# Package 'SeqArray'

November 12, 2026

**Type** Package

**Title** Data management of large-scale whole-genome sequence variant calls using GDS files

**Version** 1.52.1

**Date** 2026-05-20

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

**Imports** methods, parallel, digest, S4Vectors, IRanges, GenomicRanges, Seqinfo, Biostrings

**LinkingTo** gdsfmt

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

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

**License** GPL-3

**VignetteBuilder** knitr

**ByteCompile** TRUE

**LazyData** true

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

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

**biocViews** Infrastructure, DataRepresentation, Sequencing, Genetics

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

**git_branch** RELEASE_3_23

**git_last_commit** fbd42ac

**git_last_commit_date** 2026-05-30

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-11

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

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

# Contents

SeqArray-package . . . 3
KG_P1_SampData . . . 6
seqAddValue . . . 6
seqAlleleFreq . . . 8
seqApply . . . 10
seqAsVCF . . . 14
seqBED2GDS . . . 15
seqBlockApply . . . 17
seqCheck . . . 19
seqClose-methods . . . 20
seqDelete . . . 21
seqDigest . . . 22
seqEmptyFile . . . 23
seqExampleFileName . . . 24
seqExport . . . 25
seqGDS2SNP . . . 26
seqGDS2VCF . . . 28
seqGet2bGeno . . . 30
seqGetData . . . 31
seqGetFilter . . . 34
seqMerge . . . 35
seqMissing . . . 38
seqNewVarData . . . 39
seqNumAllele . . . 40
seqOpen . . . 41
seqOptimize . . . 42
seqParallel . . . 43
seqParallelSetup . . . 46
seqRecompress . . . 48
seqResetVariantID . . . 49
seqSetFilter-methods . . . 50
seqSetFilterCond . . . 55
seqSNP2GDS . . . 56
seqStorageOption . . . 57
seqSummary . . . 59
seqSystem . . . 62
seqTranspose . . . 63

seqUnitApply . . . . . 64
seqUnitCreate . . . . . 66
seqUnitFilterCond . . . . . 67
seqUnitSlidingWindows . . . . . 69
SeqVarGDSClass . . . . . 70
seqVCF2GDS . . . . . 72
seqVCF_Header . . . . . 75
seqVCF_SampID . . . . . 77

**Index** **78**

---

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

---

## Description

Data management of large-scale whole-genome sequencing variants.

## Details

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

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

Webpage: https://github.com/zhengxwen/SeqArray, http://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
head(seqApply(f, "allele",
    FUN=function(x) length(unlist(strsplit(x,","))), as.is="integer"))
# or
head(seqGetData(f, "$num_allele"))



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

# remove the sample and variant filters
seqResetFilter(f)

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


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

## KG_P1_SampData *Simulated sample data for 1000 Genomes Phase 1*

### Description

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

### Usage

```
KG_P1_SampData
```

### Value

An AnnotatedDataFrame

## seqAddValue *Add values to a GDS File*

### Description

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

### Usage

```
seqAddValue(gdsfile, varnm, val, desp=character(), replace=FALSE, compress="LZMA_RA",
  packed=TRUE, packed.idx=TRUE, use_float32=TRUE, verbose=TRUE, verbose.attr=TRUE)
```

### Arguments

| | |
|---|---|
| gdsfile | character for file name, or a SeqVarGDSClass object |
| varnm | the variable name, e.g., "sample.id", "variant.id", "chromosome", "annotation/info/NEW_VARIABLE" |
| val | the R value can be integers, real numbers, characters, factor, logical, raw variable, data.frame or a list; a list of vectors is used for variable-length annotation data; LogicalList, IntegerList, NumericList and CharacterList from the **IRanges** package are also supported for variable-length data; or NULL for adding a new folder |
| desp | variable description |
| replace | if TRUE, replace the existing variable silently if possible |
| compress | the compression method can be "" (no compression), see add.gdsn |
| packed | TRUE, pack data if there is any missing value |
| packed.idx | TRUE, store the index variable using integers with the fewest bits if possible |
| use_float32 | if TRUE, use 32 bits instead of 64 bits to store floating numeric values (i.e., double) |
| verbose | if TRUE, show information |
| verbose.attr | if TRUE, show attribute information in a GDS node |

## Value

Return none.

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS, seqNewVarData, add.gdsn

## Examples

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

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

show(index.gdsn(f, "sample.id"))
seqAddValue(f, "sample.id", 1:90, replace=TRUE)
show(index.gdsn(f, "sample.id"))

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

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

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

# variable-length annotation info data using IntegerList
# v2 <- IRanges::IntegerList(lapply(1:1348, function(x) seq_len(x %% 5)))
# seqAddValue(f, "annotation/info/folder/val2", v2)
# head(seqGetData(f, "annotation/info/folder/val2", .tolist=TRUE))


# sample annotation
```

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

# close the GDS file
seqClose(f)


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

---

`seqAlleleFreq` *Get Allele Frequencies or Counts*

---

## Description

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

## Usage

```
seqAlleleFreq(gdsfile, ref.allele=0L, minor=FALSE, parallel=seqGetParallel(),
    balancing=NA, verbose=FALSE)
seqAlleleCount(gdsfile, ref.allele=0L, minor=FALSE, parallel=seqGetParallel(),
    balancing=NA, verbose=FALSE)
seqGetAF_AC_Missing(gdsfile, minor=FALSE, alt=FALSE, ns=FALSE,
    parallel=seqGetParallel(), balancing=NA, verbose=FALSE)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `ref.allele` | `NULL`, a single numeric value, a numeric vector or a character vector; see Value |
| `minor` | if `TRUE`, return minor allele frequency/count |
| `alt` | if `TRUE` and `minor=FALSE`, return the frequencies and counts for the alternative alleles in `seqGetAF_AC_Missing`; otherwise, return the frequencies and counts for the reference alleles; if there are more than two alleles at a site, the alternative alleles will be collapsed |
| `ns` | if `TRUE`, return the numbers of samples without missing genotypes in the column `ns`, used in `seqGetAF_AC_Missing` |
| `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. |
| `balancing` | whether to perform workload balancing or not, only applicable when multiple cores are used; if `NA`, use `TRUE` as a default until `getOption("seqarray.balancing")` is set and not `TRUE` |
| `verbose` | if `TRUE`, show progress information |

## Details

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

## Value

If `ref.allele=NULL`, the function returns a list of allele frequencies/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).

