

Package ‘SeqArray’

October 16, 2018

Type Package

Title Big Data Management of Whole-genome Sequence Variant Calls

Version 1.20.2

Date 2018-09-10

Depends R (>= 3.5.0), gdsfmt (>= 1.10.1)

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

LinkingTo gdsfmt

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

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

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

git_branch RELEASE_3_7

git_last_commit 7277e82

git_last_commit_date 2018-09-10

Date/Publication 2018-10-15

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	2
KG_P1_SampData	5
seqAlleleFreq	5
seqApply	6
seqAsVCF	9
seqBED2GDS	11
seqBlockApply	12
seqClose-methods	14
seqDelete	15
seqDigest	16
seqExampleFileName	17
seqExport	18
seqGDS2SNP	19
seqGDS2VCF	20
seqGetData	22
seqGetFilter	24
seqMerge	25
seqMissing	27
seqNumAllele	28
seqOpen	29
seqOptimize	30
seqParallel	31
seqParallelSetup	34
seqSetFilter-methods	35
seqSetFilterCond	38
seqSNP2GDS	39
seqStorageOption	41
seqSummary	42
seqSystem	45
seqTranspose	46
SeqVarGDSClass	47
seqVCF2GDS	48
seqVCF_Header	51
seqVCF_SampID	52
Index	54

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", 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
# 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	<i>Simulated sample data for 1000 Genomes Phase 1</i>
----------------	---

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	<i>Get Allele Frequencies or Counts</i>
---------------	---

Description

Calculates the allele frequencies or counts.

Usage

```

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

```

Arguments

gdsfile	a SeqVarGDSCClass 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/counts 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).

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)
head(seqAlleleFreq(f, AA))

# allele counts
head(seqAlleleCount(f, NULL, .progress=TRUE))
head(seqAlleleCount(f, 0L, .progress=TRUE))
head(seqAlleleCount(f, AA, .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

<code>gdsfile</code>	a SeqVarGDSClass object
<code>var.name</code>	the variable name(s), see details
<code>FUN</code>	the function to be applied
<code>margin</code>	giving the dimension which the function will be applied over
<code>as.is</code>	returned value: a list, an integer vector, etc; <code>as.is</code> 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 <code>unlist(..., recursive=FALSE)</code>
<code>var.index</code>	if "none", call <code>FUN(x, ...)</code> without variable index; if "relative" or "absolute", add an argument to the user-defined function <code>FUN</code> like <code>FUN(index, x, ...)</code> where <code>index</code> is an index of variant starting from 1 if <code>margin = "by.variant"</code> : "relative" for indexing in the selection defined by seqSetFilter , "absolute" for indexing with respect to all data
<code>parallel</code>	FALSE (serial processing), TRUE (multicore processing), numeric value or other value; <code>parallel</code> is passed to the argument <code>cl</code> in seqParallel , see seqParallel for more details.
<code>.useraw</code>	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
<code>.progress</code>	if TRUE, show progress information
<code>.list_dup</code>	internal use only
<code>...</code>	optional arguments to <code>FUN</code>

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).

"\$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.

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
f1 <- file("tmp.txt", "wt")
seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is=f1)
close(f1)
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)
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")
```

Description

Create a [VCF-class](#) object

Usage

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

Arguments

x	a SeqVarGDSCClass object
chr.prefix	prefix to add to seqlevels
info	which INFO fields to return
geno	which GENO fields to return

Details

Coerces a [SeqVarGDSCClass](#) 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

*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="LZMA_RA", compress.annotation="LZMA_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, ...)
```

Arguments

<code>gdsfile</code>	a SeqVarGDSClass object
<code>var.name</code>	the variable name(s), see details
<code>FUN</code>	the function to be applied
<code>margin</code>	giving the dimension which the function will be applied over
<code>as.is</code>	returned value: a list, a vector or none; <code>as.is</code> 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 <code>unlist(..., recursive=FALSE)</code>
<code>var.index</code>	if "none", call <code>FUN(x, ...)</code> without variable index; if "relative" or "absolute", add an argument to the user-defined function <code>FUN</code> like <code>FUN(index, x, ...)</code> where <code>index</code> is an index of variant starting from 1 if <code>margin = "by.variant"</code> : "relative" for indexing in the selection defined by seqSetFilter , "absolute" for indexing with respect to all data
<code>bsize</code>	block size
<code>parallel</code>	FALSE (serial processing), TRUE (multicore processing), numeric value or other value; <code>parallel</code> is passed to the argument <code>cl</code> in seqParallel , see seqParallel for more details.
<code>.useraw</code>	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
<code>.progress</code>	if TRUE, show progress information
<code>...</code>	optional arguments to <code>FUN</code>

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.

