Package 'AnnotatedMatrix'

August 16, 2018

Title Parse rich metadata out of structured comments in Matrix files

Version 1.22.0

Date 2018-04-05

Author Charles Tilford [aut, cre]

Maintainer Charles Tilford <cran@agent.fastmail.com>

Description Primarily designed to read MatrixMarket flat files that contain structured comments describing row and column metadata.

URL https://github.com/maptracker/setfisher/tree/master/AnnotatedMatrix

BugReports https://github.com/maptracker/setfisher/issues

Depends R (>= 3.1)

Imports methods, CatMisc (>= 0.8), ParamSetI (>= 1.16), crayon, data.table, dplyr, magrittr, reshape2, biomaRt, Matrix

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat

Enhances dynamictable, canvasXpress, qgraph, igraph

NeedsCompilation no

R topics documented:

AnnotatedMatrix-class 3
annotatedMatrixExampleFile 6
annotatedMatrixParameter 7
appliedFilters 8
as.gmt 9
as.ijx 10
as.mtx 10
as.sidecar 11

autoFilter . . . 12
biomartCommonNames . . . 14
biomartGenomeVersions . . . 14
biomartVersion . . . 15
colDefs . . . 15
counts . . . 16
DOTaddAppliedFilter . . . 17
DOTautoLevel . . . 18
DOTbuildMatrix . . . 19
DOTdetailZeroedRowCol . . . 19
DOTfieldDescriptions . . . 20
DOTfilterDetails . . . 20
DOTmakeTempFile . . . 21
DOTreadMatrix . . . 22
extendDataTable . . . 22
file . . . 23
filesToMTX . . . 24
filterByCount . . . 25
filterByFactorLevel . . . 26
filterById . . . 27
filterByMetadata . . . 29
filterByScore . . . 31
filterLog . . . 32
filterSummary . . . 33
findMatrices . . . 34
fromRDS . . . 35
generateMtxRds . . . 36
inst.path.package . . . 37
is.factor . . . 37
levels . . . 38
lvlVal . . . 39
makeEnsemblMatrix . . . 40
map . . . 40
matObj . . . 44
matrixFromLists . . . 45
matrixMD . . . 46
matrixRaw . . . 47
matrixText . . . 48
matrixUse . . . 49
melt . . . 49
metadata . . . 50
metadata_keys . . . 51
nnZero . . . 52
normalizePercent . . . 53
parseMetadataSidecar . . . 54
parse_GMT_file . . . 55
parse_IJX_file . . . 55
parse_ListOfLists_file . . . 56

parse_MatrixMarket_file 57
parse_Text_file 57
populated 58
preCacheMatrixDirectory 59
print.mapFilter 60
print.mapResult 60
product 61
rdsIsCurrent 63
removeEmpty 64
reset 65
rowcolchanges 66
rowcolnames 67
setFilters 68
sidecarFiles 69
smallNumberFormatter 69
strict.unique 70
takeLowestThing 71
uniqueNames 72

Index **73**

`AnnotatedMatrix-class` *Annotated Matrix*

Description

Annotated sparse matrix for capturing query lists, identifier mappings and ontology lookups

Details

```
## Object creation
AnnotatedMatrix( help=TRUE ) # Show this help

## In general:
myAnnMat <- AnnotatedMatrix( file=NA, params=NA, autofilter=TRUE, ...)
## Specific toy example matrix:
myAnnMat <- AnnotatedMatrix( annotatedMatrixExampleFile() )

myAnnMat$help()                 # High-level help
myAnnMat$ANYMETHOD( help=TRUE ) # Each method has detailed help available
```

AnnotatedMatrix is a heavy wrapper around the Matrix package, in particular the sparse matrix dgTMatrix class. It was built with an eye toward using the MatrixMarket file format, an efficient but unfortunately simple (no structured annotation) file format. This package will parse information from the comments (' MatrixMarket files to generate rich objects for enhanced filtering and reporting.

The driving motivation for the package is to support enrichment analysis (Fisher's exact test) in a reproducible research framework. The package is also very well suited for identifier mapping (for example, gene symbols to gene accessions).

Fields

`file` Path to file the matrix was loaded from

`fromRDS` Logical, true if the loaded file was an RDS object

`log` EventLogger object holding log (activity) entries

`matrixRaw` The Matrix object as loaded from the file, will not be altered by the object.

`matrixUse` The Matrix object after manipulation by any applied filters

`matrixMD` data.table holding metadata associated with the matrix

`lvlVal` Character array of level names for factor matrices

`filterLog` data.frame storing filtering events that transpire during the pruning of matrices prior to analysis.

`setFilters` Human- and machine-readable character vector of filters that have been applied

`rowChanges` Named character vector of any row names that needed changing. Values are the original name, names are the names after processing with make.unique()

`colChanges` As per rowChanges, but for column names

`colDefs` list of definitions for metadata columns

Methods

`appliedFilters(new = NULL, help = FALSE)` Get applied filters as a vector of readable/parsable strings

`as.gmt(obj = NULL, transpose = FALSE, file = NULL, help = FALSE, ...)` Convert the active matrix into GMT text representation

`as.ijx(file = NULL, sep = "\t", obj = NULL, help = FALSE)` Serialize matrix as IJX row/col/val three-column format

`as.mtx(obj = NULL, file = NULL, help = FALSE, ...)` Convert the active matrix into Matrix Market text representation

`autoFilter(recursive = TRUE, verbose = TRUE, help = FALSE)` Automatically apply filters defined in the parameters

`cCounts(obj = NULL, help = FALSE, ...)` Return the counts of non-zero rows for each column

`cNames(new = NULL, raw = FALSE, nonzero = FALSE, reason = NA, help = FALSE)` Get/set colnames for the matrix

`filterByCount(MARGIN, min = NULL, max = NULL, relative = TRUE, filterEmpty = FALSE, reason = NA, he` Filter rows or columns based on number of non-zero connections

`filterByFactorLevel(x, keep = TRUE, ignore.case = TRUE, filterEmpty = FALSE, reason = NA, help = FA` Zero-out cells in the matrix that fail factor criteria

`filterById(id, MARGIN = NULL, keep = FALSE, ignore.case = TRUE, exact = TRUE, filterEmpty = FALSE,` Filter rows or columns based on specific IDs

`filterByMetadata(key, val, MARGIN = NULL, keep = TRUE, type = "like", op = "or", filterEmpty = FALS` Zero-out rows or columns that match metadata criteria

`filterByScore(min = NA, max = NA, filterEmpty = FALSE, reason = NA, help = FALSE)` Filter matrix cells by simple numeric tests

`filterSummary(reason = TRUE, help = FALSE)` Human-readable overview of filters and counts of rows/cols removed

`help(color = NULL, help = FALSE)` Display high-level help about all object methods

`initialize(params = NA, paramDefinitions = NA, help = FALSE, ...)` Create a new ParamSetI object; Invoked with ParamSetI(...)

`is.factor(help = FALSE)` TRUE if the matrix is a pseudo-factor, otherwise FALSE

`levels(asFactor = FALSE, help = FALSE)` Returns factor levels, if appropriate. If not, returns NULL

`map(input = NULL, via = NULL, format = "data.frame", ignore.case = TRUE, keep.best = FALSE, column.` Map (convert) names from one dimension of the matrix to the other

`matObj(raw = FALSE, transpose = FALSE, help = FALSE)` Return the current filtered state of the dgTMatrix sparse matrix

`matrixText(pad = "", useObj = NULL, fallbackVar = NULL, compact = FALSE, color = NULL, help = FALSE` Generate a compact text summary of the object, used by show()

`melt(obj = NULL, file = NULL, named.dims = TRUE, help = FALSE, ...)` Represent matrix as data.frame with three rows: Row, Col, Val

`metadata(id = NULL, key = NULL, na.rm = TRUE, drop = TRUE, verbose = TRUE, help = FALSE)` Recover metadata for specific IDs and/or columns

`metadata_keys(help = FALSE)` Get all metadata keys as a character vector

`nnZero(obj = NULL, help = FALSE, ...)` Return the count of non-zero elements in the matrix

`populatedCols(obj = NULL, help = FALSE, ...)` Determine which columns have at least one non-zero assignment

`populatedRows(obj = NULL, help = FALSE, ...)` Determine which rows have at least one non-zero assignment

`product(mat2, dim1 = NULL, dim2 = NULL, valfunc = NULL, levels = NULL, ignore.zero = TRUE, ignore.c` Take a product of this matrix AxB with another BxC to yield AxC

`rCounts(obj = NULL, help = FALSE, ...)` Return the counts of non-zero columns for each row

`removeEmpty(reason = NA, help = FALSE)` Remove all rows and columns that consist only of zeroes

`removeEmptyCols(reason = NA, help = FALSE)` Remove all rows that consist only of zeroes

`removeEmptyRows(reason = NA, help = FALSE)` Remove all rows that consist only of zeroes

`reset(help = FALSE)` Reset the matrix to the 'raw' (unfiltered) state

`rNames(new = NULL, raw = FALSE, nonzero = FALSE, reason = NA, help = FALSE)` Get/set rownames for the matrix

`show(...)` A wrapper for showLog

Examples

```
## In most circumstances you will provide only a file path, but
## optionally a list of parameters and a flag to turn off autoFilter
## can be provided as well.
```

```
## A toy (small) symbol-to-gene mapping matrix is provided with the
## package:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )

s2e$help()         # Compact reminder of fields and methods
s2e                # Summary of loaded object
s2e$map(help=TRUE) # Help for the map() method
AnnotatedMatrix(help=TRUE) # display this help

## Load a toy symbol-to-gene mapping matrix and use it to convert
## some genes to Entrez Gene IDs
demo("geneSymbolMapping", package="AnnotatedMatrix", ask=FALSE)

## Work with different file formats
demo("fileFormats", package="AnnotatedMatrix", ask=FALSE)

## Different approaches to handling non-unique mappings:
demo("managingMultiplicity", package="AnnotatedMatrix", ask=FALSE)

## Working with metadata
demo("workingWithMetadata", package="AnnotatedMatrix", ask=FALSE)
```

`annotatedMatrixExampleFile` *Annotated Matrix Example File*

Description

Path to a small example file used for demonstrations

Usage

```
annotatedMatrixExampleFile(file = "Symbol-To-Gene.mtx", list = FALSE)
```

Arguments

`file`	Default `"Symbol-To-Gene.mtx"`. The file to recover from inst/
`list`	Default `FALSE`. If TRUE, then list all available example files

Details

This is just a wrapper around the internal .makeTempFile() function that pulls up a gene symbol -to-> Entrez Gene ID mapping matrix (subset of human genes). The file is factorized and has metadata on both dimensions.

Value

The path to the file requested, unless list is TRUE, which will return a listing of all files in `extdata/`

Examples

```
ef1 <- annotatedMatrixExampleFile()
ef2 <- annotatedMatrixExampleFile("Machines.gmt")
```

`annotatedMatrixParameter` *Annotated Matrix Parameter*

Description

Get/Set a parameter associated with your AnnotatedMatrix session

Usage

```
annotatedMatrixParameter(key, value = NULL, default = NULL,
  as.list = NULL)
```

Arguments

`key`	Required, a character vector of keys to recover. Keys are case-insensitive, though the names on the returned values will honor the capitalization of the passed 'key' parameter.
`value`	Optional new value to be assigned to the keys
`default`	Default `NULL`. If the parameter is NULL, and the corresponding R option is also NULL, this value will be substitued instead
`as.list`	Default `NULL`. If `TRUE`, the return value will be a named list. `FALSE` will return the unlist()ed values (which may be odd if the parameters have mixed data types). The default of NULL will return a list if two or more keys were requested, otherwise the single value

Details

Manages values that you will likely want to be constant during an analysis, but you may wish to customize between analyses, or between users or organizations.

If a requested key is NULL, the function will also check if an R option is set for that key. For example, if you request 'dir', the option 'annotatedmatrixdir' will be checked if the parameter is NULL.

