

Package ‘gdsfmt’

January 19, 2025

Type Package

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

Version 1.42.1

Date 2024-12-25

Depends R (>= 2.15.0), methods

Suggests parallel, digest, Matrix, crayon, RUnit, knitr, markdown,
rmarkdown, BiocGenerics

Author Xiuwen Zheng [aut, cre] (<<https://orcid.org/0000-0002-1390-0708>>),
Stephanie Gogarten [ctb],
Jean-loup Gailly and Mark Adler [ctb] (for the included zlib sources),
Yann Collet [ctb] (for the included LZ4 sources),
xz contributors [ctb] (for the included liblzma sources)

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

Description Provides a high-level R interface to CoreArray Genomic Data Structure (GDS) data files. GDS is portable across platforms with 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 of less than 8 bits, since a diploid genotype, like single-nucleotide polymorphism (SNP), usually occupies fewer bits than a byte. Data compression and decompression are available with relatively efficient random access. It is also 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), LZ4 written by Yann Collet (simplified BSD), and liblzma written by Lasse Collin and other xz contributors (public domain).

VignetteBuilder knitr

ByteCompile TRUE

BugReports <https://github.com/zhengxwen/gdsfmt/issues>

URL <https://github.com/zhengxwen/gdsfmt>

biocViews Infrastructure, DataImport
git_url <https://git.bioconductor.org/packages/gdsfmt>
git_branch RELEASE_3_20
git_last_commit a1256b8
git_last_commit_date 2024-12-26
Repository Bioconductor 3.20
Date/Publication 2025-01-19

Contents

| | |
|-----------------------------|----|
| gdsfmt-package | 3 |
| add.gdsn | 5 |
| addfile.gdsn | 8 |
| addfolder.gdsn | 10 |
| append.gdsn | 12 |
| apply.gdsn | 14 |
| assign.gdsn | 18 |
| cache.gdsn | 19 |
| cleanup.gds | 21 |
| closefn.gds | 22 |
| clusterApply.gdsn | 23 |
| cnt.gdsn | 25 |
| compression.gdsn | 26 |
| copyto.gdsn | 28 |
| createfn.gds | 29 |
| delete.attr.gdsn | 30 |
| delete.gdsn | 31 |
| diagnosis.gds | 32 |
| digest.gdsn | 34 |
| exist.gdsn | 35 |
| gds.class | 36 |
| gdsn.class | 37 |
| get.attr.gdsn | 37 |
| getfile.gdsn | 38 |
| getfolder.gdsn | 39 |
| index.gdsn | 40 |
| is.element.gdsn | 41 |
| is.sparse.gdsn | 42 |
| lasterr.gds | 43 |
| ls.gdsn | 44 |
| moveto.gdsn | 45 |
| name.gdsn | 46 |
| objdsp.gdsn | 47 |
| openfn.gds | 49 |
| permdim.gdsn | 50 |
| print.gds.class | 51 |
| put.attr.gdsn | 53 |
| read.gdsn | 54 |
| readex.gdsn | 56 |
| readmode.gdsn | 57 |

| | |
|--------------------------|----|
| rename.gdsn | 59 |
| setdim.gdsn | 60 |
| showfile.gds | 61 |
| summarize.gdsn | 62 |
| sync.gds | 63 |
| system.gds | 64 |
| unload.gdsn | 65 |
| write.gdsn | 66 |

| | |
|--------------|-----------|
| Index | 68 |
|--------------|-----------|

| | |
|----------------|--|
| gdsfmt-package | <i>R Interface to CoreArray Genomic Data Structure (GDS) files</i> |
|----------------|--|

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>, or <https://github.com/zhengxwen/gdsfmt>

Copyright notice: The 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).

Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

References

<http://corearray.sourceforge.net>, <https://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

