Package 'MultiAssayExperiment'

October 16, 2018

Title Software for the integration of multi-omics experiments in
Bioconductor

Version 1.6.0

Description Multi-assay 'omics experiments on a set of samples are increasingly
commonplace in biomedical research. MultiAssayExperiment implements data
structures and methods for representing, manipulating, and integrating
multi-assay experiments via efficient construction, subsetting, and
extraction operations. These methods are implemented matching Bioconductor
user experience by straightforward extending concept and design of
single-assay classes such as SummarizedExperiment.

Depends R (>= 3.5.0)

Imports methods, GenomicRanges (>= 1.25.93), BiocGenerics,
SummarizedExperiment (>= 1.3.81), S4Vectors (>= 0.17.5),
IRanges, Biobase, stats, tidyr, utils

Suggests BiocStyle, testthat, knitr, rmarkdown, R.rsp, HDF5Array,
RaggedExperiment, UpSetR, survival, survminer

biocViews Infrastructure, DataRepresentation

License Artistic-2.0

Encoding UTF-8

LazyData true

VignetteBuilder knitr, R.rsp

URL https:
//github.com/waldronlab/MultiAssayExperiment/wiki/MultiAssayExperiment-API

BugReports https://github.com/waldronlab/MultiAssayExperiment/issues

Collate 'ExperimentList-class.R' 'MultiAssayExperiment-class.R'
'subsetBy-methods.R' 'MultiAssayExperiment-subset.R'
'MultiAssayExperiment-methods.R'
'MultiAssayExperiment-helpers.R' 'MultiAssayExperiment-pkg.R'
'assay-methods.R' 'data.R' 'hasAssay.R' 'listToMap.R'
'mapToList.R' 'prepMultiAssay.R' 'reexports.R' 'upsetSamples.R'

RoxygenNote 6.0.1

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

git_branch RELEASE_3_7

git_last_commit 40c8a32

git_last_commit_date 2018-04-30

Date/Publication 2018-10-15

Author Marcel Ramos [aut, cre],
     Levi Waldron [aut],
     MultiAssay SIG [ctb]

Maintainer Marcel Ramos <marcel.ramos@roswellpark.org>

R topics documented:

MultiAssayExperiment-package
                    MultiAssayExperiment: Build an integrative multi-assay container

---

Description

MultiAssayExperiment allows the manipulation of related multiassay datasets with partially over-lapping samples, associated metadata at the level of an entire study, and at the level of the "biological unit". The biological unit may be a patient, plant, yeast strain, etc.

Details

The package hierarchy of information:

- study

- experiments

- samples

Author(s)

Maintainer: Marcel Ramos <marcel.ramos@roswellpark.org>

Authors:

- Levi Waldron <lwaldron.research@gmail.com>

Other contributors:

- MultiAssay SIG <biocmultiassay@googlegroups.com> [contributor]

See Also

Useful links:

- <https://github.com/waldronlab/MultiAssayExperiment/wiki/MultiAssayExperiment-API>
- Report bugs at <https://github.com/waldronlab/MultiAssayExperiment/issues>

ExperimentList	*Construct an* ExperimentList *object for the* MultiAssayExperiment *object slot.*

Description

The ExperimentList class can contain several different types of data. The only requirements for an ExperimentList class are that the objects contained have the following set of methods: dim, [, dimnames

Usage

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

Arguments

...	A named list class object

Value

A ExperimentList class object of experiment data

Examples

```
## Create an empty ExperimentList instance
ExperimentList()

## Create array matrix and AnnotatedDataFrame to create an ExpressionSet class
arraydat <- matrix(data = seq(101, length.out = 20), ncol = 4,
    dimnames = list(
        c("ENST00000294241", "ENST00000355076",
        "ENST00000383706","ENST00000234812", "ENST00000383323"),
        c("array1", "array2", "array3", "array4")
    ))

colDat <- data.frame(slope53 = rnorm(4),
```

```
       row.names = c("array1", "array2", "array3", "array4"))

## SummarizedExperiment constructor
exprdat <- SummarizedExperiment::SummarizedExperiment(arraydat,
    colData = colDat)

## Create a sample methylation dataset
methyldat <- matrix(data = seq(1, length.out = 25), ncol = 5,
    dimnames = list(
        c("ENST00000355076", "ENST00000383706",
          "ENST00000383323", "ENST00000234812", "ENST00000294241"),
        c("methyl1", "methyl2", "methyl3",
          "methyl4", "methyl5")
    ))

## Create a sample RNASeqGene dataset
rnadat <- matrix(
    data = sample(c(46851, 5, 19, 13, 2197, 507,
        84318, 126, 17, 21, 23979, 614), size = 20, replace = TRUE),
    ncol = 4,
    dimnames = list(
        c("XIST", "RPS4Y1", "KDM5D", "ENST00000383323", "ENST00000234812"),
        c("samparray1", "samparray2", "samparray3", "samparray4")
    ))

## Create a mock RangedSummarizedExperiment from a data.frame
rangedat <- data.frame(chr="chr2", start = 11:15, end = 12:16,
    strand = c("+", "-", "+", "*", "."),
    samp0 = c(0,0,1,1,1), samp1 = c(1,0,1,0,1), samp2 = c(0,1,0,1,0),
    row.names = paste0("GENE", letters[5:1]))

rangeSE <- SummarizedExperiment::makeSummarizedExperimentFromDataFrame(rangedat)

## Combine to a named list and call the ExperimentList constructor function
ExpList <- list(Affy = exprdat, Methyl450k = methyldat, RNASeqGene = rnadat,
                GISTIC = rangeSE)

## Use the ExperimentList constructor
myExperimentList <- ExperimentList(ExpList)
```

ExperimentList-class   *A container for multi-experiment data*

Description

The ExperimentList class is a container that builds on the SimpleList with additional checks for consistency in experiment names and length. It contains a SimpleList of experiments with sample identifiers. One element present per experiment performed.

Usage

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

```
## S4 method for signature 'ExperimentList'
isEmpty(x)

## S4 method for signature 'ExperimentList'
dimnames(x)

## S4 method for signature 'ExperimentList'
mergeReplicates(x, replicates = list(),
  simplify = BiocGenerics::mean, ...)

## S4 method for signature 'ANY,missing'
assay(x, i, ...)

## S4 method for signature 'ExperimentList'
assays(x, ..., withDimnames = TRUE)

## S4 method for signature 'ExperimentList,missing'
assay(x, i, ...)

## S4 method for signature 'ExperimentList,numeric'
assay(x, i, ...)

## S4 method for signature 'ExperimentList,character'
assay(x, i, ...)
```

Arguments

object, x	An [ExperimentList](#) object
replicates	mergeReplicates: A list or [LogicalList](#) where each element represents a sample and a vector of repeated measurements for the sample
simplify	A function for merging columns where duplicates are indicated by replicates
...	Additional arguments. See details for more information.
i	A scalar character or integer index
withDimnames	logical (default TRUE) whether to return dimension names

Details

Convert from SimpleList or list to the multi-experiment data container. When using the **mergeReplicates** method, additional arguments are passed to the given simplify function argument (e.g., na.rm = TRUE)

Methods (by generic)

- show: Show method for [ExperimentList](#) class
- isEmpty: check for zero length across all experiments
- dimnames: Get the dimension names for an ExperimentList using [CharacterList](#)
- mergeReplicates: Apply the mergeReplicates method on the ExperimentList elements
- assay: Get the assay data for the default ANY class
- assays: Get the assay data from each element in the [ExperimentList](#)
- assay: Convenience function for the assay of the first element

- assay: Obtain the specified assay from ExperimentList with a `numeric` index
- assay: Get the specified assay from ExperimentList with a `character` index

coercion

Convert a `list` or S4 `List` to an ExperimentList using the `as()` function.

In the following example, x is either a `list` or List:

```
\code{as(x, "ExperimentList")}
```

Examples

```
ExperimentList()
```

---

hasAssay   *Checking assay method for any class*

Description

The `hasAssay` function is intended for developers who would like to include new classes into a `MultiAssayExperiment` instance. It checks the methods tables of the `assay` function for the specified class of the argument.

Usage

```
hasAssay(object)
```

Arguments

object   A `MultiAssayExperiment` or named `list` object instance

Value

A `logical` value indicating method availability

Examples

```
lst <- structure(list(), .Names=character())
hasAssay(lst)
```

---

listToMap   *Convert map from data.frame or DataFrame to list and vice versa*

Description

The `mapToList` function provides a convenient way of reordering a `data.frame` to a `list`. The `listToMap` function does the opposite by taking a `list` and converting it to `DataFrame`.

Usage

```
listToMap(listmap, type = "colnames")

mapToList(dfmap, assayCol = "assay")
```

Arguments

listmap	A named `list` object containing either experiments (`assays`), samples (`colnames`) or features (`rownames`)
type	Any of the valid types of maps including colnames, rownames, and assays.
dfmap	A `data.frame` or `DataFrame` object with identifiers in the first column
assayCol	A character vector of length one indicating the assay names column

Value

A [DataFrame](#) class object of names

A `list` object of DataFrames for each assay

Functions

- listToMap: The inverse of the listToMap operation

Examples

```
example("MultiAssayExperiment")

## Create a sampleMap from a list using the listToMap function
mySampleMap <- listToMap(mylist)

## The inverse operation is also available
mylist <- mapToList(mySampleMap)
```

MatchedAssayExperiment-class

An integrative and matched-samples class for experiment data

Description

This class supports the use of matched samples where an equal number of observations per biological unit are present in all assays.

Usage

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

Arguments

...   Either a single MultiAssayExperiment or the components to create a valid MultiAssayExperiment

Value

A MatchedAssayExperiment object

Functions

- MatchedAssayExperiment: Construct a MatchedAssayExperiment class from MultiAssay-Experiment inputs.

See Also

MultiAssayExperiment

Examples

```
data("miniACC")
acc <- as(miniACC, "MatchedAssayExperiment")
acc
```

---

miniACC   *Adrenocortical Carcinoma (ACC) MultiAssayExperiment*

Description

A MultiAssayExperiment object providing a reduced version of the TCGA ACC dataset for all 92 patients. RNA-seq, copy number, and somatic mutations are included only for genes whose proteins are included in the reverse-phase protein array. The MicroRNA-seq dataset is also included, with infrequently expressed microRNA removed. Clinical, pathological, and subtype information are provided by colData(miniACC), and some additional details are provided by metadata(miniACC).