Value

If as.list=TRUE, a named list. If as.list=FALSE, a single value if a single key was requested, otherwise an unlist()ed vector.

Recognized Keys

- `dir` - A default directory where matrix files are stored
- `ensemblhost` - Default Ensmebl host domain to use

`appliedFilters` *Applied Filters*

Description

AnnotatedMatrix object method to show/add applied filters

Arguments

`new`	Default NULL, optional character vector of new filters to apply. NOT YET IMPLEMENTED
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$appliedFilters( help=TRUE )

myObject$appliedFilters( new=NULL )
```

Each time a method is run that filters the matrix, the criteria used are serialized as a text snippet, and the snippet is added to the setFilters vector. These snippets can be used as a record of filtering operations. In addition, some or all of them can be passed to this function to re-run the filters

The parsing feature (via the `new` parameter) is aspirational and is not yet implemented.

Value

The setFilters field, a character vector added to setFilters

See Also

setFilters, filterByScore, filterByFactorLevel, filterByMetadata, rNames, cNames, reset, filterSummary,

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## The sample file has automated filters applied to it:
s2e$appliedFilters()
```

as.gmt *As GMT*

Description

AnnotatedMatrix object method to represent matrix as GMT format

Arguments

obj	Default NULL, the matrix object to serialize. If NULL, will call matObj to recover the current filtered matrix.
transpose	Default FALSE, which will treat rows as sets and columns as set members. TRUE will instead treat the columns as sets.
file	Default NULL. If not NULL, will be treated as a file path and passed to cat.
...	Will be passed to matObj.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$as.gmt( help=TRUE )

myObject$as.gmt( obj=NULL, transpose=FALSE, file=NULL, ...)
```

Will represent the matrix as GMT format, a simple flat file format that allows for basic metadata (names and descriptions).

Value

The GMT text, invisibly if file is not NULL

See Also

GMT Format at Broad Institute

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## Generate GMT data for the symbols associated for each gene in
## the example:
s2e$as.gmt( transpose=TRUE )
```

as.ijx *As IJX*

Description

AnnotatedMatrix object method to convert matrix into three column format

Arguments

file	Default NULL. When NULL, a character matrix is returned. If not NULL, the serialized data are written to the specified file, as well as being invisibly returned.
sep	Default "\t", the field separator
obj	Default NULL, will be passed to melt. If NULL, then a character matrix will be returned, otherwise a string representing the text is generated.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

This method is very similar to melt, which it calls to structure the data. Unlike melt, it is primarily designed to return a block of text (rather than a data.frame), or a character matrix. It can also be used to write the text block to a file.

See Also

melt to get a three column data.frame

as.mtx *As MTX*

Description

AnnotatedMatrix object method to represent matrix as Matrix Market format

Arguments

obj	Default NULL, the matrix object to serialize. If NULL, will call matObj to recover the current filtered matrix.
file	Default NULL. If not NULL, will be treated as a file path and passed to cat.
...	Will be passed to matObj
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$as.mtx( help=TRUE )

myObject$as.mtx( obj=NULL, file=NULL, ...)
```

Will represent the matrix as Matrix Market format, a basic flat file format that is well-suited for encoding sparse matrices. Additionally, rich metadata will be added as comments that can then be parsed by AnnotatedMatrix.

################################## ### THIS FEATURE IS NOT YET CODED ##############################

Value

The MTX text, invisibly if file is not NULL

See Also

MTX Format at Broad Institute

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## Filter the matrix so that it is smaller:
s2e$rNames(c("HFH2","AIS1","p63"))
s2e$as.mtx()
```

---

as.sidecar *As Metadata Sidecar*

Description

AnnotatedMatrix object method to export metadata as a 'sidecar' file

Arguments

obj	Default NULL, the matrix object to serialize. If NULL, will call matObj to recover the current filtered matrix.
file	Default NULL. If not NULL, will be treated as a file path and passed to cat.
sep	Default "\t", the separator to use between columns. The special value NULL will return a character matrix object rather than a text representation.
...	Will be passed to matObj. If you wish to transpose the matrix, you can pass the argument here.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$as.sidecar( help=TRUE )

myObject$as.sidecar( obj=NULL, file=NULL, sep="\t", ...)
################################## ### THIS FEATURE IS NOT YET CODED ##############################
```

Value

If sep is NULL, a character matrix. Otherwise the text, invisibly if file is not NULL

---

autoFilter *Automatic Filters*

Description

AnnotatedMatrix object method to apply filters described in parameters

Arguments

recursive	Default TRUE. Because some filters check the number of populated rows or columns, it is possible that the application of a latter filter will result in rows or columns that passed a prior one to now be considered failing. Recursive will keep reapplying filters into no further changes (based on zeroed cell count) are observed. If you desire the final matrix to pass all filters, leave recursive as TRUE.
verbose	Default TRUE, which displays a message summarizing the count of zeroed elements (cells, rows and columns).
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$autoFilter( help=TRUE )

myObject$autoFilter( recursive=TRUE, verbose=TRUE )
```

The matrix will recognize several parameters as defining filter operations. While these parameters could be set 'manually' in code, they are of primary utility in allowing annotated matrix files to "self describe" default reasonable starting filters for the matrix.

While you can run `autoFilter()` manually (for example, after a `reset()`, it is by default run automatically on Matrix load. To **prevent** autofiltering on load, pass `autofilter=FALSE` when calling `AnnotatedMatrix()`.

The following parameters are identified as specifying a filter request:

- MinScore MaxScore

- MinRowCount MaxRowCount
- MinColCount MaxColCount
- KeepRow TossRow
- KeepCol TossCol
- KeepLevel TossLevel
- TossMeta

Calling `$showParameters( na.rm=FALSE)` will list all standard parameters and their values; Not all of these will be relevant to autoFilter, but those that are will indicate as much in the description.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterByScore, filterByFactorLevel, filterByMetadata, filterByCount, filterById, reset, filterSummary, appliedFilters, showParameters

Examples

```
## The example matrix has two default filters defined:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e # Summary of the matrix

## Only one removed any entries, though:
s2e$filterSummary()
## This is because the cells impacted by the second filter (on
## "Description" metadata) are fully contained by those already
## removed by the first (a factor filter removing level "Unknown")

## We can remove all filters:
s2e$reset()
s2e

## ... and then reapply the automatic filters if we wish:
s2e$autoFilter()

## Show the currently set parameters:
s2e$showParameters()

## Show all recognized parameters:
s2e$showParameters( na.rm=FALSE )
```

biomartCommonNames *BioMart Common Names*

Description

Dataset-to-Common-Name lookup, providing "human friendly" species names

Usage

```
biomartCommonNames(...)
```

Arguments

...	Passed to biomartVersion

Details

Recovers the species common names used in BioMart, keyed by dataset name associated with each.

Value

A vector of species common names (eg "Crab-eating macaque") with dataset names as vector names

biomartGenomeVersions *Current BioMart Genome Versions*

Description

Return a vector of genome versions in the current version of BioMart

Usage

```
biomartGenomeVersions(...)
```

Arguments

...	Passed to biomartVersion

Details

Recovers the genomic versions used in BioMart, keyed by dataset name associated with each.

Value

A vector of genome build tokens (eg "") with dataset names as vector names

See Also

biomartVersion

biomartVersion *Current BioMart Version*

Description

Report the current 'live' version of BioMart

Usage

```
biomartVersion(host = annotatedMatrixParameter("ensemblhost", default =
  "www.ensembl.org"), ...)
```

Arguments

host	Will be passed to biomaRt, defaults to the annotatedMatrixParameter "ensemblhost", falling back to www.ensembl.org.
...	Passed to listMarts. In particular, 'host' may be useful to access a preferred Ensembl server.

Details

Gets available Marts and looks for the one named "Ensembl Genes ##", returning the actual value of '##' as the version.

Value

NA if there is a problem, otherwise a single integer value representing the version, with a name representing the name of the Mart.

colDefs *Column Definitions*

Description

Internal AnnotatedMatrix field holding metadata column descriptions

Details

This list holds optional descriptive text for the metadata columns associated with the matrix. It can be set 'automatically' by ' ColumnDescription' comments in an MTX file, but in theory could be 'manually' set by code as well.

The descriptions will be shown as 'comments' when available metadata columns are summarized, and will be included in the "Columns" attribute of map output.

Value

A list, keyed to column names with descriptive text as values

See Also

matrixMD, map

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$colDefs

# Definitions are also shown in summary text:
s2e

# They are also embedded in the attributes of $map() results:
myMap <- s2e$map('AIRE1')
attr(myMap, "Columns")[[ "Output Symbol" ]]
```

---

counts *Row and Column Counts*

Description

AnnotatedMatrix object methods to count non-zero entries for rows or columns

Arguments

obj	Default NULL, which will recover the matrix using matObj. Optionally can be a user-provided sparse Matrix.
help	Default FALSE. If TRUE, show this help and perform no other actions.
...	Passed to matObj (for example, raw)

Details

```
## Method Usage:
myObject$rCounts( help=TRUE )
myObject$cCounts( help=TRUE )

myObject$rCounts( obj=NULL )
myObject$cCounts( obj=NULL )
```

For $rCounts(), return the counts of non-zero columns for each row. Same for $cCounts(), which will return the counts of non-zero rows for each column.

A value of zero indicates that the row (or column) does not have any "connections" to the other dimension. A value of 1 indicates a single unique connection.

Value

A named vector of integers, which will be as long as the number of rows (for rCounts) or columns (for cCounts). Names are taken from the dimension names.

See Also

populated, nnZero

Examples

```
## Example matrix has gene symbols in rows, gene IDs in columns
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
symCount <- s2e$rCounts()

# The matrix is loaded with an automatic filter. Which symbols no
# longer connect to any IDs?
names( symCount[ symCount == 0 ] )

# Which symbols are highly ambiguous (assigned to many genes)?
symCount[ symCount > 3 ]
# (Note that this matrix is case-sensitive, and includes "p40" and "P40")

idCount <- s2e$cCounts()
# Which genes have a lot of symbols assigned to them?
idCount[ idCount > 7 ]
# What are they?
s2e$map( names(idCount[ idCount > 7 ]) )
```

DOTaddAppliedFilter *Add Applied Filter*

Description

Internal AnnotatedMatrix object method to extend the list of applied filters

Arguments

key	Required, a key indicating the filter type, eg 'SCORE' or 'COUNT'
val	Required, text defining the filter parameters
com	Default NULL, optional comment (reason) for the filter
pre	Default NULL, optional text (eg operator) to go before the values
pro	Default NULL, text (eg modifiers) to go after the values
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly

Helper method to add new filter entries to setFilters.

Value

A single character string representing the fitler text added to setFilters

See Also

filterLog, filterSummary, appliedFilters

`DOTautoLevel` *Auto Level*

Description

Internal AnnotatedMatrix object method to generate new hybrid factor levels

Arguments

`vals`	Required, vector of one or more factor levels
`sep`	Default ',', the token used to separate multiple level names
`decreasing`	Default TRUE, which will sort factor values from largest to smallest (in order to put the 'best'-scored factor at the front of the list)
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly

Takes a set of factor levels (character names), combines them via paste, generates a new factor level if needed, and returns the factor index. Will extend the `lvlVal` internal field as needed.

This function is used by map to generate synthetic factor levels when `collapse` causes multiple rows to be merged into a single one.

Value

An integer representing the factor level

See Also

map

DOTbuildMatrix *Build Matrix*

Description

Internal AnnotatedMatrix object method to generate matrix 'by hand'

Arguments

metadata	Default NULL. Optional metadata table. Should be a data.table keyed to column 'id'.
levels	Default NULL. Optional character vector of levels. Will cause the matrix to be treated as a pseudo-factor.
cols	Default NULL. Optional metadata column definitions. Should be a list with column names as keys, human-readable descriptions as values.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly. It is called automatically when a new object is created with AnnotatedMatrix(someDataStructure).

Value

At the moment, just TRUE.

See Also

.readMatrix

DOTdetailZeroedRowCol *Detail Zeroed Rows and Columns*

Description

Internal AnnotatedMatrix object method to tally filtered rows and cols

Details

Internal method, should not be called directly

Takes a SparseMatrix and a (previously calculated) logical vector of 'failed' i+j triples, and determines which (if any) rows and columns went from populated to unpopulated (all zeroes). Adds entries to filterLog to record them

DOTfieldDescriptions *Field Descriptions*

Description

Internal AnnotatedMatrix object method providing descriptive text to object fields

Arguments

update	Default TRUE, which will modify the internal object field to have the two attached objects.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly

Helper method that adds descriptive attributes to the fields held by the object. This is an attempt to make the object self-documenting. Two fields are added: "Description", which provides a one-sentance description of the field, and "Help", which is a copy-and-pastable help command to recover the man page for the field

Value

A list, with field variable names as names, and descriptive text as values.

See Also

help

DOTfilterDetails *Filter Details*

Description

Internal AnnotatedMatrix object method to add filtered names to the log

Arguments

id	Required, a vector of one or more IDs (row or column names) that have been filtered out.
type	Optional type of filter, generally Row, Col or Cell
metric	Optional human-readable criteria for the filter, such as "Score > 3.14"
reason	Optional human-readable rationale for why the filter was applied, such as "Remove low-confidence assignments"
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly

Helper method to add new filter entries to filterLog.

Value

The filterLog filed (data.table) itself

See Also

filterLog, filterSummary, appliedFilters

DOTmakeTempFile *Make Temp File*

Description

Internal method to help manage example files in extdata/

Usage

```
.makeTempFile(name, pkg = "AnnotatedMatrix")
```

Arguments

name	Required, the basename of the file
pkg	Default "AnnotatedMatrix", the package name

Details

When matrix files are loaded, a .rds version is generated to aid in rapid loading in the future. These files are made alongside the original flat file. This could cause problems if the file is from inst/extdata, so this method takes a requested file and copies it to a temporary location before loading. It will also copy sidecar files used by the 'primary' file.

See Also

sidecarFiles

Examples

```
tmpFile <- AnnotatedMatrix:::.makeTempFile("Symbol-To-Gene.mtx")
```

DOTreadMatrix *Read Matrix Data*

Description

Internal AnnotatedMatrix object method to parse file contents

Arguments

format	Default "" (empty string). Optional format name, if empty the format will be deduced by the file suffix.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

Internal method, should not be called directly. It is called automatically when a new object is created with `AnnotatedMatrix(pathToSomeFile)`.

This method will take the file field as a file path and attempt to parse it as a matrix. Once parsed, the method will create a serailized RDS object. Future attempts to parse the matrix file will recognize the RDS object and quickly parse it instead.

Value

A list containing the Matrix object, metadata as a data.table, rowChanges and colChanges, levels as lvlVal, and any default parameters parsed from the file.

See Also

The individual format parsing functions: parse_MatrixMarket_file, linkfilesToMTX, linkparse_ListOfLists_file, linkparse_Text_file, linkparse_GMT_file. Also .buildMatrix, used to generate objects from data structures (rather than files)

extendDataTable *Extend Data Table*

Description

Append both new rows and new columns from one data.table to another

Usage

```
extendDataTable(x, y, key = "id")
```

Arguments

x	Required, the 'target' data.table
y	Required, the 'source' data.table, which will be added to or over-write the corresponding rows/columns in x
key	Default 'id', the column to merge on

Details

Designed to grow the metadata data.table as both new rows and columns are added, while keeping both rows and columns distinct. There may be a more elegant way to do this with native data.table methods, but I haven't found it, and not for want of trying.

Bad things can happen with data.tables if the key column is an integer - when subsetting, these will generally be interpreted as row numbers, which will oftenot correlate to the rows holding the anticipated value, and are also fluid depending on the key(s) being set on the DT (and perhaps on prior operations?) For this reason, the key should be a column of type character.

Value

A new data.table with new rows and columns merged. Will not alter input values.

Examples

```
library('data.table')
## Again, the key column should be character mode
dt1 <- data.table(a = c("1","2","3"), b = c("apple","banana", "cherry"),
                  key='a')
dt2 <- data.table(a = c("2","4","5"), b = c("BLUEBERRY","DATE","EGGPLANT"),
                  c = c("beep","ding","eek"), key='a')
extendDataTable(dt1, dt2, key='a')
```

---

file *Matrix File Path*

Description

Internal AnnotatedMatrix field holding the path to the matrix file

Details

This is simply the path to the raw file (or directory of files) that define the matrix.

Value

A character vector with a single value

See Also

fromRDS, .readMatrix

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$file
```

`filesToMTX` *Files to MatrixMarket*

Description

Converts a collection of files to a single MatrixMarket file

Usage

```
filesToMTX(files, output = "SparseMatrix.mtx", filter = NULL,
  rename = NULL)
```

Arguments

`files`	Required, a vector of file paths
`output`	Default "SparseMatrix.mtx", the filename of the MatrixMarket file being generated
`filter`	Optional function used to filter set elements. It should take as input a character vector and return a character vector representing the desired (kept) elements.
`rename`	Optional function used to generate list names. It should take as input the file path (single string) and return a single string that will be used as a name for that list

Examples

```
## Not run:
  # Find all files names "geneList#.txt" and extact Ensembl Gene IDs
  # from them into a single MTX file
  filesToMTX(files=list.files(pattern='geneList[0-9]+.txt'),
             filter=function(x) x[ grepl("^ENSG\\d+$", x) ],
             output="myEnsemblLists.mtx")

## End(Not run)
```

`filterByCount` *Filter by Count*

Description

AnnotatedMatrix object method to filter rows and columns by number of connections

Arguments

`MARGIN`	Required, defines the dimension to be filtered. Can specify rows with '1' or 'row', columns with '2' or 'col'.
`min`	Default NULL, optional minimum count. Will be normalized with normalizePercent, which will interpret integers and numeric values as absolute counts, and numeric strings ending with '%' as percentages of the total dimension length.
`max`	Default NULL, optional maximum score, same input considerations as `min`.
`relative`	Default TRUE. If true, then relative values (percentages) provided for `min` and `max` will be computed against the currently populated state of the matrix. For example, if a matrix has 1000 rows, and the minimum threshold is 10 the minimum threshold is 40 when `relative=TRUE`, but 100 when `relative=FALSE`. This setting can dramatically change the course of recursive filtering, where two or more filters are repeatedly reapplied until no further changes are seen. Note that autoFilter by default will operate recursively.
`filterEmpty`	Default FALSE. If true, then rows or columns that are empty after filtering will be removed from the matrix using removeEmpty. If no cells were affected by the `filterByCount` call, then `removeEmpty` will not be called. If you wish to assure that empty rows and columns are removed after this filter, you should call `removeEmpty` explicitly.
`reason`	Default NA. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterByCount( help=TRUE )

myObject$filterByCount( MARGIN, min=NULL, max=NULL, relative=TRUE,
                        filterEmpty=FALSE, reason=NA )
```

Updates the used matrix to remove rows and columns when elements in a dimension have too few or too many connections. Minimum and maximum connection counts can be specified absolutely, or as a percentage of the length of the dimenstion.

This filter was created initially to support hypergeometric-based gene enrichment analysis. In that context it can remove ontology terms that are assigned to "few" (too specific) or "many" (too generic) genes. It is also be used to remove genes with few ontology terms, as a mechanism to exclude genes with poor biological support.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterLog, filterSummary, autoFilter, appliedFilters

Examples

```
## Fitler the example matrix to find symbols that have 3 or more genes:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$reset()
s2e$filterByCount("row", min=3, filterEmpty=TRUE,
                  reason="Keep only symbols with 3+ genes")
## Use the GMT output as a simple way to summarize the symbols:
message(s2e$as.gmt())
s2e$filterSummary()
```

---

filterByFactorLevel *Filter by Factor Level*

Description

AnnotatedMatrix object method to filter cells by factor level assignment

Arguments

x	Required, a list of factor levels. Can be either the factor level names (character) or integer level assignments.
keep	Default TRUE, which will presume that x represents levels that should be kept. If FALSE, then levels in x will be removed.
ignore.case	Default TRUE, which will match factor levels regardless of case.
filterEmpty	Default FALSE. If true, then rows or columns that are empty after filtering will be removed from the matrix using removeEmpty. If no cells were affected by the filterByFactorLevel call, then removeEmpty will not be called. If you wish to assure that empty rows and columns are removed after this filter, you should call removeEmpty explicitly.
reason	Default NA. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterByFactorLevel( help=TRUE )

myObject$filterByFactorLevel(x, keep=TRUE, ignore.case=TRUE,
                             filterEmpty=FALSE)
```

Updates the used matrix to remove (zero-out) cells that do not have the requested factor levels assinged. Like all filters, will not alter matrixRaw.

If the matrix is not factorized, an error will be reported, no action will be taken, and NA will be invisibly returned.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterLog, filterSummary, appliedFilters, autoFilter

Examples

```
# The example matrix has some highly atypical symbols. To look at
# them, we will need to revert filters, since the file has
# autofilters that exclude them from the start:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$reset()

# Check the levels to make sure we recall what we want:
s2e$levels()

s2e$filterByFactorLevel(c("Unknown"), keep=TRUE, filterEmpty=TRUE,
                        reason="Inspecting odd symbols")
s2e$filterSummary()

# We set filterEmpty to TRUE in order to more easily inspect the
# matrix iteslf:
s2e$matrixUse
```

---

filterById *Filter by ID*

Description

AnnotatedMatrix object method to filter rows and columns by their names

Arguments

id	Required, a vector of one or more IDs (names) to match against.
MARGIN	Default NULL, defines the dimensions to be filtered. Can specify rows with '1' or 'row', columns with '2' or 'col'. If NULL, then both rows and columns will be matched. **BE CAREFUL** - When keep=TRUE, you will almost always want to set a margin! Otherwise it is unlikely that your desired IDs are represented on both the rows and columns, and you will end up with an empty matrix (a warning will be shown in such cases to remind you that you might want MARGIN to be defined).
keep	Default FALSE, which will cause matching IDs to be removed. If TRUE, then non-matching IDs will be removed.
ignore.case	Default TRUE, which ignores the capitilazation of IDs
exact	Default TRUE, which will consider each value of id as an exact match. If FALSE, then the values provided in id will be treated as regular expressions. A name will match if **any** of the regular expressions matches it.
filterEmpty	Default FALSE. If true, then rows or columns that are empty after filtering will be removed from the matrix using removeEmpty. If no cells were affected by the filterByCount call, then removeEmpty will not be called. If you wish to assure that empty rows and columns are removed after this filter, you should call removeEmpty explicitly.
reason	Default NA. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterById( help=TRUE )

myObject$filterById( id, MARGIN=NULL, keep=FALSE,
                     ignore.case=TRUE, filterEmpty=FALSE, reason=NA)
```

Updates the used matrix to remove rows and columns based on their IDs (names).

This filter was created initially to support hypergeometric-based gene enrichment analysis. In that context it can remove ontology terms that are assigned to "few" (too specific) or "many" (too generic) genes. It is also be used to remove genes with few ontology terms, as a mechanism to exclude genes with poor biological support.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterLog, filterSummary, autoFilter, appliedFilters, rNames, cNames

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$reset()
# Filter down to just a handful of symbols
s2e$filterById(c("AR","AIRE1","APSI"), MARGIN='row', keep=TRUE)
message(s2e$as.gmt(transpose=TRUE))

s2e$reset()
# Use RegExp pattern to remove genes with LocusLink IDs longer than 3 digits
s2e$filterById(c('^LOC[0-9]{4,}'), exact=FALSE)
message(s2e$as.gmt(transpose=TRUE))
s2e$filterSummary()

s2e$reset()
## Unintended removal of all data can occur when you pass a
## specific keep list and fail to specify the margin. You will be
## warned when this happens, though.
s2e$filterById(c("AR","AIRE1","APSI"), keep=TRUE)
```