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

# common 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))

# close the GDS file
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", "LZMA", "LZMA_RA", "LZ4", "LZ4_RA"),
         closezip=FALSE, check=TRUE, replace=FALSE, visible=TRUE, ...)
```

Arguments

| | |
|----------|--|
| node | an object of class <code>gdsn.class</code> or <code>gds.class</code> : <code>"gdsn.class"</code> – the node of hierarchical structure; <code>"gds.class"</code> – the root of hierarchical structure |
| name | the variable name; if it is not specified, a temporary name is assigned |
| 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), signed integer: <code>"int8"</code> , <code>"int16"</code> , <code>"int24"</code> , <code>"int32"</code> , <code>"int64"</code> , <code>"sbit2"</code> , <code>"sbit3"</code> , ..., <code>"sbit16"</code> , <code>"sbit24"</code> , <code>"sbit32"</code> , <code>"sbit64"</code> , <code>"vl_int"</code> (encoding variable-length signed integer); unsigned integer: <code>"uint8"</code> , <code>"uint16"</code> , <code>"uint24"</code> , <code>"uint32"</code> , <code>"uint64"</code> , <code>"bit1"</code> , <code>"bit2"</code> , <code>"bit3"</code> , ..., <code>"bit15"</code> , <code>"bit16"</code> , <code>"bit24"</code> , <code>"bit32"</code> , <code>"bit64"</code> , <code>"vl_uint"</code> (encoding variable-length unsigned integer); floating-point number (<code>"float32"</code> , <code>"float64"</code>); packed real number (<code>"packedreal8"</code> , <code>"packedreal16"</code> , <code>"packedreal24"</code> , <code>"packedreal32"</code>): pack a floating-point number to a signed 8/16/24/32-bit integer with two attributes <code>"offset"</code> and <code>"scale"</code> , representing <code>"(signed int)*scale + offset"</code> , where the minimum of the signed integer is used to represent NaN; <code>"packedreal8u"</code> , <code>"packedreal16u"</code> , <code>"packedreal24u"</code> , <code>"packedreal32u"</code> : pack a floating-point number to an unsigned 8/16/24/32-bit integer with two attributes <code>"offset"</code> and <code>"scale"</code> , representing <code>"(unsigned int)*scale + offset"</code> , where the maximum of the unsigned integer is used to represent NaN; sparse array (<code>"sp.int"</code> (<code>"sp.int32"</code>), <code>"sp.int8"</code> , <code>"sp.int16"</code> , <code>"sp.int32"</code> , <code>"sp.int64"</code> , <code>"sp.uint8"</code> , <code>"sp.uint16"</code> , <code>"sp.uint32"</code> , <code>"sp.uint64"</code> , <code>"sp.real"</code> (<code>"sp.real64"</code>), <code>"sp.real32"</code> , <code>"sp.real64"</code>); string (variable-length: <code>"string"</code> , <code>"string16"</code> , <code>"string32"</code> ; C [null-terminated] string: <code>"cstring"</code> , <code>"cstring16"</code> , <code>"cstring32"</code> ; fixed-length: <code>"fstring"</code> , <code>"fstring16"</code> , <code>"fstring32"</code>); Or <code>"char"</code> (<code>"int8"</code>), <code>"int"/"integer"</code> (<code>"int32"</code>), <code>"single"</code> (<code>"float32"</code>), <code>"float"</code> (<code>"float32"</code>), <code>"double"</code> (<code>"float64"</code>), <code>"character"</code> (<code>"string"</code>), <code>"logical"</code> , <code>"list"</code> , <code>"factor"</code> , <code>"folder"</code> ; Or a <code>gdsn.class</code> object, the storage mode is set to be the same as the object specified by <code>storage</code> . |
| 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 <code>"</code> (no compression), <code>"ZIP"</code> , <code>"ZIP.fast"</code> , <code>"ZIP.def"</code> , <code>"ZIP.max"</code> or <code>"ZIP.none"</code> (original zlib); <code>"ZIP_RA"</code> , <code>"ZIP_RA.fast"</code> , <code>"ZIP_RA.def"</code> , <code>"ZIP_RA.max"</code> or <code>"ZIP_RA.none"</code> (zlib with efficient random access); <code>"LZ4"</code> , <code>"LZ4.none"</code> , <code>"LZ4.fast"</code> , <code>"LZ4.hc"</code> or <code>"LZ4.max"</code> (LZ4 compression/decompression library); <code>"LZ4_RA"</code> , <code>"LZ4_RA.none"</code> , <code>"LZ4_RA.fast"</code> , <code>"LZ4_RA.hc"</code> or <code>"LZ4_RA.max"</code> (with efficient random access); <code>"LZMA"</code> , <code>"LZMA.fast"</code> , <code>"LZMA.def"</code> , <code>"LZMA.max"</code> , |

| | |
|----------|--|
| | "LZMA_RA", "LZMA_RA.fast", "LZMA_RA.def", "LZMA_RA.max" (lzma compression/decompression algorithm). 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.def". "ZIP.fast", "ZIP.def" and "ZIP.max" correspond to different compression levels.

To support efficient random access of Z stream, "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.def" or "ZIP_RA.max" should be specified. "ZIP_RA" option is equivalent to "ZIP_RA.def:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K", "1M", "2M", "4M" and "8M" are allowed, like "ZIP_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

LZ4 fast lossless compression algorithm is allowed when compress="LZ4" (<https://github.com/lz4/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" or "ZIP_RA.max" 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", "1M", "2M", "4M" and "8M" are allowed, like "LZ4_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

LZMA compression algorithm (<https://tukaani.org/xz/>) is available since gdsfmt_v1.7.18, which has a higher compression ratio than ZIP algorithm. "LZMA", "LZMA.fast", "LZMA.def" and "LZMA.max" available. To support efficient random access of LZMA stream, "LZMA_RA", "LZMA_RA.fast", "LZMA_RA.def" and "LZMA_RA.max" can be used. The block size can be specified by following colon. "LZMA_RA" is equivalent to "LZMA_RA.def:256K".

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

the parameter details with equivalent command lines can be found at `compression.gdsn`.

`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", "packedreal8u", "packedreal16", "packedreal16u", "packedreal32" or "packedreal32u"`, users can define offset and scale to represent real numbers by "`val*scale + offset`" where "`val`" is a 8/16/32-bit integer. By default, `offset=0, scale=0.01` for "`packedreal8`" and "`packedreal8u`", `scale=0.0001` for "`packedreal16`" and "`packedreal16u`", `scale=0.00001` for "`packedreal24`" and "`packedreal24u`", `scale=0.000001` for "`packedreal32`" and "`packedreal32u`". For example, `packedreal8: scale=1/127, offset=0, packedreal16: scale=1/32767, offset=0` for correlation `[-1, 1]`; `packedreal8u: scale=1/254, offset=0, packedreal16u: scale=1/65534, offset=0` for a probability `[0, 1]`.

Value

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

Author(s)

Xiuwen Zheng

References

<http://zlib.net>, <https://github.com/lz4/lz4>, <https://tukaani.org/xz/>

See Also

`addfile.gdsn`, `addfolder.gdsn`, `compression.gdsn`, `index.gdsn`, `read.gdsn`, `readex.gdsn`, `write.gdsn`, `append.gdsn`

Examples

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

#####
# common 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"))

# close the GDS file
closefn.gdsn(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", "LZMA", "LZMA_RA", "LZ4", "LZ4_RA"),
             replace=FALSE, visible=TRUE)
```


Arguments

| | |
|----------|---|
| node | an object of class <code>gdsn.class</code> or <code>gds.class</code> |
| name | the variable name; if it is not specified, a temporary name is assigned |
| 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 <code>print(, all=TRUE)</code> |

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", "1M", "2M", "4M" and "8M" are allowed, like "ZIP_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

LZ4 fast lossless compression algorithm is allowed when `compress="LZ4"` (<https://github.com/lz4/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", "1M", "2M", "4M" and "8M" are allowed, like "LZ4_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

Value

An object of class `gdsn.class`.

Author(s)

Xiuwen Zheng

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")

# create 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; if it is not specified, a temporary name is assigned |
| 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

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, 400 bytes }
# |--+ virtual_folder [ --> test1.gds ]
# | |--+ NULL
# | |--+ dir [ ]
# | |--+ int { Int32 100, 400 bytes }

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, 400 bytes }
# |--+ virtual_folder [ -X- test1.gds ]

closefn.gds(f)
## End(Not run)

# 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 <code>gdsn.class</code> |
| val | R primitive data, like integer; or an object of class <code>gdsn.class</code> |
| 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

See Also

[read.gdsn](#), [write.gdsn](#), [add.gdsn](#)

Examples

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

# common 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", "gdsnnode"), var.index=c("none", "relative", "absolute"),
           target.node=NULL, .useraw=FALSE, .value=NULL, .substitute=NULL, ...)
```

Arguments

| | |
|-------------|--|
| node | an object of class <code>gdsn.class</code> , or a list of objects of class <code>gdsn.class</code> |
| 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; "gdsnnode" – the returned value from the user-defined function will be appended to <code>target.node</code> . |
| var.index | if "none", call <code>FUN(x, ...)</code> without an index; if "relative" or "absolute", add an argument to the user-defined function <code>FUN</code> like <code>FUN(index, x, ...)</code> where <code>index</code> in the function is an index starting from 1: "relative" for indexing in the selection defined by <code>selection</code> , "absolute" for indexing with respect to all data |
| target.node | NULL, an object of class <code>gdsn.class</code> or a list of <code>gdsn.class</code> : output to the target GDS node(s) when <code>as.is="gdsnnode"</code> . See details |
| .useraw | use R RAW storage mode if integers can be stored in a byte, to reduce memory usage |
| .value | a vector of values to be replaced in the original data array, or NULL for nothing |
| .substitute | a vector of values after replacing, or NULL for nothing; <code>length(.substitute)</code> should be one or <code>length(.value)</code> ; if <code>length(.substitute) = length(.value)</code> , it is a mapping from <code>.value</code> to <code>.substitute</code> |
| ... | 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="gdsnnode"` 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

See Also

[read.gdsn](#), [readex.gdsn](#), [clusterApply.gdsn](#)

Examples

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

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

(n2 <- 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(n1, margin=1, FUN=function(x) print(x), as.is="none")
apply.gdsn(n1, margin=1,
  selection = list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)),
  FUN=function(x) print(x), as.is="none")
apply.gdsn(n1, 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(n1, 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(n2, margin=1, FUN=function(x) print(x), as.is="none")

# Apply functions over columns of matrix
apply.gdsn(n1, margin=2, 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")
apply.gdsn(n2, 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=1, FUN=function(x) print(x), as.is="none",
  .value=16:40, .substitute=NA)
apply.gdsn(n1, margin=2, FUN=function(x) print(x), as.is="none",
  .value=16:40, .substitute=NA)

# close
```

```

closefn.gds(f)

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

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

(n2 <- 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"))

n2.1 <- 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(n2, margin=1, FUN=`c`, as.is="gdsnode", target.node=n2.1)

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

# 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")

(n2 <- add.gdsn(f, "matrix", val=matrix(1:(10*6), nrow=10)))

n2.1 <- add.gdsn(f, "transpose.matrix", storage="int", valdim=c(6,0))
n2.2 <- add.gdsn(f, "n.matrix", storage="int", valdim=c(0))

# Apply functions over rows of matrix
apply.gdsn(n2, margin=1, FUN=function(x) list(x, x[1]),
  as.is="gdsnode", target.node=list(n2.1, n2.2))

