Package 'cmapR'

November 11, 2026

Type Package

Title CMap Tools in R

Date 2023-04-03

Version 1.24.0

Description The Connectivity Map (CMap) is a massive resource of perturbational gene expression profiles built by researchers at the Broad Institute and funded by the NIH Library of Integrated Network-Based Cellular Signatures (LINCS) program. Please visit https://clue.io for more information.
The cmapR package implements methods to parse, manipulate, and write common CMap data objects, such as annotated matrices and collections of gene sets.

License file LICENSE

Depends R (>= 4.0)

Imports methods, rhdf5, data.table, flowCore, SummarizedExperiment, matrixStats

Suggests knitr, testthat, BiocStyle, rmarkdown

VignetteBuilder knitr

biocViews DataImport, DataRepresentation, GeneExpression

URL https://github.com/cmap/cmapR

BugReports https://github.com/cmap/cmapR/issues

LazyData true

RoxygenNote 7.1.1

git_url https://git.bioconductor.org/packages/cmapR

git_branch RELEASE_3_23

git_last_commit 049877f

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Ted Natoli [aut, cre] (ORCID: <https://orcid.org/0000-0002-0953-0206>)

Maintainer Ted Natoli <ted.e.natoli@gmail.com>

Contents

align_matrices . . . 3
annotate.gct . . . 4
append.dim . . . 5
cdesc_char . . . 6
check_colnames . . . 6
check_dups . . . 7
distil . . . 7
ds . . . 8
extract.gct . . . 8
fix.datatypes . . . 10
GCT . . . 11
GCT-class . . . 12
gene_set . . . 12
ids . . . 13
is.wholenumber . . . 14
kd_gct . . . 14
lxb2mat . . . 15
mat . . . 16
melt.gct . . . 17
merge.gct . . . 18
merge_with_precedence . . . 19
meta . . . 20
na_pad_matrix . . . 21
parse.gctx . . . 21
parse.gmt . . . 22
parse.gmx . . . 23
parse.grp . . . 24
process_ids . . . 25
rank.gct . . . 26
read.gctx.ids . . . 27
read.gctx.meta . . . 28
robust_zscore . . . 29
subset.gct . . . 29
subset_to_ids . . . 30
threshold . . . 31
transpose.gct . . . 31
update.gctx . . . 32
write.gct . . . 33
write.gctx . . . 34
write.gctx.meta . . . 35
write.tbl . . . 36
write_gmt . . . 37
write_grp . . . 38

Index **39**

`align_matrices` *Align the rows and columns of two (or more) matrices*

Description

Align the rows and columns of two (or more) matrices

Usage

```
align_matrices(m1, m2, ..., L = NULL, na.pad = TRUE, as.3D = TRUE)
```

Arguments

`m1`	a matrix with unique row and column names
`m2`	a matrix with unique row and column names
`...`	additional matrices with unique row and column names
`L`	a list of matrix objects. If this is given, m1, m2, and ... are ignored
`na.pad`	boolean indicating whether to pad the combined matrix with NAs for rows/columns that are not shared by m1 and m2.
`as.3D`	boolean indicating whether to return the result as a 3D array. If FALSE, will return a list.

Value

an object containing the aligned matrices. Will either be a list or a 3D array

Examples

```
# construct some example matrices
m1 <- matrix(rnorm(20), nrow=4)
rownames(m1) <- letters[1:4]
colnames(m1) <- LETTERS[1:5]
m2 <- matrix(rnorm(20), nrow=5)
rownames(m2) <- letters[1:5]
colnames(m2) <- LETTERS[1:4]
m1
m2

# align them, padding with NA and returning a 3D array
align_matrices(m1, m2)

# align them, not padding and retuning a list
align_matrices(m1, m2, na.pad=FALSE, as.3D=FALSE)
```

annotate.gct *Add annotations to a GCT object*

Description

Given a GCT object and either a data.frame or a path to an annotation table, apply the annotations to the gct using the given keyfield.

Usage

```
annotate.gct(...)

annotate_gct(g, annot, dim = "row", keyfield = "id")

## S4 method for signature 'GCT'
annotate_gct(g, annot, dim = "row", keyfield = "id")
```

Arguments

...	arguments passed on to annotate_gct
g	a GCT object
annot	a data.frame or path to text table of annotations
dim	either 'row' or 'column' indicating which dimension of g to annotate
keyfield	the character name of the column in annot that matches the row or column identifiers in g

Value

a GCT object with annotations applied to the specified dimension

See Also

Other GCT utilities: melt.gct(), merge.gct(), rank.gct(), subset.gct()

Examples

```
gct_path <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
# read the GCT file, getting the matrix only
g <- parse_gctx(gct_path, matrix_only=TRUE)
# separately, read the column annotations and then apply them using
# annotate_gct
cdesc <- read_gctx_meta(gct_path, dim="col")
g <- annotate_gct(g, cdesc, dim="col", keyfield="id")
```

append.dim *Append matrix dimensions to filename*

Description

Append matrix dimensions to filename

Usage

```
append.dim(...)

append_dim(ofile, mat, extension = "gct")
```

Arguments

...	arguments passed on to append_dim
ofile	the file name
mat	the matrix
extension	the file extension

Details

This is a helper function that most users will not use directly

Value

a character string of the filename with matrix dimensions appended

See Also

Other GCTX parsing functions: GCT, fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gctx(), write.gct()

Examples

```
(filename <- cmapR:::append_dim("my.gctx.filename",
  matrix(nrow=10, ncol=15)))
```

`cdesc_char` *An example table of metadata, as would be parsed from or parse.gctx. Initially all the columns are of type character.*

Description

An example table of metadata, as would be parsed from or parse.gctx. Initially all the columns are of type character.

Usage

```
cdesc_char
```

Format

An object of class `data.frame` with 368 rows and 8 columns.

`check_colnames` *Check whether* `test_names` *are columns in the* `data.frame` *df*

Description

Check whether `test_names` are columns in the `data.frame` df

Usage

```
check_colnames(test_names, df, throw_error = TRUE)
```

Arguments

`test_names`	a vector of column names to test
`df`	the `data.frame` to test against
`throw_error`	boolean indicating whether to throw an error if any `test_names` are not found in `df`

Value

boolean indicating whether or not all `test_names` are columns of `df`

Examples

```
check_colnames(c("pert_id", "pert_iname"), cdesc_char) # TRUE
check_colnames(c("pert_id", "foobar"),
  cdesc_char, throw_error=FALSE)# FALSE, suppress error
```

check_dups *Check for duplicates in a vector*

Description

Check for duplicates in a vector

Usage

```
check_dups(x, name = "")
```

Arguments

x	the vector
name	the name of the object to print in an error message if duplicates are found

Value

silently returns NULL

Examples

```
# this will throw an erorr, let's catch it
tryCatch(
  check_dups(c("a", "b", "c", "a", "d")),
  error=function(e) print(e)
  )
```

distil *Collapse the rows or columns of a matrix using weighted averaging*

Description

This is equivalent to the 'modz' procedure used in collapsing replicates in traditional L1000 data processing. The weight for each replicate is computed as its normalized average correlation to the other replicates in the set.

Usage

```
distil(m, dimension = "col", method = "spearman")
```

Arguments

m	a numeric matrix where the rows or columns are assumed to be replicates
dimension	the dimension to collapse. either 'row' or 'col'
method	the correlation method to use

Value

a list with the following elements

values a vector of the collapsed values

correlations a vector of the pairwise correlations

weights a vector of the computed weights

Examples

```
m <- matrix(rnorm(30), ncol=3)
distil(m)
```

---

ds *An example of a GCT object with row and column metadata and gene expression values in the matrix.*

Description

An example of a GCT object with row and column metadata and gene expression values in the matrix.

Usage

```
ds
```

Format

An object of class `GCT` of length 1.

---

extract.gct *Exract elements from a GCT matrix*

Description

extract the elements from a `GCT` object where the values of `row_field` and `col_field` are the same. A concrete example is if g represents a matrix of signatures of genetic perturbations, and you wan to extract all the values of the targeted genes.

Usage

```
extract.gct(...)

extract_gct(
  g,
  row_field,
  col_field,
  rdesc = NULL,
  cdesc = NULL,
  row_keyfield = "id",
  col_keyfield = "id"
)
```

Arguments

...	arguments passed on to extract_gct
g	the GCT object
row_field	the column name in rdesc to search on
col_field	the column name in cdesc to search on
rdesc	a data.frame of row annotations
cdesc	a data.frame of column annotations
row_keyfield	the column name of rdesc to use for annotating the rows of g
col_keyfield	the column name of cdesc to use for annotating the rows of g

Value

a list of the following elements

mask a logical matrix of the same dimensions as ds@mat indicating which matrix elements have been extracted

idx an array index into ds@mat representing which elements have been extracted

vals a vector of the extracted values

Examples

```
# get the values for all targeted genes from a
# dataset of knockdown experiments
res <- extract_gct(kd_gct, row_field="pr_gene_symbol",
  col_field="pert_mfc_desc")
str(res)
stats::quantile(res$vals)
```

`fix.datatypes` *Adjust the data types for columns of a meta data frame*

Description

GCT(X) parsing initially returns data frames of row and column descriptors where all columns are of type character. This is inconvenient for analysis, so the goal of this function is to try and guess the appropriate data type for each column.

Usage

```
fix.datatypes(...)

fix_datatypes(meta)
```

Arguments

`...`	arguments passed on to `fix_datatypes`
`meta`	a data.frame

Details

This is a low-level helper function which most users will not need to access directly

Value

meta the same data frame with (potentially) adjusted column types.

See Also

Other GCTX parsing functions: `GCT`, `append.dim()`, `parse.gctx()`, `process_ids()`, `read.gctx.ids()`, `read.gctx.meta()`, `write.gctx.meta()`, `write.gctx()`, `write.gct()`

Examples

```
# meta data table with all character types
str(cdesc_char)
fixed <- cmapR:::fix_datatypes(cdesc_char)
# note how some column classes have changed
str(fixed)
```

GCT *Initialize an object of class* GCT

Description

Initialize an object of class GCT

Usage

```
GCT(
  mat = NULL,
  rdesc = NULL,
  cdesc = NULL,
  src = NULL,
  rid = NULL,
  cid = NULL,
  matrix_only = FALSE
)
```

Arguments

mat	a matrix
rdesc	a data.frame of row metadata
cdesc	a data.frame of column metadata
src	path to a GCT file to read
rid	vector of character identifiers for rows
cid	vector of character identifiers for columns
matrix_only	logical indicating whether to read just the matrix data from src

Details

If mat is provided, rid and cid are treated as the row and column identifiers for the matrix and are assigned to the rid and cid slots of the GCT object.