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

## Author(s)

Xiuwen Zheng

## See Also

`seqMissing`, `seqNumAllele`, `seqParallel`, `seqGetParallel`

## Examples

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

# display
f <- seqOpen(gds.fn)

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

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

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

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

# allele counts
head(seqAlleleCount(f, NULL, verbose=TRUE))
```

```
head(seqAlleleCount(f, 0L, verbose=TRUE))
head(seqAlleleCount(f, 0L, minor=TRUE, verbose=TRUE))
head(seqAlleleCount(f, AA, verbose=TRUE))
head(seqAlleleCount(f, AA, minor=TRUE, verbose=TRUE))

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

# close the GDS file
seqClose(f)

# A GDS file with imputed dosages
ds_fn <- seqExampleFileName("dosage")
ds_fn

str(seqGetAF_AC_Missing(ds_fn, alt=FALSE))
str(seqGetAF_AC_Missing(ds_fn, alt=TRUE))
str(seqGetAF_AC_Missing(ds_fn, minor=TRUE))
```

---

`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, .balancing=FALSE, ...)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object or a GDS file name |
| `var.name` | the variable name(s), see details |
| `FUN` | the function to be applied |
| `margin` | giving the dimension which the function will be applied over; `margin="by.variant"` by default |
| `as.is` | returned value: a list, an integer vector, etc; return nothing by default `as.is="none"`; `as.is` can be a `connection` object, or a GDS node `gdsn.class` object; if "unlist" is used, produces a vector which contains all the atomic components, via `unlist(..., recursive=FALSE)` |

| | |
|---|---|
| var.index | if "none" (by default), call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1 if margin = "by.variant": "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data |
| parallel | FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details. |
| .useraw | TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW for small numbers instead of INTEGER if possible, it is needed to detect data type (RAW or INTEGER) in the user-defined function; for genotypes, 0xFF is missing value if RAW is used |
| .progress | if TRUE, show progress information; or a file name for updating progress information |
| .list_dup | internal use only |
| .balancing | whether to perform workload balancing or not, only applicable when multiple cores are used; if NA, use TRUE as a default until getOption("seqarray.balancing") is set and not TRUE |
| ... | optional arguments to FUN |

## Details

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

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

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

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

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

"$ref" returns a character vector of reference alleles

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

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

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

## Value

A vector, a list of values or none.

## Author(s)

Xiuwen Zheng

## See Also

seqBlockApply, seqSetFilter, seqGetData, seqParallel, seqGetParallel

## Examples

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

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

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

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


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

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

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


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

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

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

seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is=stdout())
seqApply(f, "genotype", FUN=sum, na.rm=TRUE, as.is="integer")
```

```
# should be identical



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

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



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

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



######################################################################
# apply the user-defined function sample by sample

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

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

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

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

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


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

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

---

seqAsVCF *VariantAnnotation objects*

---

## Description

Create a VCF-class object

## Usage

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

## Arguments

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

## Details

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

The **VariantAnnotation** package should be loaded to explore this object.

## Value

A CollapsedVCF object.

## Author(s)

Stephanie Gogarten, Xiuwen Zheng

## See Also

VCF-class

## Examples

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

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

## End(Not run)

seqClose(gds)
```

---

`seqBED2GDS` *Conversion between PLINK BED and SeqArray GDS*

---

## Description

Conversion between PLINK BED format and SeqArray GDS format.

## Usage

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

## Arguments

| | |
|---|---|
| `bed.fn` | the file name of PLINK binary file, genotype information |
| `fam.fn` | the file name of first six columns of `".ped"`, sample or family information; if missing, determine the file name using `bed.fn` |
| `bim.fn` | the file name of extended MAP file with 6 columns, variant information; if missing, determine the file name using `bed.fn` |
| `gdsfile` | character (a GDS file name), or a `SeqVarGDSClass` object |
| `out.gdsfn` | the file name, output a file of SeqArray format |
| `out.fn` | the file name of PLINK binary format without extended names |
| `compress.geno` | the compression method for "genotype"; optional values are defined in the function `add.gdsn` |
| `compress.annotation` | the compression method for the GDS variables, except "genotype"; optional values are defined in the function `add.gdsn` |
| `chr.conv` | if `TRUE`, convert numeric chromosome codes 23 to X, 24 to Y, 25 to XY, and 26 to MT |

| | |
|---|---|
| include.pheno | if TRUE, add 'family', 'father', 'mother', 'sex' and 'phenotype' in the FAM file to the output GDS file; FALSE for no phenotype; or a character vector to specify which of the family, father, mother, sex and phenotype variables to be added |
| optimize | if TRUE, optimize the access efficiency by calling cleanup.gds |
| digest | a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add hash codes to the GDS file if TRUE or a digest algorithm is specified |
| parallel | FALSE (serial processing), TRUE (parallel processing), a numeric value indicating the number of cores, or a cluster object for parallel processing; parallel is passed to the argument cl in seqParallel, see seqParallel for more details |
| write.rsid | "annot_id": use the node "annotation/id" for the variant IDs; "chr_pos_ref_alt": use the format "chrom_position_ref_alt"; "auto": use "annotation/id" for the variant IDs if it is not a blank string or ".", otherwise use "chrom_position_ref_alt" |
| multi.row | if TRUE, a multiallelic site is converted to multiple rows in PLINK bim and bed files |
| verbose | if TRUE, show information |

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqSNP2GDS, seqVCF2GDS

## Examples

```
library(SNPRelate)

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

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

seqSummary("tmp.gds")


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

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

---

seqBlockApply *Apply Functions Over Array Margins via Blocking*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object or a GDS file name |
| var.name | the variable name(s), see details |
| FUN | the function to be applied |
| margin | giving the dimension which the function will be applied over |
| as.is | returned value: a list, an integer vector, etc; return nothing by default as.is="none"; as.is can be a connection object, or a GDS node gdsn.class object; if "unlist" is used, produces a vector which contains all the atomic components, via unlist(..., recursive=FALSE) |
| var.index | if "none" (by default), call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1 if margin="by.variant": "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data |
| bsize | block size |
| parallel | FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details. |
| .useraw | TRUE, force to use RAW instead of INTEGER for genotypes and dosages; FALSE, use INTEGER; NA, use RAW instead of INTEGER if possible; for genotypes, 0xFF is missing value if RAW is used |
| .padNA | TRUE, pad a variable-length vector with NA if the number of data points for each variant is not greater than 1 |
| .tolist | if TRUE, return a list of vectors instead of the structure list(length, data) for variable-length data; NA , return a compressed List defined in **IRanges** when it is applicable |

| | |
|---|---|
| .balancing | whether to perform workload balancing or not, only applicable when multiple cores are used; if NA, use TRUE as a default until getOption("seqarray.balancing") is set and not TRUE |
| .progress | if TRUE, show progress information |
| ... | optional arguments to FUN |

## Details

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

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

"$chromosome" returns chromosome codes in an object of S4Vectors::Rle.

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

"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles. "$dosage_alt2" allow the alleles are partially missing (e.g., genotypes on chromosome X for males)

"$dosage_sp" returns a sparse matrix (dgCMatrix) for the dosages of alternative allele without distinguishing different alternative alleles. "$dosage_sp2" allow the alleles are partially missing (e.g., genotypes on chromosome X for males)

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

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

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

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

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 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)
```

---

seqCheck *Data Integrity Checking*

---

## Description

Performs data integrity on a SeqArray GDS file.