```



```

# matrix transpose
read.gdsn(n2)
read.gdsn(n2.1)
read.gdsn(n2.2)

# close
closefn.gds(f)

#####
#
# Multiple variables
#

# create 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=print, as.is="none")

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

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z2), margin=c(2, 2, 1),
  FUN=print, .value=35:45, .substitute=NA)

v <- apply.gdsn(list(X=node.X, Y=node.Y, Z=node.Z2), margin=c(2, 2, 1),
  FUN=print, .value=35:45, .substitute=NA)

# 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 the GDS file
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(node, src.node=NULL, resize=TRUE, seldim=NULL, append=FALSE,
            .value=NULL, .substitute=NULL)
```

Arguments

| | |
|-------------|---|
| node | an object of class gdsn.class , a target GDS node |
| src.node | an object of class gdsn.class , a source GDS node |
| resize | whether call setdim.gdsn to reset the dimension(s) |
| seldim | the selection of src.obj with numeric or logical indicators, or NULL for all data |
| append | if TRUE, append data by calling append.gdsn ; otherwise, replace the old one |
| .value | a vector of values to be replaced in the original data array, or NULL for nothing |
| .substitute | a vector of values after replacing, or NULL for nothing; length(.substitute) should be one or length(.value); if length(.substitute) = length(.value), it is a mapping from .value to .substitute |

Value

None.

Author(s)

Xiuwen Zheng

See Also

[read.gdsn](#), [readex.gdsn](#), [apply.gdsn](#), [write.gdsn](#), [append.gdsn](#)

Examples

```

f <- createfn.gds("test.gds")

n1 <- add.gdsn(f, "n1", 1:100)
n2 <- add.gdsn(f, "n2", storage="int", valdim=c(20, 0))
n3 <- add.gdsn(f, "n3", storage="int", valdim=c(0))
n4 <- add.gdsn(f, "n4", matrix(1:48, 6))
f

assign.gdsn(n2, n1, resize=FALSE, append=TRUE)

read.gdsn(n1)
read.gdsn(n2)

assign.gdsn(n2, n1, resize=FALSE, append=TRUE)
append.gdsn(n2, n1)
read.gdsn(n2)

assign.gdsn(n3, n2, seldim=
  list(rep(c(TRUE, FALSE), 10), c(rep(c(TRUE, FALSE), 7), TRUE)))
read.gdsn(n3)

setdim.gdsn(n2, c(25,0))
assign.gdsn(n2, n1, append=TRUE, seldim=rep(c(TRUE, FALSE), 50))
read.gdsn(n2)

assign.gdsn(n2, n1); read.gdsn(n2)
f

##

read.gdsn(n4)

# substitute
assign.gdsn(n4, .value=c(3:8,35:40), .substitute=NA); read.gdsn(n4)

# subset
assign.gdsn(n4, seldim=list(c(4,2,6,NA), c(5,6,NA,2,8,NA,4))); read.gdsn(n4)

n4 <- add.gdsn(f, "n4", matrix(1:48, 6), replace=TRUE)
read.gdsn(n4)
# sort into descending order
assign.gdsn(n4, seldim=list(6:1, 8:1)); read.gdsn(n4)

# close the GDS file
closefn.gds(f)

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

```

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

See Also

[read.gdsn](#)

Examples

```
# create a GDS file
f <- createfn.gdsn("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.gdsn(f)

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

| | |
|-------------|---------------------------|
| cleanup.gds | <i>Clean up fragments</i> |
|-------------|---------------------------|

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

See Also

[openfn.gds](#), [createfn.gds](#), [closefn.gds](#)

Examples

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

# common 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

# close the GDS file
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

See Also

[createfn.gds](#), [openfn.gds](#), [sync.gds](#)

Examples

```
# create 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,
  .value=NULL, .substitute=NULL, ...)
```

Arguments

| | |
|--------------------------|--|
| <code>cl</code> | a cluster object, created by this package or by the package parallel |
| <code>gds.fn</code> | the file name of a GDS file |
| <code>node.name</code> | a character vector indicating GDS node path |
| <code>margin</code> | an integer giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns |
| <code>FUN</code> | the function to be applied |
| <code>selection</code> | a list or NULL; if a list, it is a list of logical vectors according to dimensions indicating selection; if NULL, uses all data |
| <code>as.is</code> | returned value: a list, an integer vector, etc |
| <code>var.index</code> | 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 |
| <code>.useraw</code> | use R RAW storage mode if integers can be stored in a byte, to reduce memory usage |
| <code>.value</code> | a vector of values to be replaced in the original data array, or NULL for nothing |
| <code>.substitute</code> | a vector of values after replacing, or NULL for nothing; length(.substitute) should be one or length(.value); if length(.substitute) = length(.value), it is a mapping from .value to .substitute |
| <code>...</code> | 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

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", 2L))

# 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, include.hidden=FALSE)
```

Arguments

`node` an object of class `gdsn.class`, a GDS node
`include.hidden` whether including hidden variables or folders

Value

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

Author(s)

Xiuwen Zheng

See Also

[objdesp.gdsn](#), [ls.gdsn](#), [index.gdsn](#), [delete.gdsn](#), [add.gdsn](#)

