Package: MoleculeExperiment (via r-universe)

June 29, 2024

Title Prioritising a molecule-level storage of Spatial Transcriptomics Data

Version 1.5.1

Description MoleculeExperiment contains functions to create and work with objects from the new MoleculeExperiment class. We introduce this class for analysing molecule-based spatial transcriptomics data (e.g., Xenium by 10X, Cosmx SMI by Nanostring, and Merscope by Vizgen). This allows researchers to analyse spatial transcriptomics data at the molecule level, and to have standardised data formats accross vendors.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

URL https://github.com/SydneyBioX/MoleculeExperiment

BugReports https://github.com/SydneyBioX/MoleculeExperiment/issues

Imports SpatialExperiment, Matrix, purrr, data.table, dplyr (>= 1.1.1), magrittr, rjson, utils, methods, terra, ggplot2, rlang, cli, EBImage, rhdf5, BiocParallel, S4Vectors, stats

Suggests knitr, BiocStyle, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews DataImport, DataRepresentation, Infrastructure, Software, Spatial, Transcriptomics

Config/testthat/edition 3

Depends R (>= 2.10)

Repository https://bioc.r-universe.dev

RemoteUrl https://github.com/bioc/MoleculeExperiment

RemoteRef HEAD

RemoteSha 8ad146e1ccdeb7632fee64a25c3979db89e71ec2

Contents

`.generateBPParam` *Utility function to generate BPPARAM object.*

Description

Utility function to generate BPPARAM object.

Usage

```
.generateBPParam(cores = 1)
```

Arguments

`cores`	Desired number of cores for BPPARAM object.

Value

A BPPPARAM object.

accessors *Accessor functions to work with MoleculeExperiment objects*

Description

Accessor functions to work with MoleculeExperiment objects

Usage

```
## S4 method for signature 'MoleculeExperiment'
molecules(x, assayName = NULL, flatten = FALSE)

## S4 method for signature 'MoleculeExperiment'
boundaries(object, assayName = NULL, flatten = FALSE)

## S4 method for signature 'MoleculeExperiment'
features(object, assayName = NULL)

## S4 method for signature 'MoleculeExperiment'
segmentIDs(object, assayName = NULL)

## S4 replacement method for signature 'MoleculeExperiment'
molecules(x, assayName = NULL) <- value

## S4 replacement method for signature 'MoleculeExperiment'
boundaries(object, assayName = NULL) <- value
```

Arguments

x	The MoleculeExperiment to access.
assayName	Character string specifying the name of the assay from which to retrieve or set information in the slot of interest.
flatten	Logical value specifying whether to flatten the ME list into a data.frame or not. Defaults to FALSE.
object	The MoleculeExperiment to access.
value	New value to be added to the slot and assay of interest.

Value

A MoleculeExperiment object slot.

getters

Accessor functions to get data from the MoleculeExperiment object. These include:

- molecules() to retrieve information from the molecules slot.

- boundaries() to retrieve information from the boundaries slot.
- features() to retrieve feature names from the molecules slot.
- segmentIDs() to retrieve segment ids from the boundaries slot.

setters

The `molecules<-` setter accesses the molecules slot, whereas the boundaries slot can be accessed with `boundaries<-`.

Examples

```
# get example data
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
me <- readXenium(repoDir,
    keepCols = "essential",
    addBoundaries = "cell"
)

# get insight into MoleculeExperiment object (e.g., see assay names)
me

# get insight into molecules slot (e.g., see the assay names)
showMolecules(me)

# for developers, use molecules() getter
# expect a large output from call below
# molecules(me, assayName = "detected")
# alternatively, return rectangular data structure with flatten = TRUE
molecules(me, assayName = "detected", flatten = TRUE)

# get insight into boundaries slot (e.g., see the assay names)
showBoundaries(me)

# for developers, use boundaries() getter
# expect a large output from call below
# boundaries(me, assayName = "cell")
# alternatively, return rectangular data structure with flatten = TRUE
boundaries(me, assayName = "cell", flatten = TRUE)

# features() getter
features(me, assayName = "detected")

# segmentIDs() getter
segmentIDs(me, assayName = "cell")

# setter example
# read in and standardise nucleus boundaries too
nucleiMEList <- readBoundaries(
    dataDir = repoDir,
    pattern = "nucleus_boundaries.csv",
    segmentIDCol = "cell_id",
```

```
    xCol = "vertex_x",
    yCol = "vertex_y",
    keepCols = "essential",
    boundariesAssay = "nucleus",
    scaleFactorVector = 1
)

# use `boundaries<-` setter to add nucleus boundaries to the boundaries slot
boundaries(me, assayName = "nucleus") <- nucleiMEList
me
```