## Usage

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

## Arguments

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

## Value

A list of the following components:

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

## Author(s)

Xiuwen Zheng

## Examples

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

seqCheck(gds.fn)
```

---

# 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.var=character(), fmt.var=character(),
    samp.var=character(), verbose=TRUE)
```

### Arguments

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

### Value

None.

### Author(s)

Xiuwen Zheng

### See Also

seqOpen, seqClose

## Examples

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

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

seqDelete(f, info.var=c("HM2", "AA"), fmt.var="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"), parallel=FALSE, verbose=FALSE)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object or a GDS file name |
| varname | the variable name(s), see details |
| algo | the digest hash algorithm: "md5" |
| parallel | FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details. |
| verbose | if TRUE, show progress information |

### Details

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.

Note that the hashing algorithm is not parallelizable. When multiple processes are used, the hash codes are calculated for each chuck of data, and then call digest() to obtain the final hash code. Hence, the returned hash codes are different when comparing parallel=FALSE and a multicore implementation.

### 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)
```

---

## seqEmptyFile *Empty GDS file*

---

### Description

Create a new empty GDS file.

### Usage

```
seqEmptyFile(outfn, sample.id=character(), numvariant=1L, verbose=TRUE)
```

### Arguments

| | |
|---|---|
| outfn | the output file name for a GDS file |
| sample.id | a list of sample IDs |
| numvariant | the number of variants |
| verbose | if TRUE, show information |

### Value

None.

### Author(s)

Xiuwen Zheng

### See Also

seqVCF2GDS

### Examples

```
seqEmptyFile("tmp.gds")

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

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

---

## seqExampleFileName *Example files*

---

### Description

The example files of VCF and GDS format.

### Usage

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

### Arguments

| | |
|---|---|
| type | "gds" (by default), "vcf", "KG_Phase1" or "dosage" |

### Value

Return the path of a VCF file in the package if type="vcf", or the path of a GDS file if type="gds"; if type="KG_Phase1", return the path of GDS file on Chromosome 22 of the 1000 Genomes Phase 1 project; if type="dosage", return the path of GDS file with imputed allele dosages.

### Author(s)

Xiuwen Zheng

### Examples

```
seqExampleFileName("gds")

seqExampleFileName("vcf")

seqExampleFileName("KG_Phase1")

seqExampleFileName("dosage")
```

---

## 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, verbose.clean=NA)
```

### 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 |
| `verbose.clean` | when `verbose.clean=NA`, set it to `verbose`; whether display information when calling `cleanup.gds` or not; only applicable when `optimize=TRUE` |

### Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS, cleanup.gds

## 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, dosage=FALSE, compress.geno="LZMA_RA",
    compress.annotation="LZMA_RA", ds.type=c("packedreal16", "float", "double"),
    optimize=TRUE, verbose=TRUE)
```

## Arguments

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

## Details

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqSNP2GDS, seqVCF2GDS, seqGDS2VCF

## Examples

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

seqGDS2SNP(gds.fn, "tmp.gds")


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

seqGDS2VCF *Convert to a VCF File*

## Description

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

## Usage

```
seqGDS2VCF(gdsfile, vcf.fn, info.var=NULL, fmt.var=NULL, chr_prefix="",
    header=TRUE, no_sample=NA, use_Rsamtools=TRUE, verbose=TRUE,
    verbose.progress=NA)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| vcf.fn | the file name, output a file of VCF format; or a connection object |
| info.var | a list of variable names in the INFO field, or NULL for using all variables; character(0) for no variable in the INFO field |
| fmt.var | a list of variable names in the FORMAT field, or NULL for using all variables; character(0) for no variable in the FORMAT field |
| chr_prefix | the prefix of chromosome, e.g., "chr"; no prefix by default |
| header | TRUE for exporting the metadata header and the header line containing mandatory column names |
| no_sample | TRUE to exclude the sample-level data and create a site-only VCF file |
| use_Rsamtools | TRUE for loading the Rsamtools package, see details |
| verbose | if TRUE, show information |
| verbose.progress | if NA, set it to verbose; if TRUE, display the progress for each variant |

## Details

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

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

## 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", "tmp.vcf.gz.tbi", "tmp1.vcf.gz", "tmp1.vcf.gz.tbi",
    "tmp2.vcf.gz", "tmp2.vcf.gz.tbi"), force=TRUE)

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

---

`seqGet2bGeno` *Get packed genotypes*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object or a GDS file name |
| `samp_by_var` | if `TRUE`, return a sample-by-variant matrix; otherwise, return a variant-by-sample matrix |
| `ext_nbyte` | additional `ext_nbyte` row(s) with missing genotypes |
| `parallel` | `FALSE` (serial processing), `TRUE` (multicore processing), numeric value or other value; `parallel` is passed to the argument `cl` in `seqParallel`, see `seqParallel` for more details. |
| `verbose` | if `TRUE`, show progress information |

## Details

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

## Value

Return a RAW matrix.

## Author(s)

Xiuwen Zheng

## See Also

seqGetData

## Examples

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

str(seqGet2bGeno(f))

str(seqGet2bGeno(f, samp_by_var=FALSE))

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

---

seqGetData *Get Data*

---

## Description

Gets data from a SeqArray GDS file.

## Usage

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

## Arguments

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

## Details

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

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

"$chromosome" returns chromosome codes in an object of S4Vectors::Rle.

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

"$dosage_alt" returns a RAW/INTEGER matrix for the dosages of alternative allele without distinguishing different alternative alleles. "$dosage_alt2" allow the alleles are partially missing (e.g., genotypes on chromosome X for males)

"$dosage_sp" returns a sparse matrix (dgCMatrix) for the dosages of alternative allele without distinguishing different alternative alleles. "$dosage_sp2" allow the alleles are partially missing (e.g., genotypes on chromosome X for males)

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

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

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

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

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilter, seqApply, seqNewVarData, seqListVarData

## Examples

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

# return a compressed list (CharacterList)
seqGetData(f, "annotation/info/AA", .padNA=FALSE, .tolist=NA)


# 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  ...


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


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

---

`seqGetFilter` *Get the Filter of GDS File*

---

## Description

Gets the filter of samples and variants.

## Usage

```
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

| | |
|---|---|
| gds.fn | the file names of multiple GDS files |
| out.fn | the output file name |
| storage.option | specify the storage and compression option, "ZIP_RA" (seqStorageOption("ZIP_RA")); or "LZMA_RA" to use LZMA compression algorithm with higher compression ratio (by default) |
| info.var | characters, the variable name(s) in the INFO field; NULL for all variables, or character() excludes all INFO variables |
| fmt.var | characters, the variable name(s) in the FORMAT field; NULL for all variables, or character() excludes all FORMAT variables |
| samp.var | characters, the variable name(s) in 'sample.annotation'; or NULL for all variables |
| optimize | if TRUE, optimize the access efficiency by calling cleanup.gds |
| digest | a logical value (TRUE/FALSE) or a character ("md5", "sha1", "sha256", "sha384" or "sha512"); add md5 hash codes to the GDS file if TRUE or a digest algorithm is specified |
| geno.pad | TRUE, pad a 2-bit genotype array in bytes to avoid recompressing genotypes if possible |
| verbose | if TRUE, show information |

## Details

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

WARNING: the functionality of seqMerge() is limited.