Examples

```
# create 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

# close the GDS file
closefn.gds(f)

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

compression.gdsn *Modify compression mode*

Description

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

Usage

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

Arguments

| | |
|----------|--|
| node | an object of class <code>gdsn.class</code> , a GDS node |
| compress | the compression method can be "" (no compression), "ZIP", "ZIP.fast", "ZIP.def", "ZIP.max" or "ZIP.none" (original zlib); "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.def", "ZIP_RA.max" or "ZIP_RA.none" (zlib with efficient random access); "LZ4", "LZ4.none", "LZ4.fast", "LZ4.hc" or "LZ4.max" (LZ4 compression/decompression library); "LZ4_RA", "LZ4_RA.none", "LZ4_RA.fast", "LZ4_RA.hc" or "LZ4_RA.max" (with efficient random access). "LZMA", "LZMA.fast", "LZMA.def", "LZMA.max", "LZMA_RA", "LZMA_RA.fast", "LZMA_RA.def", "LZMA_RA.max" (lzma compression/decompression algorithm). 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.def". "ZIP.fast", "ZIP.def" and "ZIP.max" correspond to different compression levels.

To support efficient random access of Z stream, "ZIP_RA", "ZIP_RA.fast", "ZIP_RA.def" or "ZIP_RA.max" should be specified. "ZIP_RA" option is equivalent to "ZIP_RA.def:256K". The block size can be specified by following colon, and "16K", "32K", "64K", "128K", "256K", "512K", "1M", "2M", "4M" and "8M" are allowed, like "ZIP_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

LZ4 fast lossless compression algorithm is allowed when compress="LZ4" (<https://github.com/lz4/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" or "ZIP_RA.max" 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", "1M", "2M", "4M" and "8M" are allowed, like "LZ4_RA:64K". The compression algorithm tries to keep each independent compressed data block to be about of the specified block size, like 64K.

LZMA compression algorithm (<https://tukaani.org/xz/>) is available since gdsfmt_v1.7.18, which has a higher compression ratio than ZIP algorithm. "LZMA", "LZMA.fast", "LZMA.def" and "LZMA.max" available. To support efficient random access of LZMA stream, "LZMA_RA", "LZMA_RA.fast", "LZMA_RA.def" and "LZMA_RA.max" can be used. The block size can be specified by following colon. "LZMA_RA" is equivalent to "LZMA_RA.def:256K".

| compression 1 | compression 2 | command line |
|----------------|-------------------|-----------------------|
| ZIP | ZIP_RA | gzip -6 |
| ZIP.fast | ZIP_RA.fast | gzip -fast |
| ZIP.def | ZIP_RA.def | gzip -6 |
| ZIP.max | ZIP_RA.max | gzip -best |
| LZ4 | LZ4_RA | LZ4 HC -6 |
| LZ4.min | LZ4_RA.min | LZ4 fast 0 |
| LZ4.fast | LZ4_RA.fast | LZ4 fast 2 |
| LZ4.hc | LZ4_RA.hc | LZ4 HC -6 |
| LZ4.max | LZ4_RA.max | LZ4 HC -9 |
| LZMA | LZMA_RA | xz -6 |
| LZMA.min | LZMA_RA.min | xz -0 |
| LZMA.fast | LZMA_RA.fast | xz -2 |
| LZMA.def | LZMA_RA.def | xz -6 |
| LZMA.max | LZMA_RA.max | xz -9e |
| LZMA.ultra | LZMA_RA.ultra | xz -lzma2=dict=512Mi |
| LZMA.ultra_max | LZMA_RA.ultra_max | xz -lzma2=dict=1536Mi |

Value

Return node.

Author(s)

Xiuwen Zheng

References

<http://zlib.net>, <https://github.com/lz4/lz4>, <https://tukaani.org/xz/>

See Also

[readmode.gdsn](#), [add.gdsn](#)

Examples

```
# create 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)
```

copyto.gdsn

Copy GDS nodes

Description

Copy GDS node(s) to a folder with a new name

Usage

```
copyto.gdsn(node, source, name=NULL)
```

Arguments

| | |
|--------|--|
| node | a folder of class gdsn.class or gds.class |
| source | an object of class gdsn.class or gds.class |
| name | a specified name; if NULL, it is determined by source |

Value

None.

Author(s)

Xiuwen Zheng

See Also

[moveto.gdsn](#)

Examples

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

add.gdsn(f, "label", NULL)
add.gdsn(f, "int", 1:100, compress="ZIP", closezip=TRUE)
add.gdsn(f, "int.matrix", matrix(1:100, nrow=20))
addfolder.gdsn(f, "folder1")
```

```

addfolder.gdsn(f, "folder2")

for (nm in c("label", "int", "int.matrix"))
  copyto.gdsn(index.gdsn(f, "folder1"), index.gdsn(f, nm))
f

copyto.gdsn(index.gdsn(f, "folder2"), index.gdsn(f, "folder1"))
f

# 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, use.abstractmethod=TRUE)
```

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 |
| use.abstractmethod | if TRUE, 'filename' of the gds.class object is set to be the absolute path |

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 hierarchical structure |
| readonly | whether it is read-only or not |

Author(s)

Xiuwen Zheng

See Also

[openfn.gds](#), [closefn.gds](#)

Examples

```
# create 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

# close the GDS file
closefn.gds(f)

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

delete.attr.gdsn *Delete attribute(s)*

Description

Remove the attribute(s) of a GDS node.

Usage

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

Arguments

| | |
|------|--|
| node | an object of class gdsn.class , a GDS node |
| name | the name(s) of an attribute |

Value

None.

Author(s)

Xiuwen Zheng

See Also

[put.attr.gdsn](#), [get.attr.gdsn](#)

Examples

```
# create 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, c("one.value", "bool"))
get.attr.gdsn(node)

# close the GDS file
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

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

Value

None.

Author(s)

Xiuwen Zheng

See Also

[add.gdsn](#)

Examples

```

# create 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 data information.

Usage

```
diagnosis.gds(gds, log.only=FALSE)
```

Arguments

| | |
|----------|--|
| gds | an object of class <code>gdsn.class</code> or <code>gds.class</code> |
| log.only | if TRUE, return a character vector of log only |

Value

A list with stream and chunk information.

If gds is a "gds.class" object (i.e., a GDS file), the function returns a list with components, like:

| | |
|--------|------------------------|
| stream | summary of byte stream |
| log | event log records |

If gds is a "gdsn.class" object, the function returns a list with components, like:

| | |
|------|--------------------------------------|
| head | total_size, chunk_offset, chunk_size |
| data | total_size, chunk_offset, chunk_size |

...

Author(s)

Xiuwen Zheng

Examples

```

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

set.seed(1000)
rawval <- as.raw(rep(0:99, 50))

add.gdsn(f, "label", NULL)
add.gdsn(f, "raw", rawval)

closefn.gds(f)

##

f <- openfn.gds("test.gds")

diagnosis.gds(f)
diagnosis.gds(f$root)
diagnosis.gds(index.gdsn(f, "label"))
diagnosis.gds(index.gdsn(f, "raw"))

closefn.gds(f)

## remove fragments

cleanup.gds("test.gds")

##

f <- openfn.gds("test.gds")

diagnosis.gds(f$root)
diagnosis.gds(index.gdsn(f, "label"))
(adr <- diagnosis.gds(index.gdsn(f, "raw")))

closefn.gds(f)

## read binary data directly

f <- file("test.gds", "rb")

dat <- NULL
for (i in seq_len(length(adr$data$chunk_offset)))
{
  seek(f, adr$data$chunk_offset[i])
  dat <- c(dat, readBin(f, "raw", adr$data$chunk_size[i]))
}

identical(dat, rawval) # should be TRUE

close(f)

```

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

| | |
|-------------|-------------------------------------|
| digest.gdsn | <i>create hash function digests</i> |
|-------------|-------------------------------------|

Description

Create hash function digests for a GDS node.

Usage

