Package 'gdsfmt'

October 9, 2015

Type Package

Title R Interface to CoreArray Genomic Data Structure (GDS) Files

Version 1.4.0

Date 2015-03-24

Depends R (>= 2.14.0)

Imports methods

Suggests parallel, RUnit, knitr, BiocGenerics

Author Xiuwen Zheng [aut, cre], Stephanie Gogarten [ctb], Jean-loup Gailly and Mark Adler [ctb] (for the included zlib sources), Yann Collet [ctb] (for the included LZ4 sources)

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

Description This package provides a high-level R interface to CoreArray Genomic Data Structure (GDS) data files, which are portable across platforms and include hierarchical structure to store multiple scalable array-oriented data sets with metadata information. It is suited for large-scale datasets, especially for data which are much larger than the available random-access memory. The gdsfmt package offers the efficient operations specifically designed for integers with less than 8 bits, since a single genetic/genomic variant, like single-nucleotide polymorphism (SNP), usually occupies fewer bits than a byte. Data compression and decompression are also supported with relatively efficient random access. It is allowed to read a GDS file in parallel with multiple R processes supported by the package parallel.

License LGPL-3

Copyright This package includes the sources of CoreArray C++ library written by Xiuwen Zheng (LGPL-3), zlib written by Jean-loup Gailly and Mark Adler (zlib license), and LZ4 written by Yann Collet (simplified BSD).

VignetteBuilder knitr

BugReports http://github.com/zhengxwen/gdsfmt/issues

URL http://corearray.sourceforge.net/,
http://github.com/zhengxwen/gdsfmt

biocViews Software, Infrastructure, DataImport

NeedsCompilation yes

R topics documented:

---

gdsfmt-package *R Interface to CoreArray Genomic Data Structure (GDS) files*

Description

This package provides a high-level R interface to CoreArray Genomic Data Structure (GDS) data files, which are portable across platforms and include hierarchical structure to store multiple scalable array-oriented data sets with metadata information. It is suited for large-scale datasets, especially for data which are much larger than the available random-access memory. The gdsfmt package offers the efficient operations specifically designed for integers with less than 8 bits, since a single genetic/genomic variant, such like single-nucleotide polymorphism, usually occupies fewer bits than a byte. It is also allowed to read a GDS file in parallel with multiple R processes supported by the parallel package.

Details

Package:	gdsfmt
Type:	R/Bioconductor Package
License:	LGPL version 3

R interface of CoreArray GDS is based on the CoreArray project initiated and developed from 2007 (http://corearray.sourceforge.net). The CoreArray project is to develop portable, scalable, bioinformatic data visualization and storage technologies.

R is the most popular statistical environment, but one not necessarily optimized for high performance or parallel computing which ease the burden of large-scale calculations. To support efficient data management in parallel for numerical genomic data, we developed the Genomic Data Structure (GDS) file format. gdsfmt provides fundamental functions to support accessing data in parallel, and allows future R packages to call these functions.

Webpage: http://corearray.sourceforge.net, http://github.com/zhengxwen/gdsfmt

Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

References

http://corearray.sourceforge.net, http://github.com/zhengxwen/gdsfmt

Xiuwen Zheng, David Levine, Jess Shen, Stephanie M. Gogarten, Cathy Laurie, Bruce S. Weir. A High-performance Computing Toolset for Relatedness and Principal Component Analysis of SNP Data. Bioinformatics 2012; doi: 10.1093/bioinformatics/bts606.

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")
L <- -2500:2499

# commom types
add.gdsn(f, "label", NULL)
add.gdsn(f, "int", val=1:10000, compress="ZIP", closezip=TRUE)
add.gdsn(f, "int.matrix", val=matrix(L, nrow=100, ncol=50))
add.gdsn(f, "mat", val=matrix(1:(10*6), nrow=10))
add.gdsn(f, "double", val=seq(1, 1000, 0.4))
add.gdsn(f, "character", val=c("int", "double", "logical", "factor"))
add.gdsn(f, "logical", val=rep(c(TRUE, FALSE, NA), 50))
add.gdsn(f, "factor", val=as.factor(c(letters, NA, "AA", "CC")))
add.gdsn(f, "NA", val=rep(NA, 10))
add.gdsn(f, "NaN", val=c(rep(NaN, 20), 1:20))
add.gdsn(f, "bit2-matrix", val=matrix(L[1:5000], nrow=50, ncol=100),
    storage="bit2")
# list and data.frame
add.gdsn(f, "list", val=list(X=1:10, Y=seq(1, 10, 0.25)))
add.gdsn(f, "data.frame", val=data.frame(X=1:19, Y=seq(1, 10, 0.5)))

# save a .RData object
obj <- list(X=1:10, Y=seq(1, 10, 0.1))
save(obj, file="tmp.RData")
addfile.gdsn(f, "tmp.RData", filename="tmp.RData")

f

read.gdsn(index.gdsn(f, "list"))
read.gdsn(index.gdsn(f, "list/Y"))
read.gdsn(index.gdsn(f, "data.frame"))
read.gdsn(index.gdsn(f, "mat"))

# Apply functions over columns of matrix
tmp <- apply.gdsn(index.gdsn(f, "mat"), margin=2, FUN=function(x) print(x))
tmp <- apply.gdsn(index.gdsn(f, "mat"), margin=2,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) print(x))

closefn.gds(f)


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

add.gdsn *Add a new GDS node*

Description

Add a new GDS node to the GDS file.

Usage

```
add.gdsn(node, name, val=NULL, storage=storage.mode(val), valdim=NULL,
    compress=c("", "ZIP", "ZIP_RA", "LZ4", "LZ4_RA"), closezip=FALSE,
    check=TRUE, replace=FALSE, visible=TRUE, ...)
```

Arguments

node	an object of class gdsn.class or gds.class: "gdsn.class" – the node of hierarchical structure; "gds.class" – the root of hieracrchical structure
name	the variable name of the added node; if it is not specified, "Item N" is assigned to name, where N is the number of child nodes plus one
val	the R value can be integers, real numbers, characters, factor, logical or raw variable, list and data.frame
storage	to specify data type (not case-sensitive): integer (signed: "int8", "int16", "int24", "int32", "int64", "sbit2", "sbit3", "sbit4", ..., "sbit32", "sbit64" ; unsigned: "uint8", "uint16", "uint24", "uint32", "uint64", "bit1", "bit2", "bit3", ..., "bit32", "bit64"); floating-point number ("float32", "float64"); packed real number ("packedreal8", "packedreal16", "packedreal32": pack a floating-point number to a 8/16/32-bit integer with two attributes "offset" and "scale", representing "int*scale + offset"); string (variable-length: "string", "string16", "string32" ; fixed-length: "fstring", "fstring16", "fstring32"). Or "char" (="int8"), "int"/"integer" (="int32"), "float" (="float32"), "double" (="float64"), "character" (="string"), "logical", "list", "factor", "folder"
valdim	the dimension attribute for the array to be created, which is a vector of length one or more giving the maximal indices in each dimension
compress	the compression method can be "" (no compression), "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max" or "ZIP.none" (original zlib); "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" (zlib with efficient random access); "LZ4", "LZ4.none", "LZ4.fast", "LZ4.hc" or "LZ4.max"; "LZ4_RA", "LZ4_RA.none", "LZ4_RA.fast", "LZ4_RA.hc" or "LZ4_RA.max" (with efficient random access). See details
closezip	if a compression method is specified, get into read mode after compression
check	if TRUE, a warning will be given when val is character and there are missing values in val. GDS format does not support missing characters NA, and any NA will be converted to a blank string ""
replace	if TRUE, replace the existing variable silently if possible
visible	FALSE – invisible/hidden, except print(, all=TRUE)
...	additional parameters for specific storage, see details

Details

`val`: if val is `list` or `data.frame`, the child node(s) will be added corresponding to objects in `list` or `data.frame`. If calling `add.gdsn(node, name, val=NULL)`, then a label will be added which does not have any other data except the name and attributes. If `val` is raw-type, it is interpreted as 8-bit signed integer.

