Package 'SeqArray'

October 9, 2013

Type Package

Title Big Data Management of Genome-wide Sequencing Variants

Version 1.0.0

Date 2013-03-14

Depends gdsfmt (>= 0.9.12)

Suggests parallel, snow

Author Xiuwen Zheng

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

Description Big data management of genome-wide variants using the CoreArray library, where 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

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 = FALSE, ...)
```

Arguments

gdsfile	a gds.class object in the gdsfmt package
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 TRUE, 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"; otherwise call FUN(x, ...)
...	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 <zhengx@u.washington.edu>

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=TRUE)
# 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 *Close a Sequencing GDS File*

Description

Close a Sequencing GDS file

Usage

```
seqClose(gdsfile)
```

Arguments

gdsfile	a gds.class object in the gdsfmt package

Details

It is strongly suggested to call seqClose instead of closefn.gds.

Value

Return an object of class gds.class.

Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

See Also

seqOpen

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)
```

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 <zhengx@u.washington.edu>

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 gds.class object in the gdsfmt package
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 <zhengx@u.washington.edu>

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 <zhengx@u.washington.edu>

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 gds.class object in the gdsfmt package
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
fmt.var	a list of variable names in the FORMAT field, or NULL for using all variables
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 <zhengx@u.washington.edu>

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 gds.class object in the gdsfmt package
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 <zhengx@u.washington.edu>

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 `gds.class` object in the gdsfmt package

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 <zhengx@u.washington.edu>

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)
```

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 <zhengx@u.washington.edu>

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 <zhengx@u.washington.edu>

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 `gds.class` object in the gdsfmt package
`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 <zhengx@u.washington.edu>

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, verbose=TRUE)
```

Arguments

`gdsfile`	a `gds.class` object in the gdsfmt package
`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
`verbose`	if TRUE, show information

Value

None.

Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

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)
```

`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 file name, or a `gds.class` object in the gdsfmt package
`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 <zhengx@u.washington.edu>

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)
```

`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 <zhengx@u.washington.edu>

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

```
(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 <zhengx@u.washington.edu>

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

```
(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(),
cvt.raise.err = 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`
`cvt.raise.err`	`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 <zhengx@u.washington.edu>

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 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"))

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

Index