Usage

```
miniACC
```

Format

A `MultiAssayExperiment` with 5 experiments, providing:

RNASeq2GeneNorm RNA-seq count data: an `ExpressionSet` with 198 rows and 79 columns

gistict Reccurent copy number lesions identified by GISTIC2: a `SummarizedExperiment` with 198 rows and 90 columns

RPPAArray Reverse Phase Protein Array: an `ExpressionSet` with 33 rows and 46 columns. Rows are indexed by genes, but protein annotations are available from `featureData(miniACC[["RPPAArray"]])`. The source of these annotations is noted in `abstract(miniACC[["RPPAArray"]])`

Mutations Somatic mutations: a `matrix` with 223 rows and 90 columns. 1 for any kind of non-silent mutation, zero for silent (synonymous) or no mutation.

miRNASeqGene microRNA sequencing: an `ExpressionSet` with 471 rows and 80 columns. Rows not having at least 5 counts in at least 5 samples were removed.

Author(s)

Levi Waldron <lwaldron.research@gmail.com>

Source

https://github.com/waldronlab/multiassayexperiment-tcga

References

Zheng S *et al.*: Comprehensive Pan-Genomic Characterization of Adrenocortical Carcinoma. Cancer Cell 2016, 29:723-736.

Examples

```
miniACC
metadata(miniACC)
colnames(colData(miniACC))
table(miniACC$vital_status)
longFormat(miniACC["MAPK3", , ], colDataCols = c("vital_status", "days_to_death"))
wideFormat(miniACC["MAPK3", , ], colDataCols = c("vital_status", "days_to_death"))


##
## The following is the code used to create this mini dataset from the full ACC dataset.
## The full ACC MultiAssayExperiment was created by the pipeline at
## https://github.com/waldronlab/multiassayexperiment-tcga.
## Not run:
    ## See www.tinyurl.com/MAEOurls for more pre-built TCGA MultiAssayExperiment objects
    download.file("http://s3.amazonaws.com/multiassayexperiments/accMAEO.rds",
                  destfile = "accMAEO.rds")
    library(MultiAssayExperiment)
    library(RaggedExperiment) #needed for RaggedExperiment objects by updateObject()
    library(Biobase)

    acc <- readRDS("accMAEO.rds")
    acc <- updateObject(acc)
```

```
protmap <- read.csv(paste0("http://genomeportal.stanford.edu/",
                           "pan-tcga/target_selection_send_data",
                           "?filename=Allprotein.txt"), as.is = TRUE
    )

RPPAgenes <- Filter(function(x) x != "", protmap$Genes)
RPPAgenes <- unlist(strsplit(RPPAgenes, ","))
RPPAgenes <- unique(RPPAgenes)

miniACC <-
    acc[RPPAgenes, , c("RNASeq2GeneNorm", "gistict", "RPPAArray", "Mutations")]
mut <- assay(miniACC[["Mutations"]], i = "Variant_Classification")
mut <- ifelse(is.na(mut) | mut == "Silent", 0, 1)

miniACC[["Mutations"]] <- mut
colData(miniACC) <- colData(miniACC)[, c(1:17, 810:822)]

rpparowData <-
    protmap[match(rownames(miniACC[["RPPAArray"]]), protmap$Genes),]
rpparowData <- AnnotatedDataFrame(rpparowData)
featureData(miniACC[["RPPAArray"]]) <- rpparowData

md <-
    list(
      title = "Comprehensive Pan-Genomic Characterization of Adrenocortical Carcinoma",
        PMID = "27165744",
        sourceURL = "http://s3.amazonaws.com/multiassayexperiments/accMAEO.rds",
        RPPAfeatureDataURL = paste0("http://genomeportal.stanford.edu/",
                                    "pan-tcga/show_target_selection_file",
                                    "?filename=Allprotein.txt"),
      colDataExtrasURL = "http://www.cell.com/cms/attachment/2062093088/2063584534/mmc3.xlsx"
    )
metadata(miniACC) <- md

mirna <- acc[["miRNASeqGene"]]
mirna <- mirna[rowSums(assay(mirna) >= 5) >= 5, ]
experimentData(mirna)@abstract <-
    "Note: Rows not having at least 5 counts in at least 5 samples were removed."
miniACC <- c(miniACC,
             list(miRNASeqGene = mirna),
             sampleMap = sampleMap(acc)[sampleMap(acc)$assay == "miRNASeqGene",])

 miniACC[["RNASeq2GeneNorm"]] <-
     as(miniACC[["RNASeq2GeneNorm"]], "SummarizedExperiment")
 miniACC[["RPPAArray"]] <-
     as(miniACC[["RPPAArray"]], "SummarizedExperiment")
 miniACC[["miRNASeqGene"]] <-
     as(miniACC[["miRNASeqGene"]], "SummarizedExperiment")

save(miniACC, file = "data/miniACC.RData", compress = "bzip2")

## End(Not run)
```

MultiAssayExperiment   *Construct a* MultiAssayExperiment *object*

Description

The constructor function for the [MultiAssayExperiment-class](#) combines multiple data elements from the different hierarchies of data (study, experiments, and samples). It can create instances where neither a `sampleMap` or a `colData` set is provided. Please see the MultiAssayExperiment API documentation for more information.

