Package: MultiAssaySpatialExperiment (via r-universe)

July 22, 2026

Version 0.99.2

Date 2026-07-22

Title Multi-Assay Experiment with Spatial Context

Description Extends MultiAssayExperiment with spatial context elements (images, labels, points, shapes) for integrative analysis of spatially resolved multi-omics data. Supports coercion to and from SpatialExperiment and SpatialFeatureExperiment.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.5.0), methods, MultiAssayExperiment

Imports tools, BiocGenerics, BiocParallel, S4Vectors, MatrixGenerics, SummarizedExperiment, SingleCellExperiment, SpatialExperiment, sf

Suggests knitr, rmarkdown, BiocStyle, testthat, arrow, DropletUtils, jsonlite, sfarrow, terra, ggplot2, RBioFormats, SpatialFeatureExperiment, zellkonverter

biocViews Infrastructure, DataRepresentation

VignetteBuilder knitr

Collate 'SpatialGenerics.R' 'SpatialLayerList-class.R' 'MultiAssaySpatialExperiment-class.R' 'MultiAssaySpatialExperiment-SpatialExperiment.R' 'MultiAssaySpatialExperiment-SpatialFeatureExperiment.R' 'MultiAssaySpatialExperiment-combine.R' 'MultiAssaySpatialExperiment-spatialMap.R' 'MultiAssaySpatialExperiment-helpers.R' 'MultiAssaySpatialExperiment-read-components.R' 'MultiAssaySpatialExperiment-read-config.R' 'MultiAssaySpatialExperiment-read-cosmx.R' 'MultiAssaySpatialExperiment-read-generics.R' 'MultiAssaySpatialExperiment-read-helpers.R' 'MultiAssaySpatialExperiment-read-merscope.R'

'MultiAssaySpatialExperiment-read-visium.R'
'MultiAssaySpatialExperiment-read-visiumhd.R'
'MultiAssaySpatialExperiment-read-xenium.R'
'MultiAssaySpatialExperiment-subset.R' 'spatial-aggregate.R'
'spatial-annotate.R' 'spatial-operations.R'
'spatial-raster-convert.R' 'zzz.R'

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libmagick++-dev gsfonts libicu-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev zlib1g-dev

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

Date/Publication 2026-07-22 15:54:31 UTC

RemoteUrl https://github.com/BiocStaging/MultiAssaySpatialExperiment

RemoteRef HEAD

RemoteSha 1c4601f09049897b3f6ba9294642648059e40368

Contents

aggregateByRegion . . . 3
annotateWithRegions . . . 4
buildSpatialMap . . . 6
checkSpatialMap . . . 7
labelsToShapes . . . 8
MultiAssaySpatialExperiment-class . . . 9
MultiAssaySpatialExperiment-combine . . . 12
MultiAssaySpatialExperiment-SpatialExperiment . . . 13
MultiAssaySpatialExperiment-subset . . . 15
prepMASE . . . 16
readCosMxMASE . . . 18
readCSVForMASE . . . 19
readGeoJSONForMASE . . . 20
readGeoParquetForMASE . . . 20
readHDF5ForMASE . . . 21
readMERSCOPEMASE . . . 22
readParquetForMASE . . . 23
readVisiumHDMASE . . . 24
readVisiumMASE . . . 25
readXeniumMASE . . . 26
shapesToLabels . . . 27
spatial-accessors . . . 28
spatialJoin . . . 30
SpatialLayerList-class . . . 31
spatialMatch . . . 33
spatialOverlaps . . . 34

Index **36**

aggregateByRegion *Aggregate assay values by shape region*

Description

Aggregate assay values (e.g., counts, expression) by shape region. Each assay column is linked to a point via spatialMap; the annotation column (from annotateWithRegions) links points to shapes. Values are aggregated within each shape.

Usage

```
aggregateByRegion(x, by, assays = NULL, FUN = "sum")
```

Arguments

x	A MultiAssaySpatialExperiment.
by	Character. Name of the region column in spatialMap that holds the shape instance IDs (the regionCol from annotateWithRegions, default is the shapes layer name).
assays	Character. Assay names to aggregate (default: all assays in spatialMap).
FUN	Character or function. For character values, one of "sum", "mean", or "count". For "count", returns the number of points per shape; for "sum" and "mean", aggregates assay values. When a function, it is applied to each feature-by-observation submatrix (features as rows, observations in the same shape as columns) and must return a numeric vector of length nrow(submatrix).

Details

Run annotateWithRegions first to add the by column to spatialMap. Rows with NA in by are dropped.

For "sum" and "mean", the assay matrix (features x columns) is grouped by shape: columns that map to the same shape are aggregated. The result is a list of matrices (one per assay), each with features as rows and shape instance IDs as columns.

Value

A list of matrices (or for FUN = "count" a single DataFrame), one element per assay. Each matrix has features (rows) and shape instance IDs (columns).

Polymorphism

Uses S3/S4 generics only. Table-like elements (spatialMap, experiments) use nrow, colnames, rownames, [[, [, unique, and split. Assay data use assay for SummarizedExperiment-like objects; plain matrices are passed through. Layer lists use names. Consistent with annotateWithRegions and subsetByPolygon; see vignette *Working with MultiAssaySpatialExperiment*.

Author(s)

Patrick Aboyoun

Examples

```
pts <- S4Vectors::DataFrame(
    x = c(1.5, 2.5, 3.5),
    y = c(1.5, 2.5, 3.5),
    instance_id = c("A", "B", "C"))
shp_df <- S4Vectors::DataFrame(
    instance_id = c("cell1", "cell2", "cell3"),
    geometry = sf::st_sfc(
        sf::st_polygon(list(matrix(c(1,1,2,1,2,2,1,2,1,1), ncol=2, byrow=TRUE))),
        sf::st_polygon(list(matrix(c(2,1,3,1,3,2,2,2,2,1), ncol=2, byrow=TRUE))),
        sf::st_polygon(list(matrix(c(2,2,3,2,3,3,2,3,2,2), ncol=2, byrow=TRUE)))))
mase <- MultiAssaySpatialExperiment(
    experiments = ExperimentList(assay1 = matrix(1:9, 3, 3,
        dimnames = list(paste0("G", 1:3), c("A", "B", "C")))),
    colData = S4Vectors::DataFrame(row.names = "s1"),
    sampleMap = S4Vectors::DataFrame(
        assay = "assay1", primary = "s1", colname = c("A", "B", "C")),
    points = PointsLayerList(centroids = pts),
    shapes = ShapesLayerList(cells = shp_df),
    spatialMap = S4Vectors::DataFrame(
        assay = "assay1", colname = c("A", "B", "C"),
        element_type = "points", region = "centroids", instance_id = c("A", "B", "C")))
mase <- annotateWithRegions(mase, points = "centroids", shapes = "cells")
agg <- aggregateByRegion(mase, by = "cells", FUN = "sum")
agg[["assay1"]]
```

`annotateWithRegions` *Annotate points with shape regions*

Description

Perform a spatial join between a points layer and a shapes layer. For each point, the matched shape's `instance_id` is recorded. This annotation is added to `spatialMap` and enables `aggregateByRegion`.

Usage

```
annotateWithRegions(
  x,
  points,
  shapes,
  spatialCoordsNames = c("x", "y"),
  regionCol = NULL,
  join = st_intersects
)
```

Arguments

x	A MultiAssaySpatialExperiment.
points	Character. Name of the points layer (in spatialPoints(x)).
shapes	Character. Name of the shapes layer (in spatialShapes(x)).
spatialCoordsNames	Character vector. Names of the x and y coordinate columns in the points layer (default c("x", "y")).
regionCol	Character or NULL. Name of the column to add to spatialMap with the shape instance IDs. Default is the shapes name.
join	Function. Spatial join to use, matching st_join. Default st_intersects (point-in-polygon). Use st_nearest_feature to assign each point to the nearest shape when no shape contains it.

Details

For each row in the points layer, join determines which shape (if any) matches. The shape's instance_id is written to a new column in spatialMap. Only spatialMap rows that reference the given points layer are updated; other rows get NA for the new column.

With join = st_intersects (default), points not falling within any shape receive NA. With join = st_nearest_feature, every point is assigned to its nearest shape. The user is responsible for ensuring instance_id types match between points and shapes (no coercion).

Value

x with spatialMap updated to include the new annotation column. If spatialMap is NULL or the required layers are missing, x is returned unchanged.

Polymorphism

Uses S3/S4 generics only. Table-like elements (spatialMap, points, shapes) use nrow, colnames, [[, and [. Spatial operations use the supplied join function and st_as_sfc. Layer lists use names. Consistent with subsetByPolygon; see vignette *Working with MultiAssaySpatialExperiment*.

Author(s)

Patrick Aboyoun

Examples

```
pts <- S4Vectors::DataFrame(
    x = c(1.5, 2.5, 3.5),
    y = c(1.5, 2.5, 3.5),
    instance_id = c("A", "B", "C"))
shp_df <- S4Vectors::DataFrame(
    instance_id = c("cell1", "cell2", "cell3"),
    geometry = sf::st_sfc(
        sf::st_polygon(list(matrix(c(1,1,2,1,2,2,1,2,1,1), ncol=2, byrow=TRUE))),
        sf::st_polygon(list(matrix(c(2,1,3,1,3,2,2,2,2,1), ncol=2, byrow=TRUE))),
```

```
        sf::st_polygon(list(matrix(c(2,2,3,2,3,3,2,3,2,2), ncol=2, byrow=TRUE)))))
mase <- MultiAssaySpatialExperiment(
    experiments = ExperimentList(assay1 = matrix(1:9, 3, 3,
        dimnames = list(NULL, c("A", "B", "C")))),
    colData = S4Vectors::DataFrame(row.names = "s1"),
    sampleMap = S4Vectors::DataFrame(
        assay = "assay1", primary = "s1", colname = c("A", "B", "C")),
    points = PointsLayerList(centroids = pts),
    shapes = ShapesLayerList(cells = shp_df),
    spatialMap = S4Vectors::DataFrame(
        assay = "assay1", colname = c("A", "B", "C"),
        element_type = "points", region = "centroids", instance_id = c("A", "B", "C")))
mase <- annotateWithRegions(mase, points = "centroids", shapes = "cells")
spatialMap(mase)
```

`buildSpatialMap` *Build a spatialMap table from sampleMap*

Description

Assemble a `spatialMap` DataFrame linking assay observations to spatial layer instances, analogous to `listToMap` for `sampleMap`.

Usage

```
buildSpatialMap(
  sampleMap,
  region,
  element_type = c("points", "shapes"),
  assays = NULL,
  instance_from = "colname"
)
```

Arguments

`sampleMap`	A DataFrame with MAE columns `assay`, `primary`, and `colname`.
`region`	Character. Name of the points or shapes layer (must match a name in `spatialPoints` or `spatialShapes` when the object is constructed).
`element_type`	Character. `"points"` or `"shapes"`.
`assays`	Optional character vector limiting which assays from `sampleMap` are included.
`instance_from`	Character. Column of `sampleMap` used for `instance_id` values (default `"colname"`, i.e. observation id equals spatial instance id).

Value

A DataFrame with columns `assay`, `colname`, `element_type`, `region`, and `instance_id`.

See Also

prepMASE, listToMap

Examples

```
sm <- S4Vectors::DataFrame(
    assay = factor(rep("rna", 3), "rna"),
    primary = c("S1", "S1", "S2"),
    colname = paste0("spot", 1:3))
buildSpatialMap(sm, region = "tissue1", element_type = "points")
```

---

checkSpatialMap *Check spatialMap consistency*

Description

Run spatialMap diagnostics without constructing a MultiAssaySpatialExperiment. Returns a character vector of problems (empty if valid).

Usage

```
checkSpatialMap(
  spatialMap,
  sampleMap,
  points = PointsLayerList(),
  shapes = ShapesLayerList()
)
```

Arguments

spatialMap	A spatialMap DataFrame or NULL.
sampleMap	A MAE sampleMap DataFrame.
points	A PointsLayerList or named list of point layers.
shapes	A ShapesLayerList or named list of shape layers.

Value

Character vector of error messages (length zero if no problems).

Examples

```
sm <- S4Vectors::DataFrame(
    assay = factor("rna", "rna"),
    primary = "S1",
    colname = "spot1")
spmap <- buildSpatialMap(sm, "coords", "points")
spmap[["instance_id"]] <- "missing"
checkSpatialMap(spmap, sm, points = PointsLayerList(
    coords = S4Vectors::DataFrame(
        x = 1, y = 1, instance_id = "spot1")))
```

---

labelsToShapes *Convert label rasters to polygon shapes*

Description

Dissolve contiguous cells with the same label value into polygons. Thin wrapper around `terra::as.polygons` for segmentation masks stored in `spatialLabels()` or vendor TIFFs.

Usage

```
labelsToShapes(x, dissolve = TRUE, ...)
```

Arguments

`x`	A `SpatRaster`, 2-D matrix/array, or a single element from a RasterLayerList.
`dissolve`	Logical; dissolve adjacent cells with the same value (default `TRUE`).
`...`	Additional arguments passed to `terra::as.polygons`.

Value

An **sf** object with dissolved label polygons.

See Also

`shapesToLabels`, `spatialLabels`

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
    mat <- matrix(c(1L, 1L, 2L, 2L), 2, 2)
    labelsToShapes(mat)
}
```

`MultiAssaySpatialExperiment-class`
MultiAssaySpatialExperiment objects

Description

The MultiAssaySpatialExperiment class extends MultiAssayExperiment with spatial context: raster data (images, labels), points, and shapes.

Details

MultiAssaySpatialExperiment provides slots for spatial layers that complement the `ExperimentList`. The `sampleMap` links experiment columns to primary identifiers; the optional `spatialMap` provides instance-level mapping (`assay`, `colname`, `element_type`, `region`, `instance_id`) linking experiment columns to rows in the spatial point or shape layers via explicit foreign key.

Terminology: a *specimen* is a row in `colData` (`sampleMap$primary`); an *observation* is a column in an assay (`sampleMap$colname`).

Value

The `MultiAssaySpatialExperiment()` constructor returns a MultiAssaySpatialExperiment. The layer accessors return the corresponding container: `spatialImages()` / `spatialLabels()` a RasterLayerList, `spatialPoints()` a PointsLayerList, and `spatialShapes()` a ShapesLayerList; the single-layer accessors (`spatialImage()`, `spatialPoint()`, and so on) return one layer, and the `*Names()` accessors a character vector. `imgData()` and `spatialMap()` return a `DataFrame` or `NULL`. Replacement methods (the `<-` forms) return the updated object, and `show()` returns `NULL` invisibly.

Slots

`ExperimentList`, `colData`, `sampleMap`, `drops` Inherited from MultiAssayExperiment.

`images` A RasterLayerList of user-attached image rasters (not the same as `imgData`, which stores specimen-level image metadata from vendor readers).

`labels` A RasterLayerList of label/mask layers.

`points` A PointsLayerList of point layers.

`shapes` A ShapesLayerList of shape layers.

`imgData` Optional `DataFrame` linking specimens to images (same structure as `SpatialExperiment`, including `sample_id`, `image_id`, and `scaleFactor`). Populated by vendor readers; distinct from `spatialImages()` raster layers.

`spatialMap` Optional `DataFrame` for instance-level mapping with required columns:

- `assay` Experiment name (factor; must be in `names(experiments)`)
- `colname` Column identifier within that experiment
- `element_type` Spatial slot discriminator; one of `"points"` or `"shapes"`
- `region` Layer name within that element type (e.g., `"cells"`, `"nuclei"`)
- `instance_id` Row identifier in that layer (character or integer; no NAs)

The element_type column disambiguates spatial layers with the same name in different slots (e.g., points$cells vs. shapes$cells). This explicit routing eliminates ambiguity and enables robust foreign key validation.

Terminology and spatialdata alignment

MASE adopts terminology from Python spatialdata where practical while following Bioconductor S4 conventions:

- element_type: Matches spatialdata's element type discriminator (values: "points", "shapes", "images", "labels"). Currently only "points" and "shapes" are supported in spatialMap linkage; images and labels are reference layers accessed via spatialImages() and spatialLabels().
- region: Corresponds to spatialdata's region (layer name within an element type).
- instance_id: Matches spatialdata's instance identifier (row ID within a region). Recommended types: integer, character, or factor. Must be unique within each (element_type, region) combination.
- Class naming: MASE uses "Layer" suffix (PointsLayerList, ShapesLayerList) rather than "Element" to avoid R naming collisions. See ?SpatialLayerList for details.
- Accessor naming: MASE uses spatialPoints(), spatialShapes() (with "spatial" prefix) as S4 generics, while spatialdata uses bare attribute access (sdata.points, sdata.shapes). This follows Bioconductor conventions and avoids conflicts with base R functions.

Constructor

MultiAssaySpatialExperiment(experiments = ExperimentList(), colData = DataFrame(), sampleMap = DataFram
Creates a MultiAssaySpatialExperiment object.

experiments An ExperimentList or list of experiments.

colData A DataFrame of specimen-level metadata (one row per primary identifier in sampleMap$primary).

sampleMap A DataFrame with columns assay, primary, colname.

metadata, drops Additional MultiAssayExperiment arguments.

images, labels RasterLayerList of user-attached raster layers (distinct from imgData, which vendor readers use for image metadata).

points, shapes PointsLayerList and ShapesLayerList of spatial layers.

imgData Optional DataFrame linking specimens to images (populated by vendor readers; use getImg() for SPE-compatible access).

spatialMap Optional DataFrame for instance-level mapping.

Accessors

In the code snippets below, x is a MultiAssaySpatialExperiment object:

spatialImages(x), spatialImages(x) <- value: Get or set the full RasterLayerList of image layers.

spatialLabels(x), spatialLabels(x) <- value: Get or set the full RasterLayerList of label layers.

spatialPoints(x), spatialPoints(x) <- value: Get or set the full PointsLayerList of point layers.

`spatialShapes(x)`, `spatialShapes(x) <- value`: Get or set the full ShapesLayerList of shape layers.

`spatialImage(x, i)`, `spatialImage(x, i) <- value`: Get or set a single image layer by index `i` (character or integer).

`spatialLabel(x, i)`, `spatialLabel(x, i) <- value`: Get or set a single label layer by index `i`.

`spatialPoint(x, i)`, `spatialPoint(x, i) <- value`: Get or set a single point layer by index `i`.

`spatialShape(x, i)`, `spatialShape(x, i) <- value`: Get or set a single shape layer by index `i`.

`spatialImageNames(x)`, `spatialLabelNames(x)`, `spatialPointNames(x)`, `spatialShapeNames(x)`: Get the names of the image, label, point, and shape layers, respectively.

`imgData(x)`, `imgData(x) <- value`: Get or set the optional `DataFrame` linking specimens to images (`SpatialExperiment`-compatible columns such as `sample_id`, `image_id`, and `scaleFactor`).

`spatialMap(x)`, `spatialMap(x) <- value`: Get or set the optional `DataFrame` for instance-level mapping.

Displaying

The `show()` method prints the inherited MultiAssayExperiment summary followed by counts of spatial images, labels, points, shapes, and whether `imgData` and `spatialMap` are present.

Author(s)

Patrick Aboyoun

See Also

- MultiAssaySpatialExperiment-subset for subsetting methods
- MultiAssaySpatialExperiment-combine for combining methods
- RasterLayerList, PointsLayerList, ShapesLayerList for spatial layer containers

Examples

```
mat <- matrix(rnorm(20), 5, 4,
    dimnames = list(paste0("G", 1:5), paste0("obs", 1:4)))
pts <- S4Vectors::DataFrame(
    x = 1:4, y = 1:4, instance_id = paste0("obs", 1:4))
mase <- MultiAssaySpatialExperiment(
    experiments = ExperimentList(rna = mat),
    colData = S4Vectors::DataFrame(
        tissue = c("core", "margin"),
        row.names = c("specimen_A", "specimen_B")),
    sampleMap = S4Vectors::DataFrame(
        assay = factor(rep("rna", 4), "rna"),
        primary = rep(c("specimen_A", "specimen_B"), each = 2),
        colname = paste0("obs", 1:4)),
    points = PointsLayerList(coords = pts),
    spatialMap = S4Vectors::DataFrame(
        assay = factor(rep("rna", 4), "rna"),
        colname = paste0("obs", 1:4),
```

```
        element_type = "points",
        region = "coords",
        instance_id = paste0("obs", 1:4))
)
mase
spatialPointNames(mase)
mase[, "specimen_A"]
```

`MultiAssaySpatialExperiment-combine`
MultiAssaySpatialExperiment combining

Description

Methods for combining MultiAssaySpatialExperiment objects. `c` merges two MASE objects (adding assays); `cbind` combines along columns (adding specimens); `complete.cases` identifies specimens with data in all assays.

Value

`c()` and `cbind()` return a combined MultiAssaySpatialExperiment; `complete.cases()` returns a logical vector flagging specimens (primaries) present in all assays.

Combining

`c(x, ..., sampleMap = NULL, mapFrom = NULL)`**:** When `...` is a single MultiAssaySpatialExperiment, merges the two objects (experiments, colData, sampleMap, spatial layers, imgData, spatialMap). Experiment and spatial layer names must be unique across objects. Otherwise delegates to `c,MultiAssayExperiment-method`.

`cbind(..., deparse.level = 1)`**:** Combines MASE objects column-wise (same assay names, adds specimens). Spatial layers with the same name are row-bound; `imgData` and `spatialMap` are row-bound.

`complete.cases(x)`**:** Returns a logical vector indicating which specimens (primaries) have data in all assays. Delegates to `complete.cases,MultiAssayExperiment-method`.

Author(s)

Patrick Aboyoun

See Also

- MultiAssaySpatialExperiment for the main class
- MultiAssaySpatialExperiment-subset for subsetting methods

Examples

```
mat1 <- matrix(rnorm(20), 5, 4,
    dimnames = list(paste0("G", 1:5), paste0("S", 1:4)))
mat2 <- matrix(rnorm(20), 5, 4,
    dimnames = list(paste0("G", 1:5), paste0("S", 1:4)))
sm1 <- S4Vectors::DataFrame(
    assay = factor(rep("assay1", 4), "assay1"),
    primary = paste0("S", 1:4),
    colname = paste0("S", 1:4))
sm2 <- S4Vectors::DataFrame(
    assay = factor(rep("assay2", 4), "assay2"),
    primary = paste0("S", 1:4),
    colname = paste0("S", 1:4))
cd <- S4Vectors::DataFrame(row.names = paste0("S", 1:4))
mase1 <- MultiAssaySpatialExperiment(
    ExperimentList(assay1 = mat1), cd, sm1)
mase2 <- MultiAssaySpatialExperiment(
    ExperimentList(assay2 = mat2), cd, sm2)
c(mase1, mase2)
complete.cases(mase1)
```

MultiAssaySpatialExperiment-SpatialExperiment

MultiAssaySpatialExperiment - SpatialExperiment integration

Description

Coercion between SpatialExperiment and MultiAssaySpatialExperiment, plus methods for `spatialCoords`, `imgData`, `scaleFactors`, `getImg`, `addImg`, `rmvImg`, and related SpatialExperiment-derived functionality.

Value

Coercion returns the requested object: a MultiAssaySpatialExperiment from a SpatialExperiment, or a SpatialExperiment from a MASE. `spatialCoords()` returns a numeric coordinate matrix and `scaleFactors()` a numeric vector. The image methods return image data (`getImg()`) or an updated MASE for the modifying methods (such as `addImg()`). `molecules()` returns the molecule data of the first assay, or `NULL`.

Coercion

`as(spe, "MultiAssaySpatialExperiment")`**:** Wraps a SpatialExperiment in MASE with a single assay. `spatialCoords` are copied to points; `imgData` is passed through.

`as(mase, "SpatialExperiment")`**:** Requires exactly one assay that is or can be coerced to SpatialExperiment.

spatialCoords and scaleFactors

`spatialCoords(x, assay = 1L)`: Returns spatial coordinates from the assay (if a SpatialExperiment) or from `spatialPoints(x)`.

`spatialCoordsNames(x)`: Returns coordinate column names from the first assay or points layer.

`scaleFactors(x, sample_id = TRUE, image_id = TRUE)`: Returns scale factors from `imgData`.

imgData methods

`getImg(x, sample_id = NULL, image_id = NULL)`: Retrieves image(s) from `imgData`.

`addImg(x, imageSource, scaleFactor, sample_id, image_id, load = TRUE)`: Adds an image to `imgData`.

`rmvImg(x, sample_id = NULL, image_id = NULL)`: Removes image(s) from `imgData`.

`imgSource(x, sample_id = NULL, image_id = NULL, path = FALSE)`: Returns image source path(s).

`imgRaster(x, sample_id = NULL, image_id = NULL)`: Returns raster image(s).

`rotateImg(x, sample_id = NULL, image_id = NULL, degrees = 90)`: Rotates image(s) in place.

`mirrorImg(x, sample_id = NULL, image_id = NULL, axis = c("h", "v"))`: Mirrors image(s) in place.

molecules

`molecules(x, ...)`: Returns molecule data from the first assay if it is a SpatialExperiment with a `molecules` assay; otherwise `NULL`.

Author(s)

Patrick Aboyoun

See Also

- MultiAssaySpatialExperiment for the main class
- SpatialExperiment for single-assay spatial data

Examples

```
mat <- matrix(1:12, 3, 4,
    dimnames = list(paste0("G", 1:3), paste0("cell", 1:4)))
spe <- SpatialExperiment::SpatialExperiment(
    assays = list(counts = mat),
    spatialCoords = matrix(c(1:4, 1:4), 4, 2))
mase <- as(spe, "MultiAssaySpatialExperiment")
names(mase)
spatialCoords(mase)
as(mase, "SpatialExperiment")
```

MultiAssaySpatialExperiment-subset
MultiAssaySpatialExperiment subsetting

Description

Methods for subsetting a MultiAssaySpatialExperiment. They extend `subsetBy` with propagation to spatial slots.

Value

`subsetByColData()` and `subsetByRow()` return a MultiAssaySpatialExperiment restricted to the selected specimens or rows; the linked spatial layers, `imgData`, and `spatialMap` are subset accordingly.

Subsetting

In the snippets below, `x` is a MultiAssaySpatialExperiment:

`subsetByColData(x, y)`**:** Subset by primary identifiers (specimens). `spatialMap`, `imgData`, and linked points, shapes, images, and labels are filtered to retained specimens (via `sampleMap` and `spatialMap`).

`subsetByRow(x, y, i = TRUE, ...)`**:** Subset rows of experiments. Spatial layers are assay-level and unchanged.

`subsetByColumn(x, y)`**:** Subset assay columns (observations). When `y` is a `list` (one element per assay), each element may be column names, indices, or a logical vector — the usual `subsetByColumn` interface. `spatialMap`, `imgData`, and linked points, shapes, images, and labels are filtered to retained columns. To subset by assay `colData`, build a logical vector or column names and pass a list, e.g. `subsetByColumn(mase, list(rna = colData(experiments(mase)[["rna"]])$tiss == "tumor"))`.

`subsetByAssay(x, y)`**:** Subset assays. Points, shapes, images, labels, `spatialMap`, and `imgData` are filtered to layers and rows linked to retained assays via `spatialMap`. Auxiliary shape layers not referenced in `spatialMap` are dropped.

`x[i, j, k, ..., drop = FALSE]`**:** Subset by row (`i`), column (`j`), and assay (`k`) indices. Equivalent to composing `subsetByColData`, `subsetByColumn`, `subsetByRow`, and `subsetByAssay`. If `drop = TRUE`, empty assays are removed.

`subsetByBoundingBox(x, xmin, xmax, ymin, ymax, ...)`**:** Subset by bounding box. Filters points and shapes within the rectangle; propagates to assays via `spatialMap` when present. When `crop_rasters = TRUE` (default), `spatialImages` and `spatialLabels` are cropped to the same bounding box (single-scale, pixel coordinates for matrix rasters; **terra** extent for `SpatRaster` layers).

`subsetByPolygon(x, polygon, ...)`**:** Subset by polygon. Filters points and shapes within or intersecting the `sf` geometry; propagates to assays via `spatialMap` when present. Raster layers are cropped to the polygon bounding box when `crop_rasters = TRUE`.

Polymorphism

All subsetting uses S3/S4 generics only. Table-like elements (`spatialMap`, `sampleMap`, `imgData`, points, shapes) use `nrow`, `colnames`, `[[`, `[`, `unique`, and `split` (from **BiocGenerics**). Layer lists use `names` and `length`. Spatial filtering additionally requires `sf::st_intersects` (and `st_as_sfc`, `st_bbox` for shapes / polygon). For spatial filtering, `instance_id` in points, shapes, and `spatialMap` must have matching types (no coercion). See vignette *Working with MultiAssaySpatialExperiment* for a per-operation propagation table.

Author(s)

Patrick Aboyoun

See Also

- MultiAssaySpatialExperiment for the main class
- MultiAssaySpatialExperiment-combine for combining methods

Examples

```
mat <- matrix(rnorm(20), 5, 4,
    dimnames = list(paste0("G", 1:5), paste0("obs", 1:4)))
pts <- S4Vectors::DataFrame(
    x = 1:4, y = 1:4, instance_id = paste0("obs", 1:4))
sm <- S4Vectors::DataFrame(
    assay = factor(rep("rna", 4), "rna"),
    primary = rep(c("specimen_A", "specimen_B"), each = 2),
    colname = paste0("obs", 1:4))
mase <- MultiAssaySpatialExperiment(
    experiments = ExperimentList(rna = mat),
    colData = S4Vectors::DataFrame(row.names = c("specimen_A", "specimen_B")),
    sampleMap = sm,
    points = PointsLayerList(coords = pts),
    spatialMap = buildSpatialMap(sm, "coords", "points"))
subsetByColData(mase, "specimen_A")
subsetByBoundingBox(mase, xmin = 1.5, xmax = 3.5, ymin = 1.5, ymax = 3.5)
```

---

prepMASE *Prepare components for a MultiAssaySpatialExperiment*

Description

Wraps `prepMultiAssay` and harmonizes spatial slots. Returns a `list` suitable for `MultiAssaySpatialExperiment` or `do.call`.

Usage

```
prepMASE(
  ExperimentList,
  colData,
  sampleMap,
  points = PointsLayerList(),
  shapes = ShapesLayerList(),
  images = RasterLayerList(),
  labels = RasterLayerList(),
  imgData = NULL,
  spatialMap = NULL,
  ...
)
```

Arguments

ExperimentList	A ExperimentList or named list of assays passed to prepMultiAssay.
colData	A DataFrame of specimen-level metadata (one row per sampleMap$primary), passed to prepMultiAssay.
sampleMap	A DataFrame with MAE columns assay, primary, and colname, passed to prepMultiAssay.
points	A PointsLayerList or named list of point layers.
shapes	A ShapesLayerList or named list of shape layers.
images	A RasterLayerList of image layers (default empty).
labels	A RasterLayerList of label layers (default empty).
imgData	Optional specimen-level image metadata DataFrame.
spatialMap	Optional instance-level mapping DataFrame.
...	Additional arguments passed to prepMultiAssay (metadata, drops).

Value

A list with elements experiments, colData, sampleMap, metadata, drops, and spatial slot components.

See Also

buildSpatialMap, prepMultiAssay

Examples

```
mat <- matrix(1:6, 2, 3, dimnames = list(c("G1", "G2"), paste0("spot", 1:3)))
sm <- S4Vectors::DataFrame(
    assay = factor(rep("rna", 3), "rna"),
    primary = c("S1", "S1", "S2"),
    colname = paste0("spot", 1:3))
pts <- S4Vectors::DataFrame(
    x = 1:3, y = 1:3, instance_id = paste0("spot", 1:3))
```

```
prepared <- prepMASE(
    ExperimentList(rna = mat),
    S4Vectors::DataFrame(row.names = c("S1", "S2")),
    sm,
    points = PointsLayerList(tissue1 = pts),
    spatialMap = buildSpatialMap(sm, "tissue1", "points"))
do.call(MultiAssaySpatialExperiment, prepared)
```

`readCosMxMASE` *Read NanoString CosMx SMI Spatial Data*

Description

Load a NanoString CosMx output directory into a MultiAssaySpatialExperiment.

Usage

```
readCosMxMASE(
  data_dir,
  sample_id = NULL,
  fov_ids = NULL,
  load_transcripts = FALSE,
  images = TRUE,
  load_images = FALSE,
  min_area = NULL
)
```

Arguments

`data_dir`	Character. Path to CosMx output directory.
`sample_id`	Character. Optional sample identifier (default: directory name).
`fov_ids`	Character vector or NULL. FOV IDs to load (default: NULL loads all).
`load_transcripts`	Logical. Whether to load transcript data (default: FALSE).
`images`	Logical. Whether to load image metadata (default: TRUE).
`load_images`	Logical. Whether to load image data (default: FALSE).
`min_area`	Numeric. Minimum polygon area for filtering (default: NULL).

Details

Supports multi-FOV datasets. Use `fov_ids` to load specific fields of view, or `NULL` to load all FOVs into one object. Transcript coordinates are optional (`load_transcripts`).

Value

A MultiAssaySpatialExperiment object.

See Also

readMERSCOPEMASE, readXeniumMASE

Examples

```
## A minimal mock CosMx output directory is bundled for a runnable example;
## in practice, point 'data_dir' at a real CosMx output directory.
dir <- system.file("extdata", "cosmx_mock",
                   package = "MultiAssaySpatialExperiment")
mase <- readCosMxMASE(dir, images = FALSE, load_transcripts = FALSE)
names(mase)
```

---

readCSVForMASE *Read CSV Files for MASE*

Description

Read a CSV file into a DataFrame for MASE component assembly.

Usage

```
readCSVForMASE(file_path, ...)

## S4 method for signature 'character'
readCSVForMASE(file_path, ...)
```

Arguments

file_path	Character path to CSV file
...	Additional arguments passed to format-specific methods

Value

DataFrame

Examples

```
tmp <- tempfile(fileext = ".csv")
df <- data.frame(
    cell_id = paste0("C", 1:3),
    x_centroid = 1:3,
    y_centroid = 1:3)
write.csv(df, tmp, row.names = FALSE)
readCSVForMASE(tmp)
unlink(tmp)
```

readGeoJSONForMASE *Read GeoJSON Files for MASE*

Description

Read a GeoJSON file via **sf** for shape layers in MASE readers.

Usage

```
readGeoJSONForMASE(file_path, ...)

## S4 method for signature 'character'
readGeoJSONForMASE(file_path, quiet = TRUE, ...)
```

Arguments

file_path	Character path to GeoJSON file
...	Additional arguments passed to format-specific methods
quiet	Logical, suppress sf reading messages

Value

sf or equivalent object

Examples

```
tmp <- tempfile(fileext = ".geojson")
pt <- sf::st_point(c(10, 20))
sf_obj <- sf::st_sf(id = "C1", geometry = sf::st_sfc(pt))
sf::st_write(sf_obj, tmp, quiet = TRUE, delete_dsn = TRUE)
readGeoJSONForMASE(tmp, quiet = TRUE)
unlink(tmp)
```

readGeoParquetForMASE *Read GeoParquet Files for MASE*

Description

Read a GeoParquet file via **sfarrow** for geometry layers in MASE readers.

Usage

```
readGeoParquetForMASE(file_path, ...)

## S4 method for signature 'character'
readGeoParquetForMASE(file_path, ...)
```

Arguments

file_path	Character path to GeoParquet file
...	Additional arguments passed to format-specific methods

Value

sf object or equivalent object

Examples

```
if (requireNamespace("sfarrow", quietly = TRUE) &&
    requireNamespace("sf", quietly = TRUE)) {
    tmp <- tempfile(fileext = ".parquet")
    pt <- sf::st_sfc(sf::st_point(c(1, 2)))
    sf_obj <- sf::st_sf(id = 1L, geometry = pt)
    sfarrow::st_write_parquet(sf_obj, tmp)
    readGeoParquetForMASE(tmp)
    unlink(tmp)
}
```

---

readHDF5ForMASE *Read HDF5 Files for MASE*

Description

Read an HDF5 matrix file (10x or AnnData/h5ad) into a SummarizedExperiment for MASE assembly.

Usage

```
readHDF5ForMASE(file_path, type = c("10x", "anndata"), ...)

## S4 method for signature 'character,character'
readHDF5ForMASE(file_path, type = c("10x", "anndata"), ...)
```

Arguments

file_path	Character path to HDF5 file
type	Character: "10x" for 10x HDF5, "anndata" for h5ad
...	Additional arguments passed to format-specific methods

Value

SummarizedExperiment

Examples

```
## Not run:
## Requires DropletUtils and a 10x filtered_feature_bc_matrix.h5 file
readHDF5ForMASE("filtered_feature_bc_matrix.h5", type = "10x")

## End(Not run)
```

`readMERSCOPEMASE` *Read Vizgen MERSCOPE Spatial Data*

Description

Load a Vizgen MERSCOPE (MERFISH) output directory into a MultiAssaySpatialExperiment.

Usage

```
readMERSCOPEMASE(
  data_dir,
  sample_id = NULL,
  fov_ids = NULL,
  segmentation = c("cellpose", "watershed", "both"),
  load_transcripts = FALSE,
  images = TRUE,
  load_images = FALSE,
  min_area = NULL,
  min_qv = 20
)
```

Arguments

`data_dir`	Character. Path to MERSCOPE output directory.
`sample_id`	Character. Optional sample identifier (default: directory name).
`fov_ids`	Character vector or NULL. FOV IDs to load (default: NULL loads all).
`segmentation`	Character. Segmentation method: "cellpose", "watershed", or "both" (default: "cellpose").
`load_transcripts`	Logical. Whether to load transcript data (default: FALSE).
`images`	Logical. Whether to load image metadata (default: TRUE).
`load_images`	Logical. Whether to load image data (default: FALSE).
`min_area`	Numeric. Minimum polygon area for filtering (default: NULL).
`min_qv`	Numeric. Minimum quality value for transcripts (default: 20).

Details

Supports multi-FOV datasets and multiple segmentation methods (`segmentation`: `"cellpose"`, `"watershed"`, or `"both"`). Transcript coordinates are optional (`load_transcripts`).

Value

A MultiAssaySpatialExperiment object.

See Also

`readCosMxMASE`, `readXeniumMASE`

Examples

```
## A minimal mock MERSCOPE output directory is bundled for a runnable
## example; in practice, point 'data_dir' at a real MERSCOPE region directory.
dir <- system.file("extdata", "merscope_mock",
                   package = "MultiAssaySpatialExperiment")
mase <- readMERSCOPEMASE(dir, segmentation = "cellpose",
                         images = FALSE, load_transcripts = FALSE)
names(mase)
```

`readParquetForMASE` *Read Parquet Files for MASE*

Description

Read a Parquet file into a DataFrame for use in MASE component assembly or custom readers.

Usage

```
readParquetForMASE(file_path, ...)

## S4 method for signature 'character'
readParquetForMASE(file_path, col_select = NULL, ...)
```

Arguments

`file_path`	Character path to Parquet file
`...`	Additional arguments passed to format-specific methods
`col_select`	Optional character vector of column names to select

Value

DataFrame

Examples

```
if (requireNamespace("arrow", quietly = TRUE)) {
    tmp <- tempfile(fileext = ".parquet")
    df <- data.frame(cell_id = paste0("C", 1:3), x = 1:3, y = 1:3)
    arrow::write_parquet(df, tmp)
    readParquetForMASE(tmp)
    unlