspond to the variant.id.
- `ref(x)` Returns the reference alleles as a DNAStringSet.
- `alt(x)` Returns the alternate alleles as a DNAStringSetList.
- `qual(x)` Returns the quality scores.
- `filt(x)` Returns the filter data.
- `fixed(x)` Returns the fixed fields (ref, alt, qual, filt).
- `header(x)` Returns the header as a DataFrameList.
- `rowRanges(x)` Returns a GRanges object with metadata.
- `colData(x)` Returns a DataFrame with sample identifiers and any information in the 'sample.annotation' node.
- `info(x, info=NULL)` Returns the info fields as a DataFrame. info is a character vector with the names of fields to return (default is to return all).
- `geno(x, geno=NULL)` Returns the geno (format) fields as a SimpleList. geno is a character vector with the names of fields to return (default is to return all).

Other data can be accessed with `seqGetData`.

Coercion methods

In the following code snippets `x` is a `SeqVarGDSClass` object.

```r
seqAsVCF(x, chr.prefix="", info=NULL, geno=NULL)
```

Author(s)
Stephanie Gogarten, Xiuwen Zheng

See Also
- `gds.class`, `seqOpen`

Examples

gds <- seqOpen(seqExampleFileName("gds"))
gds

## sample ID
head(seqGetData(gds, "sample.id"))

## variants
granges(gds)

## Not run:
## alleles as comma-separated character strings
head(seqGetData(gds, "allele"))

## alleles as DNAStringSet or DNAStringSetList
ref(gds)
v <- alt(gds)

## genotype
geno <- seqGetData(gds, "genotype")
dim(geno)

## dimensions are: allele, sample, variant
geno[1,1:10,1:5]

## rsID
head(seqGetData(gds, "annotation/id"))

## alternate allele count
head(seqGetData(gds, "annotation/info/AC"))

## individual read depth
depth <- seqGetData(gds, "annotation/format/DP")
names(depth)

## VCF header defined DP as variable-length data
table(depth$length)

## all length 1, so depth$data should be a sample by variant matrix
dim(depth$data)

depth$data[1:10,1:5]
Reformat Variant Call Format (VCF) files.

Usage

seqVCF2GDS(vcf.fn, out.fn, header=NULL, storage.option="LZMA_RA",
info.import=NULL, fmt.import=NULL, genotype.var.name="GT",
ignore.chr.prefix="chr", scenario=c("general", "imputation"),
reference=NULL, start=1L, count=-1L, optimize=TRUE, raise.error=TRUE,
digest=TRUE, parallel=FALSE, verbose=TRUE)

seqBCF2GDS(bcf.fn, out.fn, header=NULL, storage.option="LZMA_RA",
info.import=NULL, fmt.import=NULL, genotype.var.name="GT",
ignore.chr.prefix="chr", scenario=c("general", "imputation"),
reference=NULL, optimize=TRUE, raise.error=TRUE, digest=TRUE,
bcftools="bcftools", verbose=TRUE)

Arguments

vcf.fn the file name(s) of VCF format; or a connection object
bcf.fn a file name of binary VCF format (BCF)
out.fn the file name of output GDS file
header if NULL, header is set to be seqVCF_Header(vcf.fn)
storage.option specify the storage and compression option, "ZIP_RA" (seqStorageOption("ZIP_RA"));
or "LZMA_RA" to use LZMA compression algorithm with higher compression ratio by default; or "LZ4_RA" to use an extremely fast compression and decompression algorithm. "ZIP_RA.max", "LZMA_RA.max" and "LZ4_RA.max" correspond to the algorithms with a maximum compression level; the suffix "_RA" indicates that fine-level random access is available; see more details at seqStorageOption
info.import characters, the variable name(s) in the INFO field for import; or NULL for all variables
fmt.import characters, the variable name(s) in the FORMAT field for import; or NULL for all variables
genotype.var.name the ID for genotypic data in the FORMAT column; "GT" by default (in VCF v4)
seqVCF2GDS

ignore.chr.prefix

a vector of character, indicating the prefix of chromosome which should be ignored, like "chr"; it is not case-sensitive