`storage`: the default value is `storage.mode(val)`, "int" denotes signed integer, "uint" denotes unsigned integer, 8, 16, 24, 32 and 64 denote the number of bits. "bit1" to "bit32" denote the packed data types for 1 to 32 bits which are packed on disk, and "sbit2" to "sbit32" denote the corresponding signed integers. "float32" denotes single-precision number, and "float64" denotes double-precision number. "string" represents strings of 8-bit characters, "string16" represents strings of 16-bit characters following UTF16 industry standard, and "string32" represents a string of 32-bit characters following UTF32 industry standard. "folder" is to create a folder.

`valdim`: the values in data are taken to be those in the array with the leftmost subscript moving fastest. The last entry could be ZERO. If the total number of elements is zero, gdsfmt does not allocate storage space. `NA` is treated as 0.

`compress`: Z compression algorithm (http://www.zlib.net/) can be used to deflate the data stored in the GDS file. "ZIP" option is equivalent to "ZIP.default". "ZIP.fast", "ZIP.default" and "ZIP.max" correspond to different compression levels.

To support efficient random access of Z stream, "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" should be specified. "ZIP_RA" option is equivalent to "ZIP_RA.default:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

LZ4 fast lossless compression algorithm is allowed with `compress="LZ4"` (http://code.google.com/p/lz4/). Three compression levels can be specified, "LZ4.fast" (LZ4 fast mode), "LZ4.hc" (LZ4 high compression mode), "LZ4.max" (maximize the compression ratio). The block size can be specified by following colon, and "64K", "256K", "1M" and "4M" are allowed according to LZ4 frame format. "LZ4" is equivalent to "LZ4.hc:256K".

To support efficient random access of LZ4 stream, "LZ4_RA", "LZ4_RA.fast", "LZ4_RA.hc", "ZIP_RA.max" or "LZ4_RA.none" should be specified. "LZ4_RA" option is equivalent to "LZ4_RA.hc:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

To finish compressing, you should call `readmode.gdsn` to close the writing mode.

`closezip`: if compression option is specified, then enter a read mode after deflating the data. see `readmode.gdsn`.

`...`: if `storage = "fstring"`, `"fstring16"` or `"fstring32"`, users can set the max length of string in advance by `maxlen=`. If `storage = "packedreal8", "packedreal16"` or `"packedreal32"`, users can define `offset` and `scale` to represent real numbers by "value*scale + offset" where "value" is a 8/16/32-bit integer.

Value

An object of class `gdsn.class` of the new node.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

addfile.gdsn, addfolder.gdsn, index.gdsn, objdesp.gdsn, read.gdsn, readex.gdsn, write.gdsn, append.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")
L <- -2500:2499

##############################################################################
# commom types

add.gdsn(f, "label", NULL)
add.gdsn(f, "int", 1:10000, compress="ZIP", closezip=TRUE)
add.gdsn(f, "int.matrix", matrix(L, nrow=100, ncol=50))
add.gdsn(f, "double", seq(1, 1000, 0.4))
add.gdsn(f, "character", c("int", "double", "logical", "factor"))
add.gdsn(f, "logical", rep(c(TRUE, FALSE, NA), 50))
add.gdsn(f, "factor", as.factor(c(letters, NA, "AA", "CC")))
add.gdsn(f, "NA", rep(NA, 10))
add.gdsn(f, "NaN", c(rep(NaN, 20), 1:20))
add.gdsn(f, "bit2-matrix", matrix(L[1:5000], nrow=50, ncol=100),
    storage="bit2")
# list and data.frame
add.gdsn(f, "list", list(X=1:10, Y=seq(1, 10, 0.25)))
add.gdsn(f, "data.frame", data.frame(X=1:19, Y=seq(1, 10, 0.5)))


##############################################################################
# save a .RData object

obj <- list(X=1:10, Y=seq(1, 10, 0.1))
save(obj, file="tmp.RData")
addfile.gdsn(f, "tmp.RData", filename="tmp.RData")

f

read.gdsn(index.gdsn(f, "list"))
read.gdsn(index.gdsn(f, "list/Y"))
read.gdsn(index.gdsn(f, "data.frame"))


##############################################################################
```

```
# allocate the disk spaces

n1 <- add.gdsn(f, "n1", 1:100, valdim=c(10, 20))
read.gdsn(index.gdsn(f, "n1"))

n2 <- add.gdsn(f, "n2", matrix(1:100, 10, 10), valdim=c(15, 20))
read.gdsn(index.gdsn(f, "n2"))


#############################################################################
# replace variables

f

add.gdsn(f, "double", 1:100, storage="float", replace=TRUE)
f
read.gdsn(index.gdsn(f, "double"))


closefn.gds(f)


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

`addfile.gdsn` *Add a GDS node with a file*

Description

Add a file to a GDS file as a node.

Usage

```
addfile.gdsn(node, name, filename,
    compress=c("ZIP", "ZIP_RA", "LZ4", "LZ4_RA"), replace=FALSE, visible=TRUE)
```

Arguments

`node`	an object of class `gdsn.class` or `gds.class`
`name`	the variable name of the added node; if it is not specified, "`Item N`" is assigned to `name`, where `N` is the number of child nodes plus one
`filename`	the file name of input stream.
`compress`	the compression method can be "" (no compression), "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max" or "ZIP.none" (original zlib); "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" (zlib with efficient random access); "LZ4", "LZ4.none", "LZ4.fast", "LZ4.hc" or "LZ4.max"; "LZ4_RA", "LZ4_RA.none", "LZ4_RA.fast", "LZ4_RA.hc" or "LZ4_RA.max" (with efficient random access). See details

replace	if TRUE, replace the existing variable silently if possible
visible	FALSE – invisible/hidden, except print(, all=TRUE)

Details

compress: Z compression algorithm (http://www.zlib.net/) can be used to deflate the data stored in the GDS file. "ZIP" option is equivalent to "ZIP.default". "ZIP.fast", "ZIP.default" and "ZIP.max" correspond to different compression levels.

To support efficient random access of Z stream, "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" should be specified. "ZIP_RA" option is equivalent to "ZIP_RA.default:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

LZ4 fast lossless compression algorithm is allowed with compress="LZ4" (http://code.google.com/p/lz4/). Three compression levels can be specified, "LZ4.fast" (LZ4 fast mode), "LZ4.hc" (LZ4 high compression mode), "LZ4.max" (maximize the compression ratio). The block size can be specified by following colon, and "64K", "256K", "1M" and "4M" are allowed according to LZ4 frame format. "LZ4" is equivalent to "LZ4.hc:256K".