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS, seqExport

## Examples

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

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

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

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

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

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

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


####  merging different samples  ####

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

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

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


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

seqMissing *Missing genotype percentage*

## Description

Calculates the missing rates per variant or per sample.

## Usage

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

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| per.variant | missing rate per variant if TRUE, missing rate per sample if FALSE, or calculating missing rates for variants and samples if NA |
| parallel | FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details. |
| balancing | whether to perform workload balancing or not, only applicable when multiple cores are used; if NA, use TRUE as a default until getOption("seqarray.balancing") is set and not TRUE |
| verbose | if TRUE, show progress information |

## Details

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqAlleleFreq, seqNumAllele, seqParallel, seqGetParallel

## Examples

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

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

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

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

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

---

`seqNewVarData` *Variable-length data*

---

## Description

Gets a variable-length data object.

## Usage

```
seqNewVarData(len, data)
seqListVarData(obj, useList=FALSE)
```

## Arguments

| | |
|---|---|
| `len` | a non-negative vector for variable lengths |
| `data` | a vector of data according to `len` |
| `obj` | a `SeqVarDataList` object or a compressed list (defined in **IRanges**) |
| `useList` | if `TRUE`, return a compressed List defined in **IRanges** (e.g., `seqGetData`) |

## Details

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

## Value

Return a `SeqVarDataList` object or a `CompressedAtomicList` object (e.g., `IntegerList()`).

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqAddValue, IntegerList

## Examples

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

seqListVarData(obj)  # a list
(a <- seqListVarData(obj, useList=TRUE))  # CharacterList
seqListVarData(a)  # a list
```

---

# 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 SeqVarGDSClass inherited from gds.class.

## Author(s)

Xiuwen Zheng

## See Also

seqClose, seqGetData, seqApply

## Examples

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

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

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

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

---

`seqOptimize` *Optimize the Storage of Data Array*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object or a GDS file name |
| `target` | "chromosome", "by.sample"; see details |
| `format.var` | a character vector for selected variable names, or `TRUE` for all variables, according to "annotation/format" |
| `cleanup` | call `link{cleanup.gds}` if `TRUE` |
| `verbose` | if `TRUE`, show information |

## Details

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

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

## Value

None.

### Author(s)

Xiuwen Zheng

### See Also

seqGetData, seqApply

### Examples

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

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

# prepare data
seqOptimize("tmp.gds", target="by.sample")


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

# close
seqClose(f)


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

---

## seqParallel *Apply Functions in Parallel*

---

### Description

Applies a user-defined function in parallel.

### Usage

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

## Arguments

| | |
|---|---|
| cl | NULL or FALSE: serial processing; TRUE: multicore processing (the maximum number of cores minor one); a numeric value: the number of cores to be used; a cluster object for parallel processing, created by the functions in the package parallel, like makeCluster; a BiocParallelParam object from the BiocParallel package. See details |
| gdsfile | a SeqVarGDSClass object, or NULL |
| FUN | the function to be applied, should be like FUN(gdsfile, ...) if gdsfile is given, or FUN(...) if gdsfile=NULL |
| split | split the dataset by variant or sample according to multiple processes, or "none" for no split; split="by.variant" by default |
| .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 with one or two arguments, like "+" |
| .selection.flag | TRUE – passes a logical vector of selection to the second argument of FUN(gdsfile, selection, ...) |
| .initialize | a user-defined function for initializing workers, should have two arguments (process_id, param) |
| .finalize | a user-defined function for finalizing workers, should have two arguments (process_id, param) |
| .initparam | parameters passed to .initialize and .initialize |
| .balancing | whether to perform workload balancing or not, only applicable when multiple cores are used; if NA, use TRUE as a default until getOption("seqarray.balancing") is set and not TRUE |
| .bl_size | chuck size (# of variants or samples), the increment for load balancing, only applicable if .balancing=TRUE; if NA_integer_, automatically determined by the number of cores |
| .bl_progress | if TRUE and .balancing=TRUE, show progress information |
| .status_file | if TRUE, create empty files for saving the status of child processes, which can be used in the user-defined function |
| .proc_time | if TRUE, show the elapsed time calling the function |
| x | a vector (atomic or list), passed to FUN |
| ... | 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")` could be used to setup multiple streams of pseudo-random numbers, and see `nextRNGStream` or `nextRNGSubStream` in the package `parallel`.

## Value

A vector or list of values.

## Author(s)

Xiuwen Zheng

## See Also

`seqSetFilter`, `seqGetData`, `seqApply`, `seqParallelSetup`, `seqGetParallel`

## Examples

```
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")

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

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


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


# close the GDS file
seqClose(gdsfile)

# clear the parallel cluster
seqParallelSetup(FALSE)
```

---

`seqParallelSetup` *Setup/Get a Parallel Environment*

---

## Description

Setups a parallel environment in R for the current session.

## Usage

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

## Arguments

| | |
|---|---|
| cluster | NULL or FALSE: serial processing; TRUE: parallel processing with the maximum number of cores minor one; a numeric value: the number of cores to be used; a cluster object for parallel processing, created by the functions in the package parallel, like makeCluster. See details |
| num | the maximum number of cores used for the user-defined multicore setting; FALSE, NA or any value less than 2, to disable the pre-defined multicore cluster; see details |
| type | either PSOCK or Fork cluster setup for the multicore setting, the resulting parallel cluster will be used if 'parallel' is a number greater than one in associated functions |
| verbose | if TRUE, show information |

## Details

When cl is TRUE or a numeric value, forking techniques are used to create a new child process as a copy of the current R process, see ?parallel::mcfork. However, forking is not available on Windows, so multiple processes created by makeCluster are used instead. The R environment option seqarray.parallel will be set according to the value of cluster. Using seqParallelSetup(FALSE) removes the registered cluster, as does stopping the registered cluster.

seqMulticoreSetup sets a multicore cluster and can avoid large memory usage when forking is used for the number of cores specified in the argument 'parallel'. It is recommended to call seqMulticoreSetup before running any memory-intensive functions, as forking within such functions can significantly increase memory usage.

## Value

seqParallelSetup() has no return. seqGetParallel() returns getOption("seqarray.parallel", FALSE). seqMulticoreSetup() invisibly returns the newly created cluster object when a multicore cluster is set up, and otherwise returns invisibly with no value.

## Author(s)

Xiuwen Zheng

## See Also

seqParallel, seqApply

## Examples

```
library(parallel)

seqParallelSetup(2L)

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

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

# run in parallel
summary(seqMissing(f))

# close the GDS file
seqClose(f)

seqParallelSetup(FALSE)
```

---

`seqRecompress` *Recompress the GDS file*

---

## Description

Recompress the SeqArray GDS file.

## Usage

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

## Arguments

| | |
|---|---|
| `gds.fn` | the file name of SeqArray file |
| `compress` | the compression method, `compress="ZIP"` by default |
| `exclude` | a list of GDS nodes to be excluded, see details |
| `optimize` | if `TRUE`, optimize the access efficiency by calling `cleanup.gds` |
| `digest` | a logical value (TRUE/FALSE), whether adding md5 hash codes to the GDS file |
| `verbose` | if `TRUE`, show information |

## Details

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

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

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

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS, seqStorageOption

## Examples

```
gds.fn <- seqExampleFileName("gds")
file.copy(gds.fn, "tmp.gds")

seqRecompress("tmp.gds", "LZMA")

unlink("tmp.gds")
```

---

# seqResetVariantID *Reset Variant ID in SeqArray GDS Files*

---

## Description

Resets the variant IDs in multiple SeqArray GDS files.

## Usage

```
seqResetVariantID(gds.fn, start=1L, set=NULL, digest=TRUE, optimize=TRUE,
    verbose=TRUE)
```

## Arguments

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

## Details

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

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS

## Examples

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

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

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

f <- seqOpen("tmp1.gds")
head(seqGetData(f, "variant.id"))
seqClose(f)

f <- seqOpen("tmp2.gds")
head(seqGetData(f, "variant.id"))
seqClose(f)


# delete the temporary files
unlink(gds.fn, force=TRUE)
```

---

# 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"),
    ret.idx=FALSE, warn=TRUE, verbose=TRUE)
## S4 method for signature 'SeqVarGDSClass,GRanges'
seqSetFilter(object, variant.sel,
```

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

## Arguments

| | |
|---|---|
| object | a SeqVarGDSClass object |
| variant.sel | a logical/raw/index vector indicating the selected variants; 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 |
| ret.idx | if TRUE, return the index in the output array according to the order of 'sample.id', 'sample.sel', 'variant.id' or 'variant.sel' |
| rm.txt | a character, the characters will be removed from seqnames(variant.sel) |
| chr | a vector of character for chromsome coding |
| pos | a vector of numeric values for genome coordinate |
| sample | logical, if TRUE, include all samples |
| variant | logical, if TRUE, include all variants |
| include | NULL, or a vector of characters for specified chromosome(s) |
| is.num | a logical variable: TRUE, chromosome code is numeric; FALSE, chromosome is not numeric; is.num=TRUE is usually used to exclude non-autosomes |
| from.bp | NULL, no limit; a numeric vector, the lower bound of position |

| | |
|---|---|
| to.bp | NULL, no limit; a numeric vector, the upper bound of position |
| intersect | if FALSE, the candidate samples/variants for selection are all samples/variants (by default); if TRUE, the candidate samples/variants are from the selected samples/variants defined via the previous call |
| ref | a character vector of reference alleles for additional matching, or NULL (no reference allele filter); the length should be the same as pos, and NA for entries that match any reference allele. Must be used together with alt |
| alt | a character vector of alternative alleles for additional matching, or NULL (no alternative allele filter); the length should be the same as pos, and NA for entries that match any alternative allele. Must be used together with ref |
| multi.pos | FALSE, use the first matched position; TRUE, allow multiple variants at the same position |
| id | a character vector for RS IDs (stored in "annotation/id") |
| warn | if TRUE, show a warning when the input sample.sel or variant.sel is not ordered as the GDS file or there is any duplicate |
| 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

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

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilterCond, seqGetFilter, seqGetData, seqApply

## Examples

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

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

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

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

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

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


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

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

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

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

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


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

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



## set the intersection

seqResetFilter(f)
seqSetFilterChrom(f, 10L)
seqSummary(f, "genotype", check="none")

AF <- seqAlleleFreq(f)
table(AF <= 0.9)
```

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



## chromosome

seqResetFilter(f)

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

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

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

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

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


## intersection option

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

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


## set filter by position
seqSetFilterPos(f, chr=c(5,14,19,19,1,20),
    pos=c(7945937,93982552,57310283,42747452,158593598,52208935))
as.data.frame(seqGetData(f, c("chromosome", "position", "allele")))

seqSetFilterPos(f, chr=c(5,14,19,19,1,20),
    pos=c(7945937,93982552,57310283,42747452,158593598,52208935),
    ref=c("C","G","G","G","T","C"), alt=c("T","A","A","A","C","T"),
    ret.idx=TRUE)
# NA  1 NA NA NA  2  (NA for ref/alt not matched)
as.data.frame(seqGetData(f, c("chromosome", "position", "allele")))
```

```
# 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(), balancing=NA, .progress=FALSE, verbose=TRUE)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `maf` | minimum minor reference allele frequency, or a range of MAF `maf[1]` <= ... < `maf[2]` |
| `mac` | minimum minor reference allele count, or a range of MAC `mac[1]` <= ... < `mac[2]` |
| `missing.rate` | maximum missing genotype rate |
| `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. |
| `balancing` | whether to perform workload balancing or not, only applicable when multiple cores are used; if `NA`, use `TRUE` as a default until `getOption("seqarray.balancing")` is set and not `TRUE` |
| `.progress` | if `TRUE`, show progress information |
| `verbose` | if `TRUE`, show the number of selected variants |

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqGDS2SNP, seqVCF2GDS, seqGDS2VCF, seqBED2GDS

## Examples

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

## Details

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS, seqRecompress, seqMerge

## Examples

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

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

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

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

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


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

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

---

`seqSummary` *Summarize a SeqArray GDS File*

---

## Description

Gets the summary of SeqArray GDS file.

## Usage

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

## Arguments

| | |
|---|---|
| `gdsfile` | a `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"; `check="default"` by default |
| `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

## Examples

```
seqSystem()
```

seqTranspose *Transpose Data Array*

## Description

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

## Usage

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

## Arguments

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

## Details

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

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqApply

## 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")
```

---

`seqUnitApply` *Apply Function Over Variant Units*

---

## Description

Applies a user-defined function to each variant unit.

## Usage

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

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `units` | a list of units of selected variants, with S3 class `SeqUnitListClass` |
| `var.name` | the variable name(s), see details |
| `FUN` | the function to be applied |
| `as.is` | returned value: a list, an integer vector, etc; return nothing by default `as.is="none"`; `as.is` can be a `connection` object, or a GDS node `gdsn.class` object; if "unlist" is used, produces a vector which contains all the atomic components, via `unlist(..., recursive=FALSE)` |
| `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. |
| `.bl_size` | chuck size, the increment for load balancing, 256 for units |
| `.progress` | if `TRUE`, show progress information |
| `.useraw` | `TRUE`, force to use RAW instead of INTEGER for genotypes and dosages; `FALSE`, use INTEGER; `NA`, use RAW instead of INTEGER if possible; for genotypes, 0xFF is missing value if RAW is used |

| | |
|---|---|
| .padNA | TRUE, pad a variable-length vector with NA if the number of data points for each variant is not greater than 1 |
| .tolist | if TRUE, return a list of vectors instead of the structure list(length, data) for variable-length data |
| .envir | NULL, an environment object, or a list/data.frame |
| ... | optional arguments to FUN |

## Details

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

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

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

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

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

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

"$ref" returns a character vector of reference alleles

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

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

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

## Value

A vector, a list of values or none.

## Author(s)

Xiuwen Zheng

## See Also

seqUnitSlidingWindows, seqUnitFilterCond

## Examples

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

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

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

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


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

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


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

---

`seqUnitCreate` *Subset and merge the units*

---

## Description

Subset, merge and set-diff the variant unit(s).

## Usage

```
seqUnitCreate(idx, desp=NULL)
seqUnitSubset(units, i)
seqUnitMerge(...)
seqUnitSetDiff(ut1, ut2, drop=TRUE)
```

## Arguments

| | |
|---|---|
| `idx` | a list of numeric indexing vectors for specifying variants |
| `desp` | a `data.frame` for annotating the variant sets |

| | |
|---|---|
| units | a list of units of selected variants, with S3 class SeqUnitListClass |
| ... | SeqUnitListClass objects to be merged |
| i | a numeric or logical vector for indices specifying elements |
| ut1 | a SeqUnitListClass object |
| ut2 | a SeqUnitListClass object, whose variant indices will be removed from ut1 |
| drop | if TRUE, remove units that become empty after the set difference operation |

## Value

The variant unit of SeqUnitListClass.

## Author(s)

Xiuwen Zheng

## See Also

seqUnitSlidingWindows, seqUnitFilterCond

## Examples

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

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

(u1 <- seqUnitSubset(units, 1:10))
(u2 <- seqUnitSubset(units, 30:39))

seqUnitMerge(u1, u2)

seqUnitCreate(list(1:10, 20:30), data.frame(gene=c("g1", "g2")))

# set difference: remove variants in u2 from u1
seqUnitSetDiff(u1, u2)

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

---

# seqUnitFilterCond *Filter unit variants*

---

## Description

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

## Usage

```
seqUnitFilterCond(gdsfile, units, maf=NaN, mac=1L, missing.rate=NaN,
    minsize=1L, parallel=seqGetParallel(), balancing=NA, verbose=TRUE)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| units | a list of units of selected variants, with S3 class SeqUnitListClass |
| maf | minimum minor reference allele frequency, or a range of MAF maf[1] <= ... < maf[2] |
| mac | minimum minor reference allele count, or a range of MAC mac[1] <= ... < mac[2] |
| missing.rate | maximum missing genotype rate |
| minsize | the minimum of unit size |
| parallel | FALSE (serial processing), TRUE (multicore processing), numeric value or other value; parallel is passed to the argument cl in seqParallel, see seqParallel for more details. |
| balancing | whether to perform workload balancing or not, only applicable when multiple cores are used; if NA, use TRUE as a default until getOption("seqarray.balancing") is set and not TRUE |
| verbose | if TRUE, show information |

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

seqUnitApply, seqUnitCreate, seqUnitSubset, seqUnitMerge

## Examples

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

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

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

```
# excluded variants
exvar <- setdiff(unique(unlist(unit1$index)), unique(unlist(newunit$index)))

seqSetFilter(gdsfile, variant.sel=exvar)
maf <- seqAlleleFreq(gdsfile, minor=TRUE)
table(maf > 0)
summary(maf[maf > 0])  # > 0.01


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

---

`seqUnitSlidingWindows` *Sliding units of selected variants*

---

## Description

Generates units of selected variants via sliding windows.

## Usage

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

## Arguments

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

## Value

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

## Author(s)

Xiuwen Zheng

## See Also

`seqUnitApply`, `seqUnitFilterCond`

## Examples

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

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

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

---

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 DNAStringSet.

alt(x) Returns the alternate alleles as a DNAStringSetList.

qual(x) Returns the quality scores.

filt(x) Returns the filter data.

fixed(x) Returns the fixed fields (ref, alt, qual, filt).

header(x) Returns the header as a DataFrameList.

rowRanges(x) Returns a GRanges object with metadata.

colData(x) Returns a DataFrame with sample identifiers and any information in the 'sample.annotation' node.

info(x, info=NULL) Returns the info fields as a DataFrame. info is a character vector with the names of fields to return (default is to return all).

geno(x, geno=NULL) Returns the geno (format) fields as a SimpleList. geno is a character vector with the names of fields to return (default is to return all).

Other data can be accessed with seqGetData.

### Coercion methods

In the following code snippets x is a SeqVarGDSClass object.

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

### Author(s)

Stephanie Gogarten, Xiuwen Zheng

### See Also

gds.class, seqOpen

### Examples

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

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

## variants
granges(gds)

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

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

## genotype
geno <- seqGetData(gds, "genotype")
dim(geno)
## dimensions are: allele, sample, variant
geno[1,1:10,1:5]

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

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

## individual read depth
depth <- seqGetData(gds, "annotation/format/DP")
names(depth)
## VCF header defined DP as variable-length data
table(depth$length)
## all length 1, so depth$data should be a sample by variant matrix
dim(depth$data)
depth$data[1:10,1:5]
```

```
## End(Not run)

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",
    ploidy=NA_integer_, ignore.chr.prefix="chr",
    scenario=c("general", "imputation"), reference=NULL, start=1L, count=-1L,
    variant_count=NA_integer_, optimize=TRUE, raise.error=TRUE, digest=TRUE,
    use_Rsamtools=NA, 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",
    ploidy=NA_integer_, ignore.chr.prefix="chr",
    scenario=c("general", "imputation"), reference=NULL, optimize=TRUE,
    raise.error=TRUE, digest=TRUE, bcftools="bcftools", verbose=TRUE)
```

## Arguments

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

| | |
|---|---|
| ploidy | specify the ploidy by users or NA to determine the ploidy from the VCF header |
| ignore.chr.prefix | a vector of character, indicating the prefix of chromosome which should be ignored, e.g., "chr"; it is not case-sensitive |
| scenario | "general": use float32 to store floating-point numbers (by default); "imputation": use packedreal16 to store DS and GP in the FORMAT field with four decimal place accuracy |
| reference | genome reference, like "hg19", "GRCh37"; if the genome reference is not available in VCF files, users could specify the reference here |
| start | the starting variant if importing part of VCF files |
| count | the maximum count of variant if importing part of VCF files, -1 indicates importing to the end |
| variant_count | NA_integer_ (default) or a numeric vector specifying the numbers of variants in the VCF file(s) in vcf.fn; only applicable when multiple cores are used; if the number of variants is known, the conversion can skip counting the variants before splitting the file(s); variant_count could be an approximate |
| 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 |
| use_Rsamtools | only applicable when multiple cores are used; use_Rsamtools is passed to seqVCF_Header to get the total number of variants; NA: using Rsamtools when it is installed; FALSE: not use the Rsamtools package; TRUE: to use Rsamtools, if it is not installed, the function fails |
| parallel | FALSE (serial processing), TRUE (parallel processing), a numeric value indicating the number of cores, or a cluster object for parallel processing; parallel is passed to the argument cl in seqParallel, see seqParallel for more details |
| verbose | if TRUE, show information |
| bcftools | the path of the program bcftools |

## Details

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

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

By default, the compression method is "LZMA_RA" (https://tukaani.org/xz/, LZMA algorithm with default compression level + independent data blocks for fine-level random access). Users can maximize the compression ratio by storage.option="LZMA_RA.max" or storage.option=seqStorageOption("LZMA_ LZMA is known to have higher compression ratio than the zlib algorithm. LZ4 (https://github.com/lz4/lz4) is an option via storage.option="LZ4_RA" or storage.option=seqStorageOption("LZ4_RA").

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

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

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

## 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`, `seqRecompress`

## Examples

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

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

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


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



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

# display
```

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



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

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


# delete the temporary file
unlink(c("tmp.gds", "tmp_p2.gds"), force=TRUE)
```

---

`seqVCF_Header` *Parse the Header of a VCF/BCF File*

---

## Description

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

## Usage

```
seqVCF_Header(vcf.fn, getnum=FALSE, use_Rsamtools=NA, parallel=FALSE,
    verbose=TRUE)
```

## Arguments

| | |
|---|---|
| `vcf.fn` | the file name of VCF or BCF format; or a `connection` object for VCF format |
| `getnum` | if `TRUE`, return the total number of variants |
| `use_Rsamtools` | only applicable when `getnum=TRUE` and multiple cores are used; `NA`: using Rsamtools when it is installed; `FALSE`: not use the Rsamtools package; `TRUE`: to use Rsamtools, if it is not installed, the function fails |
| `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` | when `getnum=TRUE` and `verbose=TRUE`, show the progress information for scanning the file |

## 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.

If multiple cores are used to get the total number of variants, the Rsamtools package should be installed, and the indexing file (i.e., .csi or .tbi) should be available along with `vcf.fn`.

## Value

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

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

## Author(s)

Xiuwen Zheng

## References

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


∗ **VCF**
seqExport, 25
seqGDS2VCF, 28
seqVCF2GDS, 72
seqVCF_Header, 75
seqVCF_SampID, 77
∗ **gds**
KG_P1_SampData, 6
seqAddValue, 6
seqAlleleFreq, 8
seqApply, 10
SeqArray-package, 3
seqAsVCF, 14
seqBED2GDS, 15
seqBlockApply, 17
seqCheck, 19
seqClose-methods, 20
seqDelete, 21
seqDigest, 22
seqEmptyFile, 23
seqExampleFileName, 24
seqExport, 25
seqGDS2SNP, 26
seqGDS2VCF, 28
seqGet2bGeno, 30
seqGetData, 31
seqGetFilter, 34
seqMerge, 35
seqMissing, 38
seqNewVarData, 39
seqNumAllele, 40
seqOpen, 41
seqOptimize, 42
seqParallel, 43
seqParallelSetup, 46
seqRecompress, 48
seqResetVariantID, 49
seqSetFilter-methods, 50
seqSetFilterCond, 55
seqSNP2GDS, 56
seqStorageOption, 57
seqSummary, 59
seqSystem, 62
seqTranspose, 63
seqUnitApply, 64
seqUnitCreate, 66
seqUnitFilterCond, 67
seqUnitSlidingWindows, 69
seqVCF2GDS, 72
seqVCF_Header, 75
seqVCF_SampID, 77
∗ **genetics**
KG_P1_SampData, 6
seqAddValue, 6
seqAlleleFreq, 8
seqApply, 10
SeqArray-package, 3
seqAsVCF, 14
seqBED2GDS, 15
seqBlockApply, 17
seqCheck, 19
seqClose-methods, 20
seqDelete, 21
seqDigest, 22
seqEmptyFile, 23
seqExampleFileName, 24
seqExport, 25
seqGDS2SNP, 26
seqGDS2VCF, 28
seqGet2bGeno, 30
seqGetData, 31
seqGetFilter, 34
seqMerge, 35
seqMissing, 38
seqNumAllele, 40
seqOpen, 41
seqOptimize, 42
seqParallel, 43

seqParallelSetup, 46
seqRecompress, 48
seqResetVariantID, 49
seqSetFilter-methods, 50
seqSetFilterCond, 55
seqSNP2GDS, 56
seqStorageOption, 57
seqSummary, 59
seqSystem, 62
seqTranspose, 63
seqUnitApply, 64
seqUnitCreate, 66
seqUnitFilterCond, 67
seqUnitSlidingWindows, 69
seqVCF2GDS, 72
seqVCF_Header, 75
seqVCF_SampID, 77

∗ **sequencing**
KG_P1_SampData, 6
seqAddValue, 6
seqAlleleFreq, 8
seqApply, 10
SeqArray-package, 3
seqAsVCF, 14
seqBED2GDS, 15
seqBlockApply, 17
seqCheck, 19
seqClose-methods, 20
seqDelete, 21
seqDigest, 22
seqEmptyFile, 23
seqExampleFileName, 24
seqExport, 25
seqGDS2SNP, 26
seqGDS2VCF, 28
seqGet2bGeno, 30
seqGetData, 31
seqGetFilter, 34
seqMerge, 35
seqMissing, 38
seqNewVarData, 39
seqNumAllele, 40
seqOpen, 41
seqOptimize, 42
seqParallel, 43
seqParallelSetup, 46
seqRecompress, 48
seqResetVariantID, 49
seqSetFilter-methods, 50
seqSetFilterCond, 55
seqSNP2GDS, 56
seqStorageOption, 57
seqSummary, 59
seqSystem, 62
seqTranspose, 63
seqUnitApply, 64
seqUnitCreate, 66
seqUnitFilterCond, 67
seqUnitSlidingWindows, 69
seqVCF2GDS, 72
seqVCF_Header, 75
seqVCF_SampID, 77

add.gdsn, *6, 7*, *27*, *56*, *58*, *63*
alt *(*SeqVarGDSClass*)*, 70
alt,SeqVarGDSClass-method *(*SeqVarGDSClass*)*, 70

bgzip, *28*

cleanup.gds, *16*, *25–27*, *36*, *48, 49*, *56*, *73*
colData *(*SeqVarGDSClass*)*, 70
colData,SeqVarGDSClass-method *(*SeqVarGDSClass*)*, 70
CollapsedVCF, *14*
compression.gdsn, *58*
connection, *10*, *17*, *28*, *64*, *72*, *75*, *77*