scenario

"general": use float32 to store floating-point numbers (by default); "imputation": use packedreal16 to store DS and GP in the FORMAT field with four decimal place accuracy

reference

genome reference, like "hg19", "GRCh37"; if the genome reference is not available in VCF files, users could specify the reference here

start

the starting variant if importing part of VCF files

count

the maximum count of variant if importing part of VCF files, -1 indicates importing to the end

optimize

if TRUE, optimize the access efficiency by calling cleanup.gds

raise.error

TRUE: throw an error if numeric conversion fails; FALSE: get missing value if numeric conversion fails

digest

a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if TRUE or a digest algorithm is specified

parallel

FALSE (serial processing), TRUE (parallel processing), a numeric value indicating the number of cores, or a cluster object for parallel processing; parallel is passed to the argument cl in seqParallel, see seqParallel for more details

verbose

if TRUE, show information

bcftools

the path of the program bcftools

Details

If there are more than one files in vcf.fn, seqVCF2GDS will merge all VCF files together if they contain the same samples. It is useful to merge multiple VCF files if variant data are split by chromosomes.

The real numbers in the VCF file(s) are stored in 32-bit floating-point format by default. Users can set storage.option=seqStorageOption(float.mode="float64") to switch to 64-bit floating point format. Or packed real numbers can be adopted by setting storage.option=seqStorageOption(float.mode="packedreal16:scale=0.0001")

By default, the compression method is "LZMA_RA" (https://tukaani.org/xz/, LZMA algorithm with default compression level + independent data blocks for fine-level random access). Users can maximize the compression ratio by storage.option="LZMA_RA.max" or storage.option=seqStorageOption("LZMA_RA.max")

LZMA is known to have higher compression ratio than the zlib algorithm. LZ4 (https://github.com/lz4/lz4) is an option via storage.option="LZ4_RA" or storage.option=seqStorageOption("LZ4_RA")

If multiple cores/processes are specified in parallel, all VCF files are scanned to calculate the total number of variants before format conversion, and then split by the number of cores/processes.

storage.option="Ultra" and storage.option="UltraMax" need much larger memory than other compression methods. Users may consider using seqRecompress to recompress the GDS file after calling seqVCF2GDS() with storage.option="ZIP_RA", since seqRecompress() compresses data nodes one by one, taking much less memory than "Ultra" and "UltraMax".

If storage.option="LZMA_RA" runs out of memory (e.g., there are too many annotation fields in the VCF file), users could use storage.option="ZIP_RA" and then call seqRecompress(), compress="LZMA").
seqVCF2GDS

Value
Return the file name of GDS format with an absolute path.

Author(s)
Xiuwen Zheng

References

See Also
seqVCF_Header, seqStorageOption, seqMerge, seqGDS2VCF, seqRecompress

Examples
# the VCF file
vcf.fn <- seqExampleFileName("vcf")

# conversion
seqVCF2GDS(vcf.fn, "tmp.gds", storage.option="ZIP_RA")

# conversion in parallel
seqVCF2GDS(vcf.fn, "tmp_p2.gds", storage.option="ZIP_RA", parallel=2L)

# display
(f <- seqOpen("tmp.gds"))
seqClose(f)

# convert without the INFO fields
seqVCF2GDS(vcf.fn, "tmp.gds", storage.option="ZIP_RA", info.import=character(0))

# display
(f <- seqOpen("tmp.gds"))
seqClose(f)

# convert without the INFO and FORMAT fields
seqVCF2GDS(vcf.fn, "tmp.gds", storage.option="ZIP_RA", info.import=character(0), fmt.import=character(0))

# display
(f <- seqOpen("tmp.gds"))
seqClose(f)
# delete the temporary file
unlink(c("tmp.gds", "tmp_p2.gds"), force=TRUE)

**seqVCF_Header**

*Parse the Header of a VCF/BCF File*

**Description**

Parses the meta-information lines of a VCF or BCF file.

**Usage**

```r
seqVCF_Header(vcf.fn, getnum=FALSE)
```

**Arguments**

- `vcf.fn`: the file name of VCF or BCF format; or a connection object for VCF format
- `getnum`: if TRUE, return the total number of variants

**Details**

The ID description contains four columns: ID – variable name; Number – the number of elements, see the webpage of the 1000 Genomes Project; Type – data type; Description – a variable description.

**Value**

Return a list (with a class name "SeqVCFHeaderClass", S3 object):

- `fileformat`: the file format
- `info`: the ID description in the INFO field
- `filter`: the ID description in the FILTER field
- `format`: the ID description in the FORMAT field
- `alt`: the ID description in the ALT field
- `contig`: the description in the contig field
- `assembly`: the link of assembly
- `reference`: genome reference, or NULL if unknown
- `header`: the other header lines
- `ploidy`: ploidy, two for humans
- `num.sample`: the number of samples
- `num.variant`: the number of variants, applicable only if getnum=TRUE
- `sample.id`: a vector of sample IDs in the VCF/BCF file
Author(s)

Xiuwen Zheng

References


See Also

seqVCF_SampID, seqVCF2GDS

Examples

```r
# the VCF file
(vcf.fn <- seqExampleFileName("vcf"))
# or vcf.fn <- "C:/YourFolder/Your_VCF_File.vcf"

# get sample id
seqVCF_Header(vcf.fn, getnum=TRUE)

# use a connection object
f <- file(vcf.fn, "r")
seqVCF_Header(f, getnum=TRUE)
close(f)
```

seqVCF_SampID    Get the Sample IDs

Description

Returns the sample IDs of a VCF file.

Usage

seqVCF_SampID(vcf.fn)

Arguments

vcf.fn    the file name, output a file of VCF format; or a connection object

Author(s)

Xiuwen Zheng
seqVCF_SampID

References

See Also
seqVCF_Header, seqVCF2GDS

Examples

# the VCF file
(vcf.fn <- seqExampleFileName("vcf"))

# get sample id
seqVCF_SampID(vcf.fn)
Index

* VCF
  seqExport, 24
  seqGDS2VCF, 27
  seqVCF2GDS, 70
  seqVCF_Header, 73
  seqVCF_SampID, 74

* gds
  KG_P1_SampData, 6
  seqAddValue, 6
  seqAlleleFreq, 8
  seqApply, 10
  SeqArray-package, 3
  seqAsVCF, 13
  seqBED2GDS, 14
  seqBlockApply, 16
  seqCheck, 18
  seqClose-methods, 19
  seqDelete, 20
  seqDigest, 21
  seqEmptyFile, 22
  seqExampleFileName, 23
  seqExport, 24
  seqGDS2SNP, 25
  seqGDS2VCF, 27
  seqGet2bGeno, 29
  seqGetData, 30
  seqGetFilter, 33
  seqMerge, 34
  seqMissing, 36
  seqNewVarData, 38
  seqNumAllele, 39
  seqOpen, 40
  seqOptimize, 41
  seqParallel, 42
  seqParallelSetup, 45
  seqRecompress, 46
  seqResetVariantID, 48
  seqSetFilter-methods, 49
  seqSetFilterCond, 53
  seqSNP2GDS, 54
  seqStorageOption, 56
  seqSummary, 57
  seqSystem, 60
  seqTranspose, 61
  seqUnitApply, 62
  seqUnitCreate, 64
  seqUnitFilterCond, 65
  seqUnitSlidingWindows, 67
  seqVCF2GDS, 70
  seqVCF_Header, 73
  seqVCF_SampID, 74