To support efficient random access of LZ4 stream, "LZ4_RA", "LZ4_RA.fast", "LZ4_RA.hc", "ZIP_RA.max" or "LZ4_RA.none" should be specified. "LZ4_RA" option is equivalent to "LZ4_RA.hc:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

Value

An object of class gdsn.class.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

getfile.gdsn, add.gdsn

Examples

```
# save a .RData object
obj <- list(X=1:10, Y=seq(1, 10, 0.1))
save(obj, file="tmp.RData")

# cteate a GDS file
f <- createfn.gds("test.gds")
```

```
add.gdsn(f, "double", val=seq(1, 1000, 0.4))
addfile.gdsn(f, "tmp.RData", "tmp.RData")

# open the GDS file
closefn.gds(f)


# open the existing file
(f <- openfn.gds("test.gds"))

getfile.gdsn(index.gdsn(f, "tmp.RData"), "tmp1.RData")
(obj <- get(load("tmp1.RData")))

# open the GDS file
closefn.gds(f)


# delete the temporary files
unlink(c("test.gds", "tmp.RData", "tmp1.RData"), force=TRUE)
```

`addfolder.gdsn` *Add a folder to the GDS node*

Description

Add a directory or a virtual folder to the GDS node.

Usage

```
addfolder.gdsn(node, name, type=c("directory", "virtual"), gds.fn="",
    replace=FALSE, visible=TRUE)
```

Arguments

`node`	an object of class `gdsn.class` or `gds.class`
`name`	the variable name of the added node; if it is not specified, "`Item N`" is assigned to `name`, where `N` is the number of child nodes plus one
`type`	"directory" (default) – create a directory of GDS node; "virtual" – create a virtual folder linking another GDS file by mapping all of the content to this virtual folder
`gds.fn`	the name of another GDS file; it is applicable only if `type="virtual"`
`replace`	if `TRUE`, replace the existing variable silently if possible
`visible`	`FALSE` – invisible/hidden, except `print(, all=TRUE)`

Value

An object of class `gdsn.class`.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

add.gdsn, addfile.gdsn

Examples

```
# create the first GDS file
f1 <- createfn.gds("test1.gds")

add.gdsn(f1, "NULL")
addfolder.gdsn(f1, "dir")
add.gdsn(f1, "int", 1:100)
f1

# open the GDS file
closefn.gds(f1)

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

# create the second GDS file
f2 <- createfn.gds("test2.gds")

add.gdsn(f2, "int", 101:200)

# link to the first file
addfolder.gdsn(f2, "virtual_folder", type="virtual", gds.fn="test1.gds")

f2

# open the GDS file
closefn.gds(f2)


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

# open the second file (writable)
(f <- openfn.gds("test2.gds", FALSE))
# +     [  ]
# |--+ int  { Int32 100 }
# |--+ virtual_folder   [ --> test1.gds ]
# |  |--+ NULL
# |  |--+ dir   [  ]
# |  |--+ int   { Int32 100 }

read.gdsn(index.gdsn(f, "int"))
```

```
read.gdsn(index.gdsn(f, "virtual_folder/int"))
add.gdsn(index.gdsn(f, "virtual_folder/dir"), "nm", 1:10)

f

# open the GDS file
closefn.gds(f)


#############################################
# open 'test1.gds', there is a new variable "dir/nm"

(f <- openfn.gds("test1.gds"))
closefn.gds(f)


#############################################
# remove 'test1.gds'

file.remove("test1.gds")

## Not run: (f <- openfn.gds("test2.gds"))
# +     [  ]
# |--+ int  { Int32 100 }
# |--+ virtual_folder   [ -X- test1.gds ]

## Not run: closefn.gds(f)


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

---

append.gdsn *Append data to a specified variable*

Description

Append new data to the data field of a GDS node.

Usage

```
append.gdsn(node, val, check=TRUE)
```

Arguments

node	an object of class gdsn.class
val	new data to be appended
check	whether a warning is given, when appended data can not match the capability of data field; if val is character-type, a warning will be shown if there is any NA in val

Details

`storage.mode(val)` should be "integer", "double", "character" or "logical". GDS format does not support missing characters `NA`, and any `NA` will be converted to a blank string `""`.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`read.gdsn`, `write.gdsn`, `add.gdsn`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# commom types
n <- add.gdsn(f, "int", val=matrix(1:10000, nrow=100, ncol=100),
    compress="ZIP")

# no warning, and add a new column
append.gdsn(n, -1:-100)
f

# a warning
append.gdsn(n, -1:-50)
f

# no warning here, and add a new column
append.gdsn(n, -51:-100)
f

# you should call "readmode.gdsn" before reading, since compress="ZIP"
readmode.gdsn(n)

# check the last column
read.gdsn(n, start=c(1, 102), count=c(-1, 1))


# characters
n <- add.gdsn(f, "string", val=as.character(1:100))
append.gdsn(n, as.character(rep(NA, 25)))
```

```
read.gdsn(n)


# close the gds file
closefn.gds(f)


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

---

apply.gdsn *Apply functions over margins*

Description

Return a vector or list of values obtained by applying a function to margins of a GDS matrix or array.

Usage

