Package ‘SeqArray’

April 15, 2017

Type Package

Title Big Data Management of Whole-genome Sequence Variant Calls

Version 1.14.1

Date 2017-01-15

Depends R (>= 3.3.0), gdsfmt (>= 1.10.1)

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

LinkingTo gdsfmt

Suggests BiocParallel, digest, crayon, RUnit, Biobase, BiocGenerics, knitr, Rcpp, SNPRelate

Description Big data management of whole-genome sequencing variant calls with thousands of individuals: genotypic data (e.g., SNVs, indels and structural variation calls) and annotations in SeqArray 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

URL `http://github.com/zhengxwen/SeqArray`

BugReports `http://github.com/zhengxwen/SeqArray/issues`

biocViews Infrastructure, DataRepresentation, Sequencing, Genetics

NeedsCompilation yes

Author Xiuwen Zheng [aut, cre],
Stephanie Gogarten [aut],
David Levine [ctb],
Cathy Laurie [ctb]

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

R topics documented:

`SeqArray-package` *Big Data Management of Whole-Genome Sequence Variant Calls*

Description

Big-data management of 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.

Webpage: http://github.com/zhengxwen/SeqArray, http://www.bioconductor.org/packages/SeqArray/

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")
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
# sample [,1] [,2] [,3] [,4] [,5] [,6] ...
#  [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
seqApply(f, "allele",
    FUN=function(x) length(unlist(strsplit(x,","))), as.is="integer")


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

# 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==0, 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

```
KG_P1_SampData
```

Value

An AnnotatedDataFrame

`seqAlleleFreq` *Get Allele Frequencies or Counts*

Description

Calculates the allele frequencies or counts.

Usage

```
seqAlleleFreq(gdsfile, ref.allele=0L, .progress=FALSE, parallel=seqGetParallel())
seqAlleleCount(gdsfile, .progress=FALSE, parallel=seqGetParallel())
```

Arguments

`gdsfile`	a `SeqVarGDSClass` object
`ref.allele`	`NULL`, a single numeric value, a numeric vector or a character vector; see Value
`.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.

Value

If `ref.allele=NULL`, the function returns a list of allele frequencies according to all allele per site. If `ref.allele` is a single numeric value (like `0L`), it returns a numeric vector for the specified alleles (`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).

Author(s)

Xiuwen Zheng

See Also

seqNumAllele, seqMissing, seqParallel, seqGetParallel

Examples

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

# display
f <- seqOpen(gds.fn)

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

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

# return a numeric vector, AA is ancestral allele
AA <- toupper(seqGetData(f, "annotation/info/AA")$data)
summary(seqAlleleFreq(f, AA))

# allele counts
head(seqAlleleCount(f, .progress=TRUE))

# 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
`as.is`	returned value: a list, an integer vector, etc; `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"`, 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
`.progress`	if `TRUE`, show progress information
`.list_dup`	internal use only
`...`	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"`.

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

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

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", qual="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", qual="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)
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")
```

seqBED2GDS *Convert PLINK BED Format to SeqArray Format*

Description

Converts a PLINK BED file to a SeqArray GDS file.

Usage

```
seqBED2GDS(bed.fn, fam.fn, bim.fn, out.gdsfn,
    compress.geno="ZIP_RA", compress.annotation="ZIP_RA",
    optimize=TRUE, digest=TRUE, 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
`out.gdsfn`	the file name, output a file of SeqArray format
`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`
`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.

Author(s)

Xiuwen Zheng

See Also

`seqSNP2GDS`, `seqVCF2GDS`

Examples

```
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
seqBED2GDS(bed.fn, fam.fn, bim.fn, "tmp.gds")

seqSummary("tmp.gds")

# remove the temporary file
unlink("tmp.gds", 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

```
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, .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
`as.is`	returned value: a list, a vector or 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"`, 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
`.progress`	if `TRUE`, show progress information
`.list_dup`	internal use only
`...`	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"`.

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

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

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

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

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

---

seqDelete *Delete GDS Variables*

Description

Deletes variables in the SeqArray GDS file.

Usage

```
seqDelete(gdsfile, info.varname=character(), format.varname=character(),
    samp.varname=character(), verbose=TRUE)
```

Arguments

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

Value

None.