DataFrame, *70*
DataFrameList, *70*
DNAStringSet, *70*
DNAStringSetList, *70*

filt *(*SeqVarGDSClass*)*, 70
filt,SeqVarGDSClass-method *(*SeqVarGDSClass*)*, 70
fixed *(*SeqVarGDSClass*)*, 70
fixed,SeqVarGDSClass-method *(*SeqVarGDSClass*)*, 70

gds.class, *20*, *41*, *70, 71*
gdsn.class, *10*, *17*, *64*
geno *(*SeqVarGDSClass*)*, 70
geno,SeqVarGDSClass,ANY-method *(*SeqVarGDSClass*)*, 70
geno,SeqVarGDSClass-method *(*SeqVarGDSClass*)*, 70
GRanges, *51*, *70*

granges,SeqVarGDSClass-method
(SeqVarGDSClass), 70
GRangesList, *51*

header (SeqVarGDSClass), 70
header,SeqVarGDSClass-method
(SeqVarGDSClass), 70

info (SeqVarGDSClass), 70
info,SeqVarGDSClass-method
(SeqVarGDSClass), 70
IntegerList, *40*
IRanges, *51*

KG_P1_SampData, 6

makeCluster, *44*, *47*

nextRNGStream, *45*
nextRNGSubStream, *45*

openfn.gds, *41*

parallel, *44*, *47*

qual (SeqVarGDSClass), 70
qual,SeqVarGDSClass-method
(SeqVarGDSClass), 70

ref (SeqVarGDSClass), 70
ref,SeqVarGDSClass-method
(SeqVarGDSClass), 70
rowRanges (SeqVarGDSClass), 70
rowRanges,SeqVarGDSClass-method
(SeqVarGDSClass), 70

seqAddValue, 6, *40*
seqAlleleCount, *55*
seqAlleleCount (seqAlleleFreq), 8
seqAlleleFreq, 8, *38*, *40*, *55*
seqApply, 10, *18*, *23*, *32*, *41*, *43*, *45*, *47*, *52*, *61*, *63*
SeqArray (SeqArray-package), 3
SeqArray-package, 3
seqAsVCF, 14
seqBCF2GDS (seqVCF2GDS), 72
seqBED2GDS, 15, *57*
seqBlockApply, *12*, 17
seqCheck, 19
seqClose, *21*, *41*
seqClose (seqClose-methods), 20
seqClose,gds.class-method
(seqClose-methods), 20
seqClose,SeqVarGDSClass-method
(seqClose-methods), 20
seqClose-methods, 20
seqDelete, 21
seqDigest, 22
seqEmptyFile, 23
seqExampleFileName, 24
seqExport, 25, *36*
seqFilterPop (seqSetFilter-methods), 50
seqFilterPush (seqSetFilter-methods), 50
seqGDS2BED (seqBED2GDS), 15
seqGDS2SNP, 26, *57*
seqGDS2VCF, *27*, 28, *57*, *74*
seqGet2bGeno, 30
seqGetAF_AC_Missing (seqAlleleFreq), 8
seqGetData, *12*, *18*, *31*, 31, *39–41*, *43*, *45*, *52*, *61*, *63*, *70*
seqGetFilter, 34, *52*
seqGetParallel, *9*, *12*, *18*, *38*, *45*
seqGetParallel (seqParallelSetup), 46
seqListVarData, *32*
seqListVarData (seqNewVarData), 39
seqMerge, 35, *58*, *74*
seqMissing, *9*, 38, *40*, *55*
seqMulticoreSetup (seqParallelSetup), 46
seqNewVarData, *7*, *32*, 39
seqNumAllele, *9*, *38*, 40
seqOpen, *21*, 41, *70*, *71*
seqOptimize, 42
seqParallel, *8*, *9*, *11*, *12*, *16–18*, *22*, *30*, *38*, 43, *47*, *55*, *64*, *68*, *73*, *75*
seqParallelSetup, *45*, 46
seqParApply (seqParallel), 43
seqRecompress, 48, *58*, *74*
seqResetFilter (seqSetFilter-methods), 50
seqResetVariantID, 49
seqSetFilter, *11*, *12*, *14*, *17*, *18*, *23*, *27*, *28*, *32*, *35*, *45*, *55*
seqSetFilter (seqSetFilter-methods), 50
seqSetFilter,SeqVarGDSClass,ANY-method
(seqSetFilter-methods), 50
seqSetFilter,SeqVarGDSClass,GRanges-method
(seqSetFilter-methods), 50
seqSetFilter,SeqVarGDSClass,GRangesList-method

(seqSetFilter-methods), 50
seqSetFilter,SeqVarGDSClass,IRanges-method
(seqSetFilter-methods), 50
seqSetFilter-methods, 50
seqSetFilterAnnotID
(seqSetFilter-methods), 50
seqSetFilterChrom, *55*
seqSetFilterChrom
(seqSetFilter-methods), 50
seqSetFilterCond, *52*, 55
seqSetFilterPos (seqSetFilter-methods), 50
seqSNP2GDS, *16*, *27*, 56
seqStorageOption, *36*, *49*, 57, *72*, *74*
seqSummary, 59
seqSystem, 62
seqTranspose, 63
seqUnitApply, 64, *68, 69*
seqUnitCreate, 66, *68*
seqUnitFilterCond, *65*, *67*, 67, *69*
seqUnitMerge, *68*
seqUnitMerge (seqUnitCreate), 66
seqUnitSetDiff (seqUnitCreate), 66
seqUnitSlidingWindows, *65*, *67*, 69
seqUnitSubset, *68*
seqUnitSubset (seqUnitCreate), 66
SeqVarGDSClass, *6*, *8*, *10*, *14, 15*, *17*, *19–22*, *25*, *27, 28*, *30, 31*, *34*, *38*, *40*, *42*, *44*, *51*, *55*, *59*, *63, 64*, *68, 69*, 70
SeqVarGDSClass-class (SeqVarGDSClass), 70
seqVCF2GDS, *7*, *16*, *24*, *26, 27*, *29*, *36*, *48–50*, *57, 58*, *70*, 72, *76, 77*
seqVCF_Header, *72–74*, 75, *77*
seqVCF_SampID, *76*, 77
SimpleList, *70*

VCF-class, *14*