```
apply.gdsn(node, margin, FUN, selection=NULL, as.is=c("list", "none",
    "integer", "double", "character", "logical", "raw", "gdsnode"),
    var.index=c("none", "relative", "absolute"), target.node=NULL,
    .useraw=FALSE, ...)
```

Arguments

node	an object of class gdsn.class, or a list of objects of class gdsn.class
margin	an integer giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns
FUN	the function to be applied
selection	a list or NULL; if a list, it is a list of logical vectors according to dimensions indicating selection; if NULL, uses all data
as.is	returned value: a list, an integer vector, etc; "gdsnode" – the returned value from the user-defined function will be appended to target.node.
var.index	if "none", call FUN(x, ...) without an index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index in the function is an index starting from 1: "relative" for indexing in the selection defined by selection, "absolute" for indexing with respect to all data
target.node	NULL, an object of class gdsn.class or a list of gdsn.class: output to the target GDS node(s) when as.is="gdsnode". See details
.useraw	use R RAW storage mode if integers can be stored in a byte, to reduce memory usage
...	optional arguments to FUN

Details

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

When `as.is="gdsnode"` and there are more than one `gdsn.class` object in `target.node`, the user-defined function should return a list with elements corresponding to `target.node`, or `NULL` indicating no appending.

Value

A vector or list of values.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`read.gdsn`, `readex.gdsn`, `clusterApply.gdsn`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

(n <- add.gdsn(f, "matrix", val=matrix(1:(10*6), nrow=10)))
read.gdsn(index.gdsn(f, "matrix"))

(n1 <- add.gdsn(f, "string",
    val=matrix(paste("L", 1:(10*6), sep=","), nrow=10)))
read.gdsn(index.gdsn(f, "string"))

# Apply functions over rows of matrix
apply.gdsn(n, margin=1, FUN=function(x) print(x), as.is="none")
apply.gdsn(n, margin=1,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) print(x), as.is="none")
apply.gdsn(n, margin=1, var.index="relative",
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(i, x) { cat("index: ", i, ", ", sep=""); print(x) },
    as.is="none")
apply.gdsn(n, margin=1, var.index="absolute",
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(i, x) { cat("index: ", i, ", ", sep=""); print(x) },
    as.is="none")
apply.gdsn(n1, margin=1, FUN=function(x) print(x), as.is="none")

# Apply functions over columns of matrix
```

```
apply.gdsn(n, margin=2, FUN=function(x) print(x), as.is="none")
apply.gdsn(n, margin=2,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) print(x), as.is="none")
apply.gdsn(n1, margin=2,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) print(x), as.is="none")

# close
closefn.gds(f)



############################################################
#
# Append to a target GDS node
#

# cteate a GDS file
f <- createfn.gds("test.gds")

(n1 <- add.gdsn(f, "matrix", val=matrix(1:(10*6), nrow=10)))

(n2 <- add.gdsn(f, "string",
    val=matrix(paste("L", 1:(10*6), sep=","), nrow=10)))
read.gdsn(index.gdsn(f, "string"))

n1.1 <- add.gdsn(f, "transpose.matrix", storage="int", valdim=c(6,0))
n2.1 <- add.gdsn(f, "transpose.string", storage="string", valdim=c(6,0))

# Apply functions over rows of matrix
apply.gdsn(n1, margin=1, FUN=`c`, as.is="gdsnode", target.node=n1.1)

# matrix transpose
read.gdsn(n1)
read.gdsn(n1.1)


# Apply functions over rows of matrix
apply.gdsn(n2, margin=1, FUN=`c`, as.is="gdsnode", target.node=n2.1)

# matrix transpose
read.gdsn(n2)
read.gdsn(n2.1)

# close
closefn.gds(f)



############################################################
#
# Append to multiple target GDS node
```

```
#

# cteate a GDS file
f <- createfn.gds("test.gds")

(n1 <- add.gdsn(f, "matrix", val=matrix(1:(10*6), nrow=10)))

n1.1 <- add.gdsn(f, "transpose.matrix", storage="int", valdim=c(6,0))
n1.2 <- add.gdsn(f, "n.matrix", storage="int", valdim=c(0))

# Apply functions over rows of matrix
apply.gdsn(n1, margin=1, FUN=function(x) list(x, x[1]),
    as.is="gdsnode", target.node=list(n1.1, n1.2))

# matrix transpose
read.gdsn(n1)
read.gdsn(n1.1)
read.gdsn(n1.2)

# close
closefn.gds(f)



############################################################
#
# Multiple variables
#

# cteate a GDS file
f <- createfn.gds("test.gds")

X <- matrix(1:50, nrow=10)
Y <- matrix((1:50)/100, nrow=10)
Z1 <- factor(c(rep(c("ABC", "DEF", "ETD"), 3), "TTT"))
Z2 <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

node.X <- add.gdsn(f, "X", X)
node.Y <- add.gdsn(f, "Y", Y)
node.Z1 <- add.gdsn(f, "Z1", Z1)
node.Z2 <- add.gdsn(f, "Z2", Z2)

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z1), margin=c(1, 1, 1),
    FUN=function(x) { print(x) }, as.is="none")

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z2), margin=c(2, 2, 1),
    FUN=function(x) print(x))


# with selection

s1 <- rep(c(FALSE, TRUE), 5)
```

```
s2 <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z1), margin=c(1, 1, 1),
    selection = list(list(s1, s2), list(s1, s2), list(s1)),
    FUN=function(x) print(x))

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z2), margin=c(2, 2, 1),
    selection = list(list(s1, s2), list(s1, s2), list(s2)),
    FUN=function(x) print(x))


# close
closefn.gds(f)


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

`assign.gdsn` *Assign/append data to a GDS node*

Description

Assign data to a GDS node, or append data to a GDS node

Usage

```
assign.gdsn(dest.obj, src.obj, append=TRUE)
```

Arguments

`dest.obj`	an object of class `gdsn.class`, a destination GDS node
`src.obj`	an object of class `gdsn.class`, a source GDS node
`append`	if `TRUE`, append data; otherwise, replace the old one

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

read.gdsn, write.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

n1 <- add.gdsn(f, "int", val=matrix(1:10000, nrow=100, ncol=100),
    compress="ZIP")

n2 <- add.gdsn(f, "int2", storage="int")

## Not run:
assign.gdsn(n2, n1)

## End(Not run)

# close the gds file
closefn.gds(f)


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

---

cache.gdsn *Caching variable data*

Description

Caching the data associated with a GDS variable

Usage

```
cache.gdsn(node)
```

Arguments

node	an object of class gdsn.class, a GDS node

Details

If random access of array-based data is required, it is possible to speed up the access time by caching data in memory. This function tries to force the operating system to cache the data associated with the GDS node, however how to cache data depends on the configuration of operating system, including system memory and caching strategy. Note that this function does not explicitly allocate memory for the data.

If the data has been compressed, caching strategy almost has no effect on random access, since the data has to be decompressed serially.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

read.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

n <- add.gdsn(f, "int.matrix", matrix(1:50*100, nrow=100, ncol=50))
n

cache.gdsn(n)

# close the GDS file
closefn.gds(f)

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

---

cleanup.gds *Clean up fragments*

Description

Clean up the fragments of a CoreArray Genomic Data Structure (GDS) file.

Usage

```
cleanup.gds(filename, verbose=TRUE)
```

Arguments

filename	the file name of a GDS file to be opened
verbose	if TRUE, show information

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

openfn.gds, createfn.gds, closefn.gds

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# commom types
add.gdsn(f, "int", val=1:10000)
L <- -2500:2499
add.gdsn(f, "int.matrix", val=matrix(L, nrow=100, ncol=50))

# save a .RData object
obj <- list(X=1:10, Y=seq(1, 10, 0.1))
save(obj, file="tmp.RData")
addfile.gdsn(f, "tmp.RData", filename="tmp.RData")

f

closefn.gds(f)


# clean up fragments
cleanup.gds("test.gds")


# open ...
(f <- openfn.gds("test.gds"))
closefn.gds(f)


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

`closefn.gds` *Close a GDS file*

Description

Close a CoreArray Genomic Data Structure (GDS) file.

Usage

```
closefn.gds(gdsfile)
```

Arguments

`gdsfile`	an object of class `gds.class`, a GDS file

Details

For better performance, data in a GDS file are usually cached in memory. Keep in mind that the new file may not actually be written to disk, until `closefn.gds` or `sync.gds` is called. Anyway, when R shuts down, all GDS files created or opened would be automatically closed.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`createfn.gds`, `openfn.gds`, `sync.gds`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "int.matrix", matrix(1:50*100, nrow=100, ncol=50))

# close the GDS file
closefn.gds(f)

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

clusterApply.gdsn *Apply functions over matrix margins in parallel*

Description

Return a vector or list of values obtained by applying a function to margins of a GDS matrix in parallel.

Usage

```
clusterApply.gdsn(cl, gds.fn, node.name, margin, FUN, selection=NULL,
    as.is=c("list", "none", "integer", "double", "character", "logical", "raw"),
    var.index=c("none", "relative", "absolute"), .useraw=FALSE, ...)
```

Arguments

cl	a cluster object, created by this package or by the package parallel
gds.fn	the file name of a GDS file
node.name	a character vector indicating GDS node path
margin	an integer giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns
FUN	the function to be applied
selection	a list or NULL; if a list, it is a list of logical vectors according to dimensions indicating selection; if NULL, uses all data
as.is	returned value: a list, an integer vector, etc
var.index	if "none", call FUN(x, ...) without an index; if "relative" or "absolute", add an argument to the user-defined function FUN like FUN(index, x, ...) where index in the function is an index starting from 1: "relative" for indexing in the selection defined by selection, "absolute" for indexing with respect to all data
.useraw	use R RAW storage mode if integers can be stored in a byte, to reduce memory usage
...	optional arguments to FUN

Details

The algorithm of applying is 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

References

http://github.com/zhengxwen/gdsfmt

See Also

apply.gdsn

Examples