Author(s)

Xiuwen Zheng

See Also

seqOpen, seqClose

Examples

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

# convert
seqVCF2GDS(vcf.fn, "tmp.gds")

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

seqDelete(f, info.varname=c("HM2", "AA"), format.varname="DP")
f

# close the GDS file
seqClose(f)

# clean up the fragments, reduce the file size
cleanup.gds("tmp.gds")


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

`seqDigest` *Hash function digests*

Description

Create hash function digests for all or a subset of data

Usage

```
seqDigest(gdsfile, varname, algo=c("md5"))
```

Arguments

`gdsfile`	a `SeqVarGDSClass` object
`varname`	the variable name(s), see details
`algo`	the digest hash algorithm: "md5"

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

```
library(SeqArray)

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

# display
f <- seqOpen(gds.fn)

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

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

`seqExampleFileName` *Example files*

Description

The example files of VCF and GDS format.

Usage

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

Arguments

`type`	either "gds" or "vcf"

Details

The SeqArray GDS file was created from a subset of VCF data of the 1000 Genomes Phase 1 Project.

Value

Return the file name of a VCF file shipped with the package if `type = "vcf"`, or the file name of a GDS file if `type = "gds"`.

Author(s)

Xiuwen Zheng

Examples

```
seqExampleFileName("gds")

seqExampleFileName("vcf")

seqExampleFileName("KG_Phase1")
```

seqExport *Export to a GDS File*

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

```
# 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

```
seqGDS2SNP(gdsfile, out.gdsfn, compress.geno="ZIP_RA",
    compress.annotation="ZIP_RA", optimize=TRUE, verbose=TRUE)
```

Arguments

`gdsfile`	character (GDS file name), or a `SeqVarGDSClass` object
`out.gdsfn`	the file name, output a file of VCF format
`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`
`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

```
# 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 VCF file.

Usage

```
seqGDS2VCF(gdsfile, vcf.fn, info.var=NULL, fmt.var=NULL, 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
`verbose`	if `TRUE`, show information

Details

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

GDS – Genomic Data Structures used for storing genetic array-oriented data, and the file format defined in the gdsfmt package.

VCF – The Variant Call Format (VCF), which is a generic format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations.

Value

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

Author(s)

Xiuwen Zheng

References

Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., et al. (2011). The variant call format and VCFtools. Bioinformatics 27, 2156-2158.

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)

# using all samples and variants
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)
```

`seqGetData` *Get Data*

Description

Gets data from a SeqArray GDS file.

Usage

```
seqGetData(gdsfile, var.name, .useraw=FALSE)
```

Arguments

`gdsfile`	a `SeqVarGDSClass` object
`var.name`	the variable name, 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

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.

`"$chrom_pos"` returns characters with the combination of chromosome and position, e.g., "1_1272721".

`"$dosage"` returns a RAW/INTEGER matrix for the dosages of reference allele.

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

Value

Return vectors or lists.

Author(s)

Xiuwen Zheng

See Also

`seqSetFilter`, `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" ...

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

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

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


# 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
# sample [,1] [,2] [,3] [,4] [,5] [,6] ...
#  [1,]   25   25   22    3    4   17  ...

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

seqGetFilter — *Get the Filter of GDS File*

Description

Gets the filter of samples and variants.

Usage

```
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

```
# 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

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

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], start=start[i], count=count[i])

    # merge
    seqMerge(fn, "tmp.gds")
    seqSummary("tmp.gds")


    ####

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

    # modify 'sample.id'
    f <- openfn.gds("test.gds", FALSE)
    sid <- read.gdsn(index.gdsn(f, "sample.id"))
    add.gdsn(f, "sample.id", paste("S", 1:length(sid)), replace=TRUE)
    closefn.gds(f)

    # merging
    seqMerge(c(vcf.fn, "test.gds"), "output.gds")



    # 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

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

Arguments

`gdsfile`	a `SeqVarGDSClass` object
`per.variant`	missing rate per variant if `TRUE`, or missing rate per sample if `FALSE`
`.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.

Value

A vector of missing rates.

Author(s)

Xiuwen Zheng

See Also

seqAlleleFreq, seqNumAllele, seqParallel, seqGetParallel

Examples

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

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

summary(seqMissing(f, TRUE, .progress=TRUE))

summary(seqMissing(f, FALSE, .progress=TRUE))

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

---

seqNumAllele *Number of alleles*

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 `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("by.sample"), format.var=TRUE, cleanup=TRUE,
    verbose=TRUE)
```

Arguments

`gdsfn`	a `SeqVarGDSClass` object
`target`	"by.sample" – optimize GDS file for `seqApply(..., margin="by.sample")`
`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

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")

# 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, ...)
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, ...)` or `FUN(...)`
`split`	split the dataset by variant or sample according to multiple processes, or "none" for no split
`.combine`	define a fucntion 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, like "+".

