Package 'CoreGx'

February 1, 2026

Type Package

Title Classes and Functions to Serve as the Basis for Other 'Gx' Packages

Version 2.15.0

Date 2024-03-11

Description A collection of functions and classes which serve as the foundation for our lab's suite of R packages, such as 'PharmacoGx' and 'RadioGx'. This package was created to abstract shared functionality from other lab package releases to increase ease of maintainability and reduce code repetition in current and future 'Gx' suite programs. Major features include a 'CoreSet' class, from which 'RadioSet' and 'PharmacoSet' are derived, along with get and set methods for each respective slot. Additional functions related to fitting and plotting dose response curves, quantifying statistical correlation and calculating area under the curve (AUC) or survival fraction (SF) are included. For more details please see the included documentation, as well as:

Smirnov, P., Safikhani, Z., El-Hachem, N., Wang, D., She, A., Olsen, C., Freeman, M., Selby, H., Gendoo, D., Grossman, P., Beck, A., Aerts, H., Lupien, M., Goldenberg, A. (2015) <doi:10.1093/bioinformatics/btv723>.

Manem, V., Labie, M., Smirnov, P., Kofia, V., Freeman, M., Koritzinksy, M., Abazeed, M., Haibe-Kains, B., Bratman, S. (2018) <doi:10.1101/449793>.

VignetteBuilder knitr

VignetteEngine knitr::rmarkdown

biocViews Software, Pharmacogenomics, Classification, Survival

Encoding UTF-8

LazyData TRUE

Depends R (>= 4.1), BiocGenerics, SummarizedExperiment

Imports Biobase, S4Vectors, MultiAssayExperiment, MatrixGenerics, piano, BiocParallel, parallel, BumpyMatrix, checkmate, methods, stats, utils, graphics, grDevices, lsa, data.table, crayon, glue, rlang, bench

Suggests pander, markdown, BiocStyle, rmarkdown, knitr, formatR, testthat

License GPL (>= 3)

Config/testthat/edition 3

Roxygen list(markdown=TRUE, r6=FALSE)

RoxygenNote 7.3.1

RCollate 'allGenerics.R' 'immutable-class.R' 'LongTable-class.R' 'CoreSet-class.R' 'CoreSet-accessors.R' 'CoreSet-utils.R' 'DataMapper-class.R' 'LongTable-accessors.R' 'LongTable-utils.R' 'LongTableDataMapper-class.R' 'LongTableDataMapper-accessors.R' 'TreatmentResponseExperiment-class.R' 'TREDataMapper-class.R' 'adaptiveMatthewCor.R' 'aggregate-methods.R' 'callingWaterfall.R' 'connectivityScore.R' 'cosinePerm.R' 'datasets.R' 'deprecated.R' 'endoaggregate-methods.R' 'globals.R' 'gwc.R' 'matthewCor.R' 'mergeAssays-method.R' 'methods-coerce.R' 'methods-dim.R' 'methods-dimnames.R' 'methods-drugSensitivitySig.R' 'methods-guessMapping.R' 'methods-metaConstruct.R' 'methods-subsetTo.R' 'optimizeCoreGx.R' 'updateObject-methods.R' 'utilities.R' 'utils-iteration.R' 'utils-messages.R' 'utils-optimization.R' 'utils-testing.R' 'utils-updateS4.R'

Collate 'allGenerics.R' 'immutable-class.R' 'LongTable-class.R' 'CoreSet-class.R' 'CoreSet-accessors.R' 'CoreSet-utils.R' 'DataMapper-class.R' 'LongTable-accessors.R' 'LongTable-utils.R' 'LongTableDataMapper-class.R' 'LongTableDataMapper-accessors.R' 'TreatmentResponseExperiment-class.R' 'TREDataMapper-class.R' 'adaptiveMatthewCor.R' 'aggregate-methods.R' 'callingWaterfall.R' 'connectivityScore.R' 'cosinePerm.R' 'datasets.R' 'deprecated.R' 'endoaggregate-methods.R' 'globals.R' 'gwc.R' 'matthewCor.R' 'mergeAssays-method.R' 'methods-coerce.R' 'methods-dim.R' 'methods-dimnames.R' 'methods-drugSensitivitySig.R' 'methods-guessMapping.R' 'methods-metaConstruct.R' 'methods-subsetTo.R' 'optimizeCoreGx.R' 'updateObject-methods.R' 'utilities.R' 'utils-iteration.R' 'utils-messages.R' 'utils-optimization.R' 'utils-testing.R' 'utils-updateS4.R'

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

git_branch devel

git_last_commit e18fa83

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-02-01

Author Jermiah Joseph [aut],
Petr Smirnov [aut],
Ian Smith [aut],

Christopher Eeles [aut],
Feifei Li [aut],
Benjamin Haibe-Kains [aut, cre]

Maintainer Benjamin Haibe-Kains <`benjamin.haibe.kains@utoronto.ca`>

Contents

. *Convenience function for converting R code to a call*

Description

This is used to pass through unevaluated R expressions into subset and `[`, where they will be evaluated in the correct context.

Usage

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

Arguments

`...`	`pairlist` One or more R expressions to convert to calls.

Value

`call` An R call object containing the quoted expression.

Examples

```
.(sample_line1 == 'A2058')
```

`.assayToBumpyMatrix` *Convert a LongTable assay into a BumpyMatrix object*

Description

Convert a LongTable assay into a BumpyMatrix object

Usage

```
.assayToBumpyMatrix(LT, assay, rows, cols, sparse = TRUE)
```

Arguments

`LT`	`LongTable` with assay to convert into `BumpyMatrix`
`assay`	`character(1)` A valid assay name in `LT`, as returned by `assayNames(LT)`.
`rows`	`character()` The rownames associated with the assay rowKey
`cols`	`character()` The names associated with the assay colKey
`sparse`	`logical(1)` Should the `BumpyMatrix` be sparse (i.e., is the assay sparse).

Value

`BumpyMatrix` containing the data from `assay`.

`.convertCSetMolecularProfilesToSE` *CSet molecularProfiles from ESets to SEs*

Description

Converts all ExpressionSet objects within the molecularProfiles slot of a CoreSet to SummarizedExperiments

Usage

```
.convertCSetMolecularProfilesToSE(cSet)
```

Arguments

`cSet`	`S4` A CoreSet containing molecular data in ExpressionSets

Value

`S4` A CoreSet containing molecular data in a SummarizedExperiments

`.distancePointLine` *Calculate shortest distance between point and line*

Description

This function calculates the shortest distance between a point and a line in 2D space.

Usage

```
.distancePointLine(x, y, a = 1, b = 1, c = 0)
```

Arguments

`x`	x-coordinate of point
`y`	y-coordinate of point
`a`	`numeric(1)` The coefficient in line equation a * x + b * y + c = 0. Defaults to 1.
`b`	`numeric(1)` The coefficient in line equation a * x + b * y + c = 0. Defaults to 1.
`c`	`numeric(1)` The intercept in line equation a * x + b * y + c = 0. Defaults to 0.

Value

`numeric` The shortest distance between a point and a line.

Examples

```
.distancePointLine(0, 0, 1, -1, 1)
```

`.distancePointSegment` *Calculate shortest distance between point and line segment*

Description

This function calculates the shortest distance between a point and a line segment in 2D space.

Usage

```
.distancePointSegment(x, y, x1, y1, x2, y2)
```

Arguments

`x`	x-coordinate of point
`y`	y-coordinate of point
`x1`	x-coordinate of one endpoint of the line segment
`y1`	y-coordinate of line segment endpoint with x-coordinate x1
`x2`	x-coordinate of other endpoint of line segment
`y2`	y-coordinate of line segment endpoint with x-coordinate x2

Value

`numeric` The shortest distance between a point and a line segment

Examples

```
.distancePointSegment(0, 0, -1, 1, 1, -1)
```

`.fitCurve2` *Curve fitting via `stats::optim` L-BFGS-B with fall-back grid/pattern search if convergence is not achieved.*

Description

Function to fit curve via stats::optim

Usage

```
.fitCurve2(
  par,
  x,
  y,
  fn,
  loss,
  lower = -Inf,
  upper = Inf,
  precision = 1e-04,
  density = c(2, 10, 5),
  step = 0.5/density,
  ...,
  loss_args = list(),
  span = 1,
  optim_only = FALSE,
 control = list(factr = 1e-08, ndeps = rep(1e-04, times = length(par)), trace = 0)
)
```

Arguments

`par`	`numeric` Vector of intial guesses for the parameters. For each index `i` of `par`, par[i] must be within the range (`lower\[i\]`, `upper\[i\]`). If only a single `upper` or `lower` value is present, that range is used for all parameters in `par`.
`x`	`numeric` Values to evaluate `fn` for.
`y`	`numeric` Target output values to optimze `fn` against.
`fn`	`function` A function to optimize. Any `fn` arguments passed via `...` will be treated as constant and removed from the optimization. It is assumed that the first argument is the `x` value to optimize over and any subsequent arguments are free parameters to be optimized. Transformed to be optim compatible via `make_optim_function` is the first arguement isn't already `par`.
`loss`	`character(1)` or `function` Either the name of one of the bundled loss functions (see details) or a custom loss function to compute for the output of `fn` over `x`.
`lower`	`numeric(1)` Lower bound for parameters. Parallel to `par`.
`upper`	`numeric(1)` Upper bound for paramteres. Parallel to `par`.
`precision`	`numeric` smallest step size used in pattern search, once step size drops below this value, the search terminates.
`density`	`numeric` how many points in the dimension of each parameter should be evaluated (density of the grid)
`step`	initial step size for pattern search.
`...`	`pairlist` Fall through arguments to `fn`.
`loss_args`	`list` Additional argument to the `loss` function. These get passed to losss via `do.call` analagously to using `...`.
`span`	`numeric` Can be safely kept at 1, multiplicative ratio for initial step size in pattern search. Must be larger than precision.

`optim_only`	`logical(1)` Should the fall back methods when optim fails be skipped? Default is `FALSE`.
`control`	`list` List of control parameters to pass to `optim`. See `?optim` for details.

Details

TODO

Value

`numeric` Vector of optimal parameters for `fn` fit against `y` on the values of `x`.

Examples

```
## Not run:
  # Four parameter hill curve equation
  hillEqn <- function(x, Emin, Emax, EC50, lambda) {
      (Emin + Emax * (x / EC50)^lambda) / (1 + (x / EC50)^lambda)
  }
  # Make some dummy data
  doses <- rev(1000 / (2^(1:20)))
  lambda <- 1
  Emin <- 1
  Emax <- 0.1
  EC50 <- median(doses)
  response <- hillEqn(doses, Emin=Emin, lambda=lambda, Emax=Emax, EC50=EC50)
  nresponse <- response + rnorm(length(response), sd=sd(response)*0.1) # add noise
  # 3-parameter optimization
  3par <- .fitCurve2(
      par=c(Emax, EC50, lambda),
      x=doses,
      y=nresponse,
      fn=hillEqn,
      Emin=Emin, # set this as constant in the function being optimized (via ...)
      loss=.normal_loss,
      loss_args=list(trunc=FALSE, n=1, scale=0.07),
      upper=c(1, max(doses), 6),
      lower=c(0, min(doses), 0)
  )
  # 2-parameter optimization
  2par <- .fitCurve2(
      par=c(Emax, EC50),
      x=doses,
      y=nresponse,
      fn=hillEqn,
      Emin=Emin, # set this as constant in the function being optimized (via ...)
      lambda=1,
      loss=.normal_loss,
      loss_args=list(trunc=FALSE, n=1, scale=0.07),
      upper=c(1, max(doses)),
      lower=c(0, min(doses))
  )
```

```
## End(Not run)
```

`.intersectList` *Intersect A List of More Than Two Vectors*

Description

Utility to find the intersection between a list of more than two vectors or lists This function extends the native intersect function to work on two or more arguments.

Usage

```
.intersectList(...)
```

Arguments

`...`	A list of or any number of vector like objects of the same mode, which could also be operated on by the native R set operations

Value

A vector like object of the same mode as the first argument, containing only the intersection common to all arguments to the function

Examples

```
list1 <- list('a', 'b', 'c')
list2 <- list('a', 'c')
list3 <- list('a', 'c', 'd')
listAll <- .intersectList(list1, list2, list3)
listAll
```

`.longTableToSummarizedExperiment` *Convert LongTable to gDR Style SummarizedExperiment*

Description

Convert LongTable to gDR Style SummarizedExperiment

Usage

```
.longTableToSummarizedExperiment(LT, assay_names)
```

Arguments

`LT`	`LongTable` to convert to gDR `SummarizedExperiment` format.
`assay_names`	`character()` Names to rename the assays to. These are assumed to be in the same order as `assayNames(LT)`.

Value

`SummarizedExperiment` object with all assay from `LT` as `BumpyMatrixes`.

`.symSetDiffList` *Utility to find the symmetric set difference of a list of two or more vectors or lists*

Description

The function finds the symmetric set differnces between all the arguments, defined as Union(args)-Intersection(args)

Usage

```
.symSetDiffList(...)
```

Arguments

`...`	A list of or any number of vector like objects of the same mode, which could also be operated on by the native R set operations

Value

A vector like object of the same mode as the first argument, containing only the symmetric set difference

Examples

```
list1 <- list('a', 'b', 'c')
list2 <- list('a', 'c')
list3 <- list('a', 'c', 'd')
listAll <- .symSetDiffList(list1, list2, list3)
listAll
```

.unionList — *Utility to find the union between a list of more than two vectors or lists*

Description

This function extends the native union function to work on two or more arguments.

Usage

```
.unionList(...)
```

Arguments

`...`	A list of or any number of vector like objects of the same mode, which could also be operated on by the native R set operations

Value

A vector like object of the same mode as the first argument, containing all the elements of all arguments passed to the function

Examples

```
list1 <- list('a', 'b')
list2 <- list('a', 'c')
list3 <- list('c', 'd')
listAll <- .unionList(list1, list2, list3)
listAll
```

aggregate,data.table-method — *Functional S4 API for aggregation over a* `data.table` *object.*

Description

Compute a group-by operation over a `data.table` in a functional, pipe compatible format.

Usage

```
## S4 method for signature 'data.table'
aggregate(
  x,
  by,
  ...,
  subset = TRUE,
  nthread = 1,
  progress = TRUE,
  BPPARAM = NULL,
  enlist = TRUE,
  moreArgs = list()
)
```

Arguments

`x`	`data.table` to compute aggregation over.
`by`	`character` One or more valid column names in `x` to compute groups using.
`...`	`call` One or more aggregations to compute for each group by in x. If you name aggregation calls, that will be the column name of the value in the resulting `data.table` otherwise a default name will be parsed from the function name and its first argument, which is assumed to be the name of the column being aggregated over.
`subset`	`call` An R call to evaluate before perfoming an aggregate. This allows you to aggregate over a subset of columns in an assay but have it be assigned to the parent object. Default is TRUE, which includes all rows. Passed through as the `i` argument in `[.data.table`.
`nthread`	`numeric(1)` Number of threads to use for split-apply-combine parallelization. Uses `BiocParllel::bplapply` if nthread > 1 or you pass in `BPPARAM`. Does not modify data.table threads, so be sure to use setDTthreads for reasonable nested parallelism. See details for performance considerations.
`progress`	`logical(1)` Display a progress bar for parallelized computations? Only works if `bpprogressbar<-` is defined for the current BiocParallel back-end.
`BPPARAM`	`BiocParallelParam` object. Use to customized the the parallization back-end of bplapply. Note, nthread over-rides any settings from BPPARAM as long as `bpworkers<-` is defined for that class.
`enlist`	`logical(1)` Default is `TRUE`. Set to `FALSE` to evaluate the first call in `...` within `data.table` groups. See details for more information.
`moreArgs`	`list()` A named list where each item is an argument one of the calls in `...` which is not a column in the table being aggregated. Use to further parameterize you calls. Please note that these are not added to your aggregate calls unless you specify the names in the call.

Details

This S4 method override the default `aggregate` method for a `data.frame` and as such you need to call `aggregate.data.frame` directly to get the original S3 method for a `data.table`.

Use of Non-Standard Evaluation:

Arguments in `...` are substituted and wrapped in a list, which is passed through to the j argument of `[.data.table` internally. The function currently tries to build informative column names for unnamed arguments in `...` by appending the name of each function call with the name of its first argument, which is assumed to be the column name being aggregated over. If an argument to `...` is named, that will be the column name of its value in the resulting `data.table`.

Enlisting:

The primary use case for `enlist=FALSE` is to allow computation of dependent aggregations, where the output from a previous aggregation is required in a subsequent one. For this case, wrap your call in `{` and assign intermediate results to variables, returning the final results as a list where each list item will become a column in the final table with the corresponding name. Name inference is disabled for this case, since it is assumed you will name the returned list items appropriately. A major advantage over multiple calls to `aggregate` is that the overhead of parallelization is paid only once even for complex multi-step computations like fitting a model, capturing its paramters, and making predictions using it. It also allows capturing arbitrarily complex calls which can be recomputed later using the `update,TreatmentResponseExperiment-method` A potential disadvantage is increased RAM usage per thread due to storing intermediate values in variables, as well as any memory allocation overhead associate therewith.

Value

`data.table` of aggregated results with an `aggregations` attribute capturing metadata about the last aggregation performed on the table.

`aggregate,LongTable-method`

Functional API for aggregation over a `LongTable` *or inheriting class*

Description

Compute a group-by operation over a `LongTable` object or it's inhering classes.

Usage

```
## S4 method for signature 'LongTable'
aggregate(
  x,
  assay,
  by,
  ...,
  subset = TRUE,
  nthread = 1,
  progress = TRUE,
  BPPARAM = NULL,
  enlist = TRUE,
  moreArgs = list()
)
```

Arguments

`x`	`LongTable` or inheriting class to compute aggregation on.
`assay`	`character(1)` The assay to aggregate over.
`by`	`character` One or more valid column names in `x` to compute groups using.