```
############################################################
# prepare a GDS file

# cteate a GDS file
f <- createfn.gds("test1.gds")

(n <- add.gdsn(f, "matrix", val=matrix(1:(10*6), nrow=10)))
read.gdsn(index.gdsn(f, "matrix"))

closefn.gds(f)


# cteate the GDS file "test2.gds"
(f <- createfn.gds("test2.gds"))

X <- matrix(1:50, nrow=10)
Y <- matrix((1:50)/100, nrow=10)
Z1 <- factor(c(rep(c("ABC", "DEF", "ETD"), 3), "TTT"))
Z2 <- c(TRUE, FALSE, TRUE, FALSE, TRUE)

node.X <- add.gdsn(f, "X", X)
node.Y <- add.gdsn(f, "Y", Y)
node.Z1 <- add.gdsn(f, "Z1", Z1)
node.Z2 <- add.gdsn(f, "Z2", Z2)
f

closefn.gds(f)



############################################################
# apply in parallel

library(parallel)

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

```
# Apply functions over rows or columns of matrix

clusterApply.gdsn(cl, "test1.gds", "matrix", margin=1, FUN=function(x) x)

clusterApply.gdsn(cl, "test1.gds", "matrix", margin=2, FUN=function(x) x)

clusterApply.gdsn(cl, "test1.gds", "matrix", margin=1,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) x)

clusterApply.gdsn(cl, "test1.gds", "matrix", margin=2,
    selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
    FUN=function(x) x)



# Apply functions over rows or columns of multiple data sets

clusterApply.gdsn(cl, "test2.gds", c("X", "Y", "Z1"), margin=c(1, 1, 1),
    FUN=function(x) x)

# with variable names
clusterApply.gdsn(cl, "test2.gds", c(X="X", Y="Y", Z="Z2"), margin=c(2, 2, 1),
    FUN=function(x) x)


# stop clusters
stopCluster(cl)


# delete the temporary file
unlink(c("test1.gds", "test2.gds"), force=TRUE)
```

`cnt.gdsn` *Return the number of child nodes*

Description

Return the number of child nodes for a GDS node.

Usage

```
cnt.gdsn(node)
```

Arguments

node	an object of class `gdsn.class`, a GDS node

Value

If node is a folder, return the numbers of variables in the folder including child folders. Otherwise, return 0.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

objdesp.gdsn, ls.gdsn, index.gdsn, delete.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T","B","C"), z=TRUE))
cnt.gdsn(node)
# 3

closefn.gds(f)


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

---

compression.gdsn *Modify compression mode*

Description

Modifie the compression mode of data field in the GDS node.

Usage

```
compression.gdsn(node, compress=c("", "ZIP", "ZIP_RA", "LZ4", "LZ4_RA"))
```

Arguments

node	an object of class gdsn.class, a GDS node
compress	the compression method can be "" (no compression), "ZIP", "ZIP.fast", "ZIP.default", "ZIP.max" or "ZIP.none" (original zlib); "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" (zlib with efficient random access); "LZ4", "LZ4.none", "LZ4.fast", "LZ4.hc" or "LZ4.max"; "LZ4_RA", "LZ4_RA.none", "LZ4_RA.fast", "LZ4_RA.hc" or "LZ4_RA.max" (with efficient random access). See details

Details

Z compression algorithm (http://www.zlib.net/) can be used to deflate the data stored in the GDS file. "ZIP" option is equivalent to "ZIP.default". "ZIP.fast", "ZIP.default" and "ZIP.max" correspond to different compression levels.

To support efficient random access of Z stream, "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.default", "ZIP_RA.max" or "ZIP_RA.none" should be specified. "ZIP_RA" option is equivalent to "ZIP_RA.default:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

LZ4 fast lossless compression algorithm is allowed with compress="LZ4" (http://code.google.com/p/lz4/). Three compression levels can be specified, "LZ4.fast" (LZ4 fast mode), "LZ4.hc" (LZ4 high compression mode), "LZ4.max" (maximize the compression ratio). The block size can be specified by following colon, and "64K", "256K", "1M" and "4M" are allowed according to LZ4 frame format. "LZ4" is equivalent to "LZ4.hc:256K".

To support efficient random access of LZ4 stream, "LZ4_RA", "LZ4_RA.fast", "LZ4_RA.hc", "ZIP_RA.max" or "LZ4_RA.none" should be specified. "LZ4_RA" option is equivalent to "LZ4_RA.hc:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K" and "1M" are allowed, such like "ZIP_RA:16K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, such like 64K.

Value

Return node.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt, http://zlib.net/

See Also

readmode.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

n <- add.gdsn(f, "int.matrix", matrix(1:50*100, nrow=100, ncol=50))
n

compression.gdsn(n, "ZIP")

# close the GDS file
closefn.gds(f)

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

`createfn.gds` *Create a GDS file*

Description

Create a new CoreArray Genomic Data Structure (GDS) file.

Usage

```
createfn.gds(filename, allow.duplicate=FALSE)
```

Arguments

`filename`	the file name of a new GDS file to be created
`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

Keep in mind that the new file may not actually be written to disk until `closefn.gds` or `sync.gds` is called.

Value

Return an object of class `gds.class`:

`filename`	the file name to be created
`id`	internal file id
`root`	an object of class `gdsn.class`, the root of hierachical structure
`readonly`	whether it is read-only or not

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

openfn.gds, closefn.gds

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, val=list(x=c(1,2), y=c("T", "B", "C"), z=TRUE))

f

closefn.gds(f)


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

---

delete.attr.gdsn *Delete an attribute*

Description

Remove an attribute of a GDS node.

Usage

```
delete.attr.gdsn(node, name)
```

Arguments

node	an object of class gdsn.class, a GDS node
name	the name of an attribute

Details

If there is not an attribute with name, a warning will be given.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

put.attr.gdsn, get.attr.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

node <- add.gdsn(f, "int", val=1:10000)
put.attr.gdsn(node, "missing.value", 10000)
put.attr.gdsn(node, "one.value", 1L)
put.attr.gdsn(node, "string", c("ABCDEF", "THIS"))
put.attr.gdsn(node, "bool", c(TRUE, TRUE, FALSE))

f
get.attr.gdsn(node)

delete.attr.gdsn(node, "one.value")
get.attr.gdsn(node)

closefn.gds(f)


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

---

delete.gdsn *Delete a GDS node*

Description

Delete a specified GDS node.

Usage

```
delete.gdsn(node, force=FALSE)
```

Arguments

node	an object of class gdsn.class, a GDS node
force	if FALSE, it is not allowed to delete a non-empty folder

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T", "B", "C"), z=TRUE))
f

## Not run:
# delete "node", but an error occurs
delete.gdsn(node)

## End(Not run)

# delete "node"
delete.gdsn(node, TRUE)

# close the GDS file
closefn.gds(f)


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

`diagnosis.gds` *Diagnose the GDS file*

Description

Diagnose the GDS file and block information

Usage

```
diagnosis.gds(gdsfile)
```

Arguments

`gdsfile`	An object of class `gds.class`, a GDS file

Value

A list with stream and chunk information.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`createfn.gds`, `openfn.gds`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")
L <- -2500:2499

# commom types

add.gdsn(f, "label", NULL)
add.gdsn(f, "int", 1:10000, compress="ZIP", closezip=TRUE)
add.gdsn(f, "int.matrix", matrix(L, nrow=100, ncol=50))

closefn.gds(f)


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

f <- openfn.gds("test.gds", FALSE)
```

```
diagnosis.gds(f)

delete.gdsn(index.gdsn(f, "int"))

diagnosis.gds(f)

closefn.gds(f)


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

cleanup.gds("test.gds")

f <- openfn.gds("test.gds", FALSE)
diagnosis.gds(f)
closefn.gds(f)


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

`gds.class` *the class of GDS file*

Description

The class of a CoreArray Genomic Data Structure (GDS) file.