If mat is not provided but src is provided, rid and cid are treated as filters. Data will be read from the file path provided to src and will then be restricted to the character ids or integer indices provided to rid and cid. In a similar manner, matrix_only controls whether the row and column metadata are also read from the src file path.

Value

a GCT object

See Also

Other GCTX parsing functions: append.dim(), fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gctx(), write.gct()

Examples

```
# an empty object
(g <- GCT())
# with a matrix
# note we must specify row and column ids
(g <- GCT(mat=matrix(rnorm(100), nrow=10),
          rid=letters[1:10], cid=letters[1:10]))
# from file
gct_file <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
(g <- GCT(src=gct_file))
```

GCT-class *An S4 class to represent a GCT object*

Description

The GCT class serves to represent annotated matrices. The `mat` slot contains said data and the `rdesc` and `cdesc` slots contain data frames with annotations about the rows and columns, respectively

Slots

`mat` a numeric matrix

`rid` a character vector of row ids

`cid` a character vector of column ids

`rdesc` a `data.frame` of row descriptors

`rdesc` a `data.frame` of column descriptors

`src` a character indicating the source (usually file path) of the data

See Also

`parse_gctx`, `write_gctx`, `read_gctx_meta`, `read_gctx_ids`

visit `http://clue.io/help` for more information on the GCT format

---

gene_set *An example collection of gene sets as used in the Lamb 2006 CMap paper.*

Description

An example collection of gene sets as used in the Lamb 2006 CMap paper.

Usage

```
gene_set
```

Format

An object of class `list` of length 8.

Source

Lamb et al 2006 doi:10.1126/science.1132939

`ids` *Extract the or set row or column ids of a GCT object*

Description

Extract the or set row or column ids of a GCT object

Usage

```
ids(g, dimension = "row")

## S4 method for signature 'GCT'
ids(g, dimension = "row")

ids(g, dimension = "row") <- value

## S4 replacement method for signature 'GCT'
ids(g, dimension = "row") <- value
```

Arguments

`g`	the GCT object
`dimension`	the dimension to extract/update ['row' or 'column']
`value`	a character vector

Value

a vector of row ids

See Also

Other GCT accessor methods: `mat()`, `meta()`

Examples

```
# extract rids
rids <- ids(ds)
# extract column ids
cids <- ids(ds, "column")
# set rids
ids(ds) <- as.character(1:length(rids))
# set cids
ids(ds, "column") <- as.character(1:length(cids))
```

`is.wholenumber` *Check if x is a whole number*

Description

Check if x is a whole number

Usage

```
is.wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

`x`	number to test
`tol`	the allowed tolerance

Value

boolean indicating whether x is tol away from a whole number value

Examples

```
is.wholenumber(1)
is.wholenumber(0.5)
```

`kd_gct` *An example GCT object of knockdown experiments targeting a subset of landmark genes.*

Description

An example GCT object of knockdown experiments targeting a subset of landmark genes.

Usage

```
kd_gct
```

Format

An object of class GCT of length 1.

---

lxb2mat *Read an LXB file and return a matrix*

Description

Read an LXB file and return a matrix

Usage

```
lxb2mat(lxb_path, columns = c("RID", "RP1"), newnames = c("barcode_id", "FI"))
```

Arguments

lxb_path	the path to the lxb file
columns	which columns in the lxb file to retain
newnames	what to name these columns in the returned matrix

Value

a matrix

See Also

Other CMap parsing functions: parse.gmt(), parse.gmx(), parse.grp(), write_gmt(), write_grp()

Examples

```
lxb_path <- system.file("extdata", "example.lxb", package="cmapR")
lxb_data <- lxb2mat(lxb_path)
str(lxb_data)
```

mat *Extract or set the matrix of GCT object*

Description

Extract or set the matrix of GCT object

Usage

```
mat(g)

## S4 method for signature 'GCT'
mat(g)

mat(g) <- value

## S4 replacement method for signature 'GCT'
mat(g) <- value
```

Arguments

g	the GCT object
value	a numeric matrix

Value

a matrix

See Also

Other GCT accessor methods: ids(), meta()

Examples

```
# get the matrix
m <- mat(ds)
# set the matrix
mat(ds) <- matrix(0, nrow=nrow(m), ncol=ncol(m))
```

melt.gct *Transform a GCT object in to a long form* data.table *(aka 'melt')*

Description

Utilizes the melt.data.table function to transform the matrix into long form. Optionally can include the row and column annotations in the transformed data.table.

Usage

```
melt.gct(...)

melt_gct(
  g,
  suffixes = NULL,
  remove_symmetries = FALSE,
  keep_rdesc = TRUE,
  keep_cdesc = TRUE,
  ...
)