`filterByMetadata` *Filter by Metadata*

Description

AnnotatedMatrix object method to filter rows or cells by metadata

Arguments

`key`	Required, the name of the key (tag / column) to match against. Will throw an error if not a single value
`val`	Required, the value to match against. Can be multiple values.
`MARGIN`	Default NULL, which will look for matches in both rows in columns. Alternatively can be 1, 2, "row" or "col".
`keep`	Default TRUE, which will keep any matching entry, discarding (zeroing out) the others. If FALSE, then rows/cols with metadata which does not match will be removed.
`type`	Default 'like', controls the kind of matching. 'like' will allow a fragmentary match of your value to metadata (both "left" and "right" wildcards), 'regexp' will treat your input as a regular expression, and "equal" will require an exact match. Additionaly, the text "case" can be added to any of the types to force a case-sensitive match; For example, "equal case" will require an exact, case-sensitive match.
`op`	Default 'or', controls how filters resolve two or more `val` entries. 'or' will count a metadata entry as matching if any of your `val` entries match, while 'and' requires all to match. Additionally, you can use aliases 'any' (='or') or 'all (='and').

filterEmpty	Default FALSE. If true, then rows or columns that are empty after filtering will be removed from the matrix using removeEmpty. If no cells were affected by the filterByMetadata call, then removeEmpty will not be called. If you wish to assure that empty rows and columns are removed after this filter, you should call removeEmpty explicitly.
reason	Default NA. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterByMetadata( help=TRUE )

myObject$filterByMetadata(key, val, MARGIN=NULL, type="like", op='or',
                          reason=NA)
```

Updates the used matrix to remove (zero-out) all rows or columns that have metadata failing to match your request. Like all filters, will not alter matrixRaw.

Note that this is not a cell-level filter, since metadata is associated with the dimensions of the matrix, not the "connections" between dimensions. It will thus remove entire rows or columns. However, it will still tally the number of newly-zeroed cells generated by the filter operation.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterLog, filterSummary, appliedFilters, metadata, autoFilter

Examples

```
# The example matrix has some deprecated genes. These are already
# removed by the default auto filters, so we will reset the matrix
# before looking for them
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$reset()

# This tag is used in the Description field to highlight deprecated loci:
depTag <- "{Deprecated}"
s2e$filterByMetadata("description", depTag, keep=TRUE, filterEmpty=TRUE,
                       reason="Inspecting deprecated loci")
s2e$filterSummary()

# We set filterEmpty to TRUE in order to more easily inspect the
# matrix iteslf:
s2e$matrixUse

# Verify that we found what we were looking for:
```

```
s2e$metadata( colnames(s2e$matrixUse), "Description" )
```

`filterByScore` *Filter by Score*

Description

AnnotatedMatrix object method to filter cells by their numeric value

Arguments

`min`	Optional minimum score. Cells that are below this value will be set to zero
`max`	Optional maximum score. Cells that are above this value will be set to zero
`filterEmpty`	Default FALSE. If true, then rows or columns that are empty after filtering will be removed from the matrix using removeEmpty. If no cells were affected by the `filterByScore` call, then `removeEmpty` will not be called. If you wish to assure that empty rows and columns are removed after this filter, you should call `removeEmpty` explicitly.
`reason`	Default NA. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterByScore( help=TRUE )

myObject$filterByScore( min=NA, max=NA, filterEmpty=FALSE, reason=NA )
```

Updates the used matrix to remove (zero-out) cells that fail a numeric test. Like all filters, will not alter matrixRaw.

Value

Invisibly, an integer vector of newly zeroed counts for (cells, rows, cols).

See Also

filterLog, filterSummary, appliedFilters, autoFilter

Examples

```
# The example matrix is factorized, but we can still fitler it by
# the numeric factor levels
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )

# Check the levels:
s2e$levels()

# How many populated rows currently?
sum( s2e$populatedRows() )  # 151

# Let's only keep official symbols
s2e$filterByScore( min=4, reason="Focus on official symbols only" )
# We now have a more constrained matrix:
sum( s2e$populatedRows() )  # 28
s2e$filterSummary()
```

`filterLog` *Filter Log*

Description

Internal AnnotatedMatrix field holding a data.table of rows or columns removed by filters

Details

This AnnotatedMatrix object field is `NULL` if no filters have been applied. Otherwise it is a filtered form of matrixRaw. You will generally not want to access this field directly; Instead, use matObj, as it will by default provide matrixUse if filters are applied, or will fallback to matrixRaw. Using reset will reset all filters by setting matrixUse to `NULL`.

```
## THIS FIELD SHOULD NOT BE MODIFIED BY THE USER OR OTHER CODE
## Doing so will probably not cause problems, but I can't conceive
## how it might help anything, either
```

Value

A data.table

See Also

reset, filterSummary, setFilters, appliedFilters, autoFilter

Examples

```
# The example matrix includes some automatic filters:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$filterSummary()  # Overview of filter effects
# What columns have been eliminated?
s2e$filterLog[ s2e$filterLog$type == 'Col' , c("id", "metric") ]

# Frank says he can't find data for p51 anymore. Was it filtered?
s2e$filterLog[ s2e$filterLog$id == "p51", ]
# Yes; Provide Frank with guidance on avoiding sketchy gene symbols
```

`filterSummary` *Filter Summary*

Description

AnnotatedMatrix object method to summarize filters applied to the matrix

Arguments

`reason`	Default TRUE, which will cluster the filter summary by reason and type. If FALSE, then only type will be used to cluster the summary.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$filterSummary( help=TRUE )

myObject$filterSummary( reason=TRUE )
```

Returns a data.frame that has been classed to allow a pretty-print summary of the filters applied and the number of rows or columns that have been 'removed' by each filter.

Value

A data.frame classed as 'mapFilter'. Will be pretty-printed with print.mapFilter.

See Also

setFilters, filterByScore, filterByFactorLevel, filterByMetadata, rNames, cNames, reset, appliedFilters,

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
fs <- s2e$filterSummary( )
fs
str(fs)
```

`findMatrices` *Find Matrices*

Description

Searches a directory for matrices matching your criteria

Usage

```
findMatrices(ns1 = NULL, ns2 = NULL, mod = NULL, type = NULL,
  auth = NULL, vers = NULL, dir = annotatedMatrixParameter("dir"),
  recursive = TRUE, ignore.case = TRUE, most.recent = TRUE,
  regexp = FALSE, clear.cache = FALSE)
```

Arguments

`ns1`	Default `NULL`. Optional, the "first" namespace (but will be matched to either the first or second namespace field)
`ns2`	Default `NULL`. Options, the "second" namespace, will be matched to the other field relative to `ns1`
`mod`	Default `NULL`. A modifier assigned to some matrices, often a species name like "Human" or "Mouse".
`type`	Default `NULL`. The type of the matrix, generally "Map" or "Ontology"
`auth`	Default `NULL`. The authority that provided the data used to make the matrix, for example "Entrez", "GeneOntology", "PubMed" etc
`vers`	Default `NULL`. The version of the matrix, eg "2017-11-05", "GRCh37", etc
`dir`	Default `getOption("annotatedmatrixdir")`. Required, the folder to search in on your file system.
`recursive`	Default `TRUE`, should subdirectories be searched as well.
`ignore.case`	Default `TRUE`. Should matches ignore case?
`most.recent`	Default `NULL`. If NULL, keep all results. Otherwise, cluster the matrices by all fields except version, and keep only the "most recent" version of each cluster. If `most.recent` contains the substring "vers", this will be done by sorting the Version field of the data.frame. Otherwise, the Modified field (file system modification time) will be used.
`regexp`	Default `FALSE`. Should matches be performed by regular expression?
`clear.cache`	Default `FALSE`. File lists are cached for each `dir` requested (as long as `recursive` is TRUE). This allows faster recovery of subsequent searches, but will prevent discovery of newly-created matrices. Pass a value of TRUE to assure an explicit search of the directory.

Details

The expected file format will be one of:

`<TYPE>@<MOD>-<NS>_to_<NS>@<AUTH>@<VERS>.mtx` `<TYPE>@<NS>_to_<NS>@<AUTH>@<VERS>.mtx`

... keeping in mind that the relative order of NS1 and NS2 is irrelevant, and noting that some matricies will not have a modifier (MOD). It will be expected that the suffix be lower case.

Value

A data.frame with the following fields:

- SubDir - subdirectory relative to `dir`
- Type - The type of conversion matrix
- Modifier - A modifier, generally a species
- NS1 - The row namespace / database
- NS2 - The column namespace / database
- Authority - The authority for the underlying data
- Version - The version of the matrix
- Path - The path to the matrix file, relative to `dir`
- Modified - The file modification time

See Also

annotatedMatrixParameter

`fromRDS` *Matrix Source RDS Flag*

Description

Internal AnnotatedMatrix field defining if loading was from a cached RDS

Details

A logical flag, if TRUE it indicates that the matrix was loaded from a cached RDS object. The RDS file will always be the same as file, but with an additional '.rds' suffix appended.

If the RDS file does not already exist, .readMatrix will parse the 'original' files and automatically create a new RDS file. Similarly, if the RDS file is older than the original sources, it will be regenerated from source.

```
## NORMALLY YOU WILL NOT WANT TO ACCESS THIS FIELD DIRECTLY
```

Value

A character vector with a single value

See Also

file, .readMatrix

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$fromRDS
```

`generateMtxRds` *Generate MTX RDS*

Description

Little tool function to pre-calculate RDS files from MTX

Usage

```
generateMtxRds(path, recursive = TRUE)
```

Arguments

`path`	Required, the path to the folder to inspect
`recursive`	Default `TRUE`, which will cause subfolders to also be processed

Details

Matrix generation scripts will make a collection of 'raw' (text) MTX files. When these files are passed to AnnotatedMatrix, they will be parsed to extract the sparse matrix, and to process the various annotations. Since this takes time, the end result will be cached as a serialized RDS object with the same file path plus '.rds'.

This function merely takes a folder path as input, finds all .mtx files under it, and loads them in AnnotatedMatrix to pre-generate the RDS files. In some circumstances this may increase the satisfaction of particularly impatient colleagues.

Value

A character vector of file paths that were processed

inst.path.package *Development Package Path*

Description

Get the path to package file, including when the package is locally installed from a local installation

Usage

```
inst.path.package(pkg = "AnnotatedMatrix")
```

Arguments

pkg	Default "AnnotatedMatrix", the package name

Details

When developing with the uncompressed R package, some files are in different locations. In particular, exdata/ remains inside the inst/ folder. This method detects the presence of "inst" and includes it in the return path

Value

The path to the package folder, or the inst/ subfolder if it exists

See Also

path.package

is.factor *Matrix is Factor*

Description

AnnotatedMatrix object method indicating if the matrix is a factor or not

Arguments

help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$is.factor( help=TRUE )

myObject$is.factor( )
```

Matrix matrices do not support factorized content, at least at the time this package was developed. AnnotatedMatrix maintains some metadata to allow integer matrices to be treated as factors with named levels. This method will indicate if the lvlVal field has ben set.

Value

A single logical value, TRUE if lvlVal is defined with one or more levels, otherwise FALSE.

See Also

lvlVal, levels

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$is.factor( )
s2e$levels()
```

---

levels *Matrix Levels*

Description

AnnotatedMatrix object method returning the factor levels (names)

Arguments

asFactor	Default FALSE, in which case factor level names only will be returned. If TRUE, then the returned value will be a formal factor object.
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$levels( help=TRUE )

myObject$levels( )
```

If the matrix is flagged as being a factor (by having lvlVal set), this function will return the factor levels (names).

Value

NULL if the matrix is not a factor, or a character vector of level names, or a factor if `asFactor` is TRUE.