Value

There are three components:

`filename`	the file name to be created
`id`	internal file id, an integer
`root`	an object of class `gdsn.class`, the root of hierachical structure
`readonly`	whether it is read-only or not

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`createfn.gds`, `openfn.gds`, `closefn.gds`

gdsn.class *the class of variable node in the GDS file*

Description

The class of variable node in the GDS file.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

add.gdsn, read.gdsn, write.gdsn

get.attr.gdsn *Get attributes*

Description

Get the attributes of a GDS node.

Usage

```
get.attr.gdsn(node)
```

Arguments

node	an object of class gdsn.class, a GDS node

Value

A list of attributes.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

put.attr.gdsn, delete.attr.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

node <- add.gdsn(f, "int", val=1:10000)
put.attr.gdsn(node, "missing.value", 10000)

f
get.attr.gdsn(node)

# close the GDS file
closefn.gds(f)


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

---

getfile.gdsn *Output a file from a stream container*

Description

Get a file from a GDS node of stream container.

Usage

```
getfile.gdsn(node, out.filename)
```

Arguments

node	an object of class gdsn.class, a GDS node
out.filename	the file name of output stream

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

addfile.gdsn

Examples

```
# save a .RData object
obj <- list(X=1:10, Y=seq(1, 10, 0.1))
save(obj, file="tmp.RData")

# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "double", val=seq(1, 1000, 0.4))
addfile.gdsn(f, "tmp.RData", "tmp.RData")

# open the GDS file
closefn.gds(f)


# open the existing file
(f <- openfn.gds("test.gds"))

getfile.gdsn(index.gdsn(f, "tmp.RData"), "tmp1.RData")
(obj <- get(load("tmp1.RData")))

# open the GDS file
closefn.gds(f)


# delete the temporary files
unlink(c("test.gds", "tmp.RData", "tmp1.RData"), force=TRUE)
```

---

index.gdsn *Return the specified node*

Description

Return a specified GDS node.

Usage

```
index.gdsn(node, path=NULL, index=NULL, silent=FALSE)
```

Arguments

node	an object of class gdsn.class (a GDS node), or gds.class (a GDS file)
path	the path specifying a GDS node with '/' as a separator
index	a numeric vector or characters, specifying the path; it is applicable if path=NULL
silent	if TRUE, return NULL if the specified node does not exist

Details

If `index` is a numeric vector, e.g., `c(1, 2)`, the result is the second child node of the first child of node. If `index` is a vector of characters, e.g., `c("list", "x")`, the result is the child node with name `"x"` of the `"list"` child node.

Value

An object of class `gdsn.class` for the specified node.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`cnt.gdsn`, `ls.gdsn`, `name.gdsn`, `add.gdsn`, `delete.gdsn`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T","B","C"), z=TRUE))
f

index.gdsn(f, "list/x")
index.gdsn(f, index=c("list", "x"))
index.gdsn(f, index=c(1, 1))
index.gdsn(f, index=c("list", "z"))

## Not run:
# stop here, return an error
index.gdsn(f, "list/x/z")

## End(Not run)

# return NULL
index.gdsn(f, "list/x/z", silent=TRUE)

# close the file
closefn.gds(f)


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

`is.element.gdsn` *whether the elements are in a set*

Description

Determine whether the elements are in a specified set.

Usage

```
is.element.gdsn(node, set)
```

Arguments

`node`	an object of class `gdsn.class` (a GDS node)
`set`	the specified set of elements

Value

A logical vector or array.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

`read.gdsn`

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "int", val=1:100)
add.gdsn(f, "mat", val=matrix(1:12, nrow=4, ncol=3))
add.gdsn(f, "double", val=seq(1, 10, 0.1))
add.gdsn(f, "character", val=c("int", "double", "logical", "factor"))

is.element.gdsn(index.gdsn(f, "int"), c(1, 10, 20))
is.element.gdsn(index.gdsn(f, "mat"), c(2, 8, 12))
is.element.gdsn(index.gdsn(f, "double"), c(1.1, 1.3, 1.5))
is.element.gdsn(index.gdsn(f, "character"), c("int", "factor"))

# close the GDS file
closefn.gds(f)
```

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

`lasterr.gds` *Return the last error message*

Description

Get the last error message and clear the error message(s) in the gdsfmt package.

Usage

```
lasterr.gds()
```

Value

Character.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

Examples

```
lasterr.gds()
```

`ls.gdsn` *Return the names of child nodes*

Description

Get a list of names for its child nodes.

Usage

```
ls.gdsn(node)
```

Arguments

node	an object of class `gdsn.class`, a GDS node

Value

A vector of characters.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

cnt.gdsn, objdesp.gdsn, ls.gdsn, index.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T","B","C"), z=TRUE))

ls.gdsn(node)

# close the GDS file
closefn.gds(f)


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

`moveto.gdsn` *Relocate a GDS node*

Description

Move a GDS node to a new place in the same file

Usage

```
moveto.gdsn(node, loc.node, relpos=c("after", "before", "replace"))
```

Arguments

`node`	an object of class `gdsn.class` (a GDS node)
`loc.node`	an object of class `gdsn.class` (a GDS node), indicates the new location
`relpos`	`"after"`: after `loc.node`, `"before"`: before `loc.node`, `"replace"`: replace `loc.node` (`loc.node` will be deleted and node has a new name as `loc.node`)

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

createfn.gds, openfn.gds, index.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")
L <- -2500:2499

# commom types

add.gdsn(f, "label", NULL)
add.gdsn(f, "int", 1:10000, compress="ZIP", closezip=TRUE)
add.gdsn(f, "int.matrix", matrix(L, nrow=100, ncol=50))
add.gdsn(f, "double", seq(1, 1000, 0.4))
add.gdsn(f, "character", c("int", "double", "logical", "factor"))

f
# +     [  ]
# |--+ label
# |--+ int  { Int32 10000 ZIP(34.74%) }
# |--+ int.matrix   { Int32 100x50 }
# |--+ double   { Float64 2498 }
# |--+ character    { VStr8 4 }

n1 <- index.gdsn(f, "label")
n2 <- index.gdsn(f, "double")

moveto.gdsn(n1, n2, relpos="after")
f

moveto.gdsn(n1, n2, relpos="before")
f

moveto.gdsn(n1, n2, relpos="replace")
f

# close the GDS file
closefn.gds(f)
```

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

---

name.gdsn *Return the variable name of a node*

Description

Get the variable name of a GDS node.