* genetics
  KG_P1_SampData, 6
  seqAddValue, 6
  seqAlleleFreq, 8
  seqApply, 10
  SeqArray-package, 3
  seqAsVCF, 13
  seqBED2GDS, 14
  seqBlockApply, 16
  seqCheck, 18
  seqClose-methods, 19
  seqDelete, 20
  seqDigest, 21
  seqEmptyFile, 22
  seqExampleFileName, 23
  seqExport, 24
  seqGDS2SNP, 25
  seqGDS2VCF, 27
  seqGet2bGeno, 29
  seqGetData, 30
  seqGetFilter, 33
  seqMerge, 34
  seqMissing, 36
  seqNewVarData, 38
  seqNumAllele, 39
  seqOpen, 40
  seqOptimize, 41
  seqParallel, 42
<table>
<thead>
<tr>
<th>seqParallelSetup</th>
<th>seqRecompress</th>
<th>seqResetVariantID</th>
<th>seqSetFilter-methods</th>
<th>seqSetFilterCond</th>
<th>seqSNP2GDS</th>
<th>seqStorageOption</th>
<th>seqSummary</th>
<th>seqSystem</th>
<th>seqTranspose</th>
<th>seqUnitApply</th>
<th>seqUnitCreate</th>
<th>seqUnitFilterCond</th>
<th>seqUnitSlidingWindows</th>
<th>seqVCF2GDS</th>
<th>seqVCF_Header</th>
<th>seqVCF_SampID</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>46</td>
<td>48</td>
<td>49</td>
<td>53</td>
<td>54</td>
<td>56</td>
<td>57</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>64</td>
<td>65</td>
<td>67</td>
<td>70</td>
<td>73</td>
<td>74</td>
</tr>
</tbody>
</table>

*sequencing*

- KG_P1_SampData, 6
- seqAddValue, 6
- seqAlleleFreq, 8
- seqApply, 10
- SeqArray-package, 3
- seqAsVCF, 13
- seqBED2GDS, 14
- seqBlockApply, 16
- seqCheck, 18
- seqClose-methods, 19
- seqDelete, 20
- seqDigest, 21
- seqEmptyFile, 22
- seqExampleFileName, 23
- seqExport, 24
- seqGDS2SNP, 25
- seqGDS2VCF, 27
- seqGet2bGeno, 29
- seqGetData, 30
- seqGetFilter, 33
- seqMerge, 34
- seqMissing, 36
- seqNewVarData, 38
- seqNumAllele, 39
- seqOpen, 40
- seqOptimize, 41
- seqParallel, 42
- seqParallelSetup, 45
- seqRecompress, 46
- seqResetVariantID, 48
- seqSetFilter-methods, 49
- seqSetFilterCond, 53
- seqSNP2GDS, 54
- seqStorageOption, 56
- seqSummary, 57
- seqSystem, 60
- seqTranspose, 61
- seqUnitApply, 62
- seqUnitCreate, 64
- seqUnitFilterCond, 65
- seqUnitSlidingWindows, 67
- seqVCF2GDS, 70
- seqVCF_Header, 73
- seqVCF_SampID, 74

add.gdsn, 6, 26, 55, 56, 61
alt (SeqVarGDSClass), 68
alt, SeqVarGDSClass-method
(SeqVarGDSClass), 68

bgzip, 27

cleanup.gds, 15, 24, 26, 35, 47, 48, 55, 71
clusterApply, 43
clusterApplyLB, 43
colData (SeqVarGDSClass), 68
colData, SeqVarGDSClass-method
(SeqVarGDSClass), 68
CollapsedVCF, 14
compression.gdsn, 56
connection, 10, 16, 27, 62, 70, 73, 74

DataFrameList, 68
DNAStringSet, 68
DNAStringSetList, 68

filt (SeqVarGDSClass), 68
filt, SeqVarGDSClass-method
(SeqVarGDSClass), 68
fixed (SeqVarGDSClass), 68
fixed, SeqVarGDSClass-method
(SeqVarGDSClass), 68

gds.class, 20, 40, 68, 69
gdsn.class, 10, 16, 62
geno (SeqVarGDSClass), 68
geno, SeqVarGDSClass, ANY-method
(SeqVarGDSClass), 68
geno, SeqVarGDSClass-method
(SeqVarGDSClass), 68
INDEX

