# Package 'SeqArray'

April 5, 2014

**Type** Package

**Title** Big Data Management of Genome-wide Sequencing Variants

**Version** 1.2.1

**Date** 2013-09-23

**Depends** gdsfmt (>= 1.0.0)

**Imports** methods, Biostrings, GenomicRanges, IRanges, VariantAnnotation

**Suggests** parallel, snow, BiocGenerics, RUnit, Rcpp

**Author** Xiuwen Zheng, Stephanie M. Gogarten, Cathy Laurie

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

**Description** Big data management of genome-
wide variants using the CoreArray library: genotypic data and annotations are stored in an array-oriented manner, offering efficient access of genetic variants using the R language.

**License** GPL-3

**biocViews** Bioinformatics, Infrastructure, Sequencing, Genetics

**URL** http://corearray.sourceforge.net/tutorials/SeqArray/

## R topics documented:

---

`SeqArray-package` *Big Data Management of Genome-wide Sequencing Variants*

---

## Description

Big-Data Management of Genome-Wide Sequencing Variants

## Details

In the era of big data, thousands of gigabyte-size data sets are challenging scientists for data management, even on well-equipped hardware. Currently, next-generation sequencing techniques are being adopted to investigate common and rare variants, making the analyses of large-scale genotypic data challenging. For example, the 1000 Genomes Project has identified approximately 38 million single nucleotide polymorphisms (SNPs), 1.4 million short insertions and deletions, and more than 14,000 larger deletions from whole-genome sequencing technologies. In the near future, new technologies, like third-generation whole-genome sequencing, will be enabling data to be generated at an unprecedented scale. The Variant Call Format (VCF) was developed for the 1000 Genomes Project, which is a generic text format for storing DNA polymorphism data such as SNPs, insertions, deletions and structural variants, together with rich annotations. However, this format is less efficient for large-scale analyses since numeric data have to be parsed from a text VCF file before further analyses. The computational burden associated with sequencing variants is especially evident with large sample and variant sizes, and it requires efficient numerical implementation and data management.

Here I introduce a high-performance C/C++ computing library CoreArray (`http://corearray.sourceforge.net`) for big-data management of genome-wide variants. CoreArray was designed for developing portable and scalable storage technologies for bioinformatics data, allowing parallel computing at the multicore and cluster levels. It provides the genomic data structure (GDS) file format for array-oriented data: this is a universal data format to store multiple data variables in a single file. A hierarchical data structure is used to store multiple extensible data variables in the GDS format, and all datasets are stored in a single file with chunked storage layout. Here, I focus on the application of CoreArray for statisticians working in the R environment, and developed an R/Bioconductor package SeqArray to address or reduce the computational burden associated with data management of sequencing variants. The kernels of SeqArray are written in C/C++ and highly

optimized. Genotypic data and annotations are stored in an array-oriented manner, offering efficient access of genetic variants using the R language. There are five key functions in SeqArray, and most of data analyses could be done using these 6 functions:

| Function | Description |
|---|---|
| seqVCF2GDS | Imports VCF files |
| seqSummary | Gets the summary of a sequencing GDS file (# of samples, # of variants, INFO/FORMAT variables, etc) |
| seqSetFilter | Sets a filter to sample or variant (define a subset of data) |
| seqGetData | Gets data from a sequencing GDS file (from a subset of data) |
| seqApply | Applies a user-defined function over array margins |
| seqParallel | Applies functions in parallel |

The 1000 Genomes Project released 39 million genetic variants for 1092 individuals, and a 26G data file was created by SeqArray to store sequencing variants with phasing information, where 2 bits were used as an atomic data type. The file size can be further reduced to 1.3G by compression algorithms without sacrificing access efficiency, since it has a large proportion of rare variants.

SeqArray will be of great interest to scientists involved in data analyses of large-scale genomic sequencing data using R environment, particularly those with limited experience of low-level C programming and parallel computing.

Webpage: http://corearray.sourceforge.net/

Tutorial: http://corearray.sourceforge.net/tutorials/SeqArray/

Forums: http://sourceforge.net/projects/corearray/forums

## Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

## Examples

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

# parse the header
seqVCF.Header(vcf.fn)

# get sample id
seqVCF.SampID(vcf.fn)

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

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

```
seqClose(f)
unlink("tmp.gds")


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

# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.gds"

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

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

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

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

# get allele
head(seqGetData(f, "allele"))
# "T,C" "G,A" "G,A" ...


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

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

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

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

# get annotation/format/DP, a variable-length dataset
seqGetData(f, "annotation/format/DP")
# $length             <- indicating the length of each variable-length data
# [1] 1 1 1 1 1 1 ...
# $data               <- the data according to $length
#      variant
# sample [,1] [,2] [,3] [,4] [,5] [,6] ...
```

```
#  [1,]   25   25   22    3    4   17  ...


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

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


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

# remove the sample and variant filters
seqSetFilter(f)

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


######################################################################
# run in parallel

library(parallel)

# Use option cl.core to choose an appropriate cluster size or number of cores
cl <- makeCluster(getOption("cl.cores", 2))

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

length(afreq)
summary(afreq)


stopCluster(cl)


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

---

seqApply *Apply Functions Over Array Margins*

---

## Description

Returns a vector or list of values obtained by applying a function to margins of arrays or matrices

## Usage

```
seqApply(gdsfile, var.name, FUN,
margin = c("by.variant"),
as.is = c("list", "integer", "double", "character", "none"),
var.index = c("none", "relative", "absolute"), ...)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| var.name | the variable name(s), see details |
| FUN | the function to be applied |
| margin | giving the dimension which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns |
| as.is | returned value: a list, an integer vector, etc |
| var.index | if "none", call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1 if margin = "by.variant": "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data |
| ... | optional arguments to FUN |

## 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".

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

## Value

A vector or list of values.

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilter, seqGetData, seqParallel

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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 multiple variables variant by variant
seqApply(f, c(geno="genotype", phase="phase", qual="annotation/id"),
FUN=function(x) print(x), as.is="none")

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


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

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



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

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

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

---

`seqClose-methods` *Close a SeqArray object*

---

## Description

Close a SeqArray file or object

## Usage

```
## S4 method for signature SeqVarGDSClass
seqClose(object)
```

## Arguments

| | |
|---|---|
| `object` | a SeqArray object |

## Details

If `object` is

- `SeqVarGDSClass`, close the sequencing GDS file.

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

`seqOpen`

`seqCompress.Option` *Compression Options for Importing VCF File(s)*

## Description

Get compression options for importing VCF file(s)

## Usage

```
seqCompress.Option(default="ZIP.MAX", ...)
```

## Arguments

| | |
|---|---|
| `default` | the default compression level |
| `...` | optional arguments |

## Value

Return a list with a class name "seqCompress.class".

## Author(s)

Xiuwen Zheng

## See Also

`seqVCF2GDS`

## Examples

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

# convert
seqVCF2GDS(vcf.fn, "tmp1.gds", compress.option = seqCompress.Option())
(f1 <- seqOpen("tmp1.gds"))


# does not compress the genotypic data
seqVCF2GDS(vcf.fn, "tmp2.gds", compress.option = seqCompress.Option(genotype=""))
(f2 <- seqOpen("tmp2.gds"))


# close and remove
seqClose(f1)
seqClose(f2)
unlink(c("tmp1.gds", "tmp2.gds"))
```

seqDelete *Delete variables in a Sequencing GDS File*

## Description

Delete variables in a sequencing GDS file

## Usage

```
seqDelete(gdsfile, info.varname=NULL, format.varname=NULL)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| info.varname | the variables in the INFO field, i.e., "annotation/info/VARIABLE_NAME" |
| format.varname | the variables in the FORMAT field, i.e., "annotation/format/VARIABLE_NAME" |

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqOpen, seqClose

## Examples

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

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

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

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

# close the GDS file
seqClose(f)

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

## seqExampleFileName *Example GDS file*

### Description

The example GDS file for sequencing variants

### Usage

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

### Arguments

| | |
|---|---|
| type | either "gds" or "vcf" |

### Details

A GDS sequencing-variant file was created from a subset of VCF data of the 1000 Genomes Project.

### Value

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

### Author(s)

Xiuwen Zheng

### Examples

```
seqExampleFileName("gds")

seqExampleFileName("vcf")
```

## seqGDS2VCF *Convert to a VCF file*

### Description

Convert a GDS file to a VCF file

### Usage

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

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| vcf.fn | the file name, output a file of VCF format |
| info.var | a list of variable names in the INFO field, or NULL for using all variables; character(0) for no variable in the INFO field |
| fmt.var | a list of variable names in the FORMAT field, or NULL for using all variables; character(0) for no variable in the FORMAT field |
| verbose | if TRUE, show information |

## Details

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

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

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

## Value

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

## Author(s)

Xiuwen Zheng

## References

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

http://corearray.sourceforge.net/

## See Also

seqVCF2GDS

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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(0), fmt.var=character(0))

# 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(0))


# 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
seqSetFilter(f)

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


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

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

# LOOK GOOD!


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

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

---

seqGetData *Get Data from a Sequencing GDS file*

---

## Description

Get data from a sequencing GDS file

## Usage

```
seqGetData(gdsfile, var.name)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| var.name | the variable name, see details |

## 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".

## Value

Return vectors or lists.

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilter, seqApply

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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  ...

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

---

`seqGetFilter` *Get the Filter of Samples and Variants*

---

## Description

Get the filter of samples and variants

## Usage

```
seqGetFilter(gdsfile)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |

## Details

It is strongly suggested to call seqOpen instead of openfn.gds, since seqOpen will initialize the internal data for seqGetData, seqApply, etc.

## Value

Return a list:

| | |
|---|---|
| sample.sel | a logical vector for selected samples |
| variant.sel | a logical vector for selected variants |

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqApply

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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)


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

`seqInfoNewVar` *Add a variable to the INFO field*

## Description

Add a new variable to the INFO field in the specified GDS file.

## Usage

```
seqInfoNewVar(gdsfile, var.name, variant.id, val,
    description="", compress=c("ZIP.MAX", ""), no.data.index=TRUE)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `var.name` | the variable name(s), see details |
| `variant.id` | the variant ID(s), should have the same order as IDs in the GDS file |
| `val` | a vector or a matrix with the same order as `variant.id`; if it is a matrix, the number of columns should be equal to the length of `variant.id`, see details |
| `description` | the variable description |
| `compress` | to specify the compression algorithm: `""` for no compression; see `compress` in the function `add.gdsn` |
| `no.data.index` | applicable only if `val` is a numeric vector or a factor variable; if `no.data.index=TRUE`, non-exist values in `val` will be replaced by `NA` or `NaN` with respect to the variants not specified in `variant.id`, and no index data associated with this variable are created |

## 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"`.

The argument `val` should be integers, numeric values, a logical variable, characters or factors. If `val` is a logical variable, one-bit storage mode will be used to store this variable, which corresponds to the variable defined with 'Type=Flag' in the VCF format.

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqGetData

## Examples

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

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

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

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


####    add variables to the INFO field    ####

set.seed(100)

seqInfoNewVar(f, "int", variant.id[1:15], sample.int(256, 15, TRUE),
"integer variable")
seqGetData(f, "annotation/info/int")

seqInfoNewVar(f, "int.2", variant.id[1:15], sample.int(256, 15, TRUE),
"integer variable", no.data.index=FALSE)
seqGetData(f, "annotation/info/int.2")


seqInfoNewVar(f, "numeric", variant.id[15:30], rnorm(16),
"numeric variable")
seqGetData(f, "annotation/info/numeric")

seqInfoNewVar(f, "numeric.2", variant.id[15:30], rnorm(16),
"numeric variable", no.data.index=FALSE)
seqGetData(f, "annotation/info/numeric.2")


seqInfoNewVar(f, "flag", variant.id[4:9], rep(c(FALSE, TRUE), 3),
"flag variable")
# stored in bit1
seqGetData(f, "annotation/info/flag")


seqInfoNewVar(f, "factor", variant.id,
factor(c("ABC", "DDD", "CVX")[sample(1:3, length(variant.id), TRUE)]),
"string/factor variable")
# stored in int32 with attributes
seqGetData(f, "annotation/info/factor")
```

```
seqInfoNewVar(f, "string", variant.id,
c("ABC", "DDD", "CVX")[sample(1:3, length(variant.id), TRUE)],
"string variable")
seqGetData(f, "annotation/info/string")


# show the file
f


# the corresponding VCF file
seqGDS2VCF(f, "test.vcf.gz")
txt <- strsplit(readLines("test.vcf.gz", n=40), "\t")[-c(1:21)]

# the INFO field:
sapply(txt, function(x) x[8])


# close the GDS file
seqClose(f)


# delete the temporary files
unlink("test.gds", force=TRUE)
unlink("test.vcf.gz", force=TRUE)
```

---

`seqMerge` *Merge Multiple Sequencing GDS Files*

---

## Description

Merge multiple sequencing GDS files

## Usage

```
seqMerge(gds.fn, out.fn, compress.option = seqCompress.Option(),
verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `gds.fn` | the file names of multiple GDS files |
| `out.fn` | the output file name |
| `compress.option` | specify the compression options, by default `seqCompress.Option` |
| `verbose` | if TRUE, show information |

## Details

The current implementation of `seqMerge` extracts and merges the genotypic data only without any annotation. Users can specify multiple VCF files in `seqVCF2GDS` to export a single GDS file.

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqVCF2GDS

---

seqOpen *Open a Sequencing GDS File*

---

## Description

Open a Sequencing GDS file

## Usage

```
seqOpen(gds.fn, readonly=TRUE)
```

## Arguments

| | |
|---|---|
| gds.fn | the file name |
| readonly | whether read-only or not |

## Details

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

## Value

Return an object of class gds.class.

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqApply

## Examples

```
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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)
```

---

# seqParallel *Apply Functions in Parallel*

---

## Description

Apply a user-defined function in parallel

## Usage

```
seqParallel(cl, gdsfile, FUN = function(gdsfile, ...) NULL,
split=c("by.variant", "by.sample", "none"), .combine=NULL, ...)
```

## Arguments

| | |
|---|---|
| `cl` | NULL or a cluster object, created by the package parallel or snow |
| `gdsfile` | a `SeqVarGDSClass` object |
| `FUN` | the function to be applied |
| `split` | split the dataset by variant or sample according to multiple processes, or "none" for no split |
| `.combine` | define a fucntion for combining results from different processes; by default, `c` is used; if `.combine==""`, return `invisible()` |
| `...` | optional arguments to `FUN` |

## Details

If `cl = NULL` or `length(cl) == 0`, the function simply calls `FUN(gdsfile, ...)`; otherwise, it splits jobs to different processes and calls `FUN(gdsfile, ...)` on each process, the optional arguments are passed to different processes.

## Value

A vector or list of values.

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilter, seqGetData seqApply

## Examples

```
library(parallel)

# Use option cl.core to choose an appropriate cluster size or number of cores
cl <- makeCluster(getOption("cl.cores", 2))


# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.gds"

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

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

length(afreq2)
summary(afreq2)


# check
all(afreq1 == afreq2)


######################################################################
# check -- variant splits

seqParallel(cl, f, FUN = function(gdsfile) {
v <- seqGetFilter(gdsfile)
sum(v$variant.sel)
```

```
}, split = "by.variant")
# [1] 674 674


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

stopCluster(cl)

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

---

`seqSetFilter` *Set a filter to sample or variant*

---

## Description

Set a filter to sample and/or variant

## Usage

```
seqSetFilter(gdsfile, sample.id=NULL, variant.id=NULL,
    samp.sel=NULL, variant.sel=NULL, action=c("set", "push", "pop"),
    verbose=TRUE)
```

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `sample.id` | IDs of selected samples |
| `variant.id` | IDs of selected variants |
| `samp.sel` | a logical vector indicating the selected samples |
| `variant.sel` | a logical vector indicating the selected variants |
| `action` | `"set"` – set the current filter; `"push"` – save the current filter and set the filter; `"pop"` – pop up the last filter |
| `verbose` | if TRUE, show information |

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

`seqGetFilter`, `seqGetData`, `seqApply`

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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")


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

---

# seqSlidingWindow *Apply functions via a sliding window over variants*

---

## Description

Returns a vector or list of values obtained by applying a function to a sliding window over variants

## Usage

```
seqSlidingWindow(gdsfile, var.name, win.size, shift=1, FUN,
    as.is = c("list", "integer", "double", "character", "none"),
    var.index = c("none", "relative", "absolute"), ...)
```

## Arguments

| | |
|---|---|
| gdsfile | a SeqVarGDSClass object |
| var.name | the variable name(s), see details |
| win.size | the size of sliding window |
| shift | the number of variants to shift the window at each step |
| FUN | the function to be applied |
| as.is | returned value: a list, an integer vector, etc |
| var.index | if "none", call FUN(x, ...) without variable index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index is an index of variant starting from 1: "relative" for indexing in the selection defined by seqSetFilter, "absolute" for indexing with respect to all data |
| ... | optional arguments to FUN |

## 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".

In the user-defined funciton FUN(x, ...) or FUN(index, x, ...), x is a list with win.size elements, and each element includes values for the variable(s) var.name; index is the starting position of the sliding window.

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

## Value

A vector or list of values.

## Author(s)

Xiuwen Zheng

## See Also

seqSetFilter, seqGetData, seqApply

## Examples

```
# the file of GDS
gds.fn <- seqExampleFileName("gds")
# or gds.fn <- "C:/YourFolder/Your_GDS_File.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[seq(2, 16, 2)],
variant.id=sample(variant.id, 10))


# apply a function via a sliding window over variants
seqSlidingWindow(f, c(qual="annotation/id"), win.size=3,
FUN = function(x) {
# x is a list with win.size elements
print(x)
}, as.is="none")

# apply a function via a sliding window over variants
seqSlidingWindow(f, c(qual="annotation/id"), win.size=3,
FUN = function(x) {
cat(unlist(x), sep="\t"); cat("\n")
}, as.is="none")


# apply a function via a sliding window over variants
seqSlidingWindow(f, c(geno="genotype", phase="phase", qual="annotation/id"),
FUN = function(index, x) {
cat("Window ", index, ":\n", sep="")
print(x)
},
win.size=3, as.is="none", var.index="relative")


# apply a function via a sliding window over variants
seqSlidingWindow(f, "genotype", win.size=4,
FUN = function(index, x) {
z <- unlist(lapply(x, function(z) mean(z, na.rm=TRUE)))
cat("Window ", index, ", starting from Variant ", index,
"\n    ", format(round(z,3), nsmall=3, width=8), "\n", sep="")
},
as.is="none", var.index="relative")


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

---

## seqSummary *Get the summary of a GDS file*

---

## Description

Get the summary of a sequencing GDS file

## Usage

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

## Arguments

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

## Details

If `check = "check"`, this function performs regular checking: dimensions of variables, etc. If `check = "full.check"`, it performs more checking: unique sample id, unique variant id, whether genotypic data are in a valid range or not, etc.

## Value

If `varname = NULL`, then return a list:

| | |
|---|---|
| filename | the file name |
| sequence.variant.format | the sequencing format in GDS |
| num.of.sample | the number of samples |
| num.of.variant | the number of variants |
| info | the description of the INFO field: var.name, number, type and description |
| format | the description of the FORMAT field: var.name, number, type and description |

If `varname = "genotype" or "phase"`, then return a list:

| | |
|---|---|
| dim | dim[1] – ploidy, dim[2] – the number of samples, dim[3] – the number of variants |
| seldim | seldim[1] – the number of selected samples, seldim[2] – the number of selected variants |

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqApply

## Examples

```
(gds.fn <- seqExampleFileName("gds"))
# or gds.fn <- "C:/YourFolder/Your_GDS_File.gds"

seqSummary(gds.fn)

seqSummary(gds.fn, "genotype")


######################################################
# display
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)
```

---

`seqTranspose` *Transpose Data Array*

---

## Description

Transpose data array or matrix for possibly higher-speed access

## Usage

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

## Arguments

| | |
|---|---|
| `gdsfile` | a `SeqVarGDSClass` object |
| `var.name` | the variable name with '/' as a separator |
| `compress` | the compression option used in `add.gdsn`; or determine automatically if `NULL` |
| `verbose` | if TRUE, show information |

## Details

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

## Value

None.

## Author(s)

Xiuwen Zheng

## See Also

seqGetData, seqApply

## Examples

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

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

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

seqTranspose(f, "genotype/data")
f

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

# close
seqClose(f)
unlink("tmp.gds")
```

---

SeqVarGDSClass *SeqVarGDSClass*

---

## Description

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

## Details

A sequencing 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.

filt(x): Returns the filter data.

qual(x): Returns the quality scores.

Other data can be accessed with seqGetData.

### Coercion methods

In the following code snippets x is a SeqVarGDSClass object.

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

### Author(s)

Xiuwen Zheng, Stephanie Gogarten

### See Also

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

### Examples

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

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

## variants
granges(gds)

## alleles as comma-separated character strings
head(seqGetData(gds, "allele"))

## alleles as DNAStringSet or DNAStringSetList
ref(gds)
alt(gds)

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

```
dim(geno)
## dimensions are: allele, sample, variant
geno[1,1:10,1:5]

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

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

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

seqClose(gds)
```

---

`seqVCF.Header` *Parse the header of a VCF file*

---

## Description

Parse the header of a VCF file

## Usage

```
seqVCF.Header(vcf.fn)
```

## Arguments

| | |
|---|---|
| `vcf.fn` | the file name of VCF |

## Details

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

## Value

Return a list (with class name "seqvcf.header.class"):

| | |
|---|---|
| `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 |
| header | the other header lines |
| num.ploidy | the number of ploidy, two for humans |

## Author(s)

Xiuwen Zheng

## References

http://www.1000genomes.org/wiki/Analysis/VariantCallFormat/vcf-variant-call-format-version-41

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

## See Also

seqVCF.SampID, seqVCF2GDS

## Examples

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

# get sample id
seqVCF.Header(vcf.fn)
```

---

# seqVCF.SampID *Get the sample IDs of a VCF file*

---

## Description

Return the sample IDs of a VCF file

## Usage

```
seqVCF.SampID(vcf.fn)
```

## Arguments

| | |
|---|---|
| vcf.fn | the file name of VCF |

### Author(s)

Xiuwen Zheng

### References

http://www.1000genomes.org/wiki/Analysis/VariantCallFormat/vcf-variant-call-format-version-41

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

### See Also

seqVCF.Header, seqVCF2GDS

### Examples

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

# get sample id
seqVCF.SampID(vcf.fn)
```

---

## seqVCF2GDS *Reformat VCF files*

---

### Description

Reformat Variant Call Format (VCF) files.

### Usage

```
seqVCF2GDS(vcf.fn, out.fn, header=NULL, genotype.var.name="GT",
compress.option=seqCompress.Option(),
info.import=NULL, fmt.import=NULL, raise.error=TRUE, verbose=TRUE)
```

### Arguments

| | |
|---|---|
| vcf.fn | the file name(s) of VCF format |
| out.fn | the file name of output GDS file |
| header | if NULL, header is set to be seqVCF.Header(vcf.fn) |
| genotype.var.name | the ID for genotypic data in the FORMAT column; "GT" by default, VCFv4.0 |
| compress.option | specify the compression options, by default seqCompress.Option |
| info.import | characters, the variable name(s) in the INFO field for import |

| | |
|---|---|
| fmt.import | characters, the variable name(s) in the FORMAT field for import |
| raise.error | TRUE: throw an error if numeric conversion fails; FALSE: get missing value if numeric conversion fails |
| verbose | if TRUE, show information |

## Details

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

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

If there are more than one files in vcf.fn, seqVCF2GDS will merge all dataset together once they all contain the same samples. It is useful to combine genetic data if VCF data are divided by chromosomes.

## Value

Return the file name with an absolute path.

## Author(s)

Xiuwen Zheng

## References

The variant call format and VCFtools. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R; 1000 Genomes Project Analysis Group. Bioinformatics. 2011 Aug 1;27(15):2156-8. Epub 2011 Jun 7.

http://corearray.sourceforge.net/

## See Also

seqVCF.Header, seqCompress.Option, seqGDS2VCF

## 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")

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

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

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



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

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

# Index