"\$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 allele, e.g., "1:1272721_A_G".

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

<code>gdsfile</code>	a SeqVarGDSClass object
<code>info.varname</code>	the variables in the INFO field, i.e., "annotation/info/VARIABLE_NAME"
<code>format.varname</code>	the variables in the FORMAT field, i.e., "annotation/format/VARIABLE_NAME"
<code>samp.varname</code>	the variables in the sample annotation field, i.e., "sample.annotation/VARIABLE_NAME"
<code>verbose</code>	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", storage.option="ZIP_RA")

# 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	<i>Hash function digests</i>
-----------	------------------------------

Description

Create hash function digests for all or a subset of data

Usage

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

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)
```

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

<code>gdsfile</code>	a SeqVarGDSClass object
<code>out.fn</code>	the file name of output GDS file
<code>info.var</code>	characters, the variable name(s) in the INFO field for import; or NULL for all variables
<code>fmt.var</code>	characters, the variable name(s) in the FORMAT field for import; or NULL for all variables
<code>samp.var</code>	characters, the variable name(s) in the folder "sample.annotation"
<code>optimize</code>	if TRUE, optimize the access efficiency by calling cleanup.gds
<code>digest</code>	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
<code>verbose</code>	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="LZMA_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 <code>add.gdsn</code>
compress.annotation	the compression method for the GDS variables, except "genotype"; optional values are defined in the function <code>add.gdsn</code>
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 Variant Call Format (VCF) file.

Usage

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

Arguments

<code>gdsfile</code>	a SeqVarGDSClass object
<code>vcf.fn</code>	the file name, output a file of VCF format; or a connection object
<code>info.var</code>	a list of variable names in the INFO field, or NULL for using all variables; character(0) for no variable in the INFO field
<code>fmt.var</code>	a list of variable names in the FORMAT field, or NULL for using all variables; character(0) for no variable in the FORMAT field
<code>use_Rsamtools</code>	TRUE for loading the Rsamtools package, see details
<code>verbose</code>	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, 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

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

<code>gdsfile</code>	a SeqVarGDSClass object
<code>var.name</code>	the variable name, see details
<code>.useraw</code>	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

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).

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, geno.pad=TRUE, verbose=TRUE)
```

Arguments

<code>gds.fn</code>	the file names of multiple GDS files
<code>out.fn</code>	the output file name
<code>storage.option</code>	specify the storage and compression option, "ZIP_RA" (<code>seqStorageOption("ZIP_RA")</code>); or "LZMA_RA" to use LZMA compression algorithm with higher compression ratio (by default)
<code>info.var</code>	characters, the variable name(s) in the INFO field; NULL for all variables, or <code>character()</code> excludes all INFO variables
<code>fmt.var</code>	characters, the variable name(s) in the FORMAT field; NULL for all variables, or <code>character()</code> excludes all FORMAT variables
<code>samp.var</code>	characters, the variable name(s) in 'sample.annotation'; or NULL for all variables
<code>optimize</code>	if TRUE, optimize the access efficiency by calling <code>cleanup.gds</code>
<code>digest</code>	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
<code>geno.pad</code>	TRUE, pad a 2-bit genotype array in bytes to avoid recompressing genotypes if possible
<code>verbose</code>	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
seqMerge(fn, "tmp.gds", storage.option="ZIP_RA")
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", 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

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

Arguments

<code>gdsfile</code>	a SeqVarGDSClass object
<code>per.variant</code>	missing rate per variant if TRUE, or missing rate per sample if FALSE
<code>.progress</code>	if TRUE, show progress information
<code>parallel</code>	FALSE (serial processing), TRUE (multicore processing), numeric value or other value; <code>parallel</code> is passed to the argument <code>cl</code> 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

<code>gdsfile</code>	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 integrity.

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("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 <code>link{cleanup.gds}</code> 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, ...)
seqParApply(cl=seqGetParallel(), x, FUN, load.balancing=TRUE, ...)
```

Arguments

<code>cl</code>	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 <code>BiocParallelParam</code> object from the <code>BiocParallel</code> package. See details
<code>gdsfile</code>	a SeqVarGDSCClass object, or NULL
<code>FUN</code>	the function to be applied, should be like <code>FUN(gdsfile, ...)</code> or <code>FUN(...)</code>
<code>split</code>	split the dataset by variant or sample according to multiple processes, or "none" for no split
<code>.combine</code>	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 <code>unlist(..., recursive=FALSE)</code> ; "list", return a list of results created by child processes; "none", no return; or a function, like "+".
<code>.selection.flag</code>	TRUE – passes a logical vector of selection to the second argument of <code>FUN(gdsfile, selection, ...)</code>
<code>x</code>	a vector (atomic or list), passed to <code>FUN</code>
<code>load.balancing</code>	if TRUE, call clusterApplyLB instead of clusterApply
<code>...</code>	optional arguments to <code>FUN</code>

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	<i>Setup/Get a Parallel Environment</i>
------------------	---

Description

Setup 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", 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, intersect=FALSE, multi.pos=FALSE,
                verbose=TRUE)