GRanges, 50
granges, SeqVarGDSClass-method (SeqVarGDSClass), 68
GRangesList, 50
header (SeqVarGDSClass), 68
header, SeqVarGDSClass-method (SeqVarGDSClass), 68
info (SeqVarGDSClass), 68
info, SeqVarGDSClass-method (SeqVarGDSClass), 68
IRanges, 50
KG_P1_SampData, 6
makeCluster, 42, 43, 45, 46
nextRNGStream, 43
nextRNGSubStream, 43
openfn.gds, 40
parallel, 42, 45
qual (SeqVarGDSClass), 68
qual, SeqVarGDSClass-method (SeqVarGDSClass), 68
ref (SeqVarGDSClass), 68
ref, SeqVarGDSClass-method (SeqVarGDSClass), 68
rowRanges (SeqVarGDSClass), 68
rowRanges, SeqVarGDSClass-method (SeqVarGDSClass), 68
seqAddValue, 6, 38
seqAlleleCount, 54
seqAlleleCount (seqAlleleFreq), 8
seqAlleleFreq, 8, 37, 39, 54
seqApply, 10, 18, 22, 31, 40, 41, 43, 46, 51, 59, 61
SeqArray (SeqArray-package), 3
SeqArray-package, 3
seqAsVCF, 13
seqBCF2GDS (seqVCF2GDS), 70
seqBED2GDS, 14, 55
seqBlockApply, 11, 16
seqCheck, 18
seqClose, 21, 40
seqClose (seqClose-methods), 19
seqClose, gds.class-method (seqClose-methods), 19
seqClose, SeqVarGDSClass-method (seqClose-methods), 19
seqClose-methods, 19
seqDelete, 20
seqDigest, 21
seqExampleFileName, 23
seqExport, 24, 35
seqFilterPop (seqSetFilter-methods), 49
seqFilterPush (seqSetFilter-methods), 49
seqGDS2BED (seqBED2GDS), 14
seqGDS2SNP, 25, 55
seqGDS2VCF, 26, 27, 55, 72
seqGet2bGeno, 29
seqGetAF_AC_Missing (seqAlleleFreq), 8
seqGetData, 11, 18, 29, 30, 38, 40, 41, 43, 51, 59, 61, 68
seqGetFilter, 33, 51
seqGetParallel, 9, 11, 18, 37, 43
seqGetParallel (seqParallelSetup), 45
seqListVarData, 31
seqNewVarData (seqNewVarSetup), 38
seqOpen, 20, 21, 40, 68, 69
seqOptimize, 41
seqParallel, 8–11, 15, 17, 18, 37, 42, 46, 54, 63, 66, 71
seqParallelSetup, 43, 45
seqParApply (seqParallel), 42
seqRecompress, 46, 57, 71, 72
seqResetFilter (seqSetFilter-methods), 49
seqResetVariantID, 48
seqSetFilter, 10, 11, 14, 17, 18, 22, 26, 27, 31, 33, 43, 54
seqSetFilter (seqSetFilter-methods), 49
seqSetFilter, SeqVarGDSClass, ANY-method (seqSetFilter-methods), 49
seqSetFilter, SeqVarGDSClass, GRanges-method (seqSetFilter-methods), 49
seqSetFilter, SeqVarGDSClass, GRangesList-method
INDEX

(seqSetFilter-methods), 49
seqSetFilter,SeqVarGDSClass,IRanges-method
(seqSetFilter-methods), 49
seqSetFilter-methods, 49
seqSetFilterAnnotID
(seqSetFilter-methods), 49
seqSetFilterChrom, 54
seqSetFilterChrom
(seqSetFilter-methods), 49
seqSetFilterCond, 51, 53
seqSetFilterPos (seqSetFilter-methods),
49
seqSNP2GDS, 15, 26, 54
seqStorageOption, 34, 47, 56, 70, 72
seqSummary, 57
seqSystem, 60
seqTranspose, 61
seqUnitApply, 62, 66, 67
seqUnitCreate, 64, 66
seqUnitFilterCond, 64, 65, 65, 67
seqUnitMerge, 66
seqUnitMerge (seqUnitCreate), 64
seqUnitSlidingWindows, 64, 65, 67
seqUnitSubset, 66
seqUnitSubset (seqUnitCreate), 64
SeqVarGDSClass, 6, 8, 10, 13, 15, 16, 19–21,
24, 26, 27, 29, 30, 33, 36, 39, 42, 49,
53, 58, 61, 62, 66, 67, 68
SeqVarGDSClass-class (SeqVarGDSClass),
68
seqVCF2GDS, 7, 15, 23, 24, 26, 28, 35, 47, 48,
55, 57, 68, 70, 74, 75
seqVCF_Header, 70, 72, 73, 75
seqVCF_SampID, 74, 74
VCF-class, 13, 14