`...`	`call` One or more aggregations to compute for each group by in x. If you name aggregation calls, that will be the column name of the value in the resulting `data.table` otherwise a default name will be parsed from the function name and its first argument, which is assumed to be the name of the column being aggregated over.
`subset`	`call` An R call to evaluate before perfoming an aggregate. This allows you to aggregate over a subset of columns in an assay but have it be assigned to the parent object. Default is TRUE, which includes all rows. Passed through as the `i` argument in `[.data.table`.
`nthread`	`numeric(1)` Number of threads to use for split-apply-combine parallelization. Uses `BiocParllel::bplapply` if nthread > 1 or you pass in `BPPARAM`. Does not modify data.table threads, so be sure to use setDTthreads for reasonable nested parallelism. See details for performance considerations.
`progress`	`logical(1)` Display a progress bar for parallelized computations? Only works if `bpprogressbar<-` is defined for the current BiocParallel back-end.
`BPPARAM`	`BiocParallelParam` object. Use to customized the the parallization back-end of bplapply. Note, nthread over-rides any settings from BPPARAM as long as `bpworkers<-` is defined for that class.
`enlist`	`logical(1)` Default is `TRUE`. Set to `FALSE` to evaluate the first call in `...` within `data.table` groups. See details for more information.
`moreArgs`	`list()` A named list where each item is an argument one of the calls in `...` which is not a column in the table being aggregated. Use to further parameterize you calls. Please note that these are not added to your aggregate calls unless you specify the names in the call.

Details

Use of Non-Standard Evaluation:

Arguments in `...` are substituted and wrapped in a list, which is passed through to the j argument of `[.data.table` internally. The function currently tries to build informative column names for unnamed arguments in `...` by appending the name of each function call with the name of its first argument, which is assumed to be the column name being aggregated over. If an argument to `...` is named, that will be the column name of its value in the resulting `data.table`.

Enlisting:

The primary use case for `enlist=FALSE` is to allow computation of dependent aggregations, where the output from a previous aggregation is required in a subsequent one. For this case, wrap your call in { and assign intermediate results to variables, returning the final results as a list where each list item will become a column in the final table with the corresponding name. Name inference is disabled for this case, since it is assumed you will name the returned list items appropriately. A major advantage over multiple calls to `aggregate` is that the overhead of parallelization is paid only once even for complex multi-step computations like fitting a model, capturing its paramters, and making predictions using it. It also allows capturing arbitrarily complex calls which can be recomputed later using the `update,TreatmentResponseExperiment-method` A potential disadvantage is increased RAM usage per thread due to storing intermediate values in variables, as well as any memory allocation overhead associate therewith.

Value

`data.table` of aggregation results.

See Also

`data.table::[.data.table`, `BiocParallel::bplapply`

`aggregate2`	*Functional API for data.table aggregation which allows capture of associated aggregate calls so they can be recomputed later.*

Description

Functional API for data.table aggregation which allows capture of associated aggregate calls so they can be recomputed later.

Usage

```
aggregate2(
  x,
  by,
  ...,
  nthread = 1,
  progress = interactive(),
  BPPARAM = NULL,
  enlist = TRUE,
  moreArgs = list()
)
```

Arguments

`x`	`data.table`
`by`	`character` One or more valid column names in `x` to compute groups using.
`...`	`call` One or more aggregations to compute for each group by in x. If you name aggregation calls, that will be the column name of the value in the resulting `data.table` otherwise a default name will be parsed from the function name and its first argument, which is assumed to be the name of the column being aggregated over.
`nthread`	`numeric(1)` Number of threads to use for split-apply-combine parallelization. Uses `BiocParllel::bplapply` if nthread > 1 or you pass in `BPPARAM`. Does not modify data.table threads, so be sure to use setDTthreads for reasonable nested parallelism. See details for performance considerations.
`progress`	`logical(1)` Display a progress bar for parallelized computations? Only works if `bpprogressbar<-` is defined for the current BiocParallel back-end.

`BPPARAM`	`BiocParallelParam` object. Use to customized the the parallization back-end of bplapply. Note, nthread over-rides any settings from BPPARAM as long as `bpworkers<-` is defined for that class.
`enlist`	`logical(1)` Default is `TRUE`. Set to `FALSE` to evaluate the first call in `...` within `data.table` groups. See details for more information.
`moreArgs`	`list()` A named list where each item is an argument one of the calls in `...` which is not a column in the table being aggregated. Use to further parameterize you calls. Please note that these are not added to your aggregate calls unless you specify the names in the call.

Details

Use of Non-Standard Evaluation:

Arguments in `...` are substituted and wrapped in a list, which is passed through to the j argument of `[.data.table` internally. The function currently tries to build informative column names for unnamed arguments in `...` by appending the name of each function call with the name of its first argument, which is assumed to be the column name being aggregated over. If an argument to `...` is named, that will be the column name of its value in the resulting `data.table`.

Enlisting:

The primary use case for `enlist=FALSE` is to allow computation of dependent aggregations, where the output from a previous aggregation is required in a subsequent one. For this case, wrap your call in `{` and assign intermediate results to variables, returning the final results as a list where each list item will become a column in the final table with the corresponding name. Name inference is disabled for this case, since it is assumed you will name the returned list items appropriately. A major advantage over multiple calls to `aggregate` is that the overhead of parallelization is paid only once even for complex multi-step computations like fitting a model, capturing its paramters, and making predictions using it. It also allows capturing arbitrarily complex calls which can be recomputed later using the `update,TreatmentResponseExperiment-method` A potential disadvantage is increased RAM usage per thread due to storing intermediate values in variables, as well as any memory allocation overhead associate therewith.

Value

`data.table` of aggregation results.

See Also

`data.table::[.data.table`, `BiocParallel::bplapply`

`amcc` *Calculate an Adaptive Matthews Correlation Coefficient*

Description

This function calculates an Adaptive Matthews Correlation Coefficient (AMCC) for two vectors of values of the same length. It assumes the entries in the two vectors are paired. The Adaptive Matthews Correlation Coefficient for two vectors of values is defined as the Maximum Matthews Coefficient over all possible binary splits of the ranks of the two vectors. In this way, it calculates the best possible agreement of a binary classifier on the two vectors of data. If the AMCC is low, then it is impossible to find any binary classification of the two vectors with a high degree of concordance.

Usage

```
amcc(x, y, step.prct = 0, min.cat = 3, nperm = 1000, nthread = 1, ...)
```

Arguments

`x, y`	Two paired vectors of values. Could be replicates of observations for the same experiments for example.
`step.prct`	Instead of testing all possible splits of the data, it is possible to test steps of a percentage size of the total number of ranks in x/y. If this variable is 0, function defaults to testing all possible splits.
`min.cat`	The minimum number of members per category. Classifications with less members fitting into both categories will not be considered.
`nperm`	The number of perumatation to use for estimating significance. If 0, then no p-value is calculated.
`nthread`	Number of threads to parallize over. Both the AMCC calculation and the permutation testing is done in parallel.
`...`	Additional arguments

Value

Returns a list with two elements. $amcc contains the highest 'mcc' value over all the splits, the p value, as well as the rank at which the split was done.

Examples

```
x <- c(1,2,3,4,5,6,7)
y <- c(1,3,5,4,2,7,6)
amcc(x,y, min.cat=2)
```

as *Coerce a* `LongTable` *to a* `TreatmentResponseExperiment`

Description

Coerce a LongTable into a `data.table`.

Currently only supports coercing to data.table or data.frame

Coerce a data.table with the proper configuration attributes back to a LongTable

Arguments

`from`	A `LongTableDataMapper` to coerce.

Value

The data in `object`, as the child-class `TreatmentResponseExperiment`.

A `data.table` with the data from a LongTable.

`data.table` containing the data from the LongTable, with the 'longTableDataMapper' attribute containg the metadata needed to reverse the coercing operation.

`LongTable` object configured with the longTableDataMapper

`data.table` with long format of data in `from`

`data.frame` with long format of data in `from`.

`SummarizedExperiment` with each assay as a `BumpyMatrix`

A `TREDataMapper` object.

See Also

`TreatmentResponseExperiment`

`BumpyMatrix::BumpyMatrix`

Examples

```
data(clevelandSmall_cSet)
TRE <- as(treatmentResponse(clevelandSmall_cSet),
    "TreatmentResponseExperiment")
TRE

as(merckLongTable, 'data.table')

dataTable <- as(merckLongTable, 'data.table')
print(attr(dataTable, 'longTableDataMapper')) # Method doesn't work without this
as(dataTable, 'LongTable')

SE <- molecularProfilesSlot(clevelandSmall_cSet)[[1]]
as(SE, 'data.table')
```

```
SE <- molecularProfilesSlot(clevelandSmall_cSet)[[1]]
as(SE, 'data.frame')
```

`as.long.table` *Coerce from data.table to LongTable*

Description

Coerce a data.table with the proper configuration attributes back to a LongTable

Usage

```
as.long.table(x)
```

Arguments

x	A `data.frame` with the 'longTableDataMapper' attribute, containing three lists named assayCols, rowDataCols and colDataCols. This attribute is automatically created when coercing from a LongTable to a data.table.

Value

`LongTable` object configured with the longTableDataMapper

Examples

```
dataTable <- as(merckLongTable, 'data.table')
print(attr(dataTable, 'longTableDataMapper')) # Method doesn't work without this
as.long.table(dataTable)
```

`assay,LongTableDataMapper,ANY-method`
Extract the data for an assay from a `LongTableDataMapper`

Description

Extract the data for an assay from a `LongTableDataMapper`

Usage

```
## S4 method for signature 'LongTableDataMapper,ANY'
assay(x, i, withDimnames = TRUE)
```

Arguments

x	`LongTableDataMapper` The object to retrive assay data form according to the `assayMap` slot.
i	`character(1)` Name of an assay in the `assayMap` slot of `x`.
withDimnames	`logical(1)` For compatibility with `SummarizedExperiment::assay` generic. Not used.

Value

`data.table` Data for the specified assay extracted from the `rawdata` slot of `x`.

---

assay,TREDataMapper,ANY-method

Extract the data for an assay from a `TREDataMapper`

Description

Extract the data for an assay from a `TREDataMapper`

Usage

```
## S4 method for signature 'TREDataMapper,ANY'
assay(x, i, withDimnames = TRUE)
```

Arguments

x	`TREDataMapper` The object to retrive assay data form according to the `assayMap` slot.
i	`character(1)` Name of an assay in the `assayMap` slot of `x`.
withDimnames	`logical(1)` For compatibility with `SummarizedExperiment::assay` generic. Not used.

Value

`data.table` Data for the specified assay extracted from the `rawdata` slot of `x`.

`assayCols` *Generic to access the assay columns of a rectangular object.*

Description

Generic to access the assay columns of a rectangular object.

Usage

```
assayCols(object, ...)
```

Arguments

`object`	`S4` An object to get assay ids from.
`...`	Allow new arguments to this generic.

Value

Depends on the implemented method.

Examples

```
print("Generics shouldn't need examples?")
```

`assayIndex` *Retrieve and assayIndex*

Description

Retrieve and assayIndex

Usage

```
assayIndex(x, ...)
```

Arguments

`x`	An `S4` object.
`...`	`pairlist` Allow definition of new parameters for implementations of this generic.

Value

An object representing the "assayIndex" of an `S4` object.

Examples

```
print("Generics shouldn't need examples?")
```

`assayKeys` *Retrieve a set of assayKeys*

Description

Retrieve a set of assayKeys

Usage

```
assayKeys(x, ...)
```

Arguments

`x`	An `S4` object.
`...`	`pairlist` Allow definition of new parameters for implementations of this generic.

Value

An object representing the "assayKeys" of an `S4` object.

Examples

```
print("Generics shouldn't need examples?")
```

`assays,LongTableDataMapper-method`
Extract the data for all assays from a `LongTableDataMapper`

Description

Extract the data for all assays from a `LongTableDataMapper`

Usage

```
## S4 method for signature 'LongTableDataMapper'
assays(x, withDimnames = TRUE)
```

Arguments

`x`	`LongTableDataMapper` The object to retrive assay data form according to the `assayMap` slot.
`withDimNames`	`logical(1)` For compatibility with `SummarizedExperiment::assay` generic. Not used.

Value

`list` Data for all assays extracted from the `rawdata` slot of `x` as a `list` of `data.tables`, where the `keys` for each table are their `id_columns`.

`assays,TREDataMapper-method` *Extract the data for all assays from a* `TREDataMapper`

Description

Extract the data for all assays from a `TREDataMapper`

Usage

```
## S4 method for signature 'TREDataMapper'
assays(x, withDimnames = TRUE)
```

Arguments

`x`	`TREDataMapper` The object to retrive assay data form according to the `assayMap` slot.
`withDimnames`	`logical(1)` For compatibility with `SummarizedExperiment::assay` generic. Not used.

Value

`list` Data for all assays extracted from the `rawdata` slot of `x` as a `list` of `data.tables`, where the `keys` for each table are their `id_columns`.

`assignment-immutable` *Intercept assignment operations for "immutable" S3 objects.*

Description

Prevents modification of objects labelled with the "immutable" S3-class by intercepting assignment during S3-method dispatch and returning an error.

Usage

```
\method{subset}{immutable}(object, ...) <- value

## S3 replacement method for class 'immutable'
object[...] <- value

## S3 replacement method for class 'immutable'
object[[...]] <- value

## S3 replacement method for class 'immutable'
object$... <- value

## S3 replacement method for class 'immutable'
names(x) <- value

## S3 replacement method for class 'immutable'
dimnames(x) <- value

\method{colnames}{immutable}(x) <- value

\method{rownames}{immutable}(x) <- value
```

Arguments

`object, x`	An R object inherting from the "immutable" S3-class.
`...`	Catch subset arguments for various dimensions.
`value`	Not used.

Value

None, throws an error.

Examples

```
immutable_df <- immutable(data.frame(a=1:5, b=letters[1:5]))
# return immutable data.frame
immutable_df[1:4, ]
# return immutable vector
```

```
immutable_df$a
```

`buildComboProfiles` *Build an assay table with an* `S4` *object.*

Description

Build an assay table with an `S4` object.

Usage

```
buildComboProfiles(object, ...)
```

Arguments

`object`	`S4` An S4 object a list-like slot containing assays for the object.
`...`	Allow new arguments to be defined for this generic.

Value

`data.table`.

Examples

```
"This is a generic method!"
```

`buildComboProfiles,LongTable-method`
Build an assay table with selected assay profiles for drug combinations

---

Description

Build an assay table with selected assay profiles for drug combinations

Usage

```
## S4 method for signature 'LongTable'
buildComboProfiles(object, profiles)
```

Arguments

`object`	`LongTable` or inheriting class containing curated drug combination data.
`profiles`	`character` a vector of profile names, i.e., column names of assays.

Value

A `data.table` containing fields `treatment1id`, `treatment1dose`, `treatment2id`, `treatment2dose`, `sampleid`, which are used as keys to keep track of profiles, along with columns of selected profiles from their assays. Each `*_1` is the monothearpy profile of treatment 1 in the combination, and the same rule applies to treatment 2.

Examples

```
## Not run:
combo_profile_1 <- buildComboProfiles(tre, c("auc", "SCORE"))
combo_profile_2 <- buildComboProfiles(tre, c("HS", "EC50", "E_inf", "ZIP"))

## End(Not run)
```

`buildLongTable` *Build a LongTable object*

Description

Build a LongTable object

Usage

```
buildLongTable(from, ...)
```

Arguments

`from`	What to build the LongTable from?
`...`	`pairlist` Allow definition of new parameters for implementations of this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`buildLongTable,character-method`
LongTable build method from character

Description

LongTable Create a LongTable object from a single .csv file

Usage

```
## S4 method for signature 'character'
buildLongTable(from, rowDataCols, colDataCols, assayCols)
```

Arguments

`from`	`character` Path to the .csv file containing the data and metadata from which to build the `LongTable`.
`rowDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as cell identifiers (e.g., cell-line name columns) and the second containing any additional metadata columns related to the cell identifiers.
`colDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as column identifiers (e.g., drug name columns) and the second containing any additional metadata columns related to the column identifiers.
`assayCols`	`list` A named list of character vectors specifying how to parse assay columns into a list of `data.tables`. Each list data.table will be named for the name of corresponding list item and contain the columns specified in the character vector of column names in each list item.

Value

A `LongTable` object containing one or more assays, indexed by rowID and colID.

`buildLongTable,data.frame-method`
LongTable build method

Description

Create a LongTable object from a single data.table or data.frame object.

Usage

```
## S4 method for signature 'data.frame'
buildLongTable(from, rowDataCols, colDataCols, assayCols)
```

Arguments

`from`	`character` Path to the .csv file containing the data and metadata from which to build the `LongTable`.
`rowDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as cell identifiers (e.g., cell-line name columns) and the second containing any additional metadata columns related to the cell identifiers. If you wish to rename any of these columns, assign the new names to their respective character vectors.
`colDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as column identifiers (e.g., drug name columns) and the second containing any additional metadata columns related to the column identifiers. If you wish to rename any of these columns, assign the new names to their respective character vectors.
`assayCols`	`list` A named list of character vectors specifying how to parse assay columns into a list of `data.tables`. Each list data.table will be named for the name of corresponding list item and contain the columns specified in the character vector of column names in each list item. If there are no names for assayCols, the assays will be numbered by instead.

Value

A `LongTable` object containing one or more assays, indexed by rowID and colID.

---

buildLongTable,list-method

LongTable build method from list

Description

Create a LongTable object from a list containing file paths, data.frames and data.tables.

Usage

```
## S4 method for signature 'list'
buildLongTable(from, rowDataCols, colDataCols, assayCols)
```

Arguments

`from`	`list` A list containing any combination of character file paths, data.tables and data.frames which will be used to construct the LongTable.
`rowDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as cell identifiers (e.g., cell-line name columns) and the second containing any additional metadata columns related to the cell identifiers.
`colDataCols`	`list` List with two `character` vectors, the first specifying one or more columns to be used as column identifiers (e.g., drug name columns) and the second containing any additional metadata columns related to the column identifiers.

`assayCols`	`list` A named list of character vectors specifying how to parse assay columns into a list of `data.tables`. Each list data.table will be named for the name of corresponding list item and contain the columns specified in the character vector of column names in each list item.

Value

A `LongTable` object constructed with the data in `from`.

Examples

```
## Not run:
assayList <- assays(merckLongTable, withDimnames=TRUE)
rowDataCols <- list(rowIDs(merckLongTable), rowMeta(merckLongTable))
colDataCols <- list(colIDs(merckLongTable), colMeta(merckLongTable))
assayCols <- assayCols(merckLongTable)
longTable <- buildLongTable(from=assayList, rowDataCols, colDataCols, assayCols)

## End(Not run)
```

`c.immutable` *Intercept concatenation for "immutable" class objects to return another "immutable" class object.*

Description

Ensures that `c` and `append` to an "immutable" class object return an immutable class object.

Usage

```
## S3 method for class 'immutable'
c(x, ...)
```

Arguments

`x`	An R object inheriting from the "immutable" S3-clas
`...`	Objects to concatenate to `x`.

Value

x with one or more values appended to it.

callingWaterfall *Drug sensitivity calling using waterfall plots*

Description

1. Sensitivity calls were made using one of IC50, ActArea or Amax

Usage

```
callingWaterfall(
  x,
  type = c("IC50", "AUC", "AMAX"),
  intermediate.fold = c(4, 1.2, 1.2),
  cor.min.linear = 0.95,
  name = "Drug",
  plot = FALSE
)
```

Arguments

`x`	What type of object does this take in?
`type`	ic50: IC50 values in micro molar (positive values) actarea: Activity Area, that is area under the drug activity curve (positive values) amax: Activity at max concentration (positive values)
`intermediate.fold`	vector of fold changes used to define the intermediate sensitivities for ic50, actarea and amax respectively
`cor.min.linear`	`numeric` The minimum linear correlation to require?
`name`	`character` The name of the output to use in plot
`plot`	`boolean` Whether to plot the results

Details

1. Sort log IC50s (or ActArea or Amax) of the samples to generate a “waterfall distribution”
2. Identify cutoff:

3.1 If the waterfall distribution is non-linear (pearson cc to the linear fit <=0.95), estimate the major inflection point of the log IC50 curve as the point on the curve with the maximal distance to a line drawn between the start and end points of the distribution.

3.2 If the waterfall distribution appears linear (pearson cc to the linear fit > 0.95), then use the median IC50 instead.

1. Samples within a 4-fold IC50 (or within a 1.2-fold ActArea or 20% Amax difference) difference centered around this inflection point are classified as being “intermediate”, samples with lower IC50s (or ActArea/Amax values) than this range are defined as sensitive, and those with IC50s (or ActArea/Amax) higher than this range are called “insensitive”.
2. Require at least x sensitive and x insensitive samples after applying these criteria (x=5 in our case).

Value

`factor` Containing the drug sensitivity status of each sample.

Examples

```
# Dummy example
1 + 1
```

---

checkColumnCardinality

Search a data.frame for 1:`cardinality` *relationships between a group of columns (your identifiers) and all other columns.*

Description

Search a data.frame for 1:`cardinality` relationships between a group of columns (your identifiers) and all other columns.

Usage

```
checkColumnCardinality(df, group, cardinality = 1, ...)
```

Arguments

`df`	A `data.frame` to search for 1:`cardinality` mappings with the columns in `group`.
`group`	A `character` vector of one or more column names to check the cardinality of other columns against.
`cardinality`	The cardinality of to search for (i.e., 1:`cardinality`) relationships with the combination of columns in group. Defaults to 1 (i.e., 1:1 mappings).
`...`	Fall through arguments to data.table::`[`. For developer use. One use case is setting verbose=TRUE to diagnose slow data.table operations.

Value

A `character` vector with the names of the columns with cardinality of 1:`cardinality` with the columns listed in `group`.

Examples

```
df <- rawdata(exampleDataMapper)
checkColumnCardinality(df, group='treatmentid')
```

`checkCsetStructure` *A function to verify the structure of a CoreSet*

Description

This function checks the structure of a PharamcoSet, ensuring that the correct annotations are in place and all the required slots are filled so that matching of samples and drugs can be properly done across different types of data and with other studies.

Usage

```
checkCsetStructure(object, plotDist = FALSE, result.dir = tempdir())
```

Arguments

`object`	A `CoreSet` to be verified
`plotDist`	Should the function also plot the distribution of molecular data?
`result.dir`	The path to the directory for saving the plots as a string. Defaults to this R sessions `tempdir()`.

Value

Prints out messages whenever describing the errors found in the structure of the cSet object passed in.

Examples

```
checkCsetStructure(clevelandSmall_cSet)
```

`clevelandSmall_cSet` *Cleaveland_mut RadioSet subsetted and cast as CoreSet*

Description

This dataset is just a dummy object derived from the Cleveland_mut RadioSet in the RadioGx R package. It's contents should not be interpreted and it is only present to test the functions in this package and provide examples

Usage

```
data(clevelandSmall_cSet)
```

Format

CoreSet object

References

Lamb et al. The Connectivity Map: using gene-expression signatures to connect small molecules, genes, and disease. Science, 2006.

`colData,LongTableDataMapper-method`

Convenience method to subset the `colData` *out of the* `rawdata` *slot using the assigned* `colDataMap` *metadata.*

Description

Convenience method to subset the `colData` out of the `rawdata` slot using the assigned `colDataMap` metadata.

Usage

```
## S4 method for signature 'LongTableDataMapper'
colData(x, key = TRUE)
```

Arguments

`x`	`LongTableDataMapper` object with valid data in the `rawdata` and `colDataMap` slots.
`key`	`logical(1)` Should the table be keyed according to the `id_columns` of the `colDataMap` slot? This will sort the table in memory. Default is TRUE.

Value

`data.table` The `colData` as specified in the `colDataMap` slot.

`colData,TREDataMapper-method`

Convenience method to subset the `colData` *out of the* `rawdata` *slot using the assigned* `colDataMap` *metadata.*

Description

Convenience method to subset the `colData` out of the `rawdata` slot using the assigned `colDataMap` metadata.

Usage

```
## S4 method for signature 'TREDataMapper'
colData(x, key = TRUE)
```

Arguments

`x`	`TREDataMapper` object with valid data in the `rawdata` and `colDataMap` slots.
`key`	`logical(1)` Should the table be keyed according to the `id_columns` of the `colDataMap` slot? This will sort the table in memory. Default is TRUE.

Value

`data.table` The `colData` as specified in the `colDataMap` slot.

`colIDs` *Generic to access the row identifiers for an object.*

Description

Generic to access the row identifiers for an object.

Usage

```
colIDs(object, ...)
```

Arguments

`object`	`S4` An object to get column id columns from.
`...`	ALlow new arguments to this generic

Value

Depends on the implemented method.

Examples

```
print("Generics shouldn't need examples?")
```

`collect_fn_params`	*Collects all function arguments other than the first into a single list parameter.*

Description

Useful for converting a regular function into a function amenable to optimization via `stats::optim`, which requires all free parameters be passed as a single vector `par`.

Usage

```
collect_fn_params(fn)
```

Arguments

`fn`	`function` A non-primitive function to refactor such that the first argument becomes the second argument and all other parameters must be passed as a vector to the first argument of the new function via the `par` parameter.

Details

Takes a function of the form f(x, ...), where ... is any number of additional function parameters (bot not literal `...`!) and parses it to a function of the form f(par, x) where `par` is a vector of values for ... in the same order as the arguments appear in `fn`.

Value

`function` A new non-primitive function where the first argument is `par`, which takes a vector of parameters being optimized, and the second argument is the old first argument to `fn` (usually `x` since this is the independent variable to optimize the function over).

`colMeta`	*Generic to access the column identifiers for a rectangular object.*

Description

Generic to access the column identifiers for a rectangular object.

Usage

```
colMeta(object, ...)
```

Arguments

`object`	`S4` An object to get column metadata columns from.
`...`	ALlow new arguments to this generic

Value

Depends on impemented method.

Examples

```
print("Generics shouldn't need examples?")
```

`connectivityScore` *Function computing connectivity scores between two signatures*

Description

A function for finding the connectivity between two signatures, using either the GSEA method based on the KS statistic, or the gwc method based on a weighted spearman statistic. The GSEA analysis is implemented in the piano package.

Usage

```
connectivityScore(
  x,
  y,
  method = c("fgsea", "gwc"),
  nperm = 10000,
  nthread = 1,
  gwc.method = c("spearman", "pearson"),
  ...
)
```

Arguments

`x`	A `matrix` with the first gene signature. In the case of GSEA the vector of values per gene for GSEA in which we are looking for an enrichment. In the case of gwc, this should be a matrix, with the per gene responses in the first column, and the significance values in the second.
`y`	A `matrix` with the second signature. In the case of GSEA, this is the vector of up and down regulated genes we are looking for in our signature, with the direction being determined from the sign. In the case of gwc, this should be a matrix of identical size to x, once again with the per gene responses in the first column, and their significance in the second.
`method`	`character` string identifying which method to use, out of 'fgsea' and 'gwc'
`nperm`	`numeric`, how many permutations should be done to determine significance through permutation testing? The minimum is 100, default is 1e4.
`nthread`	`numeric`, how many cores to run parallel processing on.
`gwc.method`	`character`, should gwc use a weighted spearman or pearson statistic?
`...`	Additional arguments passed down to gsea and gwc functions

Value

`numeric` a numeric vector with the score and the p-value associated with it

References

F. Pozzi, T. Di Matteo, T. Aste, 'Exponential smoothing weighted correlations', The European Physical Journal B, Vol. 85, No 6, 2012. DOI: 10.1140/epjb/e2012-20697-x

Varemo, L., Nielsen, J. and Nookaew, I. (2013) Enriching the gene set analysis of genome-wide data by incorporating directionality of gene expression and combining statistical hypotheses and methods. Nucleic Acids Research. 41 (8), 4378-4391. doi: 10.1093/nar/gkt111

Examples

```
xValue <- c(1,5,23,4,8,9,2,19,11,12,13)
xSig <- c(0.01, 0.001, .97, 0.01,0.01,0.28,0.7,0.01,0.01,0.01,0.01)
yValue <- c(1,5,10,4,8,19,22,19,11,12,13)
ySig <- c(0.01, 0.001, .97,0.01, 0.01,0.78,0.9,0.01,0.01,0.01,0.01)
xx <- cbind(xValue, xSig)
yy <- cbind(yValue, ySig)
rownames(xx) <- rownames(yy) <- c('1','2','3','4','5','6','7','8','9','10','11')
data.cor <- connectivityScore(xx,yy,method='gwc', gwc.method='spearman', nperm=300)
```

`CoreGx-deprecated` *List of* `deprecated` *or* `defunct` *methods in the* `CoreGx` *R package.*

Description

List of `deprecated` or `defunct` methods in the `CoreGx` R package.

Details

deprecated:

`CoreSet`: The `CoreSet` constructor is being updated to have a new API. This API is currently available via the `CoreSet2` constructor. In Bioconductor 3.16, the old constructor will be renamed `CoreSet2` and the new constructor will be renamed `CoreSet`.

defunct:

`buildLongTable`: This function no longer works as building a `LongTable` or `TreatmentResponseExperiment` now uses a `DataMapper` and the `metaConstruct` method. See `vignette("LongTable")` for a detailed description of how to create a LongTable object.

CoreSet *CoreSet constructor*

Description

A constructor that simplifies the process of creating CoreSets, as well as creates empty objects for data not provided to the constructor. Only objects returned by this constructor are expected to work with the CoreSet methods.

Usage

```
CoreSet(
  name,
  molecularProfiles = list(),
  sample = data.frame(),
  sensitivityInfo = data.frame(),
  sensitivityRaw = array(dim = c(0, 0, 0)),
  sensitivityProfiles = matrix(),
  sensitivityN = matrix(nrow = 0, ncol = 0),
  perturbationN = array(NA, dim = c(0, 0, 0)),
  curationSample = data.frame(),
  curationTissue = data.frame(),
  curationTreatment = data.frame(),
  treatment = data.frame(),
  datasetType = c("sensitivity", "perturbation", "both"),
  verify = TRUE,
  ...
)
```

Arguments

`name`	A `character` string detailing the name of the dataset
`molecularProfiles`	A `list` of SummarizedExperiment objects containing molecular profiles for each molecular data type.
`sample`	A `data.frame` containing the annotations for all the sample profiled in the data set, across all data types. Must contain the mandatory `sampleid` column which uniquely identifies each sample in the object.
`sensitivityInfo`	A `data.frame` containing the information for the sensitivity experiments. Must contain a 'sampleid' column with unique identifiers to each sample, matching the `sample` object and a 'treatmentid' columns with unique indenifiers for each treatment, matching the `treatment` object.
`sensitivityRaw`	A 3 Dimensional `array` contaning the raw drug dose response data for the sensitivity experiments

`sensitivityProfiles`	`data.frame` containing drug sensitivity profile statistics such as IC50 and AUC
`sensitivityN`, `perturbationN`	A `data.frame` summarizing the available sensitivity/perturbation data
`curationSample`, `curationTissue`, `curationTreatment`	A `data.frame` mapping the names for samples, tissues and treatments used in the data set to universal identifiers used between different CoreSet objects
`treatment`	A `data.frame` containing annotations for all treatments profiled in the dataset. Must contain the mandatory `treatmentid` column which uniquely identifies each treatment in the object.
`datasetType`	A `character(1)` string of 'sensitivity', 'preturbation', or 'both' detailing what type of data can be found in the `CoreSet`, for proper processing of the data
`verify`	`logical(1)`Should the function verify the CoreSet and print out any errors it finds after construction?
`...`	Catch and parse any renamed constructor arguments.

Details

WARNING::

Parameters to this function have been renamed!

- cell is now sample
- drug is now treatment

Value

An object of class `CoreSet`

Examples

```
data(clevelandSmall_cSet)
clevelandSmall_cSet
```

`CoreSet-accessors` *Accessing and modifying information in a* `CoreSet`

Description

Documentation for the various setters and getters which allow manipulation of data in the slots of a `CoreSet` object.

Usage

```
## S4 method for signature 'CoreSet'
annotation(object)

## S4 replacement method for signature 'CoreSet,list'
annotation(object) <- value

## S4 method for signature 'CoreSet'
dateCreated(object)

## S4 replacement method for signature 'CoreSet,character'
dateCreated(object) <- value

## S4 method for signature 'CoreSet'
name(object)

## S4 replacement method for signature 'CoreSet'
name(object) <- value

## S4 method for signature 'CoreSet'
sampleInfo(object)

## S4 replacement method for signature 'CoreSet,data.frame'
sampleInfo(object) <- value

## S4 method for signature 'CoreSet'
sampleNames(object)

## S4 replacement method for signature 'CoreSet,character'
sampleNames(object) <- value

## S4 method for signature 'CoreSet'
treatmentInfo(object)

## S4 replacement method for signature 'CoreSet,data.frame'
treatmentInfo(object) <- value

## S4 method for signature 'CoreSet'
treatmentNames(object)

## S4 replacement method for signature 'CoreSet,character'
treatmentNames(object) <- value

## S4 method for signature 'CoreSet'
curation(object)

## S4 replacement method for signature 'CoreSet,list'
curation(object) <- value
```

```
## S4 method for signature 'CoreSet'
datasetType(object)

## S4 replacement method for signature 'CoreSet,character'
datasetType(object) <- value

## S4 method for signature 'CoreSet'
molecularProfiles(object, mDataType, assay)

## S4 replacement method for signature 'CoreSet,character,character,matrix'
molecularProfiles(object, mDataType, assay) <- value

## S4 replacement method for signature 'CoreSet,character,missing,matrix'
molecularProfiles(object, mDataType, assay) <- value

## S4 replacement method for signature 'CoreSet,missing,missing,list_OR_MAE'
molecularProfiles(object, mDataType, assay) <- value

## S4 method for signature 'CoreSet'
featureInfo(object, mDataType)

## S4 replacement method for signature 'CoreSet,character,data.frame'
featureInfo(object, mDataType) <- value

## S4 method for signature 'CoreSet,character'
phenoInfo(object, mDataType)

## S4 replacement method for signature 'CoreSet,character,data.frame'
phenoInfo(object, mDataType) <- value

## S4 method for signature 'CoreSet,character'
fNames(object, mDataType)

## S4 replacement method for signature 'CoreSet,character,character'
fNames(object, mDataType) <- value

## S4 method for signature 'CoreSet'
mDataNames(object)

## S4 replacement method for signature 'CoreSet'
mDataNames(object) <- value

## S4 method for signature 'CoreSet'
molecularProfilesSlot(object)

## S4 replacement method for signature 'CoreSet,list_OR_MAE'
molecularProfilesSlot(object) <- value
```

```
## S4 method for signature 'CoreSet'
sensitivityInfo(object, dimension, ...)

## S4 replacement method for signature 'CoreSet,data.frame'
sensitivityInfo(object, dimension, ...) <- value

## S4 method for signature 'CoreSet'
sensitivityMeasures(object)

## S4 replacement method for signature 'CoreSet,character'
sensitivityMeasures(object) <- value

## S4 method for signature 'CoreSet'
sensitivityProfiles(object)

## S4 replacement method for signature 'CoreSet,data.frame'
sensitivityProfiles(object) <- value

## S4 method for signature 'CoreSet'
sensitivityRaw(object)

## S4 replacement method for signature 'CoreSet,array'
sensitivityRaw(object) <- value

## S4 method for signature 'CoreSet'
treatmentResponse(object)

## S4 replacement method for signature 'CoreSet,list_OR_LongTable'
treatmentResponse(object) <- value

## S4 method for signature 'CoreSet'
sensNumber(object)

## S4 replacement method for signature 'CoreSet,matrix'
sensNumber(object) <- value

## S4 method for signature 'CoreSet'
pertNumber(object)

## S4 replacement method for signature 'CoreSet,array'
pertNumber(object) <- value
```

Arguments

`object`	A `CoreSet` object.
`value`	See details.
`mDataType`	`character(1)` The name of a molecular datatype to access from the `molecularProfiles`

	of a `CoreSet` object.
`assay`	`character(1)` A valid assay name in the `SummarizedExperiment` of `@molecularProfiles` of a CoreSet object for data type `mDataType`.
`dimension`	See details.
`...`	See details.

Details

@annotation:

annotation: A `list` of CoreSet annotations with items: 'name', the name of the object; 'dateCreated', date the object was created; 'sessionInfo', the `sessionInfo()` when the object was created; 'call', the R constructor call; and 'version', the object version.

annotation<-: Setter method for the annotation slot. Arguments:

- value: a `list` of annotations to update the CoreSet with.

@dateCreated:

dateCreated: `character(1)` The date the `CoreSet` object was created, as returned by the `date()` function.

dateCreated<-: Update the 'dateCreated' item in the `annotation` slot of a `CoreSet` object. Arguments:

- value: A `character(1)` vector, as returned by the `date()` function.

name: `character(1)` The name of the `CoreSet`, retreived from the `@annotation` slot.

name<-: Update the `@annotation$name` value in a `CoreSet` object.

- value: `character(1)` The name of the `CoreSet` object.

cellInfo: `data.frame` Metadata for all sample in a `CoreSet` object.

sampleInfo<-: assign updated sample annotations to the `CoreSet` object. Arguments:

- value: a `data.frame` object.

sampleNames: `character` Retrieve the rownames of the `data.frame` in the `sample` slot from a CoreSet object.

sampleNames<-: assign new rownames to the sampleInfo `data.frame` for a CoreSet object. Arguments:

- value: `character` vector of rownames for the `sampleInfo(object)` `data.frame`.

treatmentInfo: `data.frame` Metadata for all treatments in a `CoreSet` object. Arguments:

- object: `CoreSet` An object to retrieve treatment metadata from.

treatmentInfo<-: `CoreSet` object with updated treatment metadata. object. Arguments:

- object: `CoreSet` An object to set treatment metadata for.
- value: `data.frame` A new table of treatment metadata for `object`.

treatmentNames: `character` Names for all treatments in a `CoreSet` object. Arguments:

- object: `CoreSet` An object to retrieve treatment names from.

treatmentNames<-: `CoreSet` Object with updates treatment names. object. Arguments:

- object: `CoreSet` An object to set treatment names from.
- value: `character` A character vector of updated treatment names.

@curation:

curation: A `list` of curated mappings between identifiers in the CoreSet object and the original data publication. Contains two `data.frames`, 'sample' with sample ids and 'tissue' with tissue ids.

curation<-: Update the `curation` slot of a CoreSet object. Arugments:

- value: A `list` of `data.frames`, one for each type of curated identifier. For a `CoreSet` object the slot should contain tissue and sample id `data.frames`.

datasetType slot:

datasetType: `character(1)` The type treatment response in the `sensitivity` slot. Valid values are 'sensitivity', 'perturbation' or 'both'.

datasetType<-: Update the datasetType slot of a CoreSet object. Arguments:

- value: A `character(1)` vector with one of 'sensitivity', 'perturbation' or 'both'

@molecularProfiles:

molecularProfiles: `matrix()` Retrieve an assay in a `SummarizedExperiment` from the `molecularProfiles` slot of a `CoreSet` object with the specified `mDataType`. Valid `mDataType` arguments can be found with `mDataNames(object)`. Exclude `mDataType` and `assay` to access the entire slot. Arguments:

- assay: Optional `character(1)` vector specifying an assay in the `SummarizedExperiment` of the `molecularProfiles` slot of the `CoreSet` object for the specified `mDataType`. If excluded, defaults to modifying the first assay in the `SummarizedExperiment` for the given `mDataType`.

molecularProfiles<-: Update an assay in a `SummarizedExperiment` from the `molecularProfiles` slot of a CoreSet object with the specified `mDataType`. Valid `mDataType` arguments can be found with `mDataNames(object)`. Omit `mDataType` and `assay` to update the slot.

- assay: Optional `character(1)` vector specifying an assay in the `SummarizedExperiment` of the `molecularProfiles` slot of the `CoreSet` object for the specified `mDataType`. If excluded, defaults to modifying the first assay in the `SummarizedExperiment` for the given `mDataType`.
- value: A `matrix` of values to assign to the `assay` slot of the `SummarizedExperiment` for the selected `mDataType`. The rownames and column names must match the associated `SummarizedExperiment`.

featureInfo: Retrieve a `DataFrame` of feature metadata for the specified `mDataType` from the `molecularProfiles` slot of a `CoreSet` object. More specifically, retrieve the `@rowData` slot from the `SummarizedExperiment` from the `@molecularProfiles` of a `CoreSet` object with the name `mDataType`.

featureInfo<-: Update the `featureInfo(object, mDataType)` `DataFrame` with new feature metadata. Arguments:

- value: A `data.frame` or `DataFrame` with updated feature metadata for the specified molecular profile in the `molecularProfiles` slot of a `CoreSet` object.

phenoInfo: Return the `@colData` slot from the `SummarizedExperiment` of `mDataType`, containing sample-level metadata, from a `CoreSet` object.

phenoInfo<-: Update the `@colData` slot of the `SummarizedExperiment` of `mDataType` in the `@molecularProfiles` slot of a `CoreSet` object. This updates the sample-level metadata in-place.

- value: A `data.frame` or `DataFrame` object where rows are samples and columns are sample metadata.

fNames: `character()` The features names from the `rowData` slot of a `SummarizedExperiment` of `mDataType` within a `CoreSet` object.

fNames: Updates the rownames of the feature metadata (i.e., `rowData`) for a `SummarizedExperiment` of `mDataType` within a `CoreSet` object.

- value: `character()` A character vector of new features names for the `rowData` of the `SummarizedExperiment` of `mDataType` in the `@molecularProfiles` slot of a `CoreSet` object. Must be the same length as `nrow(featureInfo(object, mDataType))`, the number of rows in the feature metadata.

mDataNames: `character` Retrieve the names of the molecular data types available in the `molecularProfiles` slot of a `CoreSet` object. These are the options which can be used in the `mDataType` parameter of various `molecularProfiles` slot accessors methods.

mDataNames: Update the molecular data type names of the `molecularProfiles` slot of a CoreSet object. Arguments:

- value: `character` vector of molecular datatype names, with length equal to `length(molecularProfilesSlot(object`

molecularProfilesSlot: Return the contents of the `@molecularProfiles` slot of a `CoreSet` object. This will either be a `list` or `MultiAssayExperiment` of `SummarizedExperiments`.

molecularProfilesSlot<-: Update the contents of the `@molecularProfiles` slot of a `CoreSet` object. Arguemnts:

- value: A `list` or `MultiAssayExperiment` of `SummarizedExperiments`. The `list` and `assays` should be named for the molecular datatype in each `SummarizedExperiment`.

@treatmentResponse:

Arguments::

- `dimension`: Optional `character(1)` One of 'treatment', 'sample' or 'assay' to retrieve `rowData`, `colData` or the 'assay_metadata' assay from the `CoreSet` `@sensitvity` `LongTable` object, respectively. Ignored with warning if `@treatmentResponse` is not a `LongTable` object.
- `...`: Additional arguments to the `rowData` or `colData`. `LongTable` methods. Only used if the sensitivity slot contains a `LongTable` object instead of a `list` and the `dimension` argument is specified.

Methods::

sensitivityInfo: `DataFrame` or `data.frame` of sensitivity treatment combo by sample metadata for the `CoreSet` object. When the `dimension` parameter is used, it allows retrieval of the dimension specific metadata from the `LongTable` object in `@treatmentResponse` of a CoreSet object.

sensitivityInfo<-: Update the `@treatmentResponse` slot metadata for a `CoreSet` object. When used without the `dimension` argument is behaves similar to the old CoreSet implementation, where the `@treatmentResponse` slot contained a list with a `$info` `data.frame` item. When the `dimension` arugment is used, more complicated assignments can occur where 'sample' modifies the `@sensitvity LongTable` colData, 'treatment' the rowData and 'assay' the 'assay_metadata' assay. Arguments:

- value: A `data.frame` of treatment response experiment metadata, documenting experiment level metadata (mapping to treatments and samples). If the `@treatmentResponse` slot doesn't contain a `LongTable` and `dimension` is not specified, you can only modify existing columns as returned by `sensitivityInfo(object)`.

sensitivityMeaures: Get the 'sensitivityMeasures' available in a `CoreSet` object. Each measure reprents some summary of sample sensitivity to a given treatment, such as ic50, ec50, AUC, AAC, etc. The results are returned as a `character` vector with all available metrics for the PSet object.

sensitivityMeaures: Update the sensitivity meaure in a `CoreSet` object. Thesee values are the column names of the 'profiles' assay and represent various compued sensitviity metrics such as ic50, ec50, AUC, AAC, etc.

- value: A `character` vector of new sensitivity measure names, the then length of the character vector must matcht he number of columns of the 'profiles' assay, excluding metadata and key columns.

sensitivityProfiles: Return the sensitivity profile summaries from the sensitivity slot. This data.frame cotanins vaarious sensitivity summary metrics, such as ic50, amax, EC50, aac, HS, etc as columns, with rows as treatment by sample experiments.

sensitivityProfiles<-: Update the sensitivity profile summaries the sensitivity slot. Arguments: - value: A `data.frame` the the same number of rows as as returned by `sensitivityProfiles(object)`, but potentially modified columns, such as the computation of additional summary metrics.

sensitivityRaw: Access the raw sensitiity measurents for a CoreSet object. A 3D `array` where rows are experiment_ids, columns are doses and the third dimension is metric, either 'Dose' for the doses used or 'Viability' for the sample viability at that dose.

sensitvityRaw<-: Update the raw dose and viability data in a `CoreSet`.

- value: A 3D `array` object where rows are experiment_ids, columns are replicates and pages are c('Dose', 'Viability'), with the corresponding dose or viability measurement for that experiment_id and replicate.

sensNumber: Return a count of viability observations in a `CoreSet` object for each treatment-combo by sample combination.

sensNumber<-: Update the 'n' item, which holds a matrix with a count of treatment by sample-line experiment counts, in the `list` in `@treatmentResponse` slot of a `CoreSet` object. Will error when `@sensitviity` contains a `LongTable` object, since the counts are computed on the fly. Arguments:

- value: A `matrix` where rows are samples and columns are treatments, with a count of the number of experiments for each combination as the values.

pertNumber: `array` Summary of available perturbation experiments from in a `CoreSet` object. Returns a 3D `array` with the number of perturbation experiments per treatment and sample, and data type.

pertNumber<-: Update the `@perturbation$n` value in a `CoreSet` object, which stores a summary of the available perturbation experiments. Arguments:

- value: A new 3D `array` with the number of perturbation experiments per treatment and sample, and data type

Value

Accessors: See details.

Setters: An updated `CoreSet` object, returned invisibly.

Examples

```
data(clevelandSmall_cSet)

## @annotation

annotation(clevelandSmall_cSet)

annotation(clevelandSmall_cSet) <- annotation(clevelandSmall_cSet)

dateCreated(clevelandSmall_cSet)

## dateCreated
dateCreated(clevelandSmall_cSet) <- date()

name(clevelandSmall_cSet)

name(clevelandSmall_cSet) <- 'new_name'

sampleInfo(clevelandSmall_cSet) <- sampleInfo(clevelandSmall_cSet)

sampleNames(clevelandSmall_cSet)

sampleNames(clevelandSmall_cSet) <- sampleNames(clevelandSmall_cSet)

treatmentInfo(clevelandSmall_cSet)

treatmentInfo(clevelandSmall_cSet) <- treatmentInfo(clevelandSmall_cSet)

treatmentNames(clevelandSmall_cSet)

treatmentNames(clevelandSmall_cSet) <- treatmentNames(clevelandSmall_cSet)

## curation
curation(clevelandSmall_cSet)

curation(clevelandSmall_cSet) <- curation(clevelandSmall_cSet)

datasetType(clevelandSmall_cSet)

datasetType(clevelandSmall_cSet) <- 'both'
```

```
# No assay specified
molecularProfiles(clevelandSmall_cSet, 'rna') <- molecularProfiles(clevelandSmall_cSet, 'rna')

# Specific assay
molecularProfiles(clevelandSmall_cSet, 'rna', 'exprs') <-
    molecularProfiles(clevelandSmall_cSet, 'rna', 'exprs')

# Replace the whole slot
molecularProfiles(clevelandSmall_cSet) <- molecularProfiles(clevelandSmall_cSet)

featureInfo(clevelandSmall_cSet, 'rna')

featureInfo(clevelandSmall_cSet, 'rna') <- featureInfo(clevelandSmall_cSet, 'rna')

phenoInfo(clevelandSmall_cSet, 'rna')

phenoInfo(clevelandSmall_cSet, 'rna') <- phenoInfo(clevelandSmall_cSet, 'rna')

fNames(clevelandSmall_cSet, 'rna')

fNames(clevelandSmall_cSet, 'rna') <- fNames(clevelandSmall_cSet, 'rna')

mDataNames(clevelandSmall_cSet)

mDataNames(clevelandSmall_cSet) <- mDataNames(clevelandSmall_cSet)

molecularProfilesSlot(clevelandSmall_cSet)

molecularProfilesSlot(clevelandSmall_cSet) <- molecularProfilesSlot(clevelandSmall_cSet)

sensitivityInfo(clevelandSmall_cSet)

sensitivityInfo(clevelandSmall_cSet) <- sensitivityInfo(clevelandSmall_cSet)

sensitivityMeasures(clevelandSmall_cSet) <- sensitivityMeasures(clevelandSmall_cSet)

sensitivityMeasures(clevelandSmall_cSet) <- sensitivityMeasures(clevelandSmall_cSet)

sensitivityProfiles(clevelandSmall_cSet)

sensitivityProfiles(clevelandSmall_cSet) <- sensitivityProfiles(clevelandSmall_cSet)

head(sensitivityRaw(clevelandSmall_cSet))

sensitivityRaw(clevelandSmall_cSet) <- sensitivityRaw(clevelandSmall_cSet)

treatmentResponse(clevelandSmall_cSet)

treatmentResponse(clevelandSmall_cSet) <- treatmentResponse(clevelandSmall_cSet)

sensNumber(clevelandSmall_cSet)
sensNumber(clevelandSmall_cSet) <- sensNumber(clevelandSmall_cSet)

pertNumber(clevelandSmall_cSet)

pertNumber(clevelandSmall_cSet) <- pertNumber(clevelandSmall_cSet)
```

CoreSet-class   *CoreSet - A generic data container for molecular profiles and treatment response data*

Description

CoreSet - A generic data container for molecular profiles and treatment response data

Details

The CoreSet (cSet) class was developed as a superclass for pSets in the PharmacoGx and RadioGx packages to contain the data generated in screens of cancer sample lines for their genetic profile and sensitivities to therapy (Pharmacological or Radiation). This class is meant to be a superclass which is contained within the PharmacoSet (pSet) and RadioSet (rSet) objects exported by PharmacoGx and RadioGx. The format of the data is similar for both pSets and rSets, allowing much of the code to be abstracted into the CoreSet super-class. However, the models involved with quantifying sampleular response to Pharmacological and Radiation therapy are widely different, and extension of the cSet class allows the packages to apply the correct model for the given data.

Slots

`annotation` See Slots section.

`molecularProfiles` See Slots section.

`sample` See Slots section.

`treatment` See Slots section.

`treatmentResponse` See Slots section.

`perturbation` See Slots section.

`curation` See Slots section.

`datasetType` See Slots section.

Slots

- annotation: A `list` of annotation data about the `CoreSet`, including the `$name` and the session information for how the object was created, detailing the exact versions of R and all the packages used.
- molecularProfiles: A `list` or `MultiAssayExperiment` containing `CoreSet` object.
- sample: A `data.frame` containg the annotations for all the samples profiled in the data set, across all molecular data types and treatment response experiments.

- treatment: A `data.frame` containing the annotations for all treatments in the dataset, including the mandatory 'treatmentid' column to uniquely identify each treatment.
- treatmentResponse: A `list` or `LongTable` containing all the data for the treatment response experiment, including `$info`, a `data.frame` containing the experimental info, `$raw` a 3D `array` containing raw data, `$profiles`, a `data.frame` containing sensitivity profiles statistics, and `$n`, a `data.frame` detailing the number of experiments for each sample-drug/radiationInfo pair
- perturbation: `list` containing `$n`, a `data.frame` summarizing the available perturbation data. This slot is currently being deprecated.
- curation: `list` containing mappings for `treatment`, `sample` and `tissue` names used in the data set to universal identifiers used between different `CoreSet` objects.
- datasetType: `character` string of 'sensitivity', 'perturbation', or both detailing what type of data can be found in the `CoreSet`, for proper processing of the data

See Also

`CoreSet-accessors`

`CoreSet-utils` *Utility methods for a* `CoreSet` *object.*

Description

Documentation for utility methods for a `CoreSet` object, such as set operations like subset and intersect. See @details for information on different types of methods and their implementations.

Usage

```
## S4 method for signature 'CoreSet'
subsetBySample(x, samples)

## S4 method for signature 'CoreSet'
subsetByTreatment(x, treatments)

## S4 method for signature 'CoreSet'
subsetByFeature(x, features, mDataTypes)
```

Arguments

`x`	A `CoreSet` object.
`samples`	`character()` vector of sample names. Must be valid rownames from `sampleInfo(x)`.
`treatments`	`character()` vector of treatment names. Must be valid rownames from `treatmentInfo(x)`. This method does not work with `CoreSet` objects yet.
`features`	`character()` vector of feature names. Must be valid feature names for a given `mDataType`
`mDataTypes`	`character()` One or more molecular data types to to subset features by. Must be valid rownames for the selected SummarizedExperiment mDataTypes.

Details

subset methods:

subsetBySample: Subset a `CoreSet` object by sample identifier.

- value: a `CoreSet` object containing only `samples`.

subset methods:

subsetByTreatment: Subset a `CoreSet` object by treatment identifier.

- value: a `CoreSet` object containing only `treatments`.

subset methods:

subsetByFeature: Subset a `CoreSet` object by molecular feature identifier.

- value: a `CoreSet` object containing only `features`.

Value

See details.

Examples

```
data(clevelandSmall_cSet)

## subset methods

### subsetBySample
samples <- sampleInfo(clevelandSmall_cSet)$sampleid[seq_len(10)]
clevelandSmall_cSet_sub <- subsetBySample(clevelandSmall_cSet, samples)


## subset methods

### subsetByTreatment
#treatments <- treatmentInfo(clevelandSmall_cSet)$treatmentid[seq_len(10)]
#clevelandSmall_cSet_sub <- subsetByTreatment(clevelandSmall_cSet, treatments)


## subset methods

### subsetByFeature
features <- fNames(clevelandSmall_cSet, 'rna')[seq_len(5)]
clevelandSmall_cSet_sub <- subsetByFeature(clevelandSmall_cSet, features, 'rna')
```

CoreSet2 — *Make a CoreSet with the updated class structure*

Description

New implementation of the CoreSet constructor to support MAE and TRE. This constructor will be swapped with the original `CoreSet` constructor as part of an overhaul of the CoreSet class structure.

Usage

```
CoreSet2(
  name = "emptySet",
  treatment = data.frame(),
  sample = data.frame(),
  molecularProfiles = MultiAssayExperiment(),
  treatmentResponse = LongTable(),
  datasetType = "sensitivity",
  perturbation = list(n = array(dim = 3), info = "No perturbation data!"),
  curation = list(sample = data.frame(), treatment = data.frame())
)
```

Arguments

`name`	A `character(1)` vector with the `CoreSet` objects name.
`treatment`	A `data.frame` with treatment level metadata.
`sample`	A `data.frame` with sample level metadata for the union of samples in `treatmentResponse` and `molecularProfiles`.
`molecularProfiles`	A `MultiAssayExperiment` containing one `SummarizedExperiment` object for each molecular data type.
`treatmentResponse`	A `LongTable` or `LongTableDataMapper` object containing all treatment response data associated with the `CoreSet` object.
`datasetType`	A deprecated slot in a `CoreSet` object included for backwards compatibility. This may be removed in future releases.
`perturbation`	A deprecated slot in a `CoreSet` object included for backwards compatibility. This may be removed in future releases.
`curation`	A `list(2)` object with two items named `treatment` and `sample` with mappings from publication identifiers to standardized identifiers for both annotations, respectively.

Value

A `CoreSet` object storing standardized and curated treatment response and multiomic profile data associated with a given publication.

Examples

```
data(clevelandSmall_cSet)
clevelandSmall_cSet
```

`cosinePerm` *Cosine Permutations*

Description

Computes the cosine similarity and significance using permutation test. This function uses random numbers, to ensure reproducibility please call `set.seed()` before running the function.

Usage

```
cosinePerm(
  x,
  y,
  nperm = 1000,
  alternative = c("two.sided", "less", "greater"),
  include.perm = FALSE,
  nthread = 1,
  ...
)
```

Arguments

`x`	`factor` is the factors for the first variable
`y`	`factor` is the factors for the second variable
`nperm`	`integer` is the number of permutations to compute the null distribution of MCC estimates
`alternative`	`string` indicates the alternative hypothesis and must be one of ‘’two.sided”, ‘’greater” or ‘’less”. You can specify just the initial letter. ‘’greater” corresponds to positive association, ‘’less” to negative association. Options are ’two.sided’, ’less’, or ’greater’
`include.perm`	`boolean` indicates whether the estimates for the null distribution should be returned. Default set to ’FALSE’
`nthread`	`integer` is the number of threads to be used to perform the permutations in parallel
`...`	A `list` of fallthrough parameters

Value

A `list` estimate of the cosine similarity, p-value and estimates after random permutations (null distribution) in include.perm is set to ’TRUE’

Examples

```
x <- factor(c(1,2,1,2,1))
y <- factor(c(2,2,1,1,1))
cosinePerm(x, y)
```

`DataMapper-accessors` *Accessing and modifying data in a* `DataMapper` *object.*

Description

Documentation for the various setters and getters which allow manipulation of data in the slots of a `DataMapper` object.

Usage

```
## S4 method for signature 'DataMapper'
rawdata(object)

## S4 replacement method for signature 'DataMapper,ANY'
rawdata(object) <- value
```

Arguments

`object`	A `DataMapper` object to get or set data from.
`value`	A `list`-like object to assign to the rawdata slot. Should be a `data.frame` or `data.table` with the current implementation.

Details

rawdata: Get the raw data slot from a `DataMapper` object. Returns a list-like containing one or more raw data inputs to the `DataMapper` object.

rawdata: Set the raw data slot from a `DataMapper` object. **value**: The `list`-like object to set for the rawdata slot. Note: this currently only supports `data.frame` or `data.table` objects.

Value

Accessors: See details

Setters: An update `DataMapper` object, returned invisibly.

See Also

Other DataMapper-accessors: `LongTableDataMapper-accessors`, `TREDataMapper-accessors`

`DataMapper-class`	*An S4 Class For Mapping from Raw Experimental Data to a Specific S4 Object*

Description

This object will be used as a way to abstract away data preprocessing.

Slots

- rawdata: A list-like object containing one or more pieces of raw data that will be processed and mapped to the slots of an `S4` object.
- metadata: A `List` of object level metadata.

`drop_fn_params`	*Drop parameters from a function and replace them with constants inside the function body.*

Description

Drop parameters from a function and replace them with constants inside the function body.

Usage

```
drop_fn_params(fn, args)
```

Arguments

`fn`	`function` A non-primitive function to remove parameters from (via `base::formals(fn)`).
`args`	`list` A list where names are the function arguments (parameters) to remove and the values are the appopriate value to replace the parameter with in the function body.

Value

`function` A new non-primitize function with the parameters named in `args` deleted and their values fixed with the values from `args` in the function body.

`drugSensitivitySig` *Compute the correlation between a molecular feature and treatment response*

Description

Compute the correlation between a molecular feature and treatment response

Usage

```
drugSensitivitySig(object, ...)
```

Arguments

`object`	An object inheriting form the `CoreGx::CoreSet` class
`...`	Allow definition of new arguments to this generic

Value

A 3D array of genes x drugs x metric

`endoaggregate` *Perform aggregation over an S4 object, but return an object of the same class.*

Description

Perform aggregation over an S4 object, but return an object of the same class.

Usage

```
endoaggregate(x, ...)
```

Arguments

`x`	An `S4` object to endomorphically aggregate over.
`...`	`pairlist` Allow definition of new parameters for implementations of this generic.

Value

An object with the same class as `x`.

Examples

```
print("Generics shouldn't need examples?")
```

endoaggregate,LongTable-method

Functional API for endomorphic aggregation over a LongTable *or inheriting class*

Description

Compute a group-by operation over a LongTable object or its inhering classes.

Usage

```
## S4 method for signature 'LongTable'
endoaggregate(
  x,
  ...,
  assay,
  target = assay,
  by,
  subset = TRUE,
  nthread = 1,
  progress = TRUE,
  BPPARAM = NULL,
  enlist = TRUE,
  moreArgs = list()
)
```

Arguments

x	LongTable or inheriting class to compute aggregation on.
...	call One or more aggregations to compute for each group by in x. If you name aggregation calls, that will be the column name of the value in the resulting data.table otherwise a default name will be parsed from the function name and its first argument, which is assumed to be the name of the column being aggregated over.
assay	character(1) The assay to aggregate over.
target	character(1) The assay to assign the results to. Defaults to assay.
by	character One or more valid column names in x to compute groups using.
subset	call An R call to evaluate before perfoming an aggregate. This allows you to aggregate over a subset of columns in an assay but have it be assigned to the parent object. Default is TRUE, which includes all rows. Passed through as the i argument in [.data.table.
nthread	numeric(1) Number of threads to use for split-apply-combine parallelization. Uses BiocParllel::bplapply if nthread > 1 or you pass in BPPARAM. Does not modify data.table threads, so be sure to use setDTthreads for reasonable nested parallelism. See details for performance considerations.

`progress`	`logical(1)` Display a progress bar for parallelized computations? Only works if `bpprogressbar<-` is defined for the current BiocParallel back-end.
`BPPARAM`	`BiocParallelParam` object. Use to customized the the parallization back-end of bplapply. Note, nthread over-rides any settings from BPPARAM as long as `bpworkers<-` is defined for that class.
`enlist`	`logical(1)` Default is `TRUE`. Set to `FALSE` to evaluate the first call in `...` within `data.table` groups. See details for more information.
`moreArgs`	`list()` A named list where each item is an argument one of the calls in `...` which is not a column in the table being aggregated. Use to further parameterize you calls. Please note that these are not added to your aggregate calls unless you specify the names in the call.

Details

Use of Non-Standard Evaluation:

Arguments in `...` are substituted and wrapped in a list, which is passed through to the j argument of `[.data.table` internally. The function currently tries to build informative column names for unnamed arguments in `...` by appending the name of each function call with the name of its first argument, which is assumed to be the column name being aggregated over. If an argument to `...` is named, that will be the column name of its value in the resulting `data.table`.

Enlisting:

The primary use case for `enlist=FALSE` is to allow computation of dependent aggregations, where the output from a previous aggregation is required in a subsequent one. For this case, wrap your call in `{` and assign intermediate results to variables, returning the final results as a list where each list item will become a column in the final table with the corresponding name. Name inference is disabled for this case, since it is assumed you will name the returned list items appropriately. A major advantage over multiple calls to `aggregate` is that the overhead of parallelization is paid only once even for complex multi-step computations like fitting a model, capturing its paramters, and making predictions using it. It also allows capturing arbitrarily complex calls which can be recomputed later using the `update,TreatmentResponseExperiment-method` A potential disadvantage is increased RAM usage per thread due to storing intermediate values in variables, as well as any memory allocation overhead associate therewith.

Value

Object with the same class as `x`, with the aggregation results assigned to `target`, using `strategy` if `target` is an existing assay in `x`.

See Also

`data.table::[.data.table`, `BiocParallel::bplapply`

`exampleDataMapper` *Example LongTableDataMapper*

Description

A dummy LongTableDataMapper object to be used in package examples.

Usage

```
data(exampleDataMapper)
```

Format

LongTableDataMapper object

`getIntern` *Retrieve the specified item from object internal metadata.*

Description

Internal slot for storing metadata relevant to the internal operation of an S4 object.

Usage

```
getIntern(object, x, ...)
```

Arguments

`object`	`S4` An object with an @.itern slot containing an environment.
`x`	`character` One or more symbol names to retrieve from the object@.intern environment.
`...`	Allow new parmeters to be defined for this generic.

Details

Warning: This method is intended for developer use and can be ignored by users.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

getIntern<- *Set the internal structural metadata for an S4 class*

Description

Set the internal structural metadata for an S4 class

Usage

```
getIntern(object, ...) <- value
```

Arguments

`object`	An R object to update internal structural metadata for.
`value`	An `immutable_list` object, being a class union between `list` and `immutable` S3 classes.

Value

Updates the object and returns invisibly.

Examples

```
print("Generics shouldn't need examples?")
```

getIntern<-,LongTable,immutable_list-method *Set the .intern slot of a LongTable*

Description

Set the .intern slot of a LongTable

Usage

```
## S4 replacement method for signature 'LongTable,immutable_list'
getIntern(object) <- value
```

Arguments

`object`	`LongTable`
`value`	An `immutable_list` object, being a class union between `list` and `immutable` S3 classes.

Value

Updates the object and returns invisibly.

`guessMapping` *Generic for Guessing the Mapping Between Some Raw Data and an S4 Object*

Description

Generic for Guessing the Mapping Between Some Raw Data and an S4 Object

Usage

```
guessMapping(object, ...)
```

Arguments

`object`	An `S4` object containing so raw data to guess data to object slot mappings for.
`...`	Allow new arguments to be defined for this generic.

Value

A `list` with mapping guesses as items.

Examples

```
"Generics shouldn't need examples!"
```

`guessMapping,LongTableDataMapper-method` *Guess which columns in raw experiment data map to which dimensions.*

Description

Checks for columns which are uniquely identified by a group of identifiers. This should be used to help identify the columns required to uniquely identify the rows, columns, assays and metadata of a `DataMapper` class object.

Usage

```
## S4 method for signature 'LongTableDataMapper'
guessMapping(object, groups, subset, data = FALSE)
```

Arguments

`object`	A `LongTableDataMapper` object.
`groups`	A `list` containing one or more vector of column names to group-by. The function uses these to determine 1:1 mappings between the combination of columns in each vector and unique values in the raw data columns.
`subset`	A `logical` vector indicating whether to to subset out mapped columns after each grouping. Must be a single `TRUE` or `FALSE` or have the same length as groups, indicating whether to subset out mapped columns after each grouping. This will prevent mapping a column to two different groups.
`data`	A `logical` vector indicating whether you would like the data for mapped columns to be returned instead of their column names. Defaults to `FALSE` for easy use assigning mapped columns to a `DataMapper` object.

Details

Any unmapped columns will be added to the end of the returned `list` in an item called unmapped.

The function automatically guesses metadata by checking if any columns have only a single value. This is returned as an additional item in the list.

Value

A `list`, where each item is named for the associated `groups` item the guess is for. The character vector in each item are columns which are uniquely identified by the identifiers from that group.

Examples

```
guessMapping(exampleDataMapper, groups=list(rows='treatmentid', cols='sampleid'),
subset=FALSE)
```

`gwc` *GWC Score*

Description

Calculate the gwc score between two vectors, using either a weighted spearman or pearson correlation

Usage

```
gwc(
  x1,
  p1,
  x2,
  p2,
  method.cor = c("pearson", "spearman"),
  nperm = 10000,
  truncate.p = 1e-16,
  ...
)
```

Arguments

x1	`numeric` vector of effect sizes (e.g., fold change or t statitsics) for the first experiment
p1	`numeric` vector of p-values for each corresponding effect size for the first experiment
x2	`numeric` effect size (e.g., fold change or t statitsics) for the second experiment
p2	`numeric` vector of p-values for each corresponding effect size for the second experiment
method.cor	`character` string identifying if a `pearson` or `spearman` correlation should be used
nperm	`numeric` how many permutations should be done to determine
truncate.p	`numeric` Truncation value for extremely low p-values
...	Other passed down to internal functions

Value

`numeric` a vector of two values, the correlation and associated p-value.

Examples

```
data(clevelandSmall_cSet)
x <- molecularProfiles(clevelandSmall_cSet,'rna')[,1]
y <- molecularProfiles(clevelandSmall_cSet,'rna')[,2]
x_p <- rep(0.05, times=length(x))
y_p <- rep(0.05, times=length(y))
names(x_p) <- names(x)
names(y_p) <- names(y)
gwc(x,x_p,y,y_p, nperm=100)
```

idCols *Generic to access the unique id columns in an S4 object used to*

Description

Generic to access the unique id columns in an S4 object used to

Usage

```
idCols(object, ...)
```

Arguments

`object`	An `S4` object to get id columns from.
`...`	Allow new arguments to this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`immutable` *Constructor for "immutable" S3-class property*

Description

This method should allow any S3 object in R to become immutable by intercepting `[<-`, `[[<-`, `$<-` and `c` during S3-method dispatch and returning an error.

Reverse with call to the `mutable` function.

Usage

```
immutable(object)

is.immutable(object)

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

show.immutable(x)
```

Arguments

`object, x`	Any R object which uses S3 method dispatch
`...`	Fallthrough arguments to `print.default`.

Details

The motivation for this class was to create pseudo-private slots in an R S4 object by preventing mutation of those slots outside of the accessors written for the class. It should behave as expected for R object which operate with 'copy-on-modify' semantics, including most base R functions and S3 objects.

An environment was not suitable for this case due to the 'copy-by-reference' semantics, such that normal R assignment, which users assume makes a copy of the object, actually references the same environment in both the original and copy of the object.

WARNING: This implementation is unable to intercept modifications to a `data.table` via the `set*` group of methods. This is because these methods are not S3 generics and therefore no mechanism exists for hooking into them to extend their functionality. In general, this helper class will only work for objects with an S3 interface.

Value

The `object` with "immutable" prepended to its class attribute.

`logical(1)` Does the object inherit from the "immutable" S3-class?

None, `invisible(NULL)`

See Also

`assignment-immutable`, `setOps-immutable`

Examples

```
immutable_list <- immutable(as.list(1:5))
class(immutable_list)
# errors during assignment operations
tryCatch({ immutable_list$new <- 1 }, error=print)

immutable_list <- immutable(as.list(1:5))
is.immutable(immutable_list)
```

`is.items` *Get the types of all items in a list*

Description

Get the types of all items in a list

Usage

```
is.items(list, ..., FUN = is)
```

Arguments

`list`	A `list` to get the types from
`...`	`pairlist` Additional arguments to FUN
`FUN`	`function` or `character` Either a function, or the name of a function which returns a single logical value. The default function uses `is`, specify the desired type in `...`. You can also use other type checking functions such as is.character, is.numeric, or is.data.frame.

Value

`logical` A vector indicating if the list item is the specified type.

Examples

```
list <- list(c(1,2,3), c('a','b','c'))
is.items(list, 'character')
```

`is_optim_compatible` *Check whether a function signature is amenable to optimization via* `stats::optim`*.*

Description

Functions compatible with `optim` have the parameter named `par` as their first formal argument where each value is a respective free parameter to be optimized.

Usage

```
is_optim_compatible(fn)
```

Arguments

`fn`	`function` A non-primitive function.

Value

`logical(1)` `TRUE` if the first value of `formalArg(fn)` is "par", otherwise `FALSE`.

`lapply,MultiAssayExperiment-method` *lapply lapply method for* `MultiAssayExperiment`

Description

lapply lapply method for `MultiAssayExperiment`

Usage

```
## S4 method for signature 'MultiAssayExperiment'
lapply(X, FUN, ...)
```

Arguments

`X`	A `MultiAssayExperiment` object.
`FUN`	A function to be applied to each `SummarizedExperiment` in a in `X`.
`...`	Fall through parameters to `FUN`

Value

A `MultiAssayExperiment` object, modifed such that `experiments(X) <- endoapply(experiments(X), FUN, ...)`.s

`list_OR_LongTable-class` *A class union to allow multiple types in a CoreSet slot*

Description

A class union to allow multiple types in a CoreSet slot

`LongTable` *LongTable constructor method*

Description

LongTable constructor method

Usage

```
LongTable(
  rowData,
  rowIDs,
  colData,
  colIDs,
  assays,
  assayIDs,
  metadata = list(),
  keep.rownames = FALSE
)
```

Arguments

`rowData`	`data.frame` A rectangular object coercible to a `data.table`.
`rowIDs`	`character` A vector of `rowData` column names needed to uniquely identify each row in a `LongTable`.
`colData`	`data.frame` A rectangular object coercible to a `data.table.`
`colIDs`	`chacter` A vector of `colData` column names needed to uniquely identify each column in a `LongTable`.
`assays`	`list` A list of rectangular objects, each coercible to a `data.table`. Must be named and item names must match the `assayIDs` list.
`assayIDs`	`list` A list of `character` vectors specifying the columns needed to uniquely identify each row in an `assay`. Names must match the `assays` list.
`metadata`	`list` A list of one or more metadata items associated with a LongTable experiment.
`keep.rownames`	`logical(1)` or `character(1)` Should rownames be retained when coercing to `data.table` inside the constructor. Default is FALSE. If TRUE, adds a 'rn' column to each rectangular object that gets coerced from `data.frame` to `data.table`. If a string, that becomes the name of the rownames column.

Value

A `LongTable` object containing the data for a treatment response experiment and configured according to the rowIDs and colIDs arguments.

Examples

```
"See vignette('The LongTable Class', package='CoreGx')"
```

`LongTable-accessors` *Accessing and modifying information in a* `LongTable`

Description

Documentation for the various setters and getters which allow manipulation of data in the slots of a `LongTable` object.

Value

Accessors: See details.

Setters: An updated `LongTable` object, returned invisibly.

Examples

```
data(merckLongTable)
```

`LongTable-class` *LongTable class definition*

Description

Define a private constructor method to be used to build a `LongTable` object.

This is used as an alternative to R attributes for storing structural metadata of an S4 objects.

Add or replace an assay in a LongTable by name. Currently this function only works when the assay has all columns in row and column data tables (i.e., when assays is retured withDimnames=TRUE).

Select an assay from within a LongTable object.

Usage

```
## S4 method for signature 'LongTable'
rowIDs(object, data = FALSE, key = FALSE)

## S4 method for signature 'LongTable'
rowMeta(object, data = FALSE, key = FALSE)

## S4 method for signature 'LongTable'
colIDs(object, data = FALSE, key = FALSE)

## S4 method for signature 'LongTable'
colMeta(object, data = FALSE, key = FALSE)

## S4 method for signature 'LongTable'
idCols(object)

## S4 method for signature 'LongTable'
assayIndex(x)

## S4 method for signature 'LongTable'
assayKeys(x, i)

## S4 method for signature 'LongTable'
assayCols(object, i)

## S4 method for signature 'LongTable,character'
getIntern(object, x)

## S4 method for signature 'LongTable,missing'
getIntern(object, x)

## S4 method for signature 'LongTable'
rowData(x, key = FALSE, use.names = FALSE, ...)
```

```
## S4 replacement method for signature 'LongTable'
rowData(x, ...) <- value

## S4 method for signature 'LongTable'
colData(x, key = FALSE, dimnames = FALSE, ...)

## S4 replacement method for signature 'LongTable,ANY'
colData(x, ...) <- value

## S4 method for signature 'LongTable'
assays(
  x,
  withDimnames = TRUE,
  metadata = withDimnames,
  key = !withDimnames,
  ...
)

## S4 replacement method for signature 'LongTable,list'
assays(x, withDimnames = TRUE, ...) <- value

## S4 method for signature 'LongTable,ANY'
assay(
  x,
  i,
  withDimnames = TRUE,
  summarize = withDimnames,
  metadata = !summarize,
  key = !(summarize || withDimnames),
  ...
)

## S4 replacement method for signature 'LongTable,ANY'
assay(x, i) <- value

## S4 method for signature 'LongTable'
assayNames(x)

## S4 method for signature 'LongTable,ANY,ANY'
x[[i]]

## S4 method for signature 'LongTable'
dim(x)

## S4 method for signature 'LongTable'
colnames(x)

## S4 method for signature 'LongTable'
```

```
rownames(x)

## S4 method for signature 'LongTable'
dimnames(x)
```

Arguments

object	LongTable
data	logical Should the colData for the metadata columns be returned instead of the column names? Default is FALSE.
key	logical Should the key columns also be returned? Defaults to !withDimnames. This is incompatible with summarize=TRUE, which will drop the key columns regardless of the value of this argument.
x	The LongTable object to retrieve the dimnames for.
i	character(1) name or integer index of the desired assay.
use.names	logical This parameter is just here to stop matching the positional argument to use.names from the rowData generic. It doesn't do anything at this time and can be ignored.
...	For developer use only! Pass raw=TRUE to return the slot for modification by reference.
value	A data.frame or data.table to update the assay data with. This must at minumum contain the row and column data identifier columns to allow correctly mapping the assay keys. We recommend modifying the results returned by assay(longTable, 'assayName', withDimnames=TRUE). For convenience, both the [[and $ LongTable accessors return an assay with the dimnames.
withDimnames	logical(1) Should the dimension names be returned joined to the assay. This retrieves both the row and column identifiers and returns them joined to the assay. For
metadata	logical(1) Should all of the metadata also be joined to the assay. This is useful when modifying assays as the resulting list has all the information needed to recreated the LongTable object. Defaults to withDimnames.
summarize	logical(1) If the assays is a summary where some of idCols(x) are not in assayKeys(x, i), then those missing columns are dropped. Defaults to FALSE. When metadata is TRUE, only metadata columns with 1:1 cardinality with the assay keys for i.
‘x‘	A LongTable or inheriting class.
‘i‘	An optional valid assay name or index in x.

Value

LongTable object containing the assay data from a treatment response experiment

A character vector of rowData column names if data is FALSE, otherwise a data.table with the data from the rowData id columns.

A character vector of rowData column names if data is FALSE, otherwise a data.table with the data from the rowData metadta columns.

A `character` vector of colData column names if data is FALSE, otherwise a `data.table` with the data from the colData id columns.

A `character` vector of colData column names if data is FALSE, otherwise a `data.table` with the data from the colData metadta columns.

`character` A character vector containing the unique rowIDs and colIDs in a LongTable object.

A `mutable` copy of the "assayIndex" for `x`

A `mutable` copy of the "assyKeys" for `x`

A `list` of `character` vectors containing the value column names for each assay if i is missing, otherwise a `character` vector of value column names for the selected assay.

`immutable` value of x if length(x) == 1 else named list of values for all symbols in x.

An `immutable` list.

A `data.table` containing rowID, row identifiers, and row metadata.

A copy of the `LongTable` object with the `rowData` slot updated.

A `data.table` containing row identifiers and metadata.

A copy of the `LongTable` object with the `colData` slot updated.

A `list` of `data.table` objects, one per assay in the object.

A copy of the `LongTable` with the assays modified.

`LongTable` With updated assays slot.

`character` Names of the assays contained in the `LongTable`.

`numeric` Vector of object dimensions.

`character` Vector of column names.

`character` Vector of row names.

`list` List with two character vectors, one for row and one for column names.

Methods (by generic)

- `rowMeta(LongTable)`: Get the names of the non-id columns from rowData.
- `colIDs(LongTable)`: Get the names of the columns in colData required to uniquely identify each row.
- `colMeta(LongTable)`: Get the names of the non-id columns in the colData `data.table`.
- `idCols(LongTable)`: Get the names of all id columns.
- `assayIndex(LongTable)`: Get the assayIndex item from the objects internal metadata.
- `assayKeys(LongTable)`: Get the assayKeys item from the objects internal metadata.
- `assayCols(LongTable)`: Get a list of column names for each assay in the object.
- `getIntern(object = LongTable, x = character)`: Access structural metadata present within a LongTable object. This is mostly for developmer use.
- `getIntern(object = LongTable, x = missing)`: Access all structural metadata present within a LongTable object. This is primarily for developmer use.
- `rowData(LongTable)`: Get the row level annotations for a `LongTable` object.

- `rowData(LongTable) <- value`: Update the row annotations for a `LongTable` object. Currently requires that all columns in rowIDs(longTable) be present in value.
- `colData(LongTable)`: Get the column level annotations for a LongTable object.
- `colData(x = LongTable) <- value`: Update the colData of a LongTable object. Currently requires that all of the colIDs(longTable) be in the value object.
- `assays(LongTable)`: Get a list containing all the assays in a `LongTable`.
- `assays(x = LongTable) <- value`: Update the assays in a LongTable object. The rowIDs and colIDs must be present in all assays to allow successfully remapping the keys. We recommend modifying the list returned by assays(longTable, withDimnames=TRUE) and the reassigning to the `LongTable`.
- `assay(x = LongTable, i = ANY)`: Retrieve an assay `data.table` object from the `assays` slot of a `LongTable` object.
- `assay(x = LongTable, i = ANY) <- value`:
- `assayNames(LongTable)`: Return the names of the assays contained in a `LongTable`
- `x[[i`: Get an assay from a LongTable object. This method returns the row and column annotations by default to make assignment and aggregate operations easiers.
- `dim(LongTable)`: Get the number of row annotations by the number of column annotations from a LongTable object. Please note that row x columns does not necessarily equal the number of rows in an assay, since it is not required for each assay to have every row or column present.
- `colnames(LongTable)`: Retrieve the pseudo-colnames of a LongTable object, these are constructed by pasting together the colIDs(longTable) and can be used in the subset method for regex based queries.
- `rownames(LongTable)`: Retrieve the pseudo-rownames of a LongTable object, these are constructed by pasting together the rowIDs(longTable) and can be used in the subset method for regex based queries.
- `dimnames(LongTable)`: Get the pseudo-dimnames for a LongTable object. See colnames and rownames for more information.

Slots

`rowData` See Slots section.

`colData` See Slots section.

`assays` See Slots section.

`metadata` See Slots section.

`.intern` See Slots section.

Slots

- *rowData*: A `data.table` containing the metadata associated with the row dimension of a `LongTable`.
- *colData*: A `data.table` containing the metadata associated with the column dimension of a `LongTable`.

- *assays*: A `list` of `data.tables`, one for each assay in a `LongTable`.
- *metadata*: An optional `list` of additional metadata for a `LongTable` which doesn't map to one of the dimensions.
- *.intern*: An `immutable list` that holds internal structural metadata about a LongTable object, such as which columns are required to key the object.

Examples

```
rowIDs(merckLongTable)

rowMeta(merckLongTable)

colIDs(merckLongTable)

colMeta(merckLongTable)

idCols(merckLongTable)

assayIndex(nci_TRE_small)

assayKeys(nci_TRE_small)
assayKeys(nci_TRE_small, "sensitivity")
assayKeys(nci_TRE_small, 1)

assayCols(merckLongTable)

getIntern(merckLongTable, 'rowIDs')
getIntern(merckLongTable, c('colIDs', 'colMeta'))

getIntern(merckLongTable)

rowData(merckLongTable)

rowData(merckLongTable) <- rowData(merckLongTable)

colData(merckLongTable)

# Get the keys as well, mostly for internal use
colData(merckLongTable, key=TRUE)

colData(merckLongTable) <- colData(merckLongTable)

assays(merckLongTable)

assays(merckLongTable) <- assays(merckLongTable, withDimnames=TRUE)

# Default annotations, just the key columns
assay(merckLongTable, 'sensitivity')
assay(merckLongTable, 1)

# With identifiers joined
assay(merckLongTable, 'sensitivity', withDimnames=TRUE)
```

```
# With identifiers and metadata
assay(merckLongTable, 'profiles', withDimnames=TRUE, metadata=TRUE)

assay(merckLongTable, 'sensitivity') <-
     assay(merckLongTable, 'sensitivity', withDimnames=TRUE)
assay(merckLongTable, 'sensitivity') <- merckLongTable$sensitivity

assayNames(merckLongTable)
names(merckLongTable)

merckLongTable[['sensitivity']]

dim(merckLongTable)

dim(merckLongTable)

head(colnames(merckLongTable))

head(rownames(merckLongTable))

lapply(dimnames(merckLongTable), head)
```

`LongTableDataMapper` *Constructor for the* `LongTableDataMapper` *class, which maps from one or more raw experimental data files to the slots of a* `LongTable` *object.*

Description

Constructor for the `LongTableDataMapper` class, which maps from one or more raw experimental data files to the slots of a `LongTable` object.

Usage

```
LongTableDataMapper(
  rawdata = data.frame(),
  rowDataMap = list(character(), character()),
  colDataMap = list(character(), character()),
  assayMap = list(list(character(), character())),
  metadataMap = list(character())
)
```

Arguments

`rawdata`	A `data.frame` of raw data from a treatment response experiment. This will be coerced to a `data.table` internally. We recommend using joins to aggregate your raw data if it is not present in a single file.

`rowDataMap`	A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each row, the second is additional columns which map to rows, but are not required to uniquely identify them. Rows should be drugs.
`colDataMap`	A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each column, the second is additional columns which map to rows, but are not required to uniquely identify them. Columns should be samples.
`assayMap`	A list-like where each item is a `list` with two `character` vectors defining an assay, the first containing the identifier columns in `rawdata` needed to uniquely identify each row an assay, and the second the `rawdata` columns to be mapped to that assay. The names of `assayMap` will be the names of the assays in the `LongTable` that is created when calling `metaConstruct` on this `DataMapper` object. If the character vectors have names, the value columns will be renamed accordingly.
`metadataMap`	A list-like where each item is a `character` vector of `rawdata` column names to assign to the `@metadata` of the `LongTable`, where the name of that assay is the name of the list item. If names are omitted, assays will be numbered by their index in the list.

Details

The `guessMapping` method can be used to test hypotheses about the cardinality of one or more sets of identifier columns. This is helpful to determine the id columns for `rowDataMap` and `colDataMap`, as well as identify columns mapping to `assays` or `metadata`.

To attach metadata not associated with `rawdata`, please use the `metadata` assignment method on your `LongTableDataMapper`. This metadata will be merged with any metadata from `metadataMap` and added to the `LongTable` which this object ultimately constructs.

Value

A `LongTable` object, with columns mapped to it's slots according to the various maps in the `LongTableDataMapper` object.

See Also

`guessMapping`

Examples

```
data(exampleDataMapper)
exampleDataMapper
```

LongTableDataMapper-accessors

Accessing and modifying data in a `LongTableDataMapper` *object.*

Description

Documentation for the various setters and getters which allow manipulation of data in the slots of a `LongTableDataMapper` object.

Usage

```
## S4 replacement method for signature 'LongTableDataMapper,list'
rawdata(object) <- value

## S4 method for signature 'LongTableDataMapper'
rowDataMap(object)

## S4 replacement method for signature 'LongTableDataMapper,list_OR_List'
rowDataMap(object) <- value

## S4 method for signature 'LongTableDataMapper'
colDataMap(object)

## S4 replacement method for signature 'LongTableDataMapper,list_OR_List'
colDataMap(object) <- value

## S4 method for signature 'LongTableDataMapper'
assayMap(object)

## S4 replacement method for signature 'LongTableDataMapper,list_OR_List'
assayMap(object) <- value

## S4 method for signature 'LongTableDataMapper'
metadataMap(object)

## S4 replacement method for signature 'LongTableDataMapper,list_OR_List'
metadataMap(object) <- value
```

Arguments

`object`	A `LongTableDataMapper` object to get or set data from.
`value`	See details.

Details

rawdata: Get the raw data slot from a `LongTableDataMapper` object. Returns a list-like containing one or more raw data inputs to the `LongTableDataMapper` object.

rawdata: Set the raw data slot from a `LongTableDataMapper` object. **value**: The `list`-like object to set for the rawdata slot. Note: this currently only supports `data.frame` or `data.table` objects.

rowDataMap: `list` of two `character` vectors, the first are the columns required to uniquely identify each row of a `LongTableDataMapper` and the second any additional row-level metadata. If the character vectors have names, the resulting columns are automatically renamed to the item name of the specified column.

rowDataMap<-: Update the `@rowDataMap` slot of a `LongTableDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` or `List` where the first item is the names of the identifier columns – columns needed to uniquely identify each row in rowData – and the second item is the metadata associated with those the identifier columns, but not required to uniquely identify rows in the object rowData.

colDataMap: `list` of two `character` vectors, the first are the columns required to uniquely identify each row of a `LongTableDataMapper` and the second any additional col-level metadata. If the character vectors have names, the resulting columns are automatically renamed to the item name of the specified column.

colDataMap<-: Update the `@colDataMap` slot of a `LongTableDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` or `List` where the first item is the names of the identifier columns – columns needed to uniquely identify each row in colData – and the second item is the metadata associated with those the identifier columns, but not required to uniquely identify rows in the object rowData.

assayMap: A `list` of character vectors. The name of each list item will be the assay in a `LongTableDataMapper` object that the columns in the `character` vector will be assigned to. Column renaming occurs automatically when the character vectors have names (from the value to the name).

assayMap<-: Updates the `@assayMap` slot of a `LongTableDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` of character vectors, where the name of each list item is the name of an assay and the values of each character vector specify the columns mapping to the assay in the `S4` object the `LongTableDataMapper` constructs.

metadataMap: A `list` of `character` vectors. Each item is an element of the constructed objects `@metadata` slot.

metadataMap<-: Updates `LongTableDataMapper` object in-place, then returns an `invisible(NULL)`. Arguments:

- value: A `list` of `character` vectors. The name of each list item is the name of the item in the `@metadata` slot of the `LongTableDataMapper` object created when `metaConstruct` is called on the `DataMapper`, and a character vector specifies the columns of `@rawdata` to assign to each item.

Value

Accessors: See details

Setters: An update `LongTableDataMapper` object, returned invisibly.

See Also

Other DataMapper-accessors: `DataMapper-accessors`, `TREDataMapper-accessors`

Examples

```
rowDataMap(exampleDataMapper)

rowDataMap(exampleDataMapper) <- list(c('treatmentid'), c())

colDataMap(exampleDataMapper)

colDataMap(exampleDataMapper) <- list(c('sampleid'), c())

assayMap(exampleDataMapper)

assayMap(exampleDataMapper) <- list(sensitivity=c(viability1='viability'))

metadataMap(exampleDataMapper)

metadataMap(exampleDataMapper) <- list(object_metadata=c('metadata'))
```

`LongTableDataMapper-class`

A Class for Mapping Between Raw Data and an `LongTable` *Object*

Description

A Class for Mapping Between Raw Data and an `LongTable` Object

Usage

```
## S4 method for signature 'LongTableDataMapper'
show(object)
```

Arguments

`object`	A `LongTableDataMapper` to display in the console.

Value

`invisible` Prints to console.

Functions

- `show(LongTableDataMapper)`: Show method for LongTableDataMapper. Determines how the object is displayed in the console.

Slots

`rawdata` See Slots section.

`rowDataMap` See Slots section.

`colDataMap` See Slots section.

`assayMap` See Slots section.

`metadataMap` See Slots section.

Slots

- rowDataMap: A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each row, the second is additional columns which map to rows, but are not required to uniquely identify them. Rows should be drugs.
- colDataMap: A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each column, the second is additional columns which map to rows, but are not required to uniquely identify them. Columns should be samples.
- assayMap A list-like where each item is a `list` with two elements specifying an assay, the first being the identifier columns in `rawdata` needed to uniquely identify each row an assay, and the second a list of `rawdata` columns to be mapped to that assay. The names of `assayMap` will be the names of the assays in the `LongTable` that is created when calling `metaConstruct` on this `DataMapper` object.
- metadataMap: A list-like where each item is a `character` vector of `rawdata` column names to assign to the `@metadata` of the `LongTable`, where the name of that assay is the name of the list item. If names are omitted, assays will be numbered by their index in the list.

- rawdata: A list-like object containing one or more pieces of raw data that will be processed and mapped to the slots of an `S4` object.
- metadata: A `List` of object level metadata.

Examples

```
show(exampleDataMapper)
```

`make_optim_function` *Takes a non-primitive R function and refactors it to be compatible with optimization via* `stats::optim`*.*

Description

Takes a non-primitive R function and refactors it to be compatible with optimization via `stats::optim`.

Usage

```
make_optim_function(fn, ...)
```

Arguments

`fn`	`function` A non-primitive function
`...`	Arguments to `fn` to fix for before building the function to be optimized. Useful for reducing the number of free parameters in an optimization if there are insufficient degrees of freedom.

See Also

`drop_fn_params`, `collect_fn_params`

`mcc` *Compute a Mathews Correlation Coefficient*

Description

The function computes a Matthews correlation coefficient for two factors provided to the function. It assumes each factor is a factor of class labels, and the enteries are paired in order of the vectors.

Usage

```
mcc(
  x,
  y,
  nperm = 1000,
  nthread = 1,
  alternative = c("two.sided", "less", "greater"),
  ...
)
```

Arguments

`x, y`	`factor` of the same length with the same number of levels
`nperm`	`numeric` number of permutations for significance estimation. If 0, no permutation testing is done
`nthread`	`numeric` can parallelize permutation texting using BiocParallels bplapply
`alternative`	indicates the alternative hypothesis and must be one of ‘"two.sided"’, ‘"greater"’ or ‘"less"’. You can specify just the initial letter. ‘"greater"’ corresponds to positive association, ‘"less"’ to negative association.
`...`	`list` Additional arguments

Details

Please note: we recommend you call set.seed() before using this function to ensure the reproducibility of your results. Write down the seed number or save it in a script if you intend to use the results in a publication.

Value

A list with the MCC as the $estimate, and p value as $p.value

Examples

```
x <- factor(c(1,2,1,2,3,1))
y <- factor(c(2,1,1,1,2,2))
mcc(x,y)
```

`merckLongTable` *Merck Drug Combination Data LongTable*

Description

This is a LongTable object created from some drug combination data provided to our lab by Merck.

Usage

```
data(merckLongTable)
```

Format

LongTable object

References

TODO:: Include a reference

`mergeAssays` *Merge assays with an `S4` object.*

Description

Merge assays with an `S4` object.

Usage

```
mergeAssays(object, ...)
```

Arguments

`object`	`S4` An S4 object a list-like slot containing assays for the object.
`...`	Allow new arguments to be defined for this generic.

Value

A modified version of `object`.

Examples

```
"This is a generic method!"
```

`mergeAssays,LongTable-method`

Endomorphically merge assays within a `LongTable` *or inheriting class*

Description

Endomorphically merge assays within a `LongTable` or inheriting class

Usage

```
## S4 method for signature 'LongTable'
mergeAssays(object, x, y, target = x, ..., metadata = FALSE)
```

Arguments

`object`	A `LongTable` or inheriting class.
`x`	`character(1)` A valid assay name in `object`.
`y`	`character(1)` A valid assay name in `object`.
`target`	`character(1)` Name of the assay to assign the result to. Can be a new or existing assay. Defaults to `x`.
`...`	Fallthrough arguments to merge.data.table to specify the join type. Use this to specify which columns to merge on. If excluded, defaults to by=assayKeys(objecty, y).
`metadata`	`logical` A logical vector indicating whether to attach metadata to either assay before the merge occurs. If only one value is passed that value is used for both assays. Defaults to `FALSE`.

Value

A copy of `object` with assays `x` and `y` merged and assigned to `target`.

Author(s)

Christopher Eeles

See Also

`merge.data.table`

`metaConstruct` *Generic for preprocessing complex data before being used in the constructor of an* `S4` *container object.*

Description

This method is intended to abstract away complex constructor arguments and data preprocessing steps needed to transform raw data, such as that produced in a treatment-response or next-gen sequencing experiment, and automate building of the appropriate `S4` container object. This is is intended to allow mapping between different experimental designs, in the form of an `S4` configuration object, and various S4 class containers in the Bioconductor community and beyond.

Usage

```
metaConstruct(mapper, ...)

## S4 method for signature 'LongTableDataMapper'
metaConstruct(mapper)

## S4 method for signature 'TREDataMapper'
metaConstruct(mapper)
```

Arguments

`mapper`	An `TREDataMapper` object abstracting arguments to an the `TreatmentResponseExperiment` constructor.
`...`	Allow new arguments to be defined for this generic.

Value

An `S4` object for which the class corresponds to the type of the build configuration object passed to this method.

A `LongTable` object, as specified in the mapper.

A `TreatmentResponseExperiment` object, as specified in the mapper.

Examples

```
data(exampleDataMapper)
rowDataMap(exampleDataMapper) <- list(c('treatmentid'), c())
colDataMap(exampleDataMapper) <- list(c('sampleid'), c())
assayMap(exampleDataMapper) <- list(sensitivity=list(c("treatmentid", "sampleid"), c('viability')))
metadataMap(exampleDataMapper) <- list(experiment_metadata=c('metadata'))
longTable <- metaConstruct(exampleDataMapper)
longTable

data(exampleDataMapper)
exampleDataMapper <- as(exampleDataMapper, "TREDataMapper")
```

```
rowDataMap(exampleDataMapper) <- list(c('treatmentid'), c())
colDataMap(exampleDataMapper) <- list(c('sampleid'), c())
assayMap(exampleDataMapper) <- list(sensitivity=list(c("treatmentid", "sampleid"), c('viability')))
metadataMap(exampleDataMapper) <- list(experiment_metadata=c('metadata'))
tre <- metaConstruct(exampleDataMapper)
tre
```

metadata,LongTable-method

Getter method for the metadata slot of a `LongTable` *object*

Description

Getter method for the metadata slot of a `LongTable` object

Usage

```
## S4 method for signature 'LongTable'
metadata(x)
```

Arguments

x	The `LongTable` object from which to retrieve the metadata list.

Value

`list` The contents of the `metadata` slot of the `LongTable` object.

metadata<-,LongTable-method

Setter method for the metadata slot of a `LongTable` *object*

Description

Setter method for the metadata slot of a `LongTable` object

Usage

```
## S4 replacement method for signature 'LongTable'
metadata(x) <- value
```

Arguments

x	`LongTable` The LongTable to update
value	`list` A list of new metadata associated with a `LongTable` object.

Value

`LongTable` A copy of the `LongTable` object with the `value` in the metadata slot.

`mutable` *Remove the "immutable" S3-class from an R object, allowing it to be modified normally again.*

Description

Remove the "immutable" S3-class from an R object, allowing it to be modified normally again.

Usage

```
mutable(object)
```

Arguments

`object`	An R object inheriting from the "immutable" class.

Value

The `object` with the "immutable" class stripped from it.

Examples

```
immut_list <- immutable(list())
mutable(immut_list)
```

`nci_TRE_small` *NCI-ALMANAC Drug Combination Data TreatmentResponseExperiment Subset*

Description

This is a `TreatmentResponseExperiment` object containing a subset of NCI-ALMANAC drug combination screening data, with 2347 unique treatment combinations on 10 cancer cell lines selected.

Usage

```
data(nci_TRE_small)
```

Format

TreatmentResponseExperiment object

References

Susan L. Holbeck, Richard Camalier, James A. Crowell, Jeevan Prasaad Govindharajulu, Melinda Hollingshead, Lawrence W. Anderson, Eric Polley, Larry Rubinstein, Apurva Srivastava, Deborah Wilsker, Jerry M. Collins, James H. Doroshow; The National Cancer Institute ALMANAC: A Comprehensive Screening Resource for the Detection of Anticancer Drug Pairs with Enhanced Therapeutic Activity. Cancer Res 1 July 2017; 77 (13): 3564–3576. https://doi.org/10.1158/0008-5472.CAN-17-0489

`optimizeCoreGx` *A helper method to find the best multithreading configuration for your computer*

Description

A helper method to find the best multithreading configuration for your computer

Usage

```
optimizeCoreGx(sample_data, set = FALSE, report = !set)
```

Arguments

`sample_data`	`TreatmentResponseExperiment`
`set`	`logical(1)` Should the function modify your R environment with the predicted optimal settings? This changes the global state of your R session!
`report`	`logical(1)` Should a `data.frame` of results be returned by number of threads and operation be returned? Defaults to `!set`.

Value

If `set=TRUE`, modifies `data.table` threads via `setDTthreads()`, otherwise displays a message indicating the optimal number of threads. If `report=TRUE`, also returns a `data.frame` of the benchmark results.

Examples

```
data(merckLongTable)
optimizeCoreGx(merckLongTable)
```

`printSlot` *Helper function to print slot information*

Description

Helper function to print slot information

Usage

```
printSlot(slotName, slotData)
```

Arguments

`slotName`	`character` The name of the slot to print.
`slotData`	`data.table` The data to print.

`reindex` *Generic method for resetting indexing in an S4 object*

Description

This method allows integer indexes used to maintain referential integrity internal to an S4 object to be reset. This is useful particularly after subsetting an object, as certain indexes may no longer be present in the object data. Reindexing removes gaps integer indexes and ensures that the smallest contiguous integer values are used in an objects indexes.

Usage

```
reindex(object, ...)
```

Arguments

`object`	`S4` An object to redo indexing for
`...`	`pairlist` Allow definition of new parameters to this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`reindex,LongTable-method`
Redo indexing for a LongTable object to remove any gaps in integer indexes

---

Description

After subsetting a LongTable, it is possible that values of rowKey or colKey could no longer be present in the object. As a result there the indexes will no longer be contiguous integers. This method will calcualte a new set of rowKey and colKey values such that integer indexes are the smallest set of contiguous integers possible for the data.

Usage

```
## S4 method for signature 'LongTable'
reindex(object)
```

Arguments

`object`	The `LongTable` object to recalcualte indexes (rowKey and colKey values) for.

Value

A copy of the `LongTable` with all keys as the smallest set of contiguous integers possible given the current data.

`rowData,LongTableDataMapper-method`
Convenience method to subset the `rowData` *out of the* `rawdata` *slot using the assigned* `rowDataMap` *metadata.*

Description

Convenience method to subset the `rowData` out of the `rawdata` slot using the assigned `rowDataMap` metadata.

Usage

```
## S4 method for signature 'LongTableDataMapper'
rowData(x, key = TRUE)
```

Arguments

`x`	`LongTableDataMapper` object with valid data in the `rawdata` and `colDataMap` slots.
`key`	`logical(1)` Should the table be keyed according to the `id_columns` of the `rowDataMap` slot? This will sort the table in memory. Default is TRUE.

Value

`data.table` The `rowData` as specified in the `rowDataMap` slot.

`rowData,TREDataMapper-method`

Convenience method to subset the `rowData` *out of the* `rawdata` *slot using the assigned* `rowDataMap` *metadata.*

Description

Convenience method to subset the `rowData` out of the `rawdata` slot using the assigned `rowDataMap` metadata.

Usage

```
## S4 method for signature 'TREDataMapper'
rowData(x, key = TRUE)
```

Arguments

`x`	`TREDataMapper` object with valid data in the `rawdata` and `colDataMap` slots.
`key`	`logical(1)` Should the table be keyed according to the `id_columns` of the `rowDataMap` slot? This will sort the table in memory. Default is TRUE.

Value

`data.table` The `rowData` as specified in the `rowDataMap` slot.

`rowIDs`

Generic to access the row identifiers from

Description

Generic to access the row identifiers from

Usage

```
rowIDs(object, ...)
```

Arguments

`object`	`S4` An object to get row id columns from.
`...`	Allow new arguments to this generic.

Value

Depends on the implemented method.

Examples

```
print("Generics shouldn't need examples?")
```

`rowMeta` *Generic to access the row identifiers from*

Description

Generic to access the row identifiers from

Usage

```
rowMeta(object, ...)
```

Arguments

`object`	`S4` An object to get row metadata columns from.
`...`	Allow new arguments to this generic.

Value

Depends on the implemented method.

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityInfo` *Generic function to get the annotations for a treatment response experiment from an S4 class*

Description

Generic function to get the annotations for a treatment response experiment from an S4 class

Usage

```
sensitivityInfo(object, ...)
```

Arguments

`object`	An S4 object to get treatment response experiment annotations from.
`...`	Allow new arguments to be defined for this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityInfo<-` *sensitivityInfo<- Generic Method*

Description

Generic function to get the annotations for a treatment response experiment from an S4 class.

Usage

```
sensitivityInfo(object, ...) <- value
```

Arguments

`object`	An S4 object to set treatment response experiment annotations for.
`...`	Allow new arguments to be defined for this generic.
`value`	The new treatment response experiment annotations.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityMeasures` *sensitivityMeasures Generic*

Description

Get the names of the sensitivity summary metrics available in an S4 object.

Usage

```
sensitivityMeasures(object, ...)
```

Arguments

`object`	An S4 object to retrieve the names of sensitivty summary measurements for.
`...`	Fallthrough arguements for defining new methods

Value

Depends on the implemented method

Examples

```
sensitivityMeasures(clevelandSmall_cSet)
```

`sensitivityMeasures<-` *sensitivityMeasures<- Generic*

Description

Set the names of the sensitivity summary metrics available in an S4 object.

Usage

```
sensitivityMeasures(object, ...) <- value
```

Arguments

`object`	An S4 object to update.
`...`	Allow new methods to be defined for this generic.
`value`	A set of names for sensitivity measures to use to update the object with.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityProfiles` *sensitivityProfiles Generic*

Description

A generic for sensitivityProfiles getter method

Usage

```
sensitivityProfiles(object, ...)
```

Arguments

`object`	The `S4` object to retrieve sensitivity profile summaries from.
`...`	`pairlist` Allow defining new arguments for this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityProfiles<-` *sensitivityProfiles<- Generic*

Description

A generic for the sensitivityProfiles replacement method

Usage

```
sensitivityProfiles(object, ...) <- value
```

Arguments

`object`	An `S4` object to update the sensitivity profile summaries for.
`...`	Fallthrough arguments for defining new methods
`value`	An object with the new sensitivity profiles. If a matrix object is passed in, converted to data.frame before assignment

Value

Updated `CoreSet`

`sensitivityRaw` *sensitivityRaw Generic Method*

Description

Generic function to get the raw data array for a treatment response experiment from an S4 class.

Usage

```
sensitivityRaw(object, ...)
```

Arguments

`object`	An S4 object to extract the raw sensitivity experiment data from.
`...`	`pairlist` Allow new parameters to be defined for this generic.

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`sensitivityRaw<-` *sensitivityRaw<- Generic*

Description

Generic function to set the raw data array for a treatment response experiment in an S4 class.

Usage

```
sensitivityRaw(object, ...) <- value
```

Arguments

`object`	An S4 object to extract the raw sensitivity data from.
`...`	`pairlist` Allow new parameters to be defined for this generic.
`value`	An object containing dose and viability metrics to update the object with.

Value

Depends on the implemented method

sensitivitySlotToLongTable *sensitivitySlotToLongTable Generic*

Description

Convert the sensitivity slot in an object inheriting from a CoreSet from a list to a LongTable.

Usage

```
sensitivitySlotToLongTable(object, ...)
```

Arguments

object	CoreSet Object inheriting from CoreSet.
...	Allow new arguments to be defined on this generic.

Value

A LongTable object containing the data in the sensitivity slot.

Examples

```
print("Generics shouldn't need examples?")
```

setOps-immutable *Subset an immutable object, returning another immutable object.*

Description

Subset an immutable object, returning another immutable object.

Usage

```
subset.immutable(x, ...)

## S3 method for class 'immutable'
x[...]

## S3 method for class 'immutable'
x[[...]]

## S3 method for class 'immutable'
x$...
```

Arguments

`x`	An R object inheriting from the "immutable" S3-class.
`...`	Catch any additional parameters. Lets objects with arbitrary dimensions be made immutable.

Value

An immutable subset of `x`.

Examples

```
immut_mat <- immutable(matrix(1:100, 10, 10))
immut_mat[1:5, 1:5]
```

`show,CoreSet-method` *Show a CoreSet*

Description

Show a CoreSet

Usage

```
## S4 method for signature 'CoreSet'
show(object)
```

Arguments

`object`	`CoreSet` object to show via `cat`.

Value

Prints the CoreSet object to the output stream, and returns invisible NULL.

See Also

`cat`

Examples

```
show(clevelandSmall_cSet)
```

show,LongTable-method *Show method for the LongTable class*

Description

Show method for the LongTable class

Usage

```
## S4 method for signature 'LongTable'
show(object)
```

Arguments

object	A LongTable object to print the results for.

Value

invisible Prints to console.

Examples

```
show(merckLongTable)
```

showSigAnnot *Get the annotations for a Signature class object, as returned by drugSensitivitysig or radSensitivtySig functions available in PharmacoGx and RadioGx, respectively.*

Description

Get the annotations for a Signature class object, as returned by drugSensitivitysig or radSensitivtySig functions available in PharmacoGx and RadioGx, respectively.

Usage

```
showSigAnnot(object, ...)
```

Arguments

object	A Signature class object
...	Allow definition of new arguments to this generic

Value

NULL Prints the signature annotations to console

Examples

```
print("Generics shouldn't need examples?")
```

`subset,LongTable-method`
Subset method for a LongTable object.

Description

Allows use of the colData and rowData `data.table` objects to query based on rowID and colID, which is then used to subset all assay `data.tables` stored in the `assays` slot. This function is endomorphic, it always returns a LongTable object.

Usage

```
## S4 method for signature 'LongTable'
subset(x, i, j, assays = assayNames(x), reindex = TRUE)
```

Arguments

`x`	`LongTable` The object to subset.
`i`	`character`, `numeric`, `logical` or `call` Character: pass in a character vector of rownames for the `LongTable` object or a valid regex query which will be evaluated against the rownames. Numeric or Logical: vector of indices or a logical vector to subset the rows of a `LongTable`. Call: Accepts valid query statements to the `data.table` i parameter, this can be used to make complex queries using the `data.table` API for the `rowData` data.table.
`j`	`character`, `numeric`, `logical` or `call` Character: pass in a character vector of colnames for the `LongTable` object or a valid regex query which will be evaluated against the colnames. Numeric or Logical: vector of indices or a logical vector to subset the columns of a `LongTable`. Call: Accepts valid query statements to the `data.table` i parameter, this can be used to make complex queries using the `data.table` API for the `colData` data.table.
`assays`	`character`, `numeric` or `logical` Optional list of assay names to subset. Can be used to subset the assays list further, returning only the selected items in the new LongTable.
`reindex`	`logical(1)` Should index values be reset such that they are the smallest possible set of consecutive integers. Modifies the "rowKey", "colKey", and all assayKey columns. Initial benchmarks indicate `reindex=FALSE` saves ~20% of both execution time and memory allocation. The cost of reindexing decreases the smaller your subset gets.

Value

`LongTable` A new `LongTable` object subset based on the specified parameters.

Examples

```
# Character
subset(merckLongTable, 'ABT-888', 'CAOV3')
# Numeric
subset(merckLongTable, 1, c(1, 2))
# Logical
subset(merckLongTable, , colData(merckLongTable)$sampleid == 'A2058')
# Call
subset(merckLongTable, drug1id == 'Dasatinib' & drug2id != '5-FU',
    sampleid == 'A2058')
```

`subsetTo` *Subset a CoreSet object based on various parameters, such as cell lines, molecular features*

Description

Subset a CoreSet object based on various parameters, such as cell lines, molecular features

Usage

```
subsetTo(object, ...)
```

Arguments

`object`	An object inheriting from the `CoreGx::CoreSet` class
`...`	Allow definition of new arguments to this generic

Value

A subsetted version of the original `object`

Examples

```
"Generics shouldn't need examples!"
```

`summarizeMolecularProfiles`
Summarize molecular profile data such that there is a single entry for each sample line/treatment combination

---

Description

Summarize molecular profile data such that there is a single entry for each sample line/treatment combination

Usage

```
summarizeMolecularProfiles(object, ...)
```

Arguments

`object`	An S4 object to summarize the molecular profiles for.
`...`	Allow definition of new arguments to this generic

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

`summarizeSensitivityProfiles`
Summarize across replicates for a sensitivity dose-response experiment

---

Description

Summarize across replicates for a sensitivity dose-response experiment

Usage

```
summarizeSensitivityProfiles(object, ...)
```

Arguments

`object`	An S4 object to summarize sensitivity profiles for.
`...`	Allow definition of new arguments to this generic

Value

Depends on the implemented method

Examples

```
print("Generics shouldn't need examples?")
```

TreatmentResponseExperiment *TreatmentResponseExperiment constructor method*

Description

Builds a `TreatmentResponseExperiment` object from rectangular objects. The `rowData` argument should contain row level metadata, while the `colData` argument should contain column level metadata, for the experimental assays in the `assays` list. The `rowIDs` and `colIDs` lists are used to configure the internal keys mapping rows or columns to rows in the assays. Each list should contain at minimum one character vector, specifying which columns in `rowData` or `colData` are required to uniquely identify each row. An optional second character vector can be included, specifying any metadata columns for either dimension. These should contain information about each row but NOT be required to uniquely identify a row in the `colData` or `rowData` objects. Additional metadata can be attached to a `TreatmentResponseExperiment` by passing a list to the metadata argument.

Usage

```
TreatmentResponseExperiment(
  rowData,
  rowIDs,
  colData,
  colIDs,
  assays,
  assayIDs,
  metadata = list(),
  keep.rownames = FALSE
)
```

Arguments

`rowData`	`data.table`, `data.frame`, `matrix` A table like object coercible to a `data.table` containing the a unique `rowID` column which is used to key assays, as well as additional row metadata to subset on.
`rowIDs`	`character`, `integer` A vector specifying the names or integer indexes of the row data identifier columns. These columns will be pasted together to make up the rownames of the `TreatmentResponseExperiment` object.

`colData`	`data.table`, `data.frame`, `matrix` A table like object coercible to a `data.table` containing the a unique `colID` column which is used to key assays, as well as additional column metadata to subset on.
`colIDs`	`character`, `integer` A vector specifying the names or integer indexes of the column data identifier columns. These columns will be pasted together to make up the colnames of the `TreatmentResponseExperiment` object.
`assays`	A `list` containing one or more objects coercible to a `data.table`, and keyed by rowIDs and colIDs corresponding to the rowID and colID columns in colData and rowData.
`assayIDs`	`list` A list of `character` vectors specifying the columns needed to uniquely identify each row in an `assay`. Names must match the `assays` list.
`metadata`	A `list` of metadata associated with the `TreatmentResponseExperiment` object being constructed
`keep.rownames`	`logical`, `character` Logical: whether rownames should be added as a column if coercing to a `data.table`, default is FALSE. If TRUE, rownames are added to the column 'rn'. Character: specify a custom column name to store the rownames in.

Details

For now this class is simply a wrapper around a `LongTable` class. In the future we plan to refactor CoreGx such that the `LongTable` class is in a separate pacakge. We can then specialize the implementation of `TreatmentResponseExperiment` to better capture the biomedical nature of this object.

Value

A `TreatmentResponseExperiment` object containing the data for a treatment response experiment configured according to the rowIDs and colIDs arguments.

`TreatmentResponseExperiment-class`
TreatmentResponseExperiment class definition

Description

Define a private constructor method to be used to build a `TreatmentResponseExperiment` object.

Value

`TreatmentResponseExperiment` object containing the assay data from a treatment response experiment

Slots

`rowData` See Slots section.

`colData` See Slots section.

`assays` See Slots section.

`metadata` See Slots section.

`.intern` See Slots section.

Slots

- *rowData*: A `data.table` containing the metadata associated with the row dimension of a `TreatmentResponseExperiment`.
- *colData*: A `data.table` containing the metadata associated with the column dimension of a `TreatmentResponseExperiment`.
- *assays*: A `list` of `data.tables`, one for each assay in a `TreatmentResponseExperiment`.
- *metadata*: An optional `list` of additional metadata for a `TreatmentResponseExperiment` which doesn't map to one of the dimensions.
- *.intern*: An `environment` that holds internal structural metadata about a `TreatmentResponseExperiment` object, such as which columns are required to key the object. An environment has been used to allow locking items, which can prevent accidental modification of a property required for the class to work.

`TREDataMapper` *Constructor for the* `TREDataMapper` *class, which maps from one or more raw experimental data files to the slots of a* `LongTable` *object.*

Description

Constructor for the `TREDataMapper` class, which maps from one or more raw experimental data files to the slots of a `LongTable` object.

Usage

```
TREDataMapper(
  rawdata = data.frame(),
  rowDataMap = list(character(), character()),
  colDataMap = list(character(), character()),
  assayMap = list(list(character(), character())),
  metadataMap = list(character())
)
```

Arguments

`rawdata`	A `data.frame` of raw data from a treatment response experiment. This will be coerced to a `data.table` internally. We recommend using joins to aggregate your raw data if it is not present in a single file.
`rowDataMap`	A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each row, the second is additional columns which map to rows, but are not required to uniquely identify them. Rows should be treatments.
`colDataMap`	A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each column, the second is additional columns which map to rows, but are not required to uniquely identify them. Columns should be samples.
`assayMap`	A list-like where each item is a `list` with two `character` vectors defining an assay, the first containing the identifier columns in `rawdata` needed to uniquely identify each row an assay, and the second the `rawdata` columns to be mapped to that assay. The names of `assayMap` will be the names of the assays in the `TreatmentResponseExperiment` that is created when calling `metaConstruct` on this `DataMapper` object. If the character vectors have names, the value columns will be renamed accordingly.
`metadataMap`	A list-like where each item is a `character` vector of `rawdata` column names to assign to the `@metadata` of the `LongTable`, where the name of that assay is the name of the list item. If names are omitted, assays will be numbered by their index in the list.

Details

The `guessMapping` method can be used to test hypotheses about the cardinality of one or more sets of identifier columns. This is helpful to determine the id columns for `rowDataMap` and `colDataMap`, as well as identify columns mapping to `assays` or `metadata`.

To attach metadata not associated with `rawdata`, please use the `metadata` assignment method on your `TREDataMapper`. This metadata will be merge with any metadata from `metadataMap` and added to the `LongTable` which this object ultimately constructs.

Value

A `TREDataMapper` object, with columns mapped to it's slots according to the various maps in the `LongTableDataMapper` object.

See Also

`guessMapping`

TREDataMapper-accessors

Accessing and modifying data in a `TREDataMapper` *object.*

Description

Documentation for the various setters and getters which allow manipulation of data in the slots of a `TREDataMapper` object.

Usage

```
## S4 replacement method for signature 'TREDataMapper,list'
rawdata(object) <- value

## S4 method for signature 'TREDataMapper'
rowDataMap(object)

## S4 replacement method for signature 'TREDataMapper,list_OR_List'
rowDataMap(object) <- value

## S4 method for signature 'TREDataMapper'
colDataMap(object)

## S4 replacement method for signature 'TREDataMapper,list_OR_List'
colDataMap(object) <- value

## S4 method for signature 'TREDataMapper'
assayMap(object)

## S4 replacement method for signature 'TREDataMapper,list_OR_List'
assayMap(object) <- value

## S4 method for signature 'TREDataMapper'
metadataMap(object)

## S4 replacement method for signature 'TREDataMapper,list_OR_List'
metadataMap(object) <- value
```

Arguments

`object`	A `TREDataMapper` object to get or set data from.
`value`	See details.

Details

rawdata: Get the raw data slot from a `TREDataMapper` object. Returns a list-like containing one or more raw data inputs to the `TREDataMapper` object.

rawdata: Set the raw data slot from a `TREDataMapper` object. **value**: The `list`-like object to set for the rawdata slot. Note: this currently only supports `data.frame` or `data.table` objects.

rowDataMap: `list` of two `character` vectors, the first are the columns required to uniquely identify each row of a `TREDataMapper` and the second any additional row-level metadata. If the character vectors have names, the resulting columns are automatically renamed to the item name of the specified column.

rowDataMap<-: Update the `@rowDataMap` slot of a `TREDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` or `List` where the first item is the names of the identifier columns – columns needed to uniquely identify each row in rowData – and the second item is the metadata associated with those the identifier columns, but not required to uniquely identify rows in the object rowData.

colDataMap: `list` of two `character` vectors, the first are the columns required to uniquely identify each row of a `TREDataMapper` and the second any additional col-level metadata. If the character vectors have names, the resulting columns are automatically renamed to the item name of the specified column.

colDataMap<-: Update the `@colDataMap` slot of a `TREDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` or `List` where the first item is the names of the identifier columns – columns needed to uniquely identify each row in colData – and the second item is the metadata associated with those the identifier columns, but not required to uniquely identify rows in the object rowData.

assayMap: A `list` of character vectors. The name of each list item will be the assay in a `LongTableDataMapper` object that the columns in the `character` vector will be assigned to. Column renaming occurs automatically when the character vectors have names (from the value to the name).

assayMap<-: Updates the `@assayMap` slot of a `TREDataMapper` object, returning an invisible NULL. Arguments:

- value: A `list` of character vectors, where the name of each list item is the name of an assay and the values of each character vector specify the columns mapping to the assay in the `S4` object the `TREDataMapper` constructs.

metadataMap: A `list` of `character` vectors. Each item is an element of the constructed objects `@metadata` slot.

metadataMap<-: Updates `TREDataMapper` object in-place, then returns an `invisible(NULL)`. Arguments:

- value: A `list` of `character` vectors. The name of each list item is the name of the item in the `@metadata` slot of the `TREDataMapper` object created when `metaConstruct` is called on the `DataMapper`, and a character vector specifies the columns of `@rawdata` to assign to each item.

Value

Accessors: See details

Setters: An update `TREDataMapper` object, returned invisibly.

See Also

Other DataMapper-accessors: DataMapper-accessors, LongTableDataMapper-accessors

Examples

```
rowDataMap(exampleDataMapper)

rowDataMap(exampleDataMapper) <- list(c('treatmentid'), c())

colDataMap(exampleDataMapper)

colDataMap(exampleDataMapper) <- list(c('sampleid'), c())

assayMap(exampleDataMapper)

assayMap(exampleDataMapper) <- list(sensitivity=c(viability1='viability'))

metadataMap(exampleDataMapper)

metadataMap(exampleDataMapper) <- list(object_metadata=c('metadata'))
```

`TREDataMapper-class` *A Class for Mapping Between Raw Data and an* `TreatmentResponseExperiment` *Object*

Description

A Class for Mapping Between Raw Data and an `TreatmentResponseExperiment` Object

Slots

`rawdata` See Slots section.

`rowDataMap` See Slots section.

`colDataMap` See Slots section.

`assayMap` See Slots section.

`metadataMap` See Slots section.

Slots

- rowDataMap: A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each row, the second is additional columns which map to rows, but are not required to uniquely identify them. Rows should be drugs.
- colDataMap: A list-like object containing two `character` vectors. The first is column names in `rawdata` needed to uniquely identify each column, the second is additional columns which map to rows, but are not required to uniquely identify them. Columns should be samples.

- assayMap A list-like where each item is a `list` with two elements specifying an assay, the first being the identifier columns in `rawdata` needed to uniquely identify each row an assay, and the second a list of `rawdata` columns to be mapped to that assay. The names of `assayMap` will be the names of the assays in the `LongTable` that is created when calling `metaConstruct` on this `DataMapper` object.
- metadataMap: A list-like where each item is a `character` vector of `rawdata` column names to assign to the `@metadata` of the `LongTable`, where the name of that assay is the name of the list item. If names are omitted, assays will be numbered by their index in the list.
- rawdata: A list-like object containing one or more pieces of raw data that will be processed and mapped to the slots of an `S4` object.
- metadata: A `List` of object level metadata.

`updateObject,CoreSet-method`
Update the `CoreSet` *class after changes in it struture or API*

Description

Update the `CoreSet` class after changes in it struture or API

Usage

```
## S4 method for signature 'CoreSet'
updateObject(object, verify = FALSE, verbose = FALSE)
```

Arguments

`object`	A `CoreSet` object to update the class structure for.
`verify`	A `logical(1)` indicating is `validObject` should be called after updating the object. Defaults to `TRUE`, only set `FALSE` for debugging.
`verbose`	TRUE or FALSE, indicating whether information about the update should be reported

Value

`CoreSet` with update class structure.

updateObject,LongTable-method

Update the `LongTable` *class after changes in it struture or API*

Description

Update the `LongTable` class after changes in it struture or API

Usage

```
## S4 method for signature 'LongTable'
updateObject(object, verify = FALSE, verbose = FALSE)
```

Arguments

`object`	A `LongTable` object to update the class structure for.
`verify`	A `logical(1)` indicating is `validObject` should be called after updating the object. Defaults to `TRUE`, only set `FALSE` for debugging.
`verbose`	TRUE or FALSE, indicating whether information about the update should be reported

Value

`LongTable` with update class structure.

updateSampleId

Update the sample ids in a cSet object

Description

Update the sample ids in a cSet object

Usage

```
updateSampleId(object, new.ids = vector("character"))
```

Arguments

`object`	The object for which the sample ids will be updated
`new.ids`	The new ids to assign to the object

Value

`CoreSet` The modified CoreSet object

Examples

```
updateSampleId(clevelandSmall_cSet, sampleNames(clevelandSmall_cSet))
```

`updateTreatmentId` *Update the treatment ids in a cSet object*

Description

Update the treatment ids in a cSet object

Usage

```
updateTreatmentId(object, new.ids = vector("character"))
```

Arguments

`object`	The object for which the treatment ids will be updated
`new.ids`	The new ids to assign to the object

Value

`CoreSet` The modified CoreSet object

Examples

```
updateTreatmentId(clevelandSmall_cSet, treatmentNames(clevelandSmall_cSet))
```

`[,LongTable,ANY,ANY,ANY-method` *[LongTable Method*

Description

Single bracket subsetting for a LongTable object. See subset for more details.

Usage

```
## S4 method for signature 'LongTable,ANY,ANY,ANY'
x[i, j, assays = assayNames(x), ..., drop = FALSE]
```

Arguments

`x`	`LongTable` The object to subset.
`i`	`character`, `numeric`, `logical` or `call` Character: pass in a character vector of drug names, which will subset the object on all row id columns matching the vector. This parameter also supports valid R regex query strings which will match on the colnames of `x`. For convenience, * is converted to .* automatically. Colon can be to denote a specific part of the colnames string to query. Numeric or Logical: these select based on the rowKey from the `rowData` method for the `LongTable`. Call: Accepts valid query statements to the `data.table` i parameter as a call object. We have provided the function .() to conveniently convert raw R statements into a call for use in this function.
`j`	`character`, `numeric`, `logical` or `call` Character: pass in a character vector of drug names, which will subset the object on all drug id columns matching the vector. This parameter also supports regex queries. Colon can be to denote a specific part of the colnames string to query. Numeric or Logical: these select based on the rowID from the `rowData` method for the `LongTable`. Call: Accepts valid query statements to the `data.table` i parameter as a call object. We have provided the function .() to conveniently convert raw R statements into a call for use in this function.
`assays`	`character` Names of assays which should be kept in the `LongTable` after subsetting.
`...`	Included to ensure drop can only be set by name.
`drop`	`logical` Included for compatibility with the '[' primitive, it defaults to FALSE and changing it does nothing.

Details

This function is endomorphic, it always returns a LongTable object.

Value

A `LongTable` containing only the data specified in the function parameters.

Examples

```
# Character
merckLongTable['ABT-888', 'CAOV3']
# Numeric
merckLongTable[1, c(1, 2)]
# Logical
merckLongTable[, colData(merckLongTable)$sampleid == 'A2058']
# Call
merckLongTable[
     .(drug1id == 'Dasatinib' & drug2id != '5-FU'),
     .(sampleid == 'A2058'),
 ]
```

`[[<-,LongTable,ANY,ANY-method` *`[[<-` Method for LongTable Class*

Description

Just a wrapper around assay<- for convenience. See `?'assay<-,LongTable,character-method'`.

Usage

```
## S4 replacement method for signature 'LongTable,ANY,ANY'
x[[i]] <- value
```

Arguments

`x`	A `LongTable` to update.
`i`	The name of the assay to update, must be in `assayNames(object)`.
`value`	A `data.frame`

Value

A `LongTable` object with the assay `i` updated using `value`.

Examples

```
merckLongTable[['sensitivity']] <- merckLongTable[['sensitivity']]
```

`$,LongTable-method` *Select an assay from a LongTable object*

Description

Select an assay from a LongTable object

Usage

```
## S4 method for signature 'LongTable'
x$name
```

Arguments

`x`	A `LongTable` object to retrieve an assay from
`name`	`character` The name of the assay to get.

Value

`data.frame` The assay object.

Examples

```
merckLongTable$sensitivity
```

`$<-,LongTable-method` *Update an assay from a LongTable object*

Description

Update an assay from a LongTable object

Usage

```
## S4 replacement method for signature 'LongTable'
x$name <- value
```

Arguments

`x`	A `LongTable` to update an assay for.
`name`	`character(1)` The name of the assay to update
`value`	A `data.frame` or `data.table` to update the assay with.

Value

Updates the assay `name` in `x` with `value`, returning an invisible NULL.

Examples

```
merckLongTable$sensitivity <- merckLongTable$sensitivity
```

Index