```

Arguments

object a [SeqVarGDSClass](#) object

<code>variant.sel</code>	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
<code>sample.sel</code>	a logical/raw/index vector indicating the selected samples
<code>variant.id</code>	ID of selected variants
<code>sample.id</code>	ID of selected samples
<code>action</code>	"set" – set the current filter via <code>sample.id</code> , <code>variant.id</code> , <code>samp.sel</code> or <code>variant.sel</code> ; "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 <code>sample.id</code> , <code>variant.id</code> , <code>samp.sel</code> or <code>variant.sel</code> ; "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
<code>rm.txt</code>	a character, the characters will be removed from <code>seqnames(variant.sel)</code>
<code>chr</code>	a vector of character for chromosome coding
<code>pos</code>	a vector of numeric values for genome coordinate
<code>sample</code>	logical, if TRUE, include all samples
<code>variant</code>	logical, if TRUE, include all variants
<code>include</code>	NULL, or a vector of characters for specified chromosome(s)
<code>is.num</code>	a logical variable: TRUE, chromosome code is numeric; FALSE, chromosome is not numeric; <code>is.num=TRUE</code> is usually used to exclude non-autosomes
<code>from.bp</code>	NULL, no limit; a numeric vector, the lower bound of position
<code>to.bp</code>	NULL, no limit; a numeric vector, the upper bound of position
<code>intersect</code>	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
<code>multi.pos</code>	FALSE, use the first matched position; TRUE, allow multiple variants at the same position
<code>verbose</code>	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](#), [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 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")

## 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

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

Arguments

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

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="LZMA_RA",
           major.ref=TRUE, optimize=TRUE, digest=TRUE, verbose=TRUE)
```

Arguments

<code>gds.fn</code>	the file name of SNP format
<code>out.fn</code>	the file name, output a file of SeqArray format
<code>storage.option</code>	specify the storage and compression options, "LZMA_RA" to use LZMA compression algorithm with higher compression ratio compared to "ZIP_RA"
<code>major.ref</code>	if TRUE, use the major allele as a reference allele; otherwise, use A allele in SNP GDS file as a reference allele
<code>optimize</code>	if TRUE, optimize the access efficiency by calling cleanup.gds
<code>digest</code>	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
<code>verbose</code>	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", "Ultra", "UltraMax", "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"

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` ($\geq v1.16.0$). "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 "SeqGDSSStorageClass", contains the compression algorithm for each data type.

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, or a file name
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:

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

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

<code>dim</code>	an integer vector: ploidy, # of samples, # of variants
<code>seldim</code>	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:

<code>value</code>	a <code>data.frame</code> with ID and Description
<code>table</code>	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:

<code>value</code>	a <code>data.frame</code> with ID and Description
<code>table</code>	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	<i>Get the parameters in the GDS system</i>
-----------	---

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", storage.option="ZIP_RA")

# 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 SeqArray 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 [DNAStrngSet](#).

`alt(x)`: Returns the alternate alleles as a [DNAStrngSetList](#).

`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.

```
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 DNStringSet or DNStringSetList
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]

## End(Not run)

seqClose(gds)
```


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)
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", reference=NULL, optimize=TRUE, raise.error=TRUE,
  digest=TRUE, bcftools="bcftools", verbose=TRUE)
```