`bufferBoundaries` *Create a new boundaries assay with buffers*

Description

This function takes in an existing MoleculeExperiment object and generates a new boundaries assay with added buffers. This can be useful for visualisation and for countMolecules.

Usage

```
bufferBoundaries(me, assayName = "cell", ...)
```

Arguments

`me`	A MoleculeExperiment object.
`assayName`	Character string (default "cell") specifying the existing boundaries assay that should have buffer added.
`...`	Arguments that pass to internal functions. The most relevant parameter is buffer (default 0).

Value

A boundaries assay with essential columns and vertices with added buffer.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
me <- readXenium(repoDir,
  keepCols = "essential"
)
MoleculeExperiment::boundaries(me, "cell_buffer") <- bufferBoundaries(
  me,
  assayName = "cell", buffer = 1
)

library(ggplot2)
ggplot_me() +
```

```
  geom_polygon_me(me, assayName = "cell", fill = "grey") +
  geom_polygon_me(me, assayName = "cell_buffer", fill = NA, colour = "red") +
  geom_point_me(me) +
  coord_cartesian(
    xlim = c(4900, 4919.98),
    ylim = c(6400.02, 6420)
  )
```

`countMolecules` *Count molecules per region of interest (e.g., cell)*

Description

This function takes the information from the molecules and boundaries slot, and counts the molecules per region of interest. Its input is a MoleculeExperiment object, and its output a SpatialExperiment object. That way, if one is interested in doing downstream analyses at the cell level, one can do so.

Usage

```
countMolecules(
  me,
  moleculesAssay = "detected",
  boundariesAssay = "cell",
  buffer = 0,
  matrixOnly = FALSE,
  nCores = 1
)
```

Arguments

`me`	MoleculeExperiment object containing both the transcript data as well as the boundaries data. I.e., the "molecules" and "boundaries" slots need to be filled. See MoleculeExperiment() for more information.
`moleculesAssay`	Character string naming the list of the molecules slot from which transcript information should be retrieved from. The default is the detected transcript data that is read in when creating a MoleculeExperiment object. It is possible to change it to another mode, e.g., "high_threshold" will access the transcript information that has been stored in the "high_threshold" element of the list in the molecules slot.
`boundariesAssay`	Character string naming the list of the boundaries slot from which boundary information should be retrieved from. For example, for counting transcripts per cell, the list containing the cell boundaries (e.g., "cell") should be selected.
`buffer`	Single numeric value (default 0) indicating value to buffer beyond segment boundaries, i.e. to count molecules just outside of a segment boundary

matrixOnly	Logical value indicating whether to return a matrix of the counted molecules per segment (e.g., cell). Is FALSE by default, i.e., the default output is a SpatialExperiment object.
nCores	Number of cores to use for the operation.

Value

A SpatialExperiment object derived from a MoleculeExperiment object. Alternatively, a matrix with the counted molecules per segment.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
me <- readXenium(repoDir,
    keepCols = "essential"
)

spe <- countMolecules(me)
spe
```

---

dataframeToMEList *Convert a transcript (molecule) or boundary dataframe to the ME list format*

Description

The goal of this function is to standardise transcript and boundary files for input to a MoleculeExperiment object.