Usage

```
name.gdsn(node, fullname=FALSE)
```

Arguments

node	an object of class gdsn.class, a GDS node
fullname	if FALSE, return the node name (by default); otherwise the name with a full path

Value

Characters.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

cnt.gdsn, objdesp.gdsn, ls.gdsn, rename.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T","B","C"), z=TRUE))
node <- index.gdsn(f, "list/x")

name.gdsn(node)
# "x"

name.gdsn(node, fullname=TRUE)
```

```
# "list/x"

closefn.gds(f)


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

---

objdesp.gdsn *Variable description*

Description

Get the descritpion of a GDS node.

Usage

```
objdesp.gdsn(node)
```

Arguments

node	an object of class gdsn.class, a GDS node

Value

Returns a list:

name	the variable name of a specified node
fullname	the full name of a specified node
storage	the storage mode in the GDS file
trait	the description of data field, like "Int8"
type	a factor indicating the storage mode in R: Label – a label node, Folder – a directory, VFolder – a virtual folder linking to another GDS file, Raw – raw data (addfile.gdsn), Integer – integers, Factor – factor values, Logical – logical values (FALSE, TRUE and NA), Real – floating numbers, String – characters, Unknown – unknown type
is.array	indicates whether it is array-type
dim	the dimension of data field
encoder	encoder for compressed data, such like "ZIP"
compress	the compression method: "", "ZIP.max", etc
cpratio	data compression ratio, NaN indicates no compression
size	the size of data stored in the GDS file
good	logical, indicates the state of GDS file, e.g., FALSE if the virtual folder fails to link the target GDS file
message	if applicable, messages of the GDS node, such like error messages, log information
param	the parameters, used in add.gdsn, like "maxlen", "offset", "scale"

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

cnt.gdsn, name.gdsn, ls.gdsn, index.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a vector to "test.gds"
node1 <- add.gdsn(f, name="vector1", val=1:10000)
objdesp.gdsn(node1)

# add a vector to "test.gds"
node2 <- add.gdsn(f, name="vector2", val=1:10000, compress="ZIP.max",
    closezip=FALSE)
objdesp.gdsn(node2)

# add a character to "test.gds"
node3 <- add.gdsn(f, name="vector3", val=c("A", "BC", "DEF"),
    compress="ZIP", closezip=TRUE)
objdesp.gdsn(node3)

# close the GDS file
closefn.gds(f)


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

---

openfn.gds *Open a GDS file*

Description

Open an existing file of CoreArray Genomic Data Structure (GDS) for reading or writing.

Usage

```
openfn.gds(filename, readonly=TRUE, allow.duplicate=FALSE, allow.fork=FALSE)
```

Arguments

filename	the file name of a GDS file to be opened
readonly	if TRUE, the file is opened read-only; otherwise, it is allowed to write data to the file
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
allow.fork	TRUE for parallel environment using forking, see details

Details

This function opens an existing GDS file for reading (or, if readonly=FALSE, for writing). To create a new GDS file, use createfn.gds instead.

If the file is opened read-only, all data in the file are not allowed to be changed, including hierachical structure, variable names, data fields, etc.

mclapply and mcmapply in the R package parallel rely on unix forking. However, the forked child process inherits copies of the parent's set of open file descriptors. Each file descriptor in the child refers to the same open file description as the corresponding file descriptor in the parent. This means that the two descriptors share open file status flags, current file offset, and signal-driven I/O attributes. The sharing of file description can cause a serious problem (wrong reading, even program crashes), when child processes read or write the same GDS file simultaneously. allow.fork=TRUE adds additional file operations to avoid any conflict using forking. The current implementation does not support writing in forked processes.

Value

Return an object of class gds.class.

filename	the file name to be created
id	internal file id, an integer
root	an object of class gdsn.class, the root of hierachical structure
readonly	whether it is read-only or not

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

createfn.gds, closefn.gds

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# add a list to "test.gds"
node <- add.gdsn(f, name="list", val=list(x=c(1,2), y=c("T","B","C"), z=TRUE))
# close
closefn.gds(f)

# open the same file
f <- openfn.gds("test.gds")

# read
(node <- index.gdsn(f, "list"))
read.gdsn(node)

# close
closefn.gds(f)


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

---

print.gds.class *Show the information of class "gds.class"*

Description

Displays an object of class "gds.class", a GDS file.

Usage

```
## S3 method for class 'gds.class'
print(x, all=FALSE, ...)
```

Arguments

x	an object of class gds.class, a GDS file
all	if FALSE, hide GDS nodes with an attribute "R.invisible"
...	the arguments passed to or from other methods

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

print.gdsn.class

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "int.matrix", matrix(1:50*100, nrow=100, ncol=50))

print(f, all=TRUE)

# close the GDS file
closefn.gds(f)

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

---

print.gdsn.class *Show the information of class "gdsn.class"*

Description

Display an object of class "gdsn.class", a GDS node.

Usage

```
## S3 method for class 'gdsn.class'
print(x, expand=TRUE, all=FALSE, ...)
```

Arguments

x	an object of class gdsn.class, a GDS node
expand	whether enumerate all of child nodes
all	if FALSE, hide GDS nodes with an attribute "R.invisible"
...	the arguments passed to or from other methods

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

print.gds.class

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

n <- add.gdsn(f, "int.matrix", matrix(1:50*100, nrow=100, ncol=50))

print(n, all=TRUE)

compression.gdsn(n, "ZIP")

# close the GDS file
closefn.gds(f)

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

---

put.attr.gdsn *Add an attribute into a GDS node*

Description

Add an attribute to a GDS node.

Usage

```
put.attr.gdsn(node, name, val=NULL)
```

Arguments

node	an object of class gdsn.class, a GDS node
name	the name of an attribute
val	the value of an attribute

Details

Missing values are allowed in a numerical attribute, but not allowed for characters or logical values. Missing characters are converted to "NA", and missing logical values are converted to FALSE.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

get.attr.gdsn, delete.attr.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

node <- add.gdsn(f, "int", val=1:10000)
put.attr.gdsn(node, "missing.value", 10000)
put.attr.gdsn(node, "one.value", 1L)
put.attr.gdsn(node, "string", c("ABCDEF", "THIS"))
put.attr.gdsn(node, "bool", c(TRUE, TRUE, FALSE))

f
get.attr.gdsn(node)

delete.attr.gdsn(node, "one.value")
get.attr.gdsn(node)

closefn.gds(f)


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

---

read.gdsn *Read data field of a GDS node*

Description

Get data from a GDS node.

Usage

```
read.gdsn(node, start=NULL, count=NULL, simplify=c("auto", "none", "force"),
    .useraw=FALSE)
```

Arguments

node	an object of class gdsn.class, a GDS node
start	a vector of integers, starting from 1 for each dimension component
count	a vector of integers, the length of each dimnension. As a special case, the value "-1" indicates that all entries along that dimension should be written
simplify	if "auto", the result is collapsed to be a vector if possible; "force", the result is forced to be a vector
.useraw	use R RAW storage mode if integers can be stored in a byte, to reduce memory usage

Details

start, count: the values in data are taken to be those in the array with the leftmost subscript moving fastest.