## S4 method for signature 'GCT'
melt_gct(
  g,
  suffixes = NULL,
  remove_symmetries = FALSE,
  keep_rdesc = TRUE,
  keep_cdesc = TRUE,
  ...
)
```

Arguments

...	further arguments passed along to data.table::merge
g	the GCT object
suffixes	the character suffixes to be applied if there are collisions between the names of the row and column descriptors
remove_symmetries	boolean indicating whether to remove the lower triangle of the matrix (only applies if g@mat is symmetric)
keep_rdesc	boolean indicating whether to keep the row descriptors in the final result
keep_cdesc	boolean indicating whether to keep the column descriptors in the final result

Value

a data.table object with the row and column ids and the matrix values and (optinally) the row and column descriptors

See Also

Other GCT utilities: annotate.gct(), merge.gct(), rank.gct(), subset.gct()

Examples

```
# simple melt, keeping both row and column meta
head(melt_gct(ds))

# update row/colum suffixes to indicate rows are genes, columns experiments
head(melt_gct(ds, suffixes = c("_gene", "_experiment")))

# ignore row/column meta
head(melt_gct(ds, keep_rdesc = FALSE, keep_cdesc = FALSE))
```

---

merge.gct *Merge two GCT objects together*

Description

Merge two GCT objects together

Usage

```
## S3 method for class 'gct'
merge(...)

merge_gct(g1, g2, dim = "row", matrix_only = FALSE)

## S4 method for signature 'GCT,GCT'
merge_gct(g1, g2, dim = "row", matrix_only = FALSE)
```

Arguments

...	arguments passed on to merge_gct
g1	the first GCT object
g2	the second GCT object
dim	the dimension on which to merge (row or column)
matrix_only	boolean idicating whether to keep only the data matrices from g1 and g2 and ignore their row and column meta data

Value

a GCT object

See Also

Other GCT utilities: annotate.gct(), melt.gct(), rank.gct(), subset.gct()

Examples

```
# take the first 10 and last 10 rows of an object
# and merge them back together
(a <- subset_gct(ds, rid=1:10))
(b <- subset_gct(ds, rid=969:978))
(merged <- merge_gct(a, b, dim="row"))
```

`merge_with_precedence` *Merge two* `data.frame`*s, but where there are common fields those in* `x` *are retained and those in* `y` *are dropped.*

Description

Merge two `data.frame`s, but where there are common fields those in `x` are retained and those in `y` are dropped.

Usage

```
merge_with_precedence(x, y, by, allow.cartesian = TRUE, as_data_frame = TRUE)
```

Arguments

`x`	the `data.frame` whose columns take precedence
`y`	another `data.frame`
`by`	a vector of column names to merge on
`allow.cartesian`	boolean indicating whether it's ok for repeated values in either table to merge with each other over and over again.
`as_data_frame`	boolean indicating whether to ensure the returned object is a `data.frame` instead of a `data.table`. This ensures compatibility with GCT object conventions, that is, the `rdesc` and `cdesc` slots must be strictly `data.frame` objects.

Value

a `data.frame` or `data.table` object

See Also

data.table::merge

Examples

```
(x <- data.table::data.table(foo=letters[1:10], bar=1:10))
(y <- data.table::data.table(foo=letters[1:10], bar=11:20,
  baz=LETTERS[1:10]))
# the 'bar' column from y will be dropped on merge
cmapR:::merge_with_precedence(x, y, by="foo")
```

`meta` *Extract the or set metadata of a GCT object*

Description

Extract the or set metadata of a GCT object

Usage

```
meta(g, dimension = "row")

## S4 method for signature 'GCT'
meta(g, dimension = "row")

meta(g, dimension = "row") <- value

## S4 replacement method for signature 'GCT'
meta(g, dimension = "row") <- value
```

Arguments

`g`	the GCT object
`dimension`	the dimension to extract/update ['row' or 'column']
`value`	a data.frame

Value

a data.frame

See Also

Other GCT accessor methods: `ids()`, `mat()`

Examples

```
# extract rdesc
rdesc <- meta(ds)
# extract cdesc
cdesc <- meta(ds, dim="column")
# set rdesc
meta(ds) <- data.frame(x=sample(letters, nrow(rdesc), replace=TRUE))
# set cdesc
meta(ds, dim="column") <- data.frame(x=sample(letters, nrow(cdesc),
  replace=TRUE))
```

na_pad_matrix *Pad a matrix with additional rows/columns of NA values*

Description

Pad a matrix with additional rows/columns of NA values

Usage

```
na_pad_matrix(m, row_universe = NULL, col_universe = NULL)
```

Arguments

m	a matrix with unique row and column names
row_universe	a vector with the universe of possible row names
col_universe	a vector with the universe of possible column names

Value

a matrix

Examples

```
m <- matrix(rnorm(10), nrow=2)
rownames(m) <- c("A", "B")
colnames(m) <- letters[1:5]
na_pad_matrix(m, row_universe=LETTERS, col_universe=letters)
```

parse.gctx *Parse a GCTX file into the workspace as a GCT object*

Description

Parse a GCTX file into the workspace as a GCT object

Usage