```
digest.gdsn(node, algo=c("md5", "sha1", "sha256", "sha384", "sha512"),
  action=c("none", "Robject", "add", "add.Robj", "clear", "verify", "return"))
```

Arguments

| | |
|--------|--|
| node | an object of class <code>gdsn.class</code> , a GDS node |
| algo | the algorithm to be used; currently available choices are "md5" (by default), "sha1", "sha256", "sha384", "sha512" |
| action | "none": nothing (by default); "Robject": convert to R object, i.e., raw, integer, double or character before applying hash digests; "add": add a barcode attribute; "add.Robj": add a barcode attribute generated from R object; "clear": remove all hash barcodes; "verify": verify data integrity if there is any hash code in the attributes, and stop if any fails; "return": compare the existing hash code in the attributes, and return FALSE if fails, NA if no hash code, and TRUE if the verification succeeds |

Details

The R package `digest` should be installed to perform hash function digests.

Value

A character or `NA_character_` when the hash algorithm is not available.

Author(s)

Xiuwen Zheng

Examples

```
library(digest)
library(tools)

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

val <- as.raw(rep(1:128, 1024))
n1 <- add.gdsn(f, "raw1", val)
```

```

n2 <- add.gdsn(f, "int1", as.integer(val))
n3 <- add.gdsn(f, "int2", as.integer(val), compress="ZIP", closezip=TRUE)

digest.gdsn(n1)
digest.gdsn(n1, action="Robject")
digest.gdsn(n1, action="add")
digest.gdsn(n1, action="add.Robj")
writeBin(read.gdsn(n1, .useraw=TRUE), con="test1.bin")

write.gdsn(n1, 0, start=1027, count=1)
digest.gdsn(n1, action="add")
digest.gdsn(n1, action="add.Robj")
digest.gdsn(n1, "sha1", action="add")
digest.gdsn(n1, "sha256", action="add")
# digest.gdsn(n1, "sha384", action="add") ## digest_0.6.11 does not work
digest.gdsn(n1, "sha512", action="add")
writeBin(read.gdsn(n1, .useraw=TRUE), con="test2.bin")

print(n1, attribute=TRUE)
digest.gdsn(n1, action="verify")

digest.gdsn(n1, action="clear")
print(n1, attribute=TRUE)

digest.gdsn(n2)
digest.gdsn(n2, action="Robject")

# using R object
digest.gdsn(n2) == digest.gdsn(n3) # FALSE
digest.gdsn(n2, action="Robject") == digest.gdsn(n3, action="Robject") # TRUE

# close the GDS file
closefn.gds(f)

# check with other program
md5sum(c("test1.bin", "test2.bin"))

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

```

exist.gdsn

Return whether the path exists or not

Description

Get a logical vector to show whether the path exists or not.

Usage

```
exist.gdsn(node, path)
```

Arguments

node an object of class `gdsn.class`, a GDS node
 path the path(s) specifying a GDS node with '/' as a separator

Value

A logical vector.

Author(s)

Xiuwen Zheng

See Also

[ls.gdsn](#), [index.gdsn](#)

Examples

```
# create 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)
# "x" "y" "z"

exist.gdsn(f, c("list", "list/z", "wuw/dj"))

# close the GDS file
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

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

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

See Also

[put.attr.gdsn](#), [delete.attr.gdsn](#)

Examples

```
# create 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 <code>gdsn.class</code> , a GDS node |
| out.filename | the file name of output stream |

Value

None.

Author(s)

Xiuwen Zheng

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")

# create 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.gdsn(f)

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

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

# open the GDS file
closefn.gdsn(f)

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

getfolder.gdsn

Get the folder

Description

Get the folder which contains the specified GDS node.

Usage

```
getfolder.gdsn(node)
```

Arguments

node an object of class [gdsn.class](#)

Value

An object of class [gdsn.class](#).

Author(s)

Xiuwen Zheng

See Also

[index.gdsn](#)

Examples

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

add.gdsn(f, "label", NULL)
add.gdsn(f, "double", seq(1, 1000, 0.4))
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)))

f

getfolder.gdsn(index.gdsn(f, "label"))
getfolder.gdsn(index.gdsn(f, "double"))
getfolder.gdsn(index.gdsn(f, "list/X"))
getfolder.gdsn(index.gdsn(f, "data.frame/Y"))

getfolder.gdsn(f$root)

# close the GDS file
closefn.gds(f)

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

index.gdsn

Get the specified node

Description

Get a specified GDS node.

Usage

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

Arguments

| | |
|--------|---|
| node | an object of class <code>gdsn.class</code> (a GDS node), or <code>gds.class</code> (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

See Also[cnt.gdsn](#), [ls.gdsn](#), [name.gdsn](#), [add.gdsn](#), [delete.gdsn](#)**Examples**

```
# create 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:
index.gdsn(f, "list/x/z")
# Error in index.gdsn(f, "list/x/z") : Invalid path "list/x/z"!

## End(Not run)

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

# close the GDS 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 <code>gdsn.class</code> (a GDS node) |
| set | the specified set of elements |

Value

A logical vector or array.

Author(s)

Xiuwen Zheng

See Also

[read.gdsn](#)

Examples

```
# create a GDS file
f <- createfn.gdsn("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.gdsn(f)

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

| | |
|----------------|--------------------------------------|
| is.sparse.gdsn | <i>whether a sparse array or not</i> |
|----------------|--------------------------------------|

Description

Determine whether the node is a sparse array or not.

Usage

```
is.sparse.gdsn(node)
```

Arguments

node an object of class [gdsn.class](#) (a GDS node)

Value

TRUE / FALSE.

Author(s)

Xiuwen Zheng

See Also[add.gdsn](#)**Examples**

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

cnt <- matrix(0, nrow=4, ncol=8)
set.seed(100)
cnt[sample.int(length(cnt), 8)] <- rpois(8, 4)
cnt

add.gdsn(f, "mat", val=cnt)
add.gdsn(f, "sp.mat", val=cnt, storage="sp.real")
f

is.sparse.gdsn(index.gdsn(f, "mat"))
is.sparse.gdsn(index.gdsn(f, "sp.mat"))

read.gdsn(index.gdsn(f, "sp.mat"))

# 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

Examples`lasterr.gds()`

| | |
|---------|--|
| ls.gdsn | <i>Return the names of child nodes</i> |
|---------|--|

Description

Get a list of names for its child nodes.

Usage

```
ls.gdsn(node, include.hidden=FALSE, recursive=FALSE, include.dirs=TRUE)
```

Arguments

| | |
|----------------|---|
| node | an object of class <code>gdsn.class</code> , a GDS node |
| include.hidden | whether including hidden variables or folders |
| recursive | whether the listing recurses into directories or not |
| include.dirs | whether subdirectory names should be included in recursive listings |

Value

A vector of characters, or character(0) if node is not a folder.

Author(s)

Xiuwen Zheng

See Also

[cnt.gdsn](#), [objdesp.gdsn](#), [ls.gdsn](#), [index.gdsn](#)

Examples

```
# create 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)
# "x" "y" "z"

ls.gdsn(f$root)
# "list"

ls.gdsn(f$root, recursive=TRUE)
# "list" "list/x" "list/y" "list/z"

ls.gdsn(f$root, recursive=TRUE, include.dirs=FALSE)
# "list/x" "list/y" "list/z"

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

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

| | |
|-------------|----------------------------|
| moveto.gdsn | <i>Relocate a GDS node</i> |
|-------------|----------------------------|

Description

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

Usage

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

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); "replace+rename": replace loc.node (loc.node will be deleted and node has a new name as loc.node) |

Value

None.