Arguments

<code>vcf.fn</code>	the file name(s) of VCF format; or a connection object
<code>bcf.fn</code>	a file name of binary VCF format (BCF)
<code>out.fn</code>	the file name of output GDS file
<code>header</code>	if <code>NULL</code> , header is set to be seqVCF_Header (<code>vcf.fn</code>)
<code>storage.option</code>	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; "ZIP_RA.max" and "LZMA_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
<code>info.import</code>	characters, the variable name(s) in the INFO field for import; or <code>NULL</code> for all variables
<code>fmt.import</code>	characters, the variable name(s) in the FORMAT field for import; or <code>NULL</code> for all variables
<code>genotype.var.name</code>	the ID for genotypic data in the FORMAT column; "GT" by default, VCFv4.0
<code>ignore.chr.prefix</code>	a vector of character, indicating the prefix of chromosome which should be ignored, like "chr"; it is not case-sensitive
<code>reference</code>	genome reference, like "hg19", "GRCh37"; if the genome reference is not available in VCF files, users could specify the reference here
<code>start</code>	the starting variant if importing part of VCF files
<code>count</code>	the maximum count of variant if importing part of VCF files, -1 indicates importing to the end
<code>optimize</code>	if <code>TRUE</code> , optimize the access efficiency by calling cleanup.gds
<code>raise.error</code>	<code>TRUE</code> : throw an error if numeric conversion fails; <code>FALSE</code> : get missing value if numeric conversion fails
<code>digest</code>	a logical value (<code>TRUE</code> / <code>FALSE</code>) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if <code>TRUE</code> or a digest algorithm is specified
<code>parallel</code>	<code>FALSE</code> (serial processing), <code>TRUE</code> (parallel processing), a numeric value indicating the number of cores, or a cluster object for parallel processing; <code>parallel</code> is passed to the argument <code>c1</code> 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="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", 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 File

Description

Parses the header of a VCF file.

Usage

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

Arguments

<code>vcf.fn</code>	the file name; or a connection object
<code>getnum</code>	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):

<code>fileformat</code>	the file format
<code>info</code>	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

*Topic **VCF**

- seqExport, 18
- seqGDS2VCF, 20
- seqVCF2GDS, 48
- seqVCF_Header, 51
- seqVCF_SampID, 52

*Topic **gds**

- KG_P1_SampData, 5
- seqAlleleFreq, 5
- seqApply, 6
- SeqArray-package, 2
- seqAsVCF, 9
- seqBED2GDS, 11
- seqBlockApply, 12
- seqClose-methods, 14
- seqDelete, 15
- seqDigest, 16
- seqExampleFileName, 17
- seqExport, 18
- seqGDS2SNP, 19
- seqGDS2VCF, 20
- seqGetData, 22
- seqGetFilter, 24
- seqMerge, 25
- seqMissing, 27
- seqNumAllele, 28
- seqOpen, 29
- seqOptimize, 30
- seqParallel, 31
- seqParallelSetup, 34
- seqSetFilter-methods, 35
- seqSetFilterCond, 38
- seqSNP2GDS, 39
- seqStorageOption, 41
- seqSummary, 42
- seqSystem, 45
- seqTranspose, 46
- seqVCF2GDS, 48
- seqVCF_Header, 51
- seqVCF_SampID, 52

*Topic **genetics**

- KG_P1_SampData, 5
- seqAlleleFreq, 5

- seqApply, 6
- SeqArray-package, 2
- seqAsVCF, 9
- seqBED2GDS, 11
- seqBlockApply, 12
- seqClose-methods, 14
- seqDelete, 15
- seqDigest, 16
- seqExampleFileName, 17
- seqExport, 18
- seqGDS2SNP, 19
- seqGDS2VCF, 20
- seqGetData, 22
- seqGetFilter, 24
- seqMerge, 25
- seqMissing, 27
- seqNumAllele, 28
- seqOpen, 29
- seqOptimize, 30
- seqParallel, 31
- seqParallelSetup, 34
- seqSetFilter-methods, 35
- seqSetFilterCond, 38
- seqSNP2GDS, 39
- seqStorageOption, 41
- seqSummary, 42
- seqSystem, 45
- seqTranspose, 46
- seqVCF2GDS, 48
- seqVCF_Header, 51
- seqVCF_SampID, 52

*Topic **sequencing**

- KG_P1_SampData, 5
- seqAlleleFreq, 5
- seqApply, 6
- SeqArray-package, 2
- seqAsVCF, 9
- seqBED2GDS, 11
- seqBlockApply, 12
- seqClose-methods, 14
- seqDelete, 15
- seqDigest, 16
- seqExampleFileName, 17

- seqExport, 18
 - seqGDS2SNP, 19
 - seqGDS2VCF, 20
 - seqGetData, 22
 - seqGetFilter, 24
 - seqMerge, 25
 - seqMissing, 27
 - seqNumAllele, 28
 - seqOpen, 29
 - seqOptimize, 30
 - seqParallel, 31
 - seqParallelSetup, 34
 - seqSetFilter-methods, 35
 - seqSetFilterCond, 38
 - seqSNP2GDS, 39
 - seqStorageOption, 41
 - seqSummary, 42
 - seqSystem, 45
 - seqTranspose, 46
 - seqVCF2GDS, 48
 - seqVCF_Header, 51
 - seqVCF_SampID, 52
- add.gdsn, 41, 46
- alt (SeqVarGDSCClass), 47
- alt, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- bgzip, 20
- cleanup.gds, 11, 18, 19, 26, 40, 49
- clusterApply, 32
- clusterApplyLB, 32
- colData (SeqVarGDSCClass), 47
- colData, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- CollapsedVCF, 10
- compression.gdsn, 41
- connection, 7, 12, 20, 49, 51, 53
- DataFrameList, 47
- DNAStrngSet, 47
- DNAStrngSetList, 47
- filt (SeqVarGDSCClass), 47
- filt, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- fixed (SeqVarGDSCClass), 47
- fixed, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- gds.class, 14, 29, 47, 48
- gdsn.class, 7, 12
- geno (SeqVarGDSCClass), 47
- geno, SeqVarGDSCClass, ANY-method
(SeqVarGDSCClass), 47
- geno, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- GRanges, 36
- granges, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- GRangesList, 36
- header (SeqVarGDSCClass), 47
- header, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- info (SeqVarGDSCClass), 47
- info, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- IRanges, 36
- KG_P1_SampData, 5
- makeCluster, 32, 34
- nextRNGStream, 32
- nextRNGSubStream, 32
- openfn.gds, 29
- parallel, 32, 34
- qual (SeqVarGDSCClass), 47
- qual, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- ref (SeqVarGDSCClass), 47
- ref, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- rowRanges (SeqVarGDSCClass), 47
- rowRanges, SeqVarGDSCClass-method
(SeqVarGDSCClass), 47
- seqAlleleCount, 39
- seqAlleleCount (seqAlleleFreq), 5
- seqAlleleFreq, 5, 28, 29, 39
- seqApply, 6, 13, 16, 23, 30–32, 34, 37, 44, 46
- SeqArray (SeqArray-package), 2
- SeqArray-package, 2
- seqAsVCF, 9
- seqBCF2GDS (seqVCF2GDS), 48
- seqBED2GDS, 11, 40
- seqBlockApply, 8, 12
- seqClose, 15, 30
- seqClose (seqClose-methods), 14
- seqClose, gds.class-method
(seqClose-methods), 14

- seqClose, SeqVarGDSCClass-method
(seqClose-methods), 14
- seqClose-methods, 14
- seqDelete, 15
- seqDigest, 16
- seqExampleFileName, 17
- seqExport, 18, 26
- seqGDS2SNP, 19, 40
- seqGDS2VCF, 20, 20, 40, 50
- seqGetData, 8, 13, 22, 30–32, 37, 44, 46, 47
- seqGetFilter, 24, 37
- seqGetParallel, 6, 8, 13, 28, 32
- seqGetParallel (seqParallelSetup), 34
- seqMerge, 25, 42, 50
- seqMissing, 6, 27, 29, 39
- seqNumAllele, 6, 28, 28
- seqOpen, 14, 15, 29, 47, 48
- seqOptimize, 30
- seqParallel, 5–8, 12, 13, 28, 31, 34, 39, 49
- seqParallelSetup, 32, 34
- seqParApply (seqParallel), 31
- seqResetFilter (seqSetFilter-methods),
35
- seqSetFilter, 7, 8, 10, 12, 13, 16, 19, 20, 23,
25, 32, 39
- seqSetFilter (seqSetFilter-methods), 35
- seqSetFilter, SeqVarGDSCClass, ANY-method
(seqSetFilter-methods), 35
- seqSetFilter, SeqVarGDSCClass, GRanges-method
(seqSetFilter-methods), 35
- seqSetFilter, SeqVarGDSCClass, GRangesList-method
(seqSetFilter-methods), 35
- seqSetFilter, SeqVarGDSCClass, IRanges-method
(seqSetFilter-methods), 35
- seqSetFilter-methods, 35
- seqSetFilterChrom, 37, 39
- seqSetFilterChrom
(seqSetFilter-methods), 35
- seqSetFilterCond, 37, 38
- seqSetFilterPos (seqSetFilter-methods),
35
- seqSNP2GDS, 11, 20, 39
- seqStorageOption, 26, 41, 49, 50
- seqSummary, 42
- seqSystem, 45
- seqTranspose, 46
- SeqVarGDSCClass, 5, 7, 10, 12, 14–16, 18–20,
22, 24, 28, 32, 35, 39, 42, 46, 47
- SeqVarGDSCClass-class (SeqVarGDSCClass),
47
- seqVCF2GDS, 11, 18, 20, 21, 26, 40, 42, 47, 48,
52, 53
- seqVCF_Header, 49, 50, 51, 53
- seqVCF_SampID, 52, 52
- VCF-class, 10