```
parse.gctx(...)

parse_gctx(fname, rid = NULL, cid = NULL, matrix_only = FALSE)
```

Arguments

...	arguments passed on to parse_gctx
fname	path to the GCTX file on disk
rid	either a vector of character or integer row indices or a path to a grp file containing character row indices. Only these indicies will be parsed from the file.
cid	either a vector of character or integer column indices or a path to a grp file containing character column indices. Only these indicies will be parsed from the file.
matrix_only	boolean indicating whether to parse only the matrix (ignoring row and column annotations)

Details

parse_gctx also supports parsing of plain text GCT files, so this function can be used as a general GCT parser.

Value

a GCT object

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gctx(), write.gct()

Examples

```
gct_file <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
(ds <- parse_gctx(gct_file))

# matrix only
(ds <- parse_gctx(gct_file, matrix_only=TRUE))

# only the first 10 rows and columns
(ds <- parse_gctx(gct_file, rid=1:10, cid=1:10))
```

---

parse.gmt *Read a GMT file and return a list*

Description

Read a GMT file and return a list

Usage

```
parse.gmt(...)

parse_gmt(fname)
```

Arguments

...	arguments passed on to parse_gmt
fname	the file path to be parsed

Details

parse_gmt returns a nested list object. The top level contains one list per row in fname. Each of these is itself a list with the following fields: - head: the name of the data (row in fname) - desc: description of the corresponding data - len: the number of data items - entry: a vector of the data items

Value

a list of the contents of fname. See details.

See Also

Visit http://clue.io/help for details on the GMT file format

Other CMap parsing functions: lxb2mat(), parse.gmx(), parse.grp(), write_gmt(), write_grp()

Examples

```
gmt_path <- system.file("extdata", "query_up.gmt", package="cmapR")
gmt <- parse_gmt(gmt_path)
str(gmt)
```

---

parse.gmx *Read a GMX file and return a list*

Description

Read a GMX file and return a list

Usage

```
parse.gmx(...)

parse_gmx(fname)
```

Arguments

...	arguments passed on to parse_gmx
fname	the file path to be parsed

Details

parse_gmx returns a nested list object. The top level contains one list per column in fname. Each of these is itself a list with the following fields: - head: the name of the data (column in fname) - desc: description of the corresponding data - len: the number of data items - entry: a vector of the data items

Value

a list of the contents of fname. See details.

See Also

Visit http://clue.io/help for details on the GMX file format

Other CMap parsing functions: lxb2mat(), parse.gmt(), parse.grp(), write_gmt(), write_grp()

Examples

```
gmx_path <- system.file("extdata", "lm_probes.gmx", package="cmapR")
gmx <- parse_gmx(gmx_path)
str(gmx)
```

---

parse.grp *Read a GRP file and return a vector of its contents*

Description

Read a GRP file and return a vector of its contents

Usage

```
parse.grp(...)

parse_grp(fname)
```

Arguments

...	arguments passed on to parse_grp
fname	the file path to be parsed

Value

a vector of the contents of fname

See Also

Visit http://clue.io/help for details on the GRP file format

Other CMap parsing functions: lxb2mat(), parse.gmt(), parse.gmx(), write_gmt(), write_grp()

Examples

```
grp_path <- system.file("extdata", "lm_epsilon_n978.grp", package="cmapR")
values <- parse_grp(grp_path)
str(values)
```

`process_ids` *Return a subset of requested GCTX row/colum ids out of the universe of all ids*

Description

Return a subset of requested GCTX row/colum ids out of the universe of all ids

Usage

```
process_ids(ids, all_ids, type = "rid")
```

Arguments

ids	vector of requested ids. If NULL, no subsetting is performed
all_ids	vector of universe of ids
type	flag indicating the type of ids being processed

Details

This is a low-level helper function which most users will not need to access directly

Value

a list with the following elements ids: a character vector of the processed ids idx: an integer list of their corresponding indices in all_ids

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), parse.gctx(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gctx(), write.gct()

Examples

```
gct_file <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
ids <- read_gctx_ids(gct_file)
processed_ids <- cmapR:::process_ids(ids[1:10], ids)
str(processed_ids)
```

`rank.gct` *Convert a GCT object's matrix to ranks*

Description

Convert a GCT object's matrix to ranks

Usage

```
rank.gct(...)

rank_gct(g, dim = "col", decreasing = TRUE)

## S4 method for signature 'GCT'
rank_gct(g, dim = "col", decreasing = TRUE)
```

Arguments

`...`	arguments passed on to `rank_gct`
`g`	the `GCT` object to rank
`dim`	the dimension along which to rank (row or column)
`decreasing`	boolean indicating whether higher values should get lower ranks

Value

a modified version of g, with the values in the matrix converted to ranks

See Also

Other GCT utilities: `annotate.gct()`, `melt.gct()`, `merge.gct()`, `subset.gct()`

Examples

```
(ranked <- rank_gct(ds, dim="column"))
# scatter rank vs. score for a few columns
m <- mat(ds)
m_ranked <- mat(ranked)
plot(m[, 1:3], m_ranked[, 1:3],
  xlab="score", ylab="rank")
```

`read.gctx.ids` *Read GCTX row or column ids*

Description

Read GCTX row or column ids

Usage