Usage

```
MultiAssayExperiment(experiments = ExperimentList(),
  colData = S4Vectors::DataFrame(), sampleMap = S4Vectors::DataFrame(assay =
  factor(), primary = character(), colname = character()), metadata = NULL,
  drops = list())
```

Arguments

experiments	A `list` or [ExperimentList](#) of all combined experiments
colData	A [DataFrame](#) or `data.frame` of characteristics for all biological units
sampleMap	A `DataFrame` or `data.frame` of assay names, sample identifiers, and colname samples
metadata	An optional argument of "ANY" class (usually list) for content describing the experiments
drops	A `list` of unmatched information (included after subsetting)

Value

A `MultiAssayExperiment` object that can store experiment and phenotype data

See Also

[MultiAssayExperiment-class](#)

Examples

```
## Run the example ExperimentList
example("ExperimentList")

## Create sample maps for each experiment
exprmap <- data.frame(
    primary = c("Jack", "Jill", "Barbara", "Bob"),
    colname = c("array1", "array2", "array3", "array4"),
    stringsAsFactors = FALSE)

methylmap <- data.frame(
    primary = c("Jack", "Jack", "Jill", "Barbara", "Bob"),
    colname = c("methyl1", "methyl2", "methyl3", "methyl4", "methyl5"),
    stringsAsFactors = FALSE)

rnamap <- data.frame(
    primary = c("Jack", "Jill", "Bob", "Barbara"),
    colname = c("samparray1", "samparray2", "samparray3", "samparray4"),
    stringsAsFactors = FALSE)

gistmap <- data.frame(
    primary = c("Jack", "Bob", "Jill"),
```

```
        colname = c("samp0", "samp1", "samp2"),
        stringsAsFactors = FALSE)

## Combine as a named list and convert to a DataFrame
mylist <- list(Affy = exprmap, Methyl450k = methylmap,
        RNASeqGene = rnamap, GISTIC = gistmap)

## Create a sampleMap
mySampleMap <- listToMap(mylist)
## Create an example phenotype data
colDat <- data.frame(sex = c("M", "F", "M", "F"), age = 38:41,
        row.names = c("Jack", "Jill", "Bob", "Barbara"))

## Create a MultiAssayExperiment instance
myMultiAssayExperiment <- MultiAssayExperiment(experiments = ExpList,
        colData = colDat, sampleMap = mySampleMap)
```

MultiAssayExperiment-class

An integrative multi-assay class for experiment data

Description

The MultiAssayExperiment class can be used to manage results of diverse assays on a collection of specimen. Currently, the class can handle assays that are organized instances of SummarizedExperiment, ExpressionSet, matrix, RaggedExperiment (inherits from GRangesList), and RangedVcfStack. Create new MultiAssayExperiment instances with the homonymous constructor, minimally with the argument ExperimentList, potentially also with the arguments colData (see section below) and sampleMap.

Usage

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

## S4 method for signature 'MultiAssayExperiment'
length(x)

## S4 method for signature 'MultiAssayExperiment'
names(x)

## S4 method for signature 'MultiAssayExperiment'
updateObject(object, ..., verbose = FALSE)

## S4 method for signature 'MultiAssayExperiment'
dimnames(x)

## S4 method for signature 'MultiAssayExperiment'
c(x, ..., sampleMap = NULL, mapFrom = NULL)

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

```
## S4 method for signature 'MultiAssayExperiment,missing'
assay(x, i, ...)

## S4 method for signature 'MultiAssayExperiment,numeric'
assay(x, i, ...)

## S4 method for signature 'MultiAssayExperiment,character'
assay(x, i, ...)
```

Arguments

object	A `MultiAssayExperiment` object
x	A `MultiAssayExperiment` object
...	Additional arguments for supporting functions. See details.
verbose	logical (default FALSE) whether to print extra messages
sampleMap	c method: a `sampleMap` list or `DataFrame` to guide merge
mapFrom	Either a `logical`, `character`, or `integer` vector indicating the experiment(s) that have an identical colname order as the experiment input(s)
withDimnames	logical (default TRUE) whether to return dimension names included in the object
i	An integer or character scalar indicating the assay to return

Details

The dots (`...`) argument allows the user to specify additional arguments in several instances.

- subsetting **[**: additional arguments sent to [findOverlaps.](#)
- mergeReplicates: used to specify arguments for the `simplify` functional argument
- assay: may contain withDimnames, which is forwarded to assays
- combining **c**: compatible `MultiAssayExperiment` classes passed on to the [ExperimentList](#) constructor, can be a list, [List,](#) or a series of named arguments. See the examples below.