`.selection.flag`	`TRUE` – passes a logical vector of selection to the second argument of `FUN(gdsfile, selection, ...`
`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=NULL, FUN=..., split="none")` might 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

```
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(, gdsfile, FUN = function(f) {
        v <- seqGetFilter(f)
        sum(v$variant.sel)
    }, split = "by.variant")
# [1] 674 674


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

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


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


# close the GDS file
seqClose(gdsfile)


seqParallelSetup(FALSE)
```

---

seqParallelSetup *Setup/Get a Parallel Environment*

Description

Setups a parallel environment in R for the current session.

Usage

```
seqParallelSetup(cluster=TRUE, verbose=TRUE)
seqGetParallel()
```

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
`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()` returns `getOption("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)
```

seqSetFilter-methods *Set a Filter to Sample or Variant*

Description

Sets a filter to sample and/or variant.

Usage

```
## 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"),
    verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,GRanges'
seqSetFilter(object, variant.sel,
    rm.txt="chr", verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,GRangesList'
seqSetFilter(object, variant.sel,
    rm.txt="chr", verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,IRanges'
seqSetFilter(object, variant.sel,
    chr, verbose=TRUE)
seqResetFilter(object, sample=TRUE, variant=TRUE, verbose=TRUE)
seqSetFilterChrom(object, include=NULL, is.num=NA, from.bp=NULL, to.bp=NULL,
    verbose=TRUE)
```

Arguments

`object`	a `SeqVarGDSClass` object
`variant.sel`	a logical/raw/index vector indicating the selected variants; `GRanges`, a GRanges object for the genomic locations; `GRangesList`, a GRangesList object for storing a collection of GRanges objects; `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 `sample.id`, `variant.id`, `samp.sel` or `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 `sample.id`, `variant.id`, `samp.sel` or `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
`rm.txt`	a character, the characters will be removed from `seqnames(variant.sel)`
`chr`	a vector of character for chromsome coding
`sample`	logical, if `TRUE`, include all samples

`variant`	logical, if `TRUE`, include all variants
`include`	NULL, or character for specified chromosome(s)
`is.num`	a logical variable: `TRUE`, chromosome code is numeric; `FALSE`, chromosome is not numeric; `is.num=TRUE` is usually used to exclude non-autosomes
`from.bp`	NULL, no limit; numeric, the lower bound of position
`to.bp`	NULL, no limit; numeric, the upper bound of position
`verbose`	if `TRUE`, show information

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

None.

Author(s)

Xiuwen Zheng

See Also

`seqSetFilterChrom`, `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 and variant filters
seqSetFilter(f, sample.id=samp.id[c(2,4,6,8)])
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")
```

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

---

seqSNP2GDS *Convert SNPRelate Format to SeqArray Format*

Description

Converts a SNP GDS file to a SeqArray GDS file.

Usage

```
seqSNP2GDS(gds.fn, out.fn, storage.option="ZIP_RA", optimize=TRUE,
    digest=TRUE, verbose=TRUE)
```

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, by default `seqStorageOption("ZIP_RA")`; or `"LZMA_RA"` to use LZMZ compression algorithm with higher compression ratio
`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.

Author(s)

Xiuwen Zheng

See Also

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

Examples

```
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` *Storage and Compression Options*

Description

Storage and compression options for GDS import and merging.

Usage

```
seqStorageOption(compression=c("ZIP_RA", "ZIP_RA.fast", "ZIP_RA.max",
    "LZ4_RA", "LZ4_RA.fast", "LZ4_RA.max", "LZMA_RA", "LZMA_RA.fast",
    "LZMA_RA.max", "none"), mode=NULL, float.mode="float32",
    geno.compress=NULL, info.compress=NULL, format.compress=NULL,
    index.compress=NULL, ...)
```

Arguments

`compression`	the default compression level ("ZIP_RA"), see add.gdsn for the description of compression methods
`mode`	specify storage type for corresponding variable, e.g., ‘annotation/info/HM‘="int16", ‘annotation/format/PL/data‘="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"

Value

Return a list with a class name "SeqGDSStorageClass".

Author(s)

Xiuwen Zheng

See Also

`seqVCF2GDS`, `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 SeqVarGDSClass object
varname	if 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"
verbose	if 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 `data.frame` of INFO field: ID, Number, Type, Description, Source and Version
`format`	a `data.frame` of FORMAT field: ID, Number, Type and Description
`sample.annot`	a `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 data.frame with ID and descriptions (`check="none"`), or a list with components:

`value`	a 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")
seqSummary(gds.fn, "$filter")
seqSummary(gds.fn, "$alt")
seqSummary(gds.fn, "$info")
seqSummary(gds.fn, "$format")
seqSummary(gds.fn, "$digest")


# open a GDS file
f <- seqOpen(gds.fn)

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

seqSummary(f, "genotype")

# close a GDS file
seqClose(f)
```