Author(s)

Xiuwen Zheng

See Also

[createfn.gds](#), [openfn.gds](#), [index.gdsn](#), [add.gdsn](#)

Examples

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

# common 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

n2 <- index.gdsn(f, "int")
moveto.gdsn(n1, n2, relpos="replace+rename")
f

# close the GDS file
closefn.gdsn(f)

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

| | |
|-----------|---|
| name.gdsn | <i>Return the variable name of a node</i> |
|-----------|---|

Description

Get the variable name of a GDS node.

Usage

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

Arguments

| | |
|----------|--|
| node | an object of class <code>gdsn.class</code> , 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

See Also

[cnt.gdsn](#), [objdesp.gdsn](#), [ls.gdsn](#), [rename.gdsn](#)

Examples

```
# create 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"

# close the GDS file
closefn.gds(f)

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

| objdesp.gdsn | <i>Variable description</i> |
|--------------|-----------------------------|
|--------------|-----------------------------|

Description

Get the description 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 |
| is.sparse | TRUE, if it is a sparse array |

| | |
|----------|---|
| 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 |
| hidden | logical, TRUE if it is a hidden object |
| 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

See Also

[cnt.gdsn](#), [name.gdsn](#), [ls.gdsn](#), [index.gdsn](#)

Examples

```
# create 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 | <i>Open a GDS file</i> |
|------------|------------------------|

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,
            allow.error=FALSE, use.abbrev=TRUE)
```

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 |
| allow.error | TRUE for data recovery from a crashed GDS file |
| use.abbrev | if TRUE, 'filename' of the gds.class object is set to be the absolute path |

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 hierarchical 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 <code>gdsn.class</code> , the root of hierarchical structure |
| readonly | whether it is read-only or not |

Author(s)

Xiuwen Zheng

See Also[createfn.gds](#), [closefn.gds](#)**Examples**

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

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

`permdim.gdsn`*Array Transposition*

Description

Transpose an array by permuting its dimensions.

Usage

```
permdim.gdsn(node, dimidx, target=NULL)
```

Arguments

| | |
|---------------------|---|
| <code>node</code> | an object of class <code>gdsn.class</code> , a GDS node |
| <code>dimidx</code> | the subscript permutation vector, and it should be a permutation of the integers '1:n', where 'n' is the number of dimensions |
| <code>target</code> | if it is not NULL, the transposed data are saved to target |

Value

None.

Author(s)

Xiuwen Zheng

See Also[setdim.gdsn](#)**Examples**

```

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

(node <- add.gdsn(f, "matrix", val=matrix(1:48, nrow=6),
  compress="ZIP", closezip=TRUE))
read.gdsn(node)

permdim.gdsn(node, c(2,1))
read.gdsn(node)

(node <- add.gdsn(f, "array", val=array(1:120, dim=c(5,4,3,2)),
  compress="ZIP", closezip=TRUE))
read.gdsn(node)

mat <- read.gdsn(node)
permdim.gdsn(node, c(1,2,3,4))
stopifnot(identical(mat, read.gdsn(node)))

mat <- read.gdsn(node)
permdim.gdsn(node, c(4,2,1,3))
stopifnot(identical(aperm(mat, c(4,2,1,3)), read.gdsn(node)))

mat <- read.gdsn(node)
permdim.gdsn(node, c(3,2,4,1))
stopifnot(identical(aperm(mat, c(3,2,4,1)), read.gdsn(node)))

mat <- read.gdsn(node)
permdim.gdsn(node, c(2,3,1,4))
stopifnot(identical(aperm(mat, c(2,3,1,4)), read.gdsn(node)))

# close the GDS file
closefn.gds(f)

# remove unused space after permuting dimensions
cleanup.gds("test.gds")

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

```

Description

Displays the contents of "gds.class" (a GDS file) and "gdsn.class" (a GDS node).

Usage

```
## S3 method for class 'gds.class'
print(x, path="", show=TRUE, ...)
## S3 method for class 'gdsn.class'
print(x, expand=TRUE, all=FALSE, nmax=Inf, depth=Inf,
      attribute=FALSE, attribute.trim=FALSE, ...)
## S4 method for signature 'gdsn.class'
show(object)
```

Arguments

| | |
|----------------|---|
| x | an object of class <code>gds.class</code> , a GDS file; or <code>gdsn.class</code> , a GDS node |
| object | an object of class <code>gds.class</code> , the number of elements in the preview can be specified via the option <code>getOption("gds.preview.num", 6L)</code> , while 6L is the default value |
| path | the path specifying a GDS node with '/' as a separator |
| show | if TRUE, display the preview of array node |
| expand | whether enumerate all of child nodes |
| all | if FALSE, hide GDS nodes with an attribute "R.invisible" |
| nmax | display nodes within the maximum number nmax |
| depth | display nodes under maximum depth |
| attribute | if TRUE, show the attribute(s) |
| attribute.trim | if TRUE, trim the attribute information if it is too long |
| ... | the arguments passed to or from other methods |

Value

None.

Author(s)

Xiuwen Zheng

Examples

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

add.gdsn(f, "int", 1:100)
add.gdsn(f, "int.matrix", matrix(1:(50*100), nrow=100, ncol=50))
put.attr.gdsn(index.gdsn(f, "int.matrix"), "int", 1:10)

print(f, all=TRUE)
print(f, all=TRUE, attribute=TRUE)
print(f, all=TRUE, attribute=TRUE, attribute.trim=FALSE)

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

```

show(index.gdsn(f, "int.matrix"))

# close the GDS file
closefn.gdsn(f)

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

```

| | |
|---------------|---|
| put.attr.gdsn | <i>Add an attribute into a GDS node</i> |
|---------------|---|

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, or a gdsn.class object |

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.

If val is a [gdsn.class](#) object, copy all attributes to node.

Value

None.

Author(s)

Xiuwen Zheng

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", paste(letters, collapse="")))
put.attr.gdsn(node, "bool", c(TRUE, TRUE, FALSE))

```

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

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

node2 <- add.gdsn(f, "char", val=letters)
get.attr.gdsn(node2)
put.attr.gdsn(node2, val=node)
get.attr.gdsn(node2)

# close the GDS file
closefn.gds(f)

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

| | |
|-----------|--------------------------------------|
| read.gdsn | <i>Read data field of a GDS node</i> |
|-----------|--------------------------------------|

Description

Get data from a GDS node.