```
read.gctx.ids(...)

read_gctx_ids(gctx_path, dim = "row")
```

Arguments

`...`	arguments passed on to `read_gctx_ids`
`gctx_path`	path to the GCTX file
`dim`	which ids to read (row or column)

Value

a character vector of row or column ids from the provided file

See Also

Other GCTX parsing functions: `GCT`, `append.dim()`, `fix.datatypes()`, `parse.gctx()`, `process_ids()`, `read.gctx.meta()`, `write.gctx.meta()`, `write.gctx()`, `write.gct()`

Examples

```
gct_file <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
# row ids
rid <- read_gctx_ids(gct_file)
head(rid)
# column ids
cid <- read_gctx_ids(gct_file, dim="column")
head(cid)
```

read.gctx.meta *Parse row or column metadata from GCTX files*

Description

Parse row or column metadata from GCTX files

Usage

```
read.gctx.meta(...)

read_gctx_meta(gctx_path, dim = "row", ids = NULL)
```

Arguments

...	arguments passed on to read_gctx_meta
gctx_path	the path to the GCTX file
dim	which metadata to read (row or column)
ids	a character vector of a subset of row/column ids for which to read the metadata

Value

a data.frame of metadata

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), write.gctx.meta(), write.gctx(), write.gct()

Examples

```
gct_file <- system.file("extdata", "modzs_n25x50.gctx", package="cmapR")
# row meta
row_meta <- read_gctx_meta(gct_file)
str(row_meta)
# column meta
col_meta <- read_gctx_meta(gct_file, dim="column")
str(col_meta)
# now for only the first 10 ids
col_meta_first10 <- read_gctx_meta(gct_file, dim="column",
ids=col_meta$id[1:10])
str(col_meta_first10)
```

robust_zscore *Compoute robust z-scores*

Description

robust zscore implementation takes in a 1D vector, returns 1D vector after computing robust zscores rZ = (x-med(x))/mad(x)

Usage

```
robust_zscore(x, min_mad = 1e-06, ...)
```

Arguments

x	numeric vector to z-score
min_mad	the minimum allowed MAD, useful for avoiding division by very small numbers
...	further options to median, max functions

Value

transformed version of x

Examples

```
(x <- rnorm(25))
(robust_zscore(x))

# with min_mad
(robust_zscore(x, min_mad=1e-4))
```

subset.gct *Subset a gct object using the provided row and column ids*

Description

Subset a gct object using the provided row and column ids

Usage

```
## S3 method for class 'gct'
subset(...)

subset_gct(g, rid = NULL, cid = NULL)

## S4 method for signature 'GCT'
subset_gct(g, rid = NULL, cid = NULL)
```

Arguments

...	arguments passed on to subset_gct
g	a gct object
rid	a vector of character ids or integer indices for ROWS
cid	a vector of character ids or integer indices for COLUMNS

Value

a GCT object

See Also

Other GCT utilities: annotate.gct(), melt.gct(), merge.gct(), rank.gct()

Examples

```
# first 10 rows and columns by index
(a <- subset_gct(ds, rid=1:10, cid=1:10))

# first 10 rows and columns using character ids
# use \code{ids} to extract the ids
rid <- ids(ds)
cid <- ids(ds, dimension="col")
(b <- subset_gct(ds, rid=rid[1:10], cid=cid[1:10]))

identical(a, b) # TRUE
```

---

subset_to_ids *Do a robust data.frame subset to a set of ids*

Description

Do a robust data.frame subset to a set of ids

Usage

```
subset_to_ids(df, ids)
```

Arguments

df	data.frame to subset
ids	the ids to subset to

Value

a subset version of df

threshold *Threshold a numeric vector*

Description

Threshold a numeric vector

Usage

```
threshold(x, minval, maxval)
```

Arguments

x	the vector
minval	minium allowed value
maxval	maximum allowed value

Value

a thresholded version of x

Examples

```
x <- rnorm(20)
threshold(x, -0.1, -0.1)
```

transpose.gct *Transpose a GCT object*

Description

Transpose a GCT object

Usage

```
transpose.gct(...)

transpose_gct(g)

## S4 method for signature 'GCT'
transpose_gct(g)
```

Arguments

...	arguments passed on to transpose_gct
g	the GCT object

Value

a modified verion of the input GCT object where the matrix has been transposed and the row and column ids and annotations have been swapped.

Examples

```
transpose_gct(ds)
```

---

update.gctx *Update the matrix of an existing GCTX file*

Description

Update the matrix of an existing GCTX file

Usage

```
## S3 method for class 'gctx'
update(...)

update_gctx(x, ofile, rid = NULL, cid = NULL)
```

Arguments

...	arguments passed on to update_gctx
x	an array of data
ofile	the filename of the GCTX to update
rid	integer indices or character ids of the rows to update
cid	integer indices or character ids of the columns to update

Details

Overwrite the rows and columns of ofile as indicated by rid and cid respectively. rid and cid can either be integer indices or character ids corresponding to the row and column ids in ofile.

Value

silently returns NULL

Examples

```
## Not run:
m <- matrix(rnorm(20), nrow=10)
# update by integer indices
update_gctx(m, ofile="my.gctx", rid=1:10, cid=1:2)
# update by character ids
row_ids <- letters[1:10]
col_ids <- LETTERS[1:2]
update_gctx(m, ofile="my.gctx", rid=row_ids, cid=col_ids)

## End(Not run)
```

`write.gct` *Write a GCT object to disk in GCT format*

Description

Write a GCT object to disk in GCT format

Usage

```
write.gct(...)

write_gct(ds, ofile, precision = 4, appenddim = TRUE, ver = 3)
```

Arguments

`...`	arguments passed on to `write_gct`
`ds`	the GCT object
`ofile`	the desired output filename
`precision`	the numeric precision at which to save the matrix. See `details`.
`appenddim`	boolean indicating whether to append matrix dimensions to filename
`ver`	the GCT version to write. See `details`.

Details

Since GCT is text format, the higher `precision` you choose, the larger the file size. `ver` is assumed to be 3, aka GCT version 1.3, which supports embedded row and column metadata in the GCT file. Any other value passed to `ver` will result in a GCT version 1.2 file which contains only the matrix data and no annotations.

Value

silently returns NULL

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gctx()

Examples

```
# note this will create a GCT file in your current directory
write_gct(ds, "dataset", precision=2)
```

`write.gctx` *Write a GCT object to disk in GCTX format*

Description

Write a GCT object to disk in GCTX format

Usage

```
write.gctx(...)

write_gctx(
  ds,
  ofile,
  appenddim = TRUE,
  compression_level = 0,
  matrix_only = FALSE,
  max_chunk_kb = 1024
)
```

Arguments

...	arguments passed on to write_gctx
ds	a GCT object
ofile	the desired file path for writing
appenddim	boolean indicating whether the resulting filename will have dimensions appended (e.g. my_file_n384x978.gctx)
compression_level	integer between 1-9 indicating how much to compress data before writing. Higher values result in smaller files but slower read times.
matrix_only	boolean indicating whether to write only the matrix data (and skip row, column annotations)
max_chunk_kb	for chunking, the maximum number of KB a given chunk will occupy

Value

silently returns NULL

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx.meta(), write.gct()

Examples

```
# note this will create a GCT file in your current directory
write_gctx(ds, "dataset")
```

---

write.gctx.meta *Write a* data.frame *of meta data to GCTX file*

Description

Write a data.frame of meta data to GCTX file

Usage

```
write.gctx.meta(...)

write_gctx_meta(ofile, df, dimension = "row")
```

Arguments

...	arguments passed on to write_gctx_meta
ofile	the desired file path for writing
df	the data.frame of annotations
dimension	the dimension to annotate (row or column)

Value

silently returns NULL

See Also

Other GCTX parsing functions: GCT, append.dim(), fix.datatypes(), parse.gctx(), process_ids(), read.gctx.ids(), read.gctx.meta(), write.gctx(), write.gct()

Examples

```
## Not run:
# assume ds is a GCT object
write_gctx_meta("/my/file/path", cdesc_char, dimension="col")

## End(Not run)
```

`write.tbl` *Write a* `data.frame` *to a tab-delimited text file*

Description

Write a `data.frame` to a tab-delimited text file

Usage

```
write.tbl(...)

write_tbl(tbl, ofile, ...)
```

Arguments

`...`	additional arguments passed on to `write.table`
`tbl`	the `data.frame` to be written
`ofile`	the desired file name

Details

This method simply calls `write.table` with some preset arguments that generate a unquoted, tab-delimited file without row names.

Value

silently returns NULL

See Also

`write.table`

Examples

```
## Not run:
write_tbl(cdesc_char, "col_meta.txt")

## End(Not run)
```

write_gmt *Write a nested list to a GMT file*

Description

Write a nested list to a GMT file

Usage

```
write_gmt(lst, fname)
```

Arguments

lst	the nested list to write. See details.
fname	the desired file name

Details

lst needs to be a nested list where each sub-list is itself a list with the following fields: - head: the name of the data - desc: description of the corresponding data - len: the number of data items - entry: a vector of the data items

Value

silently returns NULL

See Also

Visit http://clue.io/help for details on the GMT file format

Other CMap parsing functions: lxb2mat(), parse.gmt(), parse.gmx(), parse.grp(), write_grp()

Examples

```
## Not run:
write_gmt(gene_set, "gene_set.gmt")

## End(Not run)
```

write_grp *Write a vector to a GRP file*

Description

Write a vector to a GRP file

Usage

```
write_grp(vals, fname)
```

Arguments

vals	the vector of values to be written
fname	the desired file name

Value

silently returns NULL

See Also

Visit http://clue.io/help for details on the GRP file format

Other CMap parsing functions: lxb2mat(), parse.gmt(), parse.gmx(), parse.grp(), write_gmt()

Examples

```
## Not run:
write_grp(letters, "letter.grp")

## End(Not run)
```

Index

∗ **CMap parsing functions**
  lxb2mat, 15
  parse.gmt, 22
  parse.gmx, 23
  parse.grp, 24
  write_gmt, 37
  write_grp, 38
∗ **GCT accessor methods**
  ids, 13
  mat, 16
  meta, 20
∗ **GCT utilities**
  annotate.gct, 4
  melt.gct, 17
  merge.gct, 18
  rank.gct, 26
  subset.gct, 29
∗ **GCT utilties**
  transpose.gct, 31
∗ **GCTX parsing functions**
  append.dim, 5
  fix.datatypes, 10
  GCT, 11
  parse.gctx, 21
  process_ids, 25
  read.gctx.ids, 27
  read.gctx.meta, 28
  write.gct, 33
  write.gctx, 34
  write.gctx.meta, 35
∗ **datasets**
  cdesc_char, 6
  ds, 8
  gene_set, 12
  kd_gct, 14
∗ **internal**
  append.dim, 5
  fix.datatypes, 10
  merge_with_precedence, 19
  process_ids, 25
  subset_to_ids, 30
  write.gctx.meta, 35

align_matrices, 3
annotate.gct, 4, *18*, *26*, *30*
annotate_gct *(*annotate.gct*)*, 4
annotate_gct,GCT-method *(*annotate.gct*)*, 4
append.dim, 5, *10, 11*, *22*, *25*, *27, 28*, *34, 35*
append_dim *(*append.dim*)*, 5

cdesc_char, 6
check_colnames, 6
check_dups, 7

data.frame, *4*, *6*, *19*, *30*
data.table, *17*, *19*
distil, 7
ds, 8

extract.gct, 8
extract_gct *(*extract.gct*)*, 8

fix.datatypes, *5*, 10, *11*, *22*, *25*, *27, 28*, *34, 35*
fix_datatypes *(*fix.datatypes*)*, 10

GCT, *5*, *10*, 11, *22*, *25*, *27, 28*, *34, 35*
GCT-class, 12
gene_set, 12

ids, 13, *16*, *20*
ids,GCT-method *(*ids*)*, 13
ids<- *(*ids*)*, 13
ids<-,GCT-method *(*ids*)*, 13
is.wholenumber, 14

kd_gct, 14

lxb2mat, 15, *23–25*, *37, 38*

mat, *13*, 16, *20*
mat,GCT-method *(mat)*, 16
mat<- *(mat)*, 16
mat<-,GCT-method *(mat)*, 16
melt.data.table, *17*
melt.gct, *4*, 17, *18*, *26*, *30*
melt_gct *(melt.gct)*, 17
melt_gct,GCT-method *(melt.gct)*, 17
merge.gct, *4*, *18*, 18, *26*, *30*
merge_gct *(merge.gct)*, 18
merge_gct,GCT,GCT-method *(merge.gct)*, 18
merge_with_precedence, 19
meta, *13*, *16*, 20
meta,GCT-method *(meta)*, 20
meta<- *(meta)*, 20
meta<-,GCT-method *(meta)*, 20

na_pad_matrix, 21

parse.gctx, *5*, *10, 11*, 21, *25*, *27, 28*, *34, 35*
parse.gmt, *15*, 22, *24, 25*, *37, 38*
parse.gmx, *15*, *23*, 23, *25*, *37, 38*
parse.grp, *15*, *23, 24*, 24, *37, 38*
parse_gctx, *12*
parse_gctx *(parse.gctx)*, 21
parse_gmt *(parse.gmt)*, 22
parse_gmx *(parse.gmx)*, 23
parse_grp *(parse.grp)*, 24
process_ids, *5*, *10, 11*, *22*, 25, *27, 28*, *34, 35*

rank.gct, *4*, *18*, 26, *30*
rank_gct *(rank.gct)*, 26
rank_gct,GCT-method *(rank.gct)*, 26
read.gctx.ids, *5*, *10, 11*, *22*, *25*, 27, *28*, *34, 35*
read.gctx.meta, *5*, *10, 11*, *22*, *25*, *27*, 28, *34, 35*
read_gctx_ids, *12*
read_gctx_ids *(read.gctx.ids)*, 27
read_gctx_meta, *12*
read_gctx_meta *(read.gctx.meta)*, 28
robust_zscore, 29

subset.gct, *4*, *18*, *26*, 29
subset_gct *(subset.gct)*, 29
subset_gct,GCT-method *(subset.gct)*, 29
subset_to_ids, 30

threshold, 31

transpose.gct, 31
transpose_gct *(transpose.gct)*, 31
transpose_gct,GCT-method *(transpose.gct)*, 31

update.gctx, 32
update_gctx *(update.gctx)*, 32

write.gct, *5*, *10, 11*, *22*, *25*, *27, 28*, 33, *35*
write.gctx, *5*, *10, 11*, *22*, *25*, *27, 28*, *34*, 34, *35*
write.gctx.meta, *5*, *10, 11*, *22*, *25*, *27, 28*, *34, 35*, 35
write.table, *36*
write.tbl, 36
write_gct *(write.gct)*, 33
write_gctx, *12*
write_gctx *(write.gctx)*, 34
write_gctx_meta *(write.gctx.meta)*, 35
write_gmt, *15*, *23–25*, 37, *38*
write_grp, *15*, *23–25*, *37*, 38
write_tbl *(write.tbl)*, 36