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

References

`http://github.com/zhengxwen/SeqArray`

Examples

```
seqSystem()
```

seqTranspose *Transpose Data Array*

Description

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

Usage

```
seqTranspose(gdsfile, var.name, compress=NULL, 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`
`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`

Examples

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

# convert
seqVCF2GDS(vcf.fn, "tmp.gds")

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

seqTranspose(f, "genotype/data")
f

# the original array
index.gdsn(f, "genotype/data")
# the transposed array
```

```
index.gdsn(f, "genotype/~data")

# close
seqClose(f)

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

SeqVarGDSClass *SeqVarGDSClass*

Description

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

Details

A sequence GDS file is created from a VCF file with seqVCF2GDS. This file can be opened with seqOpen to create a SeqVarGDSClass object.

Accessors

In the following code snippets x is a SeqVarGDSClass 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.

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.

asVCF(x, chr.prefix="", info=NULL, geno=NULL): 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 asVCF to specify samples and variants to return.

Author(s)

Stephanie Gogarten, Xiuwen Zheng

See Also

`gds.class`, `seqVCF2GDS`, `seqOpen`, `seqGetData`, `seqSetFilter`, `seqClose`

Examples

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

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

## variants
# granges(gds)

## 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]

seqClose(gds)
```

`seqVCF2GDS` *Reformat VCF Files*

Description

Reformats 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", reference=NULL, start=1L, count=-1L,
    optimize=TRUE, raise.error=TRUE, digest=TRUE, parallel=FALSE,
    verbose=TRUE)
```

Arguments

`vcf.fn`	the file name(s) of VCF format; or a `connection` object
`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")`) by default; or "LZMA_RA" to use LZMA compression algorithm with higher compression ratio
`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, VCFv4.0
`ignore.chr.prefix`	a vector of character, indicating the prefix of chromosome which should be ignored, like "chr"; it is not case-sensitive
`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

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="pa`

By default, the compression method is "ZIP_RA" (zlib algorithm with default compression level + independent data blocks). Users can maximize the compression ratio by `storage.option="ZIP_RA.max"` or `storage.option=seqStorageOption("ZIP_RA.max")`. LZ4 (http://cyan4973.github.io/lz4/) is an option via `storage.option="LZ4_RA"` or `storage.option=seqStorageOption("LZ4_RA")`. LZMA (xz, http://tukaani.org/xz/) is another option via `storage.option="LZMA_RA"` or `storage.option=seqStorageOption("LZMA_RA")`, and it is known to have higher compression ratio than zlib.

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

Value

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

Author(s)

Xiuwen Zheng

References

Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., et al. (2011). The variant call format and VCFtools. Bioinformatics 27, 2156-2158.

See Also

`seqVCF_Header`, `seqStorageOption`, `seqMerge`, `seqGDS2VCF`

Examples

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

# conversion
seqVCF2GDS(vcf.fn, "tmp.gds")

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


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



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

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

```
# convert without the INFO and FORMAT fields
seqVCF2GDS(vcf.fn, "tmp.gds", 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 File*

Description

Parses the header of a VCF file.

Usage

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

Arguments

`vcf.fn`	the file name; or a `connection` object
`getnum`	if `TRUE`, return the number of samples and 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

Author(s)

Xiuwen Zheng

References

Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., et al. (2011). The variant call format and VCFtools. Bioinformatics 27, 2156-2158.

See Also

seqVCF_SampID, seqVCF2GDS

Examples

```
# 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

References

Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., et al. (2011). The variant call format and VCFtools. Bioinformatics 27, 2156-2158.

See Also

seqVCF_Header, seqVCF2GDS

Examples

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

# get sample id
seqVCF_SampID(vcf.fn)
```

Index