Usage

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

Arguments

| | |
|-------------|--|
| node | an object of class <code>gdsn.class</code> , a GDS node |
| start | a vector of integers, starting from 1 for each dimension component |
| count | a vector of integers, the length of each dimension. As a special case, the value "-1" indicates that all entries along that dimension should be read, starting from start |
| 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 | a vector of values to be replaced in the original data array, or NULL for nothing |
| .substitute | a vector of values after replacing, or NULL for nothing; <code>length(.substitute)</code> should be one or <code>length(.value)</code> ; if <code>length(.substitute) = length(.value)</code> , it is a mapping from <code>.value</code> to <code>.substitute</code> |
| .sparse | only applicable for the sparse array nodes, if TRUE and it is a vector or matrix, return a <code>Matrix::dgCMatrix</code> object |

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

See Also

[readex.gdsn](#), [append.gdsn](#), [write.gdsn](#), [add.gdsn](#)

Examples

```
# create a GDS file
f <- createfn.gdsn("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

read.gdsn(index.gdsn(f, "matrix"), start=c(2,2), count=c(-1,3))
read.gdsn(index.gdsn(f, "matrix"), start=c(2,2), count=c(-1,3),
          .value=c(12,5), .substitute=NA)

# close the GDS file
closefn.gdsn(f)

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

| | |
|-------------|---|
| readex.gdsn | <i>Read data field of a GDS node with a selection</i> |
|-------------|---|

Description

Get data from a GDS node with subset selection.

Usage

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

Arguments

| | |
|-------------|--|
| node | an object of class <code>gdsn.class</code> , 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 | a vector of values to be replaced in the original data array, or NULL for nothing |
| .substitute | a vector of values after replacing, or NULL for nothing; <code>length(.substitute)</code> should be one or <code>length(.value)</code> ; if <code>length(.substitute) = length(.value)</code> , it is a mapping from <code>.value</code> to <code>.substitute</code> |
| .sparse | only applicable for the sparse array nodes, if TRUE and it is a vector or matrix, return a <code>Matrix::dgCMatrix</code> object |

Details

If `sel` is a list of numeric vectors, the internal method converts the numeric vectors to logical vectors first, extract data with logical vectors, and then call `[` to reorder or expand data.

Value

Return an array.

Author(s)

Xiuwen Zheng

See Also

[read.gdsn](#), [append.gdsn](#), [write.gdsn](#), [add.gdsn](#)

Examples

```

# create 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))
readex.gdsn(index.gdsn(f, "vector"), sel=c(4:8, 1, 2, 12))
readex.gdsn(index.gdsn(f, "vector"), sel=-1:-10)

readex.gdsn(index.gdsn(f, "vector"), sel=c(4, 1, 10, NA, 12, NA))
readex.gdsn(index.gdsn(f, "vector"), sel=c(4, 1, 10, NA, 12, NA),
  .value=c(NA, 1, 12), .substitute=c(6, 7, NA))

# read matrix
readex.gdsn(index.gdsn(f, "matrix"))
readex.gdsn(index.gdsn(f, "matrix"),
  sel=list(rep(c(TRUE, FALSE), 5), rep(c(TRUE, FALSE), 3)))
readex.gdsn(index.gdsn(f, "matrix"), sel=list(NULL, c(1,3,6)))
readex.gdsn(index.gdsn(f, "matrix"),
  sel=list(rep(c(TRUE, FALSE), 5), c(1,3,6)))
readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(1,3,6,10), c(1,3,6)))
readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(-1,-3), -6))

readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(1,3,NA,10), c(1,3,NA,5)))
readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(1,3,NA,10), c(1,3,NA,5)),
  simplify="force")

readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(1,3,NA,10), c(1,3,NA,5)))
readex.gdsn(index.gdsn(f, "matrix"), sel=list(c(1,3,NA,10), c(1,3,NA,5)),
  .value=NA, .substitute="X")

# 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

See Also

[compression.gdsn](#), [add.gdsn](#)

Examples

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

# common 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)

# close the GDS file
closefn.gds(f)

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

| | |
|-------------|--------------------------|
| rename.gdsn | <i>Rename a GDS node</i> |
|-------------|--------------------------|

Description

Rename a GDS node.

Usage

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

Arguments

| | |
|---------|---|
| node | an object of class <code>gdsn.class</code> , 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

See Also

[name.gdsn](#), [ls.gdsn](#), [index.gdsn](#)

Examples

```
# create 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

See Also

[read.gdsn](#), [write.gdsn](#), [add.gdsn](#), [append.gdsn](#)

Examples

```
# create 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)

setdim.gdsn(n, c(3, 4), permute=TRUE)
read.gdsn(n)

n <- add.gdsn(f, "bit3", val=1:24, storage="bit3")
```

```

read.gdsn(n)

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

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

setdim.gdsn(n, c(3, 4), permute=TRUE)
read.gdsn(n)

# close the GDS file
closefn.gds(f)

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

```

| | |
|--------------|-----------------------------------|
| showfile.gds | <i>Enumerate opened GDS files</i> |
|--------------|-----------------------------------|

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 data frame with the columns "FileName", "ReadOnly" and "State", or NULL if there is no opened gds file.

Author(s)

Xiuwen Zheng

Examples

```

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

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

showfile.gds()

showfile.gds(closeall=TRUE)

```

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

summarize.gdsn *GDS object Summaries*

Description

Get the summaries of a GDS node.

Usage

```
summarize.gdsn(node)
```

Arguments

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

Value

A list including

| | |
|---------|---|
| min | the minimum value |
| max | the maximum value |
| num_na | the number of invalid numbers or NA |
| decimal | the count of each decimal (integer, 0.1, 0.01, ..., or other) |

Author(s)

Xiuwen Zheng

Examples

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

n1 <- add.gdsn(f, "x", seq(1, 10, 0.1), storage="float")
n2 <- add.gdsn(f, "y", seq(1, 10, 0.1), storage="double")
n3 <- add.gdsn(f, "int", c(1:100, NA, 112, NA), storage="int")
n4 <- add.gdsn(f, "int8", c(1:100, NA, 112, NA), storage="int8")

summarize.gdsn(n1)
summarize.gdsn(n2)
summarize.gdsn(n3)
summarize.gdsn(n4)

# close the file
closefn.gds(f)

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

| | |
|----------|-------------------------------|
| sync.gds | <i>Synchronize a GDS file</i> |
|----------|-------------------------------|

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

See Also

[createfn.gds](#), [openfn.gds](#)

Examples

```
options(gds.verbose=TRUE)

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

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

sync.gds(f)

get.attr.gdsn(node)

# close the GDS file
closefn.gds(f)

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

| | |
|------------|---|
| system.gds | <i>Get the parameters in the GDS system</i> |
|------------|---|

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 if applicable |
| compression.encoder | compression/decompression algorithms |
| compiler | information of compiler |
| compiler.flag | SIMD instructions supported by the compiler |
| class.list | class list in the GDS system |
| options | list all options associated with GDS format or package, including gds.crayon(FALSE for no stylish terminal output), gds.parallel and gds.verbose |

Author(s)

Xiuwen Zheng

Examples

```
system.gds()
```

| | |
|-------------|--------------------------|
| unload.gdsn | <i>Unload a GDS node</i> |
|-------------|--------------------------|

Description

Unload a specified GDS node.