Value

A `MultiAssayExperiment` object

Methods (by generic)

- show: Show method for a `MultiAssayExperiment`
- length: Get the length of ExperimentList
- names: Get the names of the ExperimentList
- updateObject: Update old serialized MultiAssayExperiment objects to new API
- dimnames: Get the dimension names for a `MultiAssayExperiment` object
- c: Add a supported data class to the ExperimentList
- assays: Obtain a [SimpleList](#) of assay data for all available experiments in the object
- assay: Convenience function for extracting the assay of the first element in the ExperimentList
- assay: Obtain the specified assay from the MultiAssayExperiment with a numeric index
- assay: Get the specified assay from the MultiAssayExperiment with a character index

Slots

> ExperimentList A ExperimentList class object for each assay dataset
>
> colData A DataFrame of all clinical/specimen data available across experiments
>
> sampleMap A DataFrame of translatable identifiers of samples and participants
>
> metadata Additional data describing the MultiAssayExperiment object
>
> drops A metadata list of dropped information

colData

> The colData slot is a collection of primary specimen data valid across all experiments. This slot
> is strictly of class DataFrame but arguments for the constructor function allow arguments to be of
> class data.frame and subsequently coerced.

ExperimentList

> The ExperimentList slot is designed to contain results from each experiment/assay. It contains a
> SimpleList.

sampleMap

> The sampleMap contains a DataFrame of translatable identifiers of samples and participants or
> biological units. Standard column names of the sampleMap are "assay", "primary", and "colname".

See Also

> MultiAssayExperiment-methods for slot modifying methods MultiAssayExperiment API

Examples

```
example("MultiAssayExperiment")

## Subsetting
# Rows (i) Rows/Features in each experiment
myMultiAssayExperiment[1, , ]
myMultiAssayExperiment[c(TRUE, FALSE), , ]

# Columns (j) Rows in colData
myMultiAssayExperiment[, rownames(colData(myMultiAssayExperiment))[3:2],  ]

# Assays (k)
myMultiAssayExperiment[, , "Affy"]

## Complete cases (returns logical vector)
completes <- complete.cases(myMultiAssayExperiment)
compMAE <- myMultiAssayExperiment[, completes, ]
compMAE
colData(compMAE)

example("MultiAssayExperiment")

## Add an experiment
test1 <- myMultiAssayExperiment[[1L]]
colnames(test1) <- rownames(colData(myMultiAssayExperiment))
```

```
## Combine current MultiAssayExperiment with additional experiment
## (no sampleMap)
c(myMultiAssayExperiment, newExperiment = test1)

test2 <- myMultiAssayExperiment[[1L]]
c(myMultiAssayExperiment, newExp = test2, mapFrom = 3L)
```

MultiAssayExperiment-helpers

A group of helper functions for manipulating and cleaning a MultiAssayExperiment

---

Description

A set of helper functions were created to help clean and manipulate a MultiAssayExperiment object. intersectRows also works for ExperimentList objects.

- complete.cases: Returns a logical vector corresponding to 'colData' rows that have data across all experiments

- isEmpty: Returns a logical TRUE value for zero length MultiAssayExperiment objects

- intersectRows: Takes all common rows across experiments, excludes experiments with empty rownames

- intersectColumns: A wrapper for complete.cases to return a MultiAssayExperiment with only those biological units that have measurements across all experiments

- replicated: A function that identifies multiple samples that originate from a single biological unit within each assay

- anyReplicated: Displays which assays have replicate measurements

- mergeReplicates: A function that combines replicated / repeated measurements across all experiments and is guided by the replicated return value

- longFormat: A MultiAssayExperiment method that returns a small and skinny [DataFrame](#). The colDataCols arguments allows the user to append colData columns to the data.

- wideFormat: A function to return a wide [DataFrame](#) where each row represents an observation. Optional colDataCols can be added when using a MultiAssayExperiment.

- hasRowRanges: A function that identifies ExperimentList elements that have a [rowRanges](#) method

- duplicated: (Deprecated) Returns a 'list' of 'LogicalList's that indicate what measurements originate from the same biological unit

Usage

```
## S4 method for signature 'MultiAssayExperiment'
complete.cases(...)

## S4 method for signature 'MultiAssayExperiment'
isEmpty(x)

intersectRows(x)
```

```
intersectColumns(x)

replicated(x)

## S4 method for signature 'MultiAssayExperiment'
replicated(x)

anyReplicated(x)

## S4 method for signature 'MultiAssayExperiment'
anyReplicated(x)

mergeReplicates(x, replicates = list(), simplify = BiocGenerics::mean, ...)

## S4 method for signature 'MultiAssayExperiment'
mergeReplicates(x,
  replicates = replicated(x), simplify = BiocGenerics::mean, ...)

## S4 method for signature 'ANY'
mergeReplicates(x, replicates = list(),
  simplify = BiocGenerics::mean, ...)

longFormat(object, colDataCols = NULL, i = 1L)

wideFormat(object, colDataCols = NULL, check.names = TRUE, collapse = "_",
  i = 1L)

hasRowRanges(x)

## S4 method for signature 'MultiAssayExperiment'
hasRowRanges(x)

## S4 method for signature 'ExperimentList'
hasRowRanges(x)

## S4 method for signature 'MultiAssayExperiment'
duplicated(x, incomparables = FALSE, ...)
```