Value

Return an array, list, or data.frame.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

readex.gdsn, append.gdsn, write.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "vector", 1:128)
add.gdsn(f, "list", list(X=1:10, Y=seq(1, 10, 0.25)))
add.gdsn(f, "data.frame", data.frame(X=1:19, Y=seq(1, 10, 0.5)))
add.gdsn(f, "matrix", matrix(1:12, ncol=4))

f

read.gdsn(index.gdsn(f, "vector"))
read.gdsn(index.gdsn(f, "list"))
read.gdsn(index.gdsn(f, "data.frame"))


# the effects of 'simplify'
```

```
read.gdsn(index.gdsn(f, "matrix"), start=c(2,2), count=c(-1,1))
# [1] 5 6  <- a vector

read.gdsn(index.gdsn(f, "matrix"), start=c(2,2), count=c(-1,1),
    simplify="none")
#      [,1]  <- a matrix
# [1,]    5
# [2,]    6


closefn.gds(f)


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

`readex.gdsn` *Read data field of a GDS node with a selection*

Description

Get data from a GDS node with subset selection.

Usage

```
readex.gdsn(node, sel=NULL, simplify=c("auto", "none", "force"), .useraw=FALSE)
```

Arguments

`node`	an object of class `gdsn.class`, a GDS node
`sel`	a list of `m` logical vectors, where `m` is the number of dimensions of `node` and each logical vector should have the same size of dimension in `node`
`simplify`	if `"auto"`, the result is collapsed to be a vector if possible; `"force"`, the result is forced to be a vector
`.useraw`	use R RAW storage mode if integers can be stored in a byte, to reduce memory usage

Value

Return an array.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

read.gdsn, append.gdsn, write.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "vector", 1:128)
add.gdsn(f, "matrix", matrix(as.character(1:(10*6)), nrow=10))
f

# read vector
readex.gdsn(index.gdsn(f, "vector"), sel=rep(c(TRUE, FALSE), 64))

# read matrix
readex.gdsn(index.gdsn(f, "matrix"))
readex.gdsn(index.gdsn(f, "matrix"),
    sel=list(d1=rep(c(TRUE, FALSE), 5), d2=rep(c(TRUE, FALSE), 3)))

# close the GDS file
closefn.gds(f)


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

---

readmode.gdsn *Switch to read mode in the compression settings*

Description

Switch to read mode for a GDS node with respect to its compression settings.

Usage

```
readmode.gdsn(node)
```

Arguments

node	an object of class gdsn.class, a GDS node

Details

After the compressed data field is created, it is in writing mode. Users can add new data to the compressed data field, but can not read data from the data field. Users have to call readmode.gdsn to finish writing, before reading any data from the compressed data field.

Once switch to the read mode, users can not add more data to the data field. If users would like to append more data or modify the data field, please call compression.gdsn(node, compress="") to decompress data first.

Value

Return node.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

compression.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

# commom types
n <- add.gdsn(f, "int", val=1:100, compress="ZIP")

# you can not read the variable "int" because of writing mode
# read.gdsn(n)

readmode.gdsn(n)

# now you can read "int"
read.gdsn(n)

closefn.gds(f)


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

---

rename.gdsn *Rename a GDS node*

Description

Rename a GDS node.

Usage

```
rename.gdsn(node, newname)
```

Arguments

node	an object of class gdsn.class, a GDS node
newname	the new name of a specified node

Details

CoreArray hierarchical structure does not allow duplicate names in the same folder.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

name.gdsn, ls.gdsn, index.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")
n <- add.gdsn(f, "old.name", val=1:10)
f

rename.gdsn(n, "new.name")
f

# close the GDS file
closefn.gds(f)


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

setdim.gdsn *Set the dimension of data field*

Description

Assign new dimensions to the data field of a GDS node.

Usage

```
setdim.gdsn(node, valdim, permute=FALSE)
```

Arguments

node	an object of class gdsn.class, a GDS node
valdim	the new dimension(s) for the array to be created, which is a vector of length one or more giving the maximal indices in each dimension. The values in data are taken to be those in the array with the leftmost subscript moving fastest. The last entry could be ZERO. If the total number of elements is zero, gdsfmt does not allocate storage space. NA is treated as 0.
permute	if TRUE, the elements are rearranged to preserve their relative positions in each dimension of the array

Value

Returns node.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

read.gdsn, write.gdsn, add.gdsn, append.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

n <- add.gdsn(f, "int", val=1:24)
read.gdsn(n)

setdim.gdsn(n, c(6, 4))
read.gdsn(n)
```

```
setdim.gdsn(n, c(8, 5), permute=TRUE)
read.gdsn(n)

# close the GDS file
closefn.gds(f)


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

---

showfile.gds *Enumerate opened GDS files*

Description

Enumerate all opened GDS files

Usage

```
showfile.gds(closeall=FALSE, verbose=TRUE)
```

Arguments

closeall	if TRUE, close all GDS files
verbose	if TRUE, show information

Value

A list of gds.class objects.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

add.gdsn(f, "int", val=1:10000)

showfile.gds()

# close the GDS file
```

```
closefn.gds(f)


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

---

sync.gds *Synchronize a GDS file*

Description

Write the data cached in memory to disk.

Usage

```
sync.gds(gdsfile)
```

Arguments

gdsfile	An object of class gds.class, a GDS file

Details

For better performance, Data in a GDS file are usually cached in memory. Keep in mind that the new file may not actually be written to disk, until closefn.gds or sync.gds is called. Anyway, when R shuts down, all GDS files created or opened would be automatically closed.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

createfn.gds, openfn.gds

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

node <- add.gdsn(f, "int", val=1:10000)
put.attr.gdsn(node, "missing.value", 10000)

sync.gds(f)

f
get.attr.gdsn(node)

# close the GDS file
closefn.gds(f)


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

`system.gds` *Get the parameters in the GDS system*

Description

Get a list of parameters in the GDS system

Usage

```
system.gds()
```

Value

A list including

`num.logical.core` the number of logical cores

`l1i.cache.size` L1 instruction cache

`l1d.cache.size` L1 data cache

`l2.cache.size` L2 data cache

`l3.cache.size` L3 data cache

`l4.cache.size` L4 data cache

`compression.encoder` compression/decompression algorithms

`class.list` class list in the GDS system

`compiler.flag` SIMD instructions supported by the compiler

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

Examples

```
system.gds()
```

`write.gdsn` *Write data to a GDS node*

Description

Write data to a GDS node.

Usage

```
write.gdsn(node, val, start=NULL, count=NULL, check=TRUE)
```

Arguments

`node`	an object of class `gdsn.class`, a GDS node
`val`	the data to be written
`start`	a vector of integers, starting from 1 for each dimension
`count`	a vector of integers, the length of each dimnension
`check`	if `TRUE`, a warning will be given when `val` is character and there are missing values in `val`

Details

`start`, `count`: The values in data are taken to be those in the array with the leftmost subscript moving fastest.

`start` and `count` should both exist or be missing. If `start` and `count` are both missing, the dimensions and values of `val` will be assigned to the data field.

GDS format does not support missing characters `NA`, and any `NA` will be converted to a blank string `""`.

Value

None.

Author(s)

Xiuwen Zheng

References

http://github.com/zhengxwen/gdsfmt

See Also

append.gdsn, read.gdsn, add.gdsn

Examples

```
# cteate a GDS file
f <- createfn.gds("test.gds")

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

n <- add.gdsn(f, "matrix", matrix(1:20, ncol=5))
read.gdsn(n)

write.gdsn(n, val=c(NA, NA), start=c(2, 2), count=c(2, 1))
read.gdsn(n)


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

n <- add.gdsn(f, "n", val=1:12)
read.gdsn(n)

write.gdsn(n, matrix(1:24, ncol=6))
read.gdsn(n)

write.gdsn(n, array(1:24, c(4,3,2)))
read.gdsn(n)


# close the file
closefn.gds(f)


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

Index