See Also

lvlVal, is.factor

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$levels()
str( s2e$levels( asFactor=TRUE ) )
```

---

lvlVal *Matrix Level Values*

Description

Internal AnnotatedMatrix field holding factor levels (names)

Details

AnnotatedMatrix objects can be defined as a "pseudo-factor" by setting factor level names that will be associated with numeric values of the matrix. These level names are held in the `lvlVal` field. If the field is populated the matrix will be presumed to be a factor.

```
## THIS FIELD SHOULD NOT BE MODIFIED BY THE USER OR OTHER CODE
## DOING SO COULD LEAD TO CONFUSION / CRASHES
```

Value

A character vector of level values

See Also

is.factor, levels

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$lvlVal
```

makeEnsemblMatrix *Make Ensembl Matrix*

Description

Generates an Ensembl AnnotatedMatrix file

Usage

```
makeEnsemblMatrix(dataset = NULL, ...)
```

Arguments

dataset	Default NULL, which will recover all available datasets from BioMart. Can alternatively be a vector of one or more datasets.
...	Passed on to BioMart functions

Details

WORK IN PROGRESS

map *Map*

Description

AnnotatedMatrix object method to convert names on one dimension to the other

Arguments

input	Default NULL. The method requires input to function. It can be provided here as a character vector of IDs, or if append.to has provided a data.frame, then it will be taken from the column in that object specified by append.col
via	The matrix dimension that input is from. The default is NULL, which will compare your input to both rows and columns, and chose the dimension with the most matches (defaulting to 'row' in the event of equal matches)
format	Default 'data.frame', specifies the output format. Can also be 'vector', which will return a named character vector, with values corresponding to $Output and names as $Input. A value of 'dynamictable' will render the results as an interactive HTML table (requires package 'dynamictable').
ignore.case	Default TRUE, which ignores the capitilazation of IDs in both columns and rows
keep.best	Default FALSE. If TRUE, then only the top-scored cell(s) will be kept for each input conversion

`column.func` Default `max`. If ignore.case is true, it is possible that an input ID can match multiple matrix IDs. In this case, multiple matching rows will be returned for one ID. `column.func` is applied to reduce this to a single row. The function should accept a numeric vector of scores as the first argument.

`collapse` Default NULL, which will cause every pairwise connection to be reported. If 'in', then the `$Input` column of the data.frame will be unique - any input value that results in multiple output values will result in the `$Output` IDs and `$Score` being 'collapsed' to a single value (see the `collapse.*` options below). A value of 'out' will do the same, but causes the `$Output` column to be unique, and `$Input` and `$Score` are collapsed.

`collapse.name` Default NULL, which will cause multiple names to be concatenated into a single value using paste(). Alternatively, a user function can be provided. This package also includes the crude utility function takeLowestThing which can be used here. The function should accept a character vector of names as the first argument.

`collapse.token` Default ',', a string used to concatenate collapsed IDs when using paste via `collapse.name=NULL`. An error will be thrown if it is not a single scalar string.

`collapse.score` Default NULL. Optional function to apply to the `$Score` column when two or more rows are being collapsed into one. The function should take a numeric vector as input and return a single numeric value. If NULL, the function will be mean(), unless the matrix is a pseudo-factor, in which case the object method .autoLevel will be used. This will generate new 'hybrid' factors as needed. `collapse.token` will be used as the string to join factors into a new level name.

`integer.factor` Default NULL, which will include both the `$Score` column (integer factor value) as well as a `$Factor` column (with level values as characters) in the output to be present instead. If TRUE then ONLY a `$Score` column (representing integer values, perhaps including new and likely-meaningless hybrid values from .autoLevel) will be added. If FALSE, then only the `$Factor` column will be present.

`add.metadata` Default TRUE, which will add all metadata columns that have at least some information associated with the `$Ouput` column. FALSE will prevent adding metadata, and a character vector will add those specific columns (it is up to the user to confirm that requested columns exist in the metadata store)

`input.metadata` Default FALSE. If true, then metadata columns will also be included for the input IDs.

`warn` Default TRUE, which will show warning text if matches failed to be made for the input. This information is also always captured in attributes attached to the returned data.frame

`append.to` Default NULL. If a data.frame-compliant object, it will become the return value, with the columns generated by this function being added on. If `collapse='out'` then the `$Output` column will be used to merge, otherwise the `$Input` column will be utilized. The merge column from the provided data.frame is set with `append.col`

`append.col` The column to use in `append.to` for merging. Default is `1L`, can provide another column number or name. If `input` was not specified, then the contents of this column will be taken as input.

recurse — Default 0. This option is only relevant when the identifiers on both rows and columns are the same, or at least have significant overlap. If true, discovered Output IDs will be re-mapped as Input IDs to discover more Output IDs, recursively repeated the indicated number of times or until no new IDs are found. The intended use case is to exhaustively expand a Parent-Child network, either following parents "up" to the root nodes, or following children down to transitively discover all inherited child nodes.

in.name — Default "Input". The header name to use for the input IDs. If set to NULL, will be taken from the matrix dimension name corresponding to the input.

out.name — Default "Output". As per in.name, but for the Output data.

score.name — Default NULL. The column header for the Score column. Can be set explicitly with a non-null value. Otherwise, if the 'celldim' object parameter has been set, that value will be used. If not, the "Score" will be utilized.

prefix.metadata — Default TRUE, which will prefix metadata column headers with out.name and in.name, as relevant.

help — Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$map( help=TRUE )

myObject$map(input=NULL, via=NULL, format="data.frame", ignore.case=TRUE,
             keep.best=FALSE, column.func=max, collapse=NULL,
             collapse.name=NULL, collapse.token=',',
             collapse.score=NULL, collapse.factor=NULL,
             integer.factor=FALSE, add.metadata=TRUE,
             input.metadata=FALSE, warn=TRUE,
             append.to=NULL, append.col=1L,
             in.name="Input", out.name="Output", score.name=NULL,
             prefix.metadata=TRUE,
             help=FALSE )
```

This is the primary method for AnnotatedMatrix. It takes the input, matches it to one of the dimensions (chosen either automatically or specifically) and then reports "connections" to the other dimension. The results can be returned in a variety of formats.

The method is alert for any non-unique mappings, and will record such cases in attributes of the returned value as well as by alert to the terminal. The following non-unique categories are noted:

```
 Dup.In   : IDs that were present twice or more in input
            (possibly after case removal)
 Dup.Mat  : IDs that were present twice or more in matrix
            (always after case removal, since matrix names are unique)
 Mult.In  : Input IDs that generated multiple output values
 Mult.Out : Output IDs that were generated from multiple inputs
 Unmapped : Input ID is also in the matrix, but does not have a target
            with non-zero score. Score will be zero
```

```
Unknown  : Input ID could not be matched to any in the matrix. Score
           will be \code{NA}
```

These categories also correspond to attributes of the same names that are attached to the returned value. In addition, a "Notes" attributes contains the above descriptions to aid in remembering the distinctions between categories.

Note also that there are two "not mapped" scores. `NA` indicates that the requested input value was not found in the matrix, either because it was never there at all, or because the row/col that held it was removed (through removeEmpty or rNames / cNames). Alternatively, a value of `0` indicates that the name is recognized, but now occupies a fully zeroed-out row or column (presumably through applied filters). These two distinct values can be helpful in troubleshooting why some input values are not producing output.

Value

If `format` is 'data.frame', a data.frame. If `format` is 'vector', an SVG map of the globe, with output values represented as minor bodies of water or occasionally rivers. Just kidding, you'd get a (named) vector.

See Also

.autoLevel, metadata

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )

# I have some gene symbols, what Entrez Gene IDs do they map to?
syms <- c("FOXD3", "NBP", "AR")
s2e$map( syms )

# More messy than I wanted, "NBP" and "AR" are not uniquely
# mapping. Let's filter the matrix to only include official symbols
s2e$filterByFactorLevel("Official", keep=TRUE)
s2e$map( syms )

# Now we've lost mappings to NBP. Let's try again, but now use keep.best
s2e$reset() # Clear the filters (will also clear an automatic filter)
s2e$map( syms, keep.best=TRUE )

# The two NBP mappings are both unofficial. We could work with the
# data as-is, or we could collapse on input to assure that each row
# is unique:
s2e$map( syms, keep.best=TRUE, collapse='in' )

# We can also merge these data into an existing structure. Say we
# have some gene-linked research data:
evilCRO <- data.frame(
     Client=c("Evil Inc.", "Quite Evil", "Totally Evil"),
     Mutation=c("FOXD3", "NBP", "AR"),
```

```
    Phenotype=c("Eye lasers", "Adamantine claws", "Bunny whiskers"),
    Contained=c(TRUE,TRUE,FALSE),
    stringsAsFactors=FALSE)

# We can generate an integrated data.frame with append.to. Don't
# forget to specify the merge column with append.col if it's not
# the first column.
evilLoci <- s2e$map(append.to=evilCRO, append.col="Mutation",
                    keep.best=TRUE, collapse='in' )
evilLoci

# For large results the non-unique events may not be
# transparent. Check the attributes to identify non-unique IDs
attr(evilLoci, 'Mult.In')
```

`matObj` *Get Matrix Object*

Description

AnnotatedMatrix object method to recover the 'current' matrix data

Arguments

`raw`	Default `FALSE`, which will return matrixUse if filters have been applied. If no filters are applied, or raw=TRUE, then matrixRaw will be returned
`transpose`	Default `FALSE`. if TRUE, then transpose the matrix before returning it.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$matObj( help=TRUE )

obj <- myObject$matObj( raw=FALSE, transpose=FALSE )
```

Value

a Matrix:: 'dgTMatrix' sparse matrix

See Also

reset, matrixRaw, matrixUse

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## Automatically applied filters result in fewer non-zero elements
## of the filtered matrix than the raw one:
Matrix::nnzero(s2e$matObj())         # 180
Matrix::nnzero(s2e$matObj(raw=TRUE)) # 208

# transpose is included to help you avoid using base::t(), which
# will de-sparsify the matrix and often result in RAM exhaustion
head( rownames( s2e$matObj() ) )
head( rownames( s2e$matObj( transpose=TRUE ) ) )
```

`matrixFromLists` *Matrix from Lists*

Description

Convert a list of vectors into a sparse matrix consumable by AnnotatedMatrix

Usage

```
matrixFromLists(data, meta = NULL, listNames = NULL, file = NULL,
  val = 1)
```

Arguments

`data`	Required, a list of vectors. Each list element is considered to be a row, and the members will become (in aggreagate) the columns
`meta`	Default NULL. Optional list of named character vectors representing metadata assignements. The list names represent the metadata names (eg "Description", "Notes", whatever). The vector names represent the thing being annotated (a row or column name), and the values hold the actual assignment. Metadata is "mixed" in that there are not separate tables for columns and rows.
`listNames`	Default NULL, in which case the listNames (rNames) will be taken as names(data). The option to provide the names independently is included in case some names are not unique. They will still be made so in the final matrix
`file`	Default NULL. If non-null, will be used to look for sidecar metadata files.
`val`	Default 1. The value to be assigned to non-zero cells in the matrix.

Details

Sparse matrices (eg dgTMatrix) are constructed from lists of triples; row number (i), column number (j) and value of that cell (x). This function generates such triples by providing one or more lists of "members" (strings)

Value

A list with the following members:

- mat - The sparse matrix
- metadata - data.table of any metadata assignments
- colChanges - a named character vector. The names are the column names as used in the matrix, the values are the names as originally provided. Will only contain names that had to be altered to avoid uniqueness conflicts.
- rowChanges - as above, but for rows

This structure can be read by internal methods used by AnnotatedMatrix to parse flat files

Examples

```
ingredients <- list(Potato=c("Mashed","Fried","Baked"),
                    Onion=c("Fried"),
                    Apple=c("Raw","Baked"))

meta <- list(Type=setNames(c("Vegetable","Fruit","Vegetable"),
                           c("Potato","Apple","Onion")),
             Tool=setNames(c("Tray","Pan","Pot",NA),
                           c("Baked","Fried","Mashed","Raw")),
             Appliance=setNames(c("Oven","Stove","Stove",NA),
                                c("Baked","Fried","Mashed","Raw")))

matrixFromLists(ingredients, meta)
```

---

matrixMD *Matrix Metadata Object*

Description

Internal AnnotatedMatrix field holding metadata about row and column entries

Details

This field stores metadata information for the AnnotatedMatrix object. Metadata for rows and columns are both stored in a single `data.table` (this may be a bad idea - I may split the field later into matrixMdRow and matrixMdCol)

```
## NORMALLY YOU WILL NOT WANT TO ACCESS THIS FIELD DIRECTLY
## Use methods instead, in this case $metadata()
```

Value

a data.table object

See Also

metadata

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
str(s2e$matrixMD)
```

`matrixRaw` *Raw Matrix*

Description

Internal AnnotatedMatrix field holding unaltered (as read from file) Matrix object

Details

This AnnotatedMatrix object field is populated directly with matrix data from the source file, and is unaltered. It allows filters to be cleared using reset.

```
## NORMALLY YOU WILL NOT WANT TO ACCESS THIS FIELD DIRECTLY
## Use methods instead, in this case $matObj()
## ALTERING THIS FIELD WILL LEAD TO CHAOS. PLEASE DON'T.
```

Value

a Matrix:: 'dgTMatrix' sparse matrix

See Also

matObj, matrixUse, reset

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
str(s2e$matrixRaw)
```

`matrixText` *Matrix Text*

Description

Generate text summary of the AnnotatedMatrix object

Arguments

`pad`	Default `""`, text to prefix the object. Intended to allow nested construction of the summary, such that multiple AnnotatedMatrix objects embedded in a more complex object (eg SetFisher) can be visually combined.
`useObj`	Default `NULL`, the Matrix object to summarize. When NULL `$matObj()` will be used.
`fallbackVar`	Default `NULL`, a variable name to use when `.selfVarName()` fails to extract the actual variable name.
`compact`	Default `FALSE`. If TRUE, then a more compact report (fewer details) is produced.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$matrixText( help=TRUE )

myObject$matrixText(pad = "", useObj=NULL, fallbackVar=NULL,
                    compact=FALSE, color=NULL)
```

All RefClass objects include a `show()` method, which is similar to `print` in other classes - it will generate output when the object is evaluated "as-is". AnnotatedMatrix uses `matrixText` to assemble the text shown by show().

Value

A single string holding the text to be displayed

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$matrixText()         # just the raw text, with ANSI codes
cat( s2e$matrixText() )  # ... print it pretty, which is the same as:
s2e$show()               # ... which is the same as:
s2e                      # ... how $show() is "meant to be used"
```

matrixUse *Used (Filtered) Matrix*

Description

Internal AnnotatedMatrix field holding the filtered Matrix object

Details

This AnnotatedMatrix object field is NULL if no filters have been applied. Otherwise it is a filtered form of matrixRaw. You will generally not want to access this field directly; Instead, use matObj, as it will by default provide matrixUse if filters are applied, or will fallback to matrixRaw. Using reset will reset all filters by setting matrixUse to NULL.

```
## NORMALLY YOU WILL NOT WANT TO ACCESS THIS FIELD DIRECTLY
## Use methods instead, in this case $matObj()
## ALTERING THIS FIELD WILL LEAD TO CHAOS. PLEASE DON'T.
```

Value

a Matrix:: 'dgTMatrix' sparse matrix, or NULL if no filters have been applied

See Also

matObj, matrixRaw, reset

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
# Example matrix has an autofilter, so matrixUse is populated:
str(s2e$matrixUse)
# Resetting filters will also clear $matrixUse, until new filters
# are applied.
s2e$reset()
str(s2e$matrixUse)
```

melt *Melt*

Description

AnnotatedMatrix object method to convert matrix into three column table

Arguments

obj	Default NULL, the matrix object to serialize. If NULL, will call matObj to recover the current filtered matrix.
file	Default NULL. When NULL, the data.frame is simply returned. If not NULL, the table is written to the specified file, and the data.frame is invisibly returned.
named.dims	Default TRUE. If FALSE, then the table will have a column header of "Row","Col","Value". When TRUE, Row and Col will be replaced by the relevant dimnames for the matrix, if available. If a 'CellDim' parameter has been set, it will replace Value.
...	Will be passed to matObj
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$melt( help=TRUE )

myObject$melt( obj=NULL, file=NULL, named.dims=TRUE, ...)
```

Will represent the matrix as a "melted" table with three columns; row name, column name, value. Every non-zero cell in the matrix will be represented as a row. Zero-values are excluded.

The return value is a data.frame, which can be optionally written to a file.

Value

A data.frame, invisibly if `file` is specified

See Also

`as.ijx` to generate a matrix or a block of text.

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## Strip down to a smaller size for easier illustration
s2e$rNames(c("HFH2","AIS1","p63"))
s2e$melt( )
```

---

metadata *Metadata*

Description

AnnotatedMatrix object method to recover metadata assigned to rows or columns

Arguments

id	Default NULL. Optional vector of specific IDs to query
key	Default NULL. Optional vector of keys / tags to query
na.rm	Default TRUE, which will cause rows or columns that are entirely NA to be removed.
drop	Default TRUE, which will cause a named vector to be returned if only one column of metadata is present. Otherwise a data.table will be returned.
verbose	Default TRUE, which will warn about certain issues
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$metadata( help=TRUE )

myObject$metadata(id=NULL, key=NULL, na.rm=TRUE, drop=TRUE, verbose=TRUE )
```

Query metadata associated with rows and/or columns, based on IDs and/or keys

Value

If drop=TRUE and zero or one columns of metadata a present, a named character vector. Otherwise a data.table

See Also

metadata_keys, matrixMD

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$metadata(c("LOC9588","LOC7038","LOC3921"))
s2e$metadata(key="Symbol")
```

---

metadata_keys *Metadata Keys*

Description

AnnotatedMatrix object method to return all metadata keys

Arguments

help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$metadata_keys( help=TRUE )
```

```
myObject$metadata_keys( )
```

Return all keys (tag names) from the internal metadata table. The `id` column will be excluded from the result.

Value

A character vector of metadata column names

See Also

metadata, matrixMD

Examples

```
s2e  <- AnnotatedMatrix( annotatedMatrixExampleFile() )
## Two keys assigned to the example matrix:
s2e$metadata_keys()
```

`nnZero` *Number of Non-zero Cells*

Description

AnnotatedMatrix object method to return a count of non-zero cells in the used matrix

Arguments

`obj`	Default NULL, which will recover the matrix using matObj. Optionally can be a user-provided sparse Matrix.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.
`...`	Passed to matObj (for example, `raw`)

Details

```
## Method Usage:
myObject$nnZero( help=TRUE )
```

```
myObject$nnZero( obj=NULL )
```

For sparse matrices, the non-zero values represent the data, which in AnnotatedMatrix are typically used to indicate connections of some sort between the two dimensions of the matrix.

Value

An integer

See Also

counts, populated

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e
# The example matrix begins filtered - inspecting the object should show:

# 180 non-zero cells (1.92%)
# [Raw Matrix] : 167 x 67 (90.4%x92.5%), 208 non-zero (1.86%; 86.54% survive filters)

# That is, in the filtered state 180 cells are non-zero. Confirm:
s2e$nnZero()           # 180
# We can see the total 'raw' connections by passing the raw argument:
s2e$nnZero( raw=TRUE ) # 208
# Or we can reset the filters and check again:
s2e$reset()
s2e$nnZero()           # 208
```

---

normalizePercent *Normalize percent*

Description

Convert text percentages (eg "4

Usage

```
normalizePercent(x, denom = as.numeric(NA))
```

Arguments

x	Required, a character or numeric vector containing the values to be evaluated
denom	Default NA. The "denominator" for the calculations, which will be used when a percentage is provided in x.

Details

Used to parse parameters that can be absolutely defined (eg 53) or relatively defined as a percentage (eg "2.5%"). For the latter will require a denominator to be provided as well

Examples

```
## Numeric strings and percentages can be mixed together; The denom
## parameter only applies to percentages:
AnnotatedMatrix:::normalizePercent( c("5", "0.02", "5\%", "0.02%"), 1000)
AnnotatedMatrix:::normalizePercent( 1:10 ) # Numbers are 'left alone'
```

`parseMetadataSidecar` *Parse Metadata Sidecar*

Description

Loads metadata from files accompanying the 'main' data

Usage

```
parseMetadataSidecar(file, metadata = NULL, na.strings = c("NA", "-"),
  verbose = TRUE)
```

Arguments

`file`	Required, the path to the primary data file (NOT to the metadata file)
`metadata`	Default NULL. Optional data.table that already contains metadata, presumably from another source. If not provided, an empty data.table will be made.
`na.strings`	Default `c("NA",'-')`, strings that will be parsed as NA. A dash is included as it is a common null token in biological data sets.
`verbose`	Default TRUE, which will report each found file to the terminal.

Details

This package primarily expects matrices to be encoded in MatrixMarket files that include extensive 'in-line' information embedded in comments. In some cases it may be desirable to manage metadata in other files, or you may have file formats other than MTX that can't hold metadata internally.

In those cases, one or more 'sidecar' files can be included. This function will expect those files to be named <BASENAME>-metadata<SOMETHING>, where <BASENAME> is the full path of the 'main' data file, and <SOMETHING> is any other text (including no text).

The metadata files should be TSV tabular and include a header row. The function will look for an 'id' column - if not found, it will rename the first column 'id' and use it.

Also supported are files of format <BASENAME>-parameters<SOMETHING>, which will be parsed into a list structure and added as an attribute. These will then be used to set `$param()` values.

Examples

```
## .makeTempFile() is an internal helper function; Don't use normally!
lolFile <- AnnotatedMatrix:::.makeTempFile("ListOfLists.inp")

## There are two sidecars associated with this file
parseMetadataSidecar( lolFile )
```

`parse_GMT_file` *Parse GMT File*

Description

Parses one or more lists from a Gene Matrix Transposed file

Usage

```
parse_GMT_file(file)
```

Arguments

`file`	Required, the path to the GMT file

Details

GMT files allow an arbitrary number of lists to be stored, each with a name, description, and one or more members

Generally you would not call this function directly.

Value

A list structure usable by AnnotatedMatrix

`parse_IJX_file` *Parse IJX File*

Description

Parses one or more lists from a three column "ijx" (row col val) file

Usage

```
parse_IJX_file(file, sep = "\t")
```

Arguments

file	Required, the path to the IJX file
sep	Default "\t", the field separtor in the input file.

Details

This format can be described as one row per cell. Each row represents the row identifier (first column, i) the column identifier (second column, j) and the value assigned to that row/col cell (third column, x).

Generally you would not call this function directly.

Value

A list structure usable by AnnotatedMatrix

---

parse_ListOfLists_file

Parse List-of-Lists File

Description

Parses a List-of-Lists file so it can be read by AnnotatedMatrix

Usage

```
parse_ListOfLists_file(file)
```

Arguments

file	Required - the path to the LOL file. Files can be gzipped, provided they have a .gz suffix.

Details

This method is somewhat slow; For many files, you may wish to use the Perl script "listOfList_to_MatrixMarket.pl" script in the perl/ subdirectory of the AnnotatedMatrix installation.

Once the file is parsed, an RDS version will be written "next to" it (same path, plus ".rds"), which will load rapidly, unless a change to the original file triggers an automatic reparsing.

parse_MatrixMarket_file *Parse MatrixMarket File*

Description

Parses a MatrixMarket file so it can be read by AnnotatedMatrix

Usage

```
parse_MatrixMarket_file(file)
```

Arguments

file	Required - the path to the .mtx file. Files can be gzipped, provided they have a .gz suffix.

Details

Reads the "normal" matrix data from a MatrixMarket file (the [i,j,x] triples), as well as parses ' metadata usable by AnnotatedMatrix.

parse_Text_file *Parse Text File*

Description

Parses one or more simple text files into data consumable by AnnotatedMatrix

Usage

```
parse_Text_file(files, header = FALSE, rm.suffix = TRUE, pattern = NULL)
```

Arguments

files	Required. One or more file/directory paths (can be a mixture of directories and files). Files can be gzipped, provided they have a .gz suffix.
header	Default FALSE. If true, the first row of each file will be taken as the list name for that list
rm.suffix	Default TRUE, which will remove the file suffix when the file name is used as a list name
pattern	Default NULL. Optional pattern to use when extracting files from requested directories

Details

Provided one or more files and/or directories, take all files and treat each as a single list

Generally you would not call this function directly.

Value

A list structure usable by AnnotatedMatrix

`populated` *Populated Rows and Columns*

Description

AnnotatedMatrix object methods to flag non-zero rows and columns

Arguments

`obj`	Default NULL, which will recover the matrix using matObj. Optionally can be a user-provided sparse Matrix.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.
`...`	Passed to matObj (for example, `raw`)

Details

```
## Method Usage:
myObject$populatedRows( help=TRUE )
myObject$populatedCols( help=TRUE )

myObject$populatedRows( obj=NULL )
myObject$populatedCols( obj=NULL )
```

These methods are just simple wrappers for the counts methods, which convert their output to logical using `!= 0`. TRUE values indicate at least one connection to the other dimension.

Value

A named logical vector, which will be as long as the number of rows (for rCounts) or columns (for cCounts). Names are taken from the dimension names.

See Also

counts, nnZero

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
# Reset filters, then toss in a metadata filter to make a smaller matrix
s2e$reset()
s2e$filterByMetadata("Description","superfamily")

# What genes and loci still have connections after this?
popRows <- s2e$populatedRows()
popRows[ popRows ]

popCols <- s2e$populatedCols()
s2e$metadata(names(popCols[ popCols ]), c("Symbol", "Description"))
```

`preCacheMatrixDirectory` *Precache Matrix Directory*

Description

Attempts to load all MTX files in a directory, generating RDS files

Usage

```
preCacheMatrixDirectory(dir = ".")
```

Arguments

`dir`	Default '.', the directory to scan

Details

There are two reasons to precache a folder of matrix files; First, to generate the serialized RDS files in advance, and second, to scan for files that are malformed and not loading.

Value

A list with two elements; 'pass', the matrix files that were succesfully loaded, and 'fail', those that did not.

print.mapFilter *Printing Annotated Matrix filter summarys*

Description

Print method for mapFilter objects generated by AnnotatedMatrix$filterSummary()

Usage

```
## S3 method for class 'mapFilter'
print(x, color = NULL, ...)
```

Arguments

x	The object classed as 'mapFilter'
color	Default NULL, which will read the "useCol" attribute of x. Logical flagging if output should be colorized with crayon.
...	Additional parameters, passed on to message()
	s2e <- AnnotatedMatrix(annotatedMatrixExampleFile()) s2e$filterSummary( )

Details

$filterSummary() returns a data frame detailing the number of rows and columns "zeroed-out" by each applied filter. This method formats those results into a more human-friendly format.

print.mapResult *Printing map() results*

Description

Print method for mapResult objects generated by AnnotatedMatrix$map()

Usage

```
## S3 method for class 'mapResult'
print(x, ...)
```

Arguments

x	The object classed as 'mapResult'
...	Additional parameters, passed on to print.default()
	s2e <- AnnotatedMatrix(annotatedMatrixExampleFile()) locs <- s2e$map(c("MOT8", "GARS")) locs

Details

$map() normally returns a data.frame with no additional class assigned. If called with format='vector', it will return a named vector with attributes. The attributes end up being printed by print.default, which is generally annoying. This method simply strips attributes before printing.

`product` *Matrix Product*

Description

Chain two matrices together based on a common dimension

Arguments

`mat2`	Required, the second ('right') annotated matrix object
`dim1`	Default `NULL`, the shared dimension on the left matrix. If NULL, will automatically pick the one with maximal overlap. Tie-breaking logic is not implemented. Otherwise can be either '1'/'row', or '2'/'col'
`dim2`	Default `NULL`. Like `dim1`, but for the second (right) matrix.
`valfunc`	Default `NULL`, which will default to 'maxright' AND chide you to explicitly pick a value. See the 'Value Function' section for more information.
`levels`	Default `NULL`, optional list of factor levels. If NULL, and if valfunc is using a simple request (eg 'right max') to specify that values should come from just one matrix, then the levels from that matrix will be used. Otherwise you should provide a character vector of levels if appropriate (and assure that valfunc is generating appropriate integer output)
`ignore.zero`	Default `TRUE`, which will report a value of zero for any left/right pair if all values on either side are only zero. In these situations `valfunc` will be ignored.
`ignore.case`	Default `TRUE`, which will intersect the matrices without regard to case.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$product( help=TRUE )

myObject$product(mat2, dim1=NULL, dim2=NULL, valfunc=NULL, levels=NULL,
                 ignore.zero=TRUE, ignore.case=TRUE)
```

Designed to chain / bridge two matrices that share a common dimension. For example, if you have a `Foo vs Bar` matrix, and a `Pop vs Bar` matrix, you can take their 'product' by finding shared entities in the 'Pop' dimension, resulting in a `Foo vs Pop` matrix.

Because each AnnotatedMatrix is really representing a miniature graph database, this isn't a formally traditional matrix product, but rather a graph traversal across the two matrices. The primary challenge is numericly representing the final score assigned to each Foo-vs-Pop connection, since

the scores held by the matrices are rarely going to make sense as a product. Additionally, two or more paths will be available for any Foo-Pop combination if they can traverse two or more intermediate Bar entries. Normalizing / Aggregating those multiple scores into a single useful value will not have a generic solution.

Value Function

The `valfunc` defines a function that controls how the matrix value will be calculated when multiple paths are available between two output destinations. It can either be a function with two numeric arguments reprsenting a vector of 'left' values and a vector of 'right' values, returning a single numeric value, for example:

```
   function(lv,rv) {      max(c(lv,rv)) # Takes the maximum of either left or right value
}
```

... or it can be a text string, which will be parsed by regular expression:

- 'max' will return the maximum of the values
- 'min' returns the minimum
- 'mean' returns the mean
- 'left' or 'lft' will consider only values from the left input
- 'right' or 'rgt' will consider only right matrix values
- If neither 'right' nor 'left' (nor an alias) are detected, then both matrices values will be combined

So the default 'right max' will represent the value of each new transitive pair as `max(rightMatrixValues)`.

Value Function Musings

Consider the following two matrices; We wish to make a 'foo' -> 'bar' matrix by using the shared 'boop' dimension:

```
            'boop'              'boop'              X Y                 X   Y
'foo' A 2 .     'bar' Q 5  7        B . 3           R . 11        C 1 4
```

The available 'connections' are then:

```
       foo boop bar valRgt valLft          A    X      Q         2         5
B   Y    Q     3     7      B   Y    R     3    11      C   X    Q     1     5
C   Y    Q     4     7      C   Y    R     4    11
```

Partially collapsing to remove the 'discarded' boop dimension we then would have the following distinct output dimensions, with their [left,right] score pairs:

```
     A -> Q   [ [2, 5] ]     B -> Q   [ [3, 7] ]     B -> R   [ [3,11] ]
C -> Q  [ [1, 5], [4, 7] ]    C -> R  [ [4,11] ]
```

Each of these poses a challenge; We have a minimum of two numeric values (one left, one right), and in the case of C->Q there were two paths found, for four values total.

The matrix values could represent a wide variety of measurements; p-values, percent similarities, rank order, factorized categories, or simply boolean 1/0 unqualified flags. Because of this, it is impossible to naively assure a 'relevant' mechanism to collapse / summarize the values.

The default, 'max right', presumes that 'bigger is better' (max), and that the second matrix is probably capturing the "final" value we wish to use. It's a not-unreasonable approach, but given the diversity of available approaches, will be inappropriate in many cases.

Examples

```
# The `machines` toy matrix associates machines with components
mach <- AnnotatedMatrix( annotatedMatrixExampleFile("Machines.ijx") )
# The `components`  matrix associates them with parts
comp <- AnnotatedMatrix( annotatedMatrixExampleFile("Components.ijx") )

# We can use $product to associate machines with parts. These
# particular matrices have scores that are counts, so "traditional"
# matrix math is how we should set up scores:
matProd <- function(l, r) sum( l * r )

# Now we can calcualte the 'transitive' product of the two matrices:
prod <- mach$product(comp, valfunc=matProd)

# Inspect the input matrices as well as the product:
mach$matObj()
comp$matObj()
prod$matObj()

# `prod` now shows how many parts are in each machine, having
# 'joined' via the shared "Component" dimension.
```

`rdsIsCurrent` *RDS is Current*

Description

Checks if a serialized RDS file is current relative to source files

Usage

```
rdsIsCurrent(file)
```

Arguments

`file`	Required, the path to the 'main' file (not the RDS)

Details

For speed, matrix files and their supporting annotation files are serialized to an RDS object. These RDS files will be preferentially used, unless it is observed (by file dates)

Value

- TRUE - if the RDS object exists and is newer than all supporting files
- NA - if the RDS file exists but the original (main) file does not
- FALSE - All other situations; No RDS, or RDS is older than the main file and/or any of the sidecar files

---

`removeEmpty` *Remove Empty Rows and Columns*

Description

AnnotatedMatrix object methods to remove rows or columns that are all zeroes

Arguments

`reason`	Default NA, optional human-readable text that will be included with the setFilters entry.
`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$removeEmptyRows( help=TRUE )
myObject$removeEmptyCols( help=TRUE )
myObject$removeEmpty( help=TRUE )

myObject$removeEmptyRows( reason=NA )
myObject$removeEmptyCols( reason=NA )
myObject$removeEmpty( reason=NA )
```

AnnotatedMatrix can work with matrices that have entirely "empty" rows or columns; For sparse matrices these would correspond to those that are all zeroes.

There are advantages to leaving such entries in the matrix: In `$map()`, the presence of an empty row/col can be used to differentiate between "This input no longer has connections" versus "This input is unrecognized". Additionally, empty entries allow for name-based access without throwing errors.

However, in some cases it may be desirable to entirely remvove rows or columns. In particular, crossproducts between matrices requires the length of the crossed dimension to match.

These operations, like all filters, will only affect the used matrix. They will not generate new filterLog entries, however. Rather, any prior fitlers that resulted in a row or column becoming fully zeroed-out will have noted the event. The action will create an entry in setFilters, however.

`$removeEmpty()` simply wraps both `$removeEmptyRows()` and `$removeEmptyCols()`

Value

Invisibly, a character vector of removed names

See Also

counts, nnZero, populated

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
# The example matrix has some filters already applied:
rc <- s2e$rCounts()
length(rc)    # Total number of rows = 167
rc[ rc == 0 ] # Rows that no longer connect to columns

gone <- s2e$removeEmptyRows()
gone                 # Removed names
rce <- s2e$rCounts()
length(rce)          # -> 151
rce[ rce == 0 ]      # No more empty rows
```

`reset` *Reset Filters*

Description

AnnotatedMatrix object method to revert any applied filters

Arguments

`help`	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$reset( help=TRUE )

myObject$reset()
```

AnnotatedMatrix stores two matrices: matrixRaw, which remains an unaltered form of the matrix as loaded from the source file, and matrixUse, the version resulting from application of filters. The reset() method will set matrixUse to NULL, and clear the $setFilters and $filterLog structures.

Value

NA, invisibly

See Also

matObj, matrixRaw, matrixUse, filterLog, filterSummary, appliedFilters, autoFilter

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )

## The example matrix has a default filter that suppresses a few symbols:
sketchySymbol <- "B(p51A)"
# No output, though Score=0 (not NA) shows symbol is recognized
s2e$map( sketchySymbol )

s2e$reset()
s2e$map( sketchySymbol )  # Connection reestablished to LOC8626
```

`rowcolchanges` *Changed Row and Column Names*

Description

Internal AnnotatedMatrix fields tracking row and column names that needed to be changed when building the object

Details

Row and column names on the matrix must be unique. Some input formats do not contain this restriction. If any dimension names need to be altered to force them to be unique, those changes will be captured here. The values are named character vectors, where the value of the vector represents the original name, and the names represent the uniquely-renamed values.

```
## THIS FIELD SHOULD NOT BE MODIFIED BY THE USER OR OTHER CODE
## Doing so will probably not cause problems, but I can't conceive
## how it might help anything, either
```

Value

A named character vector

See Also

rNames, linkcNames, linkuniqueNames

rowcolnames *Row and Column Names*

Description

AnnotatedMatrix object method to get/set/reorder the rownames or colnames of the filtered matrix

Arguments

new	Default NULL, which will simply return the current row or column names. Alternatively can provide a character vector of new values. This can be used to reorder, remove or add names. Added names will generate a row or column of all zeros (no connections). Setting this value is treated as a filter action, so matrixUse will be altered and actions noted in filterLog.
raw	Default FALSE, which will operate on the filtered matrix. See matObj for more information. A fatal error will be thrown if raw is TRUE and new is not NULL (can not alter raw matrix)
nonzero	Default FALSE. If TRUE, will fitler the returned list of names to only include those that have at least one non-zero cell associated with them. This parameter is ignored if changes are being made to the names.
reason	Default NA. Only relevant when row is not NULL. Optional human-readable reason for why the alteration was made, will be recorded in filterLog and used to structure filterSummary
help	Default FALSE. If TRUE, show this help and perform no other actions.

Details

```
## Method Usage:
myObject$rNames( help=TRUE )

rn <- myObject$rNames( raw=FALSE )
cn <- myObject$cNames( raw=FALSE )

myObject$rNames( newRowNames, reason=NA )
myObject$cNames( newColNames, reason=NA )
```

Normally would be used to simply recover the rownames for the active matrix. Can also be used to reorder, or to reduce the accessible names. Unrecognized names will be added to the matrix as a row or column of all zeros; In most cases this will not be useful (no connection to other dimension) but might be helpful in some circumstances.

Value

A character vector of row names

See Also

reset, rowChanges, colChanges

Examples

```
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
length( s2e$rNames() ) # 167 names known
s2e$map('LOC326')      # Maps to six symbols
s2e$rNames(c('AIRE1', 'PGA1', 'AngryPirate', 'AIRE'),
           reason="Nancy asked me to limit analysis to these four genes")
s2e$filterSummary()    # Alteration is recorded in filterLog
length( s2e$rNames() ) # We now have the above four
s2e$map('LOC326')      # We only kept three of the original matches
s2e$removeEmpty()      # Remove rows (and columns) that are only zeros
s2e$matObj()           # A much smaller matrix!
```

---

setFilters *Set Filters*

Description

Internal AnnotatedMatrix field holding a character vector describing filters that have been set

Details

Each operation applied to the matrix to filter out values will add a new string to this field. The string will be both (somewhat) human-intelligible, as well as being machine-parsable. The intention of the field is to allow reapplication of some or all of the filters, as well as serialize the filters in an ASCII format that can be stored with output of analyses.

```
## THIS FIELD SHOULD NOT BE MODIFIED BY THE USER OR OTHER CODE
## Doing so will probably not cause problems, but I can't conceive
## how it might help anything, either
```

Value

A character vector

See Also

appliedFilters, filterSummary, autoFilter

Examples

```
# The example matrix includes some automatic filters:
s2e <- AnnotatedMatrix( annotatedMatrixExampleFile() )
s2e$setFilters
```

sidecarFiles *Sidecar Files*

Description

Given a 'main' file, finds any associated / companion files

Usage

```
sidecarFiles(file)
```

Arguments

file	Required, the path to the 'main' file

Details

Sidecar files are seperate files that carry supporting information for the main data file. In this package they are used to provide additional metadata to file formats that can not store such data internal to the main file

Value

A character vector of file paths

Examples

```
lolFile <- annotatedMatrixExampleFile("ListOfLists.inp")

sidecarFiles( lolFile )
```

smallNumberFormatter *Small Number Formatter*

Description

Attempt to make small numbers more human-readable

Usage

```
smallNumberFormatter(x)
```

Arguments

x	Required, a number, presumably between 0 and 1

Details

Designed for reporting the fraction of non-zero cells in a sparse matrix

Value

If x == 0, "none". If x >= 0.001, x as a percentage with two decimal places. Otherwise, x as a fraction with 1 in the numerator, to 4 significant figures.

`strict.unique` *Strict Unique*

Description

Like make.unique, but assure that non-unique names are ALL altered

Usage

```
strict.unique(x, token = "#")
```

Arguments

`x`	Required, a character vector
`token`	Default '#', the string that will go between the original name and the numeric iterator. So if x had two entries of "foo", using the default '#' would yields output values of "foo#1" and "foo#2"

Details

codemake.unique has the property of not altering the first occurance of a non-unique member. That is, the list ('a','b','a') will generate ('a', 'b', 'a.1'). When doing operations that presume or require uniqueness, this is often undesirable. strict.unique will force all non-unique values to be altered.

Unlike make.unique, iterative application of the function to the elements vector will NOT create the same output as a single application to the full vector. For its intended use this is not a problem.

Value

A character vector where all values are unique, and all non- unique names are altered

See Also

make.unique

Examples

```
fowl <-  c("duck", "duck", "goose")

## The first duck is not modified:
make.unique( fowl )

## All non-unique names will be altered:
strict.unique( fowl )
```

---

takeLowestThing *Take Lowest Thing*

Description

From a list of strings, take the lowest/smallest thing

Usage

```
takeLowestThing(x)
```

Arguments

x	A character vector of strings to pick from

Details

This is a desperation function and should only be used when you have a character vector and absolutely must reduce it to "one thing" but you have no real mechanism to do so. It was designed to address requests to reduce a list of gene accessions (eg LOC13992, LOC93) to "one gene", but without any other guiding context. The function will extract the left-most uninterupted integer that it can find, and then pick the one that is numerically smallest. The rationale is that "LOC93" was probably annotated earlier (longer ago) than "LOC13200423", and as such is more likely to be the "more common" / "higher expressed" / "better annotated" / "more popular" of the two.

Value

A single string

See Also

map

Examples

```
x <- c("LOC828221", "LOC1234", "LOC39", "HUH?", "LOC39-BETA", "LOC99.243-X")
takeLowestThing(x)
```

uniqueNames *Unique Names*

Description

Wrapper for make.names and make.unique, with reporting of changes

Usage

```
uniqueNames(names = character(), rpt = NULL, valid = FALSE,
  verbose = TRUE)
```

Arguments

names	Required, a character vector of the names to process
rpt	Default NULL. If not NULL, will report to STDERR any changes that were required. The value of rpt will be used as a noun in the message (ie if rpt='gene' the message will be of the form "6 genes required alteration: "...)
valid	Default FALSE, whch will utilize make.unique. If TRUE, then make.names(unique=TRUE) will be used instead
verbose	Default TRUE, which will emit a warning if any names needed to be changed

Value

A list with two components: "names", the vector of (possibly) transformed names, and "changes", a named vector where the names are the new names and the values are the original ones (will only include altered names)

Examples

```
foo <- c("Apple","Banana","Cherry","Apple")
uniqueNames(foo, rpt="IDs")
```

Index

.addAppliedFilter
(DOTaddAppliedFilter), 17
.autoLevel, *41*, *43*
.autoLevel (DOTautoLevel), 18
.buildMatrix, *22*
.buildMatrix (DOTbuildMatrix), 19
.detailZeroedRowCol
(DOTdetailZeroedRowCol), 19
.fieldDescriptions
(DOTfieldDescriptions), 20
.filterDetails (DOTfilterDetails), 20
.makeTempFile (DOTmakeTempFile), 21
.readMatrix, *19*, *24*, *35*, *36*
.readMatrix (DOTreadMatrix), 22

AnnotatedMatrix
(AnnotatedMatrix-class), 3
AnnotatedMatrix-class, 3
annotatedMatrixExampleFile, 6
annotatedMatrixParameter, 7, *15*, *35*
appliedFilters, 8, *13*, *18*, *21*, *26–28*, *30–33*, *65*, *68*
as.gmt, 9
as.ijx, 10, *50*
as.mtx, 10
as.sidecar, 11
autoFilter, 12, *25–28*, *30–32*, *65*, *68*

biomartCommonNames, 14
biomartGenomeVersions, 14
biomartVersion, *14*, 15

cat, *9–11*
cCounts (counts), 16
cNames, *8*, *28*, *33*, *43*
cNames (rowcolnames), 67
colChanges, *22*, *67*
colChanges (rowcolchanges), 66
colDefs, 15
counts, 16, *53*, *58*, *65*
counts methods, *58*

data.table, *22*
DOTaddAppliedFilter, 17
DOTautoLevel, 18
DOTbuildMatrix, 19
DOTdetailZeroedRowCol, 19
DOTfieldDescriptions, 20
DOTfilterDetails, 20
DOTmakeTempFile, 21
DOTreadMatrix, 22

extendDataTable, 22

file, *22*, 23, *35*, *36*
filesToMTX, 24
filterByCount, *13*, 25
filterByFactorLevel, *8*, *13*, 26, *33*
filterById, *13*, 27
filterByMetadata, *8*, *13*, 29, *33*
filterByScore, *8*, *13*, 31, *33*
filterLog, *18*, *19*, *21*, *25–28*, *30*, *31*, 32, *64*, *65*, *67*
filterSummary, *8*, *13*, *18*, *21*, *25–28*, *30–32*, 33, *65*, *67*, *68*
findMatrices, 34
fromRDS, *24*, 35

generateMtxRds, 36

help, *20*

inst.path.package, 37
is.factor, 37, *39*

levels, *38*, 38, *39*
lvlVal, *22*, *38*, *39*, 39

make.unique, *70*
makeEnsemblMatrix, 40
map, *15*, *16*, *18*, 40, *71*

matObj, *9–11*, *16*, *32*, 44, *47*, *49, 50*, *52*, *58*, *65*, *67*
Matrix, *22*, *38*
matrixFromLists, 45
matrixMD, *16*, 46, *51, 52*
matrixRaw, *27*, *30–32*, *44*, 47, *49*, *65*
matrixText, 48
matrixUse, *44*, *47*, 49, *65*, *67*
melt, *10*, 49
metadata, *30*, *43*, *47*, 50, *52*
metadata_keys, *51*, 51

nnZero, *17*, 52, *58*, *65*
normalizePercent, *25*, 53

parse_GMT_file, 55
parse_IJX_file, 55
parse_ListOfLists_file, 56
parse_MatrixMarket_file, *22*, 57
parse_Text_file, 57
parseMetadataSidecar, 54
path.package, *37*
populated, *17*, *53*, 58, *65*
populatedCols *(*populated*)*, 58
populatedRows *(*populated*)*, 58
preCacheMatrixDirectory, 59
print.mapFilter, *33*, 60
print.mapResult, 60
product, 61

rCounts *(*counts*)*, 16
RDS, *22*
rdsIsCurrent, 63
removeEmpty, *25, 26*, *28*, *30, 31*, *43*, 64
removeEmptyCols *(*removeEmpty*)*, 64
removeEmptyRows *(*removeEmpty*)*, 64
reset, *8*, *13*, *32, 33*, *44*, *47*, *49*, 65, *67*
rNames, *8*, *28*, *33*, *43*, *66*
rNames *(*rowcolnames*)*, 67
rowChanges, *22*, *67*
rowChanges *(*rowcolchanges*)*, 66
rowcolchanges, 66
rowcolnames, 67

setFilters, *8*, *18*, *32, 33*, *64*, 68
showParameters, *13*
sidecarFiles, *21*, 69
smallNumberFormatter, 69
strict.unique, 70

takeLowestThing, *41*, 71

uniqueNames, 72
used matrix, *25*, *27, 28*, *30, 31*, *64*