Usage

```
dataframeToMEList(
  df,
  dfType = NULL,
  assayName = NULL,
  sampleCol = "sample_id",
  factorCol = NULL,
  xCol = "x_location",
  yCol = "y_location",
  keepCols = "essential",
  scaleFactor = 1
)
```

Arguments

df	A data.frame containing the transcript information or the boundary information. NOTE: this dataframe should, at a minimum, have the following 4 columns: sample_id, factorCol (e.g., feature_id in transcripts, or cell_id in boundaries), x_location and y_location.
dfType	Character string specifying contents of the dataframe. Can be either "molecules" or "boundaries".
assayName	Character string specifying the name with which to identify the information later on in an ME object.
sampleCol	Character string specifying the name of the column with the sample id.
factorCol	Character string specifying the name of the column with the factors with which to group the data in the lists. When working with molecules, this column would be e.g., "feature_id" in xenium. When working with boundaries, this column would be e.g., "cell_id" in xenium.
xCol	Character string specifying the name of the column with global x coordinates.
yCol	Character string specifying the name of the column with global y coordinates.
keepCols	Character string which can be either "essential" or "all". If "essential", the function will only work with the x and y location information.
scaleFactor	Integer specifying the scale factor by which to change the scale of the x and y locations (e.g., to change from pixel to micron). The default value is 1.

Value

A list with the format required to input it into slots of a MoleculeExperiment object.

Examples

```
moleculesDf <- data.frame(
    sample_id = rep(c("sample1", "sample2"), times = c(30, 20)),
    features = rep(c("gene1", "gene2"), times = c(20, 30)),
    x_coords = runif(50),
    y_coords = runif(50)
)

moleculesMEList <- dataframeToMEList(moleculesDf,
                                     dfType = "molecules",
                                     assayName = "detected",
                                     sampleCol = "sample_id",
                                     factorCol = "features",
                                     xCol = "x_coords",
                                     yCol = "y_coords")

moleculesMEList
```

`MoleculeExperiment-class` *MoleculeExperiment class: An S4 class container to store imaging-based spatial transcriptomics data.*

Description

This class enables the analysis of imaging-based ST data at the molecule level, and standardises data across vendors. The aim of this class is to facilitate ST data integration and comparison and, importantly, facilitate common analytical and visualisation workflows.

Usage

```
MoleculeExperiment(molecules, boundaries = NULL)
```

Arguments

`molecules`	Detected transcripts information in a standardised ME list format, as is generated by dataframeToMEList() and readMolecules() functions.
`boundaries`	Slot with boundary information in a standardised ME list format, as is generated by dataframeToMEList() and readBoundaries() functions.

Value

A MoleculeExperiment object

Slots

`molecules` Slot containing information about the detected transcripts. This slot is designed as a list of lists, where each sample contains a list of tibbles with information for each gene. The basic information required for this slot are the gene names of the transcripts, as well as their x and y locations.

`boundaries` Slot containing the boundaries defining each segmented cell. The slot is designed as a list of lists, where each sample contains a list of tibbles for each cell, consisting of the x and y coordinates of the polygon vertices defining the cell boundary.

Examples

```
# creating a simple ME object from toy data
moleculesDf <- data.frame(
    sample_id = rep(c("sample1", "sample2"), times = c(30, 20)),
    features = rep(c("gene1", "gene2"), times = c(20, 30)),
    x_coords = runif(50),
    y_coords = runif(50)
)
boundariesDf <- data.frame(
    sample_id = rep(c("sample1", "sample2"), times = c(16, 6)),
    cell_id = rep(c("cell1", "cell2", "cell3", "cell4",
```

```
                    "cell1", "cell2"),
                  times = c(4, 4, 4, 4, 3, 3)),
    vertex_x = rnorm(22),
    vertex_y = rnorm(22)
)
moleculesMEList <- dataframeToMEList(moleculesDf,
                                     dfType = "molecules",
                                     assayName = "detected",
                                     sampleCol = "sample_id",
                                     factorCol = "features",
                                     xCol = "x_coords",
                                     yCol = "y_coords")

boundariesMEList <- dataframeToMEList(boundariesDf,
                                      dfType = "boundaries",
                                      assayName = "cell",
                                      sampleCol = "sample_id",
                                      factorCol = "cell_id",
                                      xCol = "vertex_x",
                                      yCol = "vertex_y")

toyME <- MoleculeExperiment(molecules = moleculesMEList,
                            boundaries = boundariesMEList)

toyME
```

---

plotting-functions *Plotting functions for SpatialUtils*

Description

A set of ggplot functions to build customized plots for imaging based spatial transcriptomics data.

Usage

```
ggplot_me()

geom_point_me(
  me,
  assayName = "detected",
  byColour = NULL,
  selectFeatures = NULL,
  ...
)

geom_polygon_me(me, assayName = "cell", byFill = NULL, ...)

geom_raster_img(
```

```
  path = NULL,
  image = NULL,
  displacement = c(0, 0),
  pixelSize = 1,
  ...
)
```

Arguments

me	MoleculeExperiment object.
assayName	Character string specifying name of assay from which to get data.
byColour	Character string specifying the column name to colour by.
selectFeatures	character vector of features to keep for geom_point_me.
...	Additional parameters to be passed to ggplot.
byFill	Character string specifying the column name to fill by.
path	Path of the image. Default: NULL
image	Image object to be plotted as raster. Default: NULL
displacement	the x-y coordinate of the top-left pixel of the image. Default: c(0, 0)
pixelSize	the pixel size in micron, Default: 1

Value

A plot with transcripts and/or segmentation information for imaging based spatial transcriptomics data.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
me <- readXenium(repoDir,
                 keepCols = "essential",
                 addBoundaries = c("cell", "nucleus"))

g = ggplot_me() +
        geom_polygon_me(me, byFill = "segment_id", colour = "black") +
        geom_point_me(me, byColour = "feature_id", size = 0.1) +
        geom_polygon_me(me, assayName = "nucleus", fill = NA, colour = "red")
g
```

`readBoundaries` *Read in csv boundary information and convert to ME list format.*

Description

This function reads in csv boundary files and converts them to the ME list format, so that they can be added to an ME object later on. To account for different coordinate scales possible being used by the boundary versus transcript information, this function scales the coordinate values of the boundaries to match the unit of the detected transcript locations. The various arguments offer flexibility to standardise data from different molecule-based ST technologies into the ME list format.

Usage

```
readBoundaries(
  dataDir,
  pattern = NULL,
  segmentIDCol = NULL,
  xCol = NULL,
  yCol = NULL,
  keepCols = "essential",
  boundariesAssay = NULL,
  scaleFactorVector = 1
)
```

Arguments

`dataDir`	Path of the directory containing the boundary csv files.
`pattern`	Character string specifying the unique pattern with which to identify the files of interest in the directory. This is useful to work with multiple samples. Defaults to NULL.
`segmentIDCol`	Character string specifying the name of the column containing the segment IDs. Defaults to NULL.
`xCol`	Character string specifying the name of the column containing the x coordinates of the vertices defining the boundaries. Defaults to NULL.
`yCol`	Character string specifying the name of the column containing the y coordinates of the vertices defining the boundaries. Defaults to NULL.
`keepCols`	Character string specifying which columns to keep. Defaults to "essential". The other option is to select "all", or custom columns by specifying their names in a vector.
`boundariesAssay`	Character string specifying the name with which to identify the boundary data in the ME object later on. Defaults to NULL.
`scaleFactorVector`	Vector containing the scale factor/s with which to change the coordinate data from pixel to micron. It can be either a single integer, or multiple scale factors for the different samples. The default value is 1.

Value

An ME list containing the boundary information. This can be used as input to the boundaries slot of an ME object.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
nucleiMEList <- readBoundaries(dataDir = repoDir,
                               pattern = "nucleus_boundaries.csv",
                               segmentIDCol = "cell_id",
                               xCol = "vertex_x",
                               yCol = "vertex_y",
                               keepCols = "essential",
                               boundariesAssay = "nucleus",
                               scaleFactorVector = 1)
nucleiMEList
```

---

readCosmx *Read in Cosmx data (Nanostring) as an ME object.*

Description

This function is a wrapper around the readMolecules function. It can read both molecule and mask information. The segmentation masks are converted to boundaries, and these are added to the boundaries slot of the MoleculeExperiment object.

Usage

```
readCosmx(dataDir, keepCols = "essential", addBoundaries = "cell")
```

Arguments

dataDir	Character string specifying the directory with the Cosmx output files.
keepCols	Character string specifying which columns to keep. Defaults to "essential". The other option is to select "all", or custom columns by specifying their names in a vector.
addBoundaries	A string with which to specify the name of the boundary assay to be added to the me object. Can be a string, or NULL. If NULL, a simple ME object with no boundaries will be created.

Value

A MoleculeExperiment object

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/nanostring_Lung9_Rep1")
#
meCosmx <- readCosmx(repoDir,
    keepCols = "essential"
)
meCosmx
```

`readMerscope` *Read in Merscope data to an ME object*

Description

Reads in Merscope (Vizgen) molecule and boundary data from a directory, and standardises it into a MoleculeExperiment object.

Usage

```
readMerscope(dataDir, keepCols = "essential", addBoundaries = "cell")
```

Arguments

`dataDir`	Character string specifying the directory with the Cosmx output files.
`keepCols`	Vector of characters specifying the columns of interest from the transcripts file. "essential" selects columns with gene names, x and y locations. "all" will select all columns. Alternatively, specific colums of interest can be selected by specifying them as characters in a vector. Note that this personalised vector needs to contain the essential columns.
`addBoundaries`	A string with which to specify the name of the boundary assay to be added to the me object. Can be a string, or NULL. If NULL, a simple ME object with no boundaries will be created.

Value

A MoleculeExperiment object

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/vizgen_HumanOvarianCancerPatient2Slice2")
meMerscope <- readMerscope(repoDir,
    keepCols = "essential",
    addBoundaries = "cell"
)
meMerscope
```

`readMolecules` *Read in detected transcripts file/s into a MoleculeExperiment object*

Description

A function to standardise transcripts.csv files across different molecule- based ST technologies, and store them into an ME object. It is technology agnostic, so it is accompanied with wrappers for the specific technologies (e.g., see readXenium).

Usage

```
readMolecules(
  dataDir,
  pattern = NULL,
  featureCol = NULL,
  xCol = NULL,
  yCol = NULL,
  keepCols = "essential",
  moleculesAssay = NULL,
  scaleFactorVector = 1
)
```

Arguments

`dataDir`	Character string specifying the directory with the file/s containing detected transcripts for different runs/samples.
`pattern`	Character string specifying the pattern with which to find the transcripts files. For example, in Xenium data, the pattern would be "transcripts.csv". In contrast, in Cosmx data, the pattern would be "tx_file".
`featureCol`	Character string specifying the name of the column with feature names. For example, "feature_name" in xenium transcripts.csv files.
`xCol`	Character string specifying the name of the column with the x locations of the transcripts.
`yCol`	Character string specifying the name of the column with the y locations of the transcripts.
`keepCols`	Vector of characters specifying the columns of interest from the transcripts file. "essential" selects columns with gene names, x and y locations. "all" will select all columns. Alternatively, specific colums of interest can be selected by specifying them as characters in a vector. Note that this personalised vector needs to contain the essential columns.
`moleculesAssay`	Character string specifying the name of the list in which the transcript information is going to be stored in the molecules slot. The default name is "detected", as we envision that a MoleculeExperiment will usually be created with raw detected transcript information.

scaleFactorVector
: Vector containing the scale factor/s with which to change the coordinate data from pixel to micron. It can be either a single integer, or multiple scale factors for the different samples. The default value is 1.

Value

A simple MoleculeExperiment object with a filled molecules slot.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
simple_me <- readMolecules(repoDir,
    pattern = "transcripts.csv",
    featureCol = "feature_name",
    xCol = "x_location",
    yCol = "y_location",
    keepCols = "essential"
)
simple_me
```

---

readSegMask *Read a segmentation mask*

Description

Reads a segmentation mask TIFF and transforms it into a ME boundaries object. One must provide either the path or the loaded image object.

Usage

```
readSegMask(
  extent,
  path = NULL,
  image = NULL,
  assayName = "cell",
  background_value = NULL,
  sample_id = NULL
)
```

Arguments

extent	The extent of the loaded segmentation mask in micrometers. Used to align the mask with the transcripts. This must be of the form c(xmin, xmax, ymin, ymax).
path	The path of the segmenation mask, Default: NULL
image	The loaded image object, Default: NULL
assayName	The name of the segmentation (e.g. cell, or nucleus), Default: 'cell'

background_value
: The value corresponding to the background in the segmentation, Default: NULL

sample_id
: What the sample should be named, Default: NULL

Value

A boundaries object.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
segMask <- paste0(repoDir, "/BIDcell_segmask.tif")
data <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain/sample1")
me <- readXenium(data,
    keepCols = "essential",
    addBoundaries = NULL
)
boundaries(me, "BIDcell_segmentation") <- readSegMask(
    # use the molecule extent to define the boundary extent
    extent(me, assayName = "detected"),
    path = segMask, assayName = "BIDcell_segmentation",
    sample_id = "sample1", background_value = 0
)
ggplot_me() +
    geom_polygon_me(
        me,
        assayName = "BIDcell_segmentation", fill = NA, colour = "black"
    ) +
    geom_point_me(me, byColour = "feature_id", size = 0.1) +
    geom_polygon_me(
        me,
        assayName = "BIDcell_segmentation", fill = NA, colour = "red"
    )
```

---

readXenium *Read in Xenium data into a MoleculeExperiment object*

Description

Function to read in, and standardise, Xenium output data into an ME object. Detected transcripts files are required. Additionally, it is also possible to read in boundary files ("cell", "nuclei", or both). This function is a wrapper around readMolecules and readBoundaries functions.

Usage

```
readXenium(dataDir, keepCols = "essential", addBoundaries = "cell")
```

Arguments

dataDir	Character string specifying the directory with the xenium output files.
keepCols	Vector of characters specifying the columns of interest from the transcripts file and boundaries file. Can be "all" or "essential". "essential" selects columns with gene names, x and y locations in the transcripts file; "essential" selects columns with cell ids, and x and y locations for the vertices defining the boundaries in the boundaries file.
addBoundaries	Vector with which to specify the names of the boundary assays to be added to the me object. Can be "cell", "nucleus", both, or NULL. The latter will lead to the creation of a simple ME object with just the molecules slot filled.

Value

A MoleculeExperiment object containing xenium data.

Examples

```
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")

me <- readXenium(repoDir,
                 keepCols = "essential")
me
```

---

small_me *A subsetted Xenium dataset include for demostration purposes.*

Description

A subsetted Xenium dataset include for demostration purposes.

Usage

```
data(small_me)
```

Format

An object of class `MoleculeExperiment` of length 1.

subset_by_extent *Subset functions for MoleculeExperiment objects*

Description

A set of functions to subset MoleculeExperiment objects by different factors

Usage

```
subset_by_extent(me, extent)
```

Arguments

me	MoleculeExperiment object.
extent	The extent in micrometers to subset the me object. This must be of the form c(xmin, xmax, ymin, ymax).

Value

A subsetted MoleculeExperiment object

Examples

```
data(small_me)

subset_extent <- c(xmin = 3000, xman = 4000, ymin = 2000, ymax = 3000)
subset_small_me <- subset_by_extent(small_me, subset_extent)

# check the extent after subsetting
extent(subset_small_me, assayName = "detected")
```

summarization *Summarization methods to get insights into a MoleculeExperiment object*

Description

The following methods are useful to get quick view of the contents in a MoleculeExperiment object. For example, showMolecules and showBoundaries summarise the large nested ME list of lists in the molecules and boundaries slots. nFeatures and nTranscripts get the numbers of features or transcripts, respectively.

Usage

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

## S4 method for signature 'MoleculeExperiment'
showMolecules(object)

## S4 method for signature 'MoleculeExperiment'
showBoundaries(object)

## S4 method for signature 'MoleculeExperiment'
extent(object, assayName = NULL)

## S4 method for signature 'MoleculeExperiment'
nFeatures(object)

## S4 method for signature 'MoleculeExperiment'
nTranscripts(object)
```

Arguments

object	Name of MoleculeExperiment object of interest.
assayName	Character string specifying the name of the assay from which to view a summary of the contents.

Value

A MoleculeExperiment object summary.

Examples

```
# get example data
repoDir <- system.file("extdata", package = "MoleculeExperiment")
repoDir <- paste0(repoDir, "/xenium_V1_FF_Mouse_Brain")
me <- readXenium(repoDir,
    keepCols = "essential",
    addBoundaries = "cell"
)

showMolecules(me)
showBoundaries(me)

nFeatures(me)

nTranscripts(me)
```

Index