Usage

```
unload.gdsn(node)
```

Arguments

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

Value

None.

Author(s)

Xiuwen Zheng

See Also

[delete.gdsn](#)

Examples

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

# add a list to "test.gds"
node <- add.gdsn(f, "val", 1:1000)
node

## Not run:
unload.gdsn(node)
node # Error: Invalid GDS node object (it was unloaded or deleted).

## End(Not run)

index.gdsn(f, "val")

# close the GDS file
closefn.gds(f)

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

| | |
|------------|---------------------------------|
| write.gdsn | <i>Write data to a GDS node</i> |
|------------|---------------------------------|

Description

Write data to a GDS node.

Usage

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

Arguments

| | |
|-------|--|
| node | an object of class <code>gdsn.class</code> , 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 dimension |
| 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

See Also

[append.gdsn](#), [read.gdsn](#), [add.gdsn](#)

Examples

```
# create 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 GDS file  
closefn.gds(f)
```

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

Index

* GDS

- add.gdsn, 5
- addfile.gdsn, 8
- addfolder.gdsn, 10
- append.gdsn, 12
- apply.gdsn, 14
- assign.gdsn, 18
- cache.gdsn, 19
- cleanup.gdsn, 21
- closefn.gds, 22
- clusterApply.gdsn, 23
- cnt.gdsn, 25
- compression.gdsn, 26
- copyto.gdsn, 28
- createfn.gds, 29
- delete.attr.gdsn, 30
- delete.gdsn, 31
- diagnosis.gds, 32
- digest.gdsn, 34
- exist.gdsn, 35
- gds.class, 36
- gdsfmt-package, 3
- gdsn.class, 37
- get.attr.gdsn, 37
- getfile.gdsn, 38
- getfolder.gdsn, 39
- index.gdsn, 40
- is.element.gdsn, 41
- is.sparse.gdsn, 42
- lasterr.gds, 43
- ls.gdsn, 44
- moveto.gdsn, 45
- name.gdsn, 46
- objdesp.gdsn, 47
- openfn.gds, 49
- permdim.gdsn, 50
- print.gds.class, 51
- put.attr.gdsn, 53
- read.gdsn, 54
- readex.gdsn, 56
- readmode.gdsn, 57
- rename.gdsn, 59
- setdim.gdsn, 60

- showfile.gds, 61
- summarize.gdsn, 62
- sync.gds, 63
- system.gds, 64
- unload.gdsn, 65
- write.gdsn, 66

* IO

- gdsfmt-package, 3

* database

- gdsfmt-package, 3

* file

- gdsfmt-package, 3

* interface

- gdsfmt-package, 3

* utilities

- add.gdsn, 5
- addfile.gdsn, 8
- addfolder.gdsn, 10
- append.gdsn, 12
- apply.gdsn, 14
- assign.gdsn, 18
- cache.gdsn, 19
- cleanup.gds, 21
- closefn.gds, 22
- clusterApply.gdsn, 23
- cnt.gdsn, 25
- compression.gdsn, 26
- copyto.gdsn, 28
- createfn.gds, 29
- delete.attr.gdsn, 30
- delete.gdsn, 31
- diagnosis.gds, 32
- digest.gdsn, 34
- exist.gdsn, 35
- gds.class, 36
- gdsfmt-package, 3
- gdsn.class, 37
- get.attr.gdsn, 37
- getfile.gdsn, 38
- getfolder.gdsn, 39
- index.gdsn, 40
- is.element.gdsn, 41
- is.sparse.gdsn, 42

- lasterr.gds, 43
- ls.gdsn, 44
- moveto.gdsn, 45
- name.gdsn, 46
- objdesp.gdsn, 47
- openfn.gds, 49
- permdim.gdsn, 50
- print.gds.class, 51
- put.attr.gdsn, 53
- read.gdsn, 54
- readex.gdsn, 56
- readmode.gdsn, 57
- rename.gdsn, 59
- setdim.gdsn, 60
- showfile.gds, 61
- summarize.gdsn, 62
- sync.gds, 63
- system.gds, 64
- unload.gdsn, 65
- write.gdsn, 66
- [, 56
- add.gdsn, 5, 9, 11, 13, 25, 27, 31, 37, 41, 43, 45, 48, 55, 56, 58, 60, 66
- addfile.gdsn, 7, 8, 11, 38, 47
- addfolder.gdsn, 7, 10
- append.gdsn, 7, 12, 18, 55, 56, 60, 66
- apply.gdsn, 14, 18, 24
- assign.gdsn, 18
- cache.gdsn, 19
- cleanup.gds, 21
- closefn.gds, 21, 22, 22, 29, 30, 37, 50, 63
- clusterApply.gdsn, 15, 23
- cnt.gdsn, 25, 41, 44, 46, 48
- compression.gdsn, 7, 26, 58
- copyto.gdsn, 28
- createfn.gds, 21, 22, 29, 37, 45, 49, 50, 63
- delete.attr.gdsn, 30, 37, 53
- delete.gdsn, 25, 31, 41, 65
- diagnosis.gds, 32
- digest.gdsn, 34
- exist.gdsn, 35
- gds.class, 5, 9, 10, 22, 28, 29, 32, 36, 40, 49, 52, 63
- gdsfmt (gdsfmt-package), 3
- gdsfmt-package, 3
- gdsn.class, 5, 7, 9–12, 14, 18, 20, 25, 26, 28–32, 34, 36, 37, 37, 38–42, 44–47, 49, 50, 52–54, 56, 58–60, 62, 65, 66
- get.attr.gdsn, 30, 37, 53
- getfile.gdsn, 9, 38
- getfolder.gdsn, 39
- index.gdsn, 7, 25, 36, 39, 40, 44, 45, 48, 59
- is.element.gdsn, 41
- is.sparse.gdsn, 42
- lasterr.gds, 43
- ls.gdsn, 25, 36, 41, 44, 44, 46, 48, 59
- mclapply, 49
- mcmapply, 49
- moveto.gdsn, 28, 45
- name.gdsn, 41, 46, 48, 59
- objdesp.gdsn, 25, 44, 46, 47
- openfn.gds, 21, 22, 30, 37, 45, 49, 63
- parallel, 23
- permdim.gdsn, 50
- print.gds.class, 51
- print.gdsn.class (print.gds.class), 51
- put.attr.gdsn, 30, 37, 53
- read.gdsn, 7, 13, 15, 18, 20, 37, 42, 54, 56, 60, 66
- readex.gdsn, 7, 15, 18, 55, 56
- readmode.gdsn, 7, 27, 57
- rename.gdsn, 46, 59
- setdim.gdsn, 18, 51, 60
- show.gdsn.class-method (print.gds.class), 51
- showfile.gds, 61
- summarize.gdsn, 62
- sync.gds, 22, 29, 63, 63
- system.gds, 64
- unload.gdsn, 65
- write.gdsn, 7, 13, 18, 37, 55, 56, 60, 66