Arguments

...	Additional arguments. See details for more information.
x	A MultiAssayExperiment or ExperimentList
replicates	A list of [LogicalList](#)s indicating multiple / duplicate entries for each biological unit per assay, see replicated (default replicated(x)).
simplify	A function for merging repeat measurements in experiments as indicated by the replicated method for MultiAssayExperiment
object	Any supported class object
colDataCols	A character, logical, or numeric index for colData columns to be included
i	An optional assay indicator for any [SummarizedExperiment](#) objects (default 1L)

check.names	(logical default TRUE) Column names of the output `DataFrame` will be checked for syntactic validity and made unique, if necessary
collapse	(character default "_") A single string delimiter for output column names. In `wideFormat`, experiments and rownames (and when replicate samples are present, colnames) are seperated by this delimiter
incomparables	unused argument

Details

The `replicated` function finds replicate measurements in each assay and returns a list of [LogicalList](#)s. Each element in a single [LogicalList](#) corresponds to a biological or *primary* unit as in the `sampleMap`. Below is a small graphic for one particular biological unit in one assay, where the logical vector corresponds to the number of measurements/samples in the assay:

```
>      replicated(MultiAssayExperiment)
(list str)        '-- $ AssayName
(LogicalList str)     '-- [[ "Biological Unit" ]]
Replicated if sum(...) > 1         '-- TRUE TRUE FALSE FALSE
```

`anyReplicated` determines if any of the assays have at least one replicate. *Note*. These methods are not available for the ExperimentList class due to a missing `sampleMap` structure (by design).

The `mergeReplicates` function is a house-keeping method for a MultiAssayExperiment where only `complete.cases` are returned, replicate measurements are averaged (by default), and columns are aligned by the row order in `colData`. Additional arguments can be passed on to the `simplify` function.

The `mergeReplicates` "ANY" method consolidates duplicate measurements for rectangular data structures, returns object of the same class (endomorphic). The ellipsis or `...` argument allows the user to provide additional arguments to the `simplify` functional argument.

The `longFormat` "ANY" class method, works with classes such as [ExpressionSet](#) and [Summarized-Experiment](#) as well as `matrix` to provide a consistent long and skinny [DataFrame](#).

The `hasRowRanges` method identifies assays that support a [rowRanges](#) method *and* return a [GRanges](#) object.

Deprecated: For the `anyDuplicated` and `duplicated` functions, the `incomparables` and ellipsis `...` arguments are not used. Neither `duplicated` nor `anyDuplicated` is supported for ExperimentList due to an unavailable `sampleMap`.

mergeReplicates

The `mergeReplicates` function makes use of the output from `replicated` which will point out the duplicate measurements by biological unit in the MultiAssayExperiment. This function will return a MultiAssayExperiment with merged replicates. Additional arguments can be provided to the simplify argument via the ellipsis (...).

longFormat

The longFormat method takes data from the [ExperimentList](#) in a [MultiAssayExperiment](#) and returns a uniform [DataFrame](#). The resulting DataFrame has columns indicating primary, rowname, colname and value. This method can optionally include colData columns with the `colDataCols` argument (MultiAssayExperiment method only). The `...` argument allows the user to specify the assay value for the [SummarizedExperiment](#) assay function's i argument.

wideFormat

The wideFormat function returns standardized wide [DataFrame](#) where each row represents a biological unit as in the colData. Depending on the data and setup, biological units can be patients, tumors, specimens, etc. Optionally, colData columns can be added to the wide data output (see the colDataCols argument). Metadata columns are generated based on the names produced in the wide format [DataFrame](#). These can be accessed via the [mcols](#) function.

hasRowRanges

The hasRowRanges method identifies assays with associated ranged row data by directly testing the method on the object. The result from the test must be a [GRanges](#) class object to satisfy the test.

MultiAssayExperiment-methods

Accessing/modifying slot information

Description

A set of accessor and setter generic functions to extract either the sampleMap, the [ExperimentList](#), colData, or metadata slots of a [MultiAssayExperiment](#) object

Usage

```
## S4 method for signature 'MultiAssayExperiment'
sampleMap(x)

## S4 method for signature 'MultiAssayExperiment'
experiments(x)

## S4 method for signature 'MultiAssayExperiment'
colData(x, ...)

## S4 method for signature 'MultiAssayExperiment'
metadata(x)

## S4 replacement method for signature 'MultiAssayExperiment,DataFrame'
sampleMap(object) <- value

## S4 replacement method for signature 'MultiAssayExperiment,ExperimentList'
experiments(object) <- value

## S4 replacement method for signature 'MultiAssayExperiment,DataFrame'
colData(x) <- value

## S4 replacement method for signature 'MultiAssayExperiment'
metadata(x, ...) <- value

## S4 replacement method for signature 'MultiAssayExperiment'
x$name <- value

## S4 method for signature 'MultiAssayExperiment'
x$name
```

Arguments

x	A `MultiAssayExperiment` object
...	Argument not in use
object	A `MultiAssayExperiment` object
value	See details.
name	A column in `colData`

Value

Accessors: Either a `sampleMap`, `ExperimentList`, or `DataFrame` object

Setters: A `MultiAssayExperiment` object

Accessors

Eponymous names for accessing `MultiAssayExperiment` slots with the exception of the [Experi-mentList](#) accessor named `experiments`.

- colData: Access the `colData` slot
- sampleMap: Access the `sampleMap` slot
- experiments: Access the [ExperimentList](#) slot
- '[[': Access the [ExperimentList](#) slot
- '$': Access a column in `colData`

Setters

Setter method values (i.e., 'function(x) <- value'):

- experiments<-: An [`ExperimentList`](#) object containing experiment data of supported classes
- sampleMap<-: A [`DataFrame`](#) object relating samples to biological units and assays
- colData<-: A [`DataFrame`](#) object describing the biological units
- metadata<-: A `list` object of metadata
- '[[<-': Equivalent to the `experiments<-` setter method for convenience
- '$<-': A vector to replace the indicated column in `colData`

Examples

```
## Load example MultiAssayExperiment
example(MultiAssayExperiment)

## Access the sampleMap
sampleMap(myMultiAssayExperiment)

## Replacement method for a MultiAssayExperiment sampleMap
sampleMap(myMultiAssayExperiment) <- S4Vectors::DataFrame()

## Access the ExperimentList
experiments(myMultiAssayExperiment)

## Replace with an empty ExperimentList
experiments(myMultiAssayExperiment) <- ExperimentList()
```

```
## Access the metadata
metadata(myMultiAssayExperiment)

## Replace metadata with a list
metadata(myMultiAssayExperiment) <- list(runDate =
    format(Sys.time(), "%B %d, %Y"))

## Access the colData
colData(myMultiAssayExperiment)

## Access a column in colData
myMultiAssayExperiment$age

## Replace a column in colData
myMultiAssayExperiment$age <- myMultiAssayExperiment$age + 1
```

---

prepMultiAssay   *Prepare a* MultiAssayExperiment *instance*

Description

The purpose of this helper function is to faciltate the creation of a MultiAssayExperiment object by detecting any inconsistencies with all types of names in either the ExperimentList, the colData, or sampleMap.

Usage

```
prepMultiAssay(ExperimentList, colData, sampleMap, ...)
```

Arguments

ExperimentList	A list of all combined experiments
colData	A DataFrame of the phenotype data for all participants
sampleMap	A DataFrame of sample identifiers, assay samples, and assay names
...	Optional arguments for the MultiAssayExperiment constructor function such as metadata and drops.

Value

A list containing all the essential components of a MultiAssayExperiment as well as a "drops" metadata element that indicates non-matched names. The names of the resulting list correspond to the arguments of the MultiAssayExperiment constructor function.

Checks

The prepMultiAssay function checks that all columns in the sampleMap are character.

It checks that all names and lengths match in both the ExperimentList and in the unique assay names of the sampleMap.

If ExperimentList names and assay names only differ by case and are not duplicated, the function will standardize all names to lowercase.

If names cannot be matched between the colname column of the [sampleMap](#) and the colnames of the ExperimentList, those unmatched will be dropped and found in the "drops" element of the resulting list.

Names in the "primary" column of the [sampleMap](#), will be matched to those in the colData. Unmatched "primary" column rows will be dropped from the [sampleMap](#). Suggestions for name fixes in either the [ExperimentList](#) or colnames will be made when necessary.

Examples

```
## Run example
example("MultiAssayExperiment")

## Check if there are any inconsistencies within the different names
preparedMAE <- prepMultiAssay(ExpList, colDat, mySampleMap)

## Results in a list of components for the MultiAssayExperiment constructor
## function
MultiAssayExperiment(preparedMAE$experiments, preparedMAE$colData,
preparedMAE$sampleMap)

## Alternatively, use the do.call function
do.call(MultiAssayExperiment, preparedMAE)
```

| reexports | *Objects exported from other packages* |

Description

These objects are imported from other packages. Click on the function name to see their documentation.

- S4Vectors: [DataFrame](#)

Examples

```
DataFrame()
```

| subsetBy | *Subsetting a MultiAssayExperiment object* |

Description

A set of functions for extracting and dividing a MultiAssayExperiment

Usage

```
subsetByRow(x, y, ...)

subsetByColData(x, y)

subsetByColumn(x, y)

subsetByAssay(x, y)

## S4 method for signature 'ExperimentList,ANY'
subsetByRow(x, y, ...)

## S4 method for signature 'ExperimentList,list'
subsetByRow(x, y)

## S4 method for signature 'ExperimentList,List'
subsetByRow(x, y)

## S4 method for signature 'ExperimentList,logical'
subsetByRow(x, y)

## S4 method for signature 'ExperimentList,list'
subsetByColumn(x, y)

## S4 method for signature 'ExperimentList,List'
subsetByColumn(x, y)

## S4 method for signature 'ExperimentList,logical'
subsetByColumn(x, y)

## S4 method for signature 'ExperimentList'
subsetByAssay(x, y)

## S4 method for signature 'MultiAssayExperiment,ANY'
subsetByColData(x, y)

## S4 method for signature 'MultiAssayExperiment,character'
subsetByColData(x, y)

## S4 method for signature 'MultiAssayExperiment,ANY'
subsetByRow(x, y, ...)

## S4 method for signature 'MultiAssayExperiment,ANY'
subsetByColumn(x, y)

## S4 method for signature 'MultiAssayExperiment'
subsetByAssay(x, y)

## S4 method for signature 'MultiAssayExperiment,ANY,ANY,ANY'
x[i, j, k, ..., drop = TRUE]

## S4 method for signature 'MultiAssayExperiment,ANY,ANY'
```

```
x[[i, j, ...]]

## S4 replacement method for signature 'MultiAssayExperiment,ANY,ANY'
x[[i, j, ...]] <- value
```

Arguments

x	A `MultiAssayExperiment` or `ExperimentList`
y	Any argument used for subsetting, can be a `character`, `logical`, `integer`, `list` or `List` vector
...	Additional arguments passed on to lower level functions.
i	Either a `character`, `integer`, `logical` or `GRanges` object for subsetting by rows
j	Either a `character`, `logical`, or numeric vector for subsetting by `colData` rows. See details for more information.
k	Either a `character`, `logical`, or numeric vector for subsetting by assays
drop	logical (default TRUE) whether to drop empty assay elements in the `ExperimentList`
value	An assay compatible with the MultiAssayExperiment API

Details

Subsetting a MultiAssayExperiment by the **j** index can yield a call to either `subsetByColData` or `subsetByColumn`. For vector inputs, the subset will be applied to the `colData` rows. For `List`-type inputs, the List will be applied to each of the elements in the `ExperimentList`. The order of the subsetting elements in the `List` must match that of the `ExperimentList` in the `MultiAssayExperiment`.

- subsetBycolData: Select biological units by vector input types
- subsetByColumn: Select observations by assay or for each assay
- subsetByRow: Select rows by assay or for each assay
- subsetByAssay: Select experiments

Examples

```
## Load the example MultiAssayExperiment
example("MultiAssayExperiment")

## Using experiment names
subsetByAssay(myMultiAssayExperiment, "Affy")

## Using numeric indices
subsetByAssay(myMultiAssayExperiment, 1:2)

## Using a logical vector
subsetByAssay(myMultiAssayExperiment, c(TRUE, FALSE, TRUE))

## Subset by character vector (Jack)
subsetByColData(myMultiAssayExperiment, "Jack")

## Subset by numeric index of colData rows (Jack and Bob)
subsetByColData(myMultiAssayExperiment, c(1, 3))

## Subset by logical indicator of colData rows (Jack and Jill)
```

```
subsetByColData(myMultiAssayExperiment, c(TRUE, TRUE, FALSE, FALSE))

subsetByColumn(myMultiAssayExperiment, list(Affy = 1:2,
    Methyl450k = c(3,5,2), RNASeqGene = 2:4, GISTIC = 1))

subsetWith <- S4Vectors::mendoapply(`[`, colnames(myMultiAssayExperiment),
    MoreArgs = list(1:2))
subsetByColumn(myMultiAssayExperiment, subsetWith)

## Use a GRanges object to subset rows where ranged data present
egr <- GenomicRanges::GRanges(seqnames = "chr2",
    IRanges::IRanges(start = 11, end = 13), strand = "-")
subsetByRow(myMultiAssayExperiment, egr)

## Use a logical vector (recycling used)
subsetByRow(myMultiAssayExperiment, c(TRUE, FALSE))

## Use a character vector
subsetByRow(myMultiAssayExperiment, "ENST00000355076")
```

---

upsetSamples   *Create a generalized Venn Diagram analog for sample membership in*
                        multiple assays, using the upset algorithm in UpSetR

Description

Create a generalized Venn Diagram analog for sample membership in multiple assays, using the upset algorithm in UpSetR

Usage

```
upsetSamples(MultiAssayExperiment, nsets = length(MultiAssayExperiment),
  nintersects = 24, order.by = "freq", nameFilter = force,
  check.names = FALSE, ...)
```

Arguments

MultiAssayExperiment
                    instance of [MultiAssayExperiment-class](#)

nsets   integer number of sets to analyze

nintersects   Number of intersections to plot. If set to NA, all intersections will be plotted.

order.by   How the intersections in the matrix should be ordered by. Options include fre-
                    quency (entered as "freq"), degree, or both in any order.

nameFilter   function, defaulting to force, to manipulate colnames of incidence matrix

check.names   logical(1) used when incidence matrix is coerced to data.frame for use in Up-
                    SetR::upset

...   parameters passed to [upset](#)

Value

Produces a visualization of set intersections using the UpSet matrix design

Note

This function is intended to provide convenient visualization of assay availability configurations in MultiAssayExperiment instances. The [upset](#) function requires data.frame input and has many parameters to tune appearance of the result. Assay name handling is important for interpretability, and the `nameFilter` parameter may be useful to simplify resulting outputs.

Author(s)

Vincent J Carey

Examples

```
data(miniACC)
upsetSamples(miniACC)
upsetSamples(miniACC, nameFilter = function(x) substr(x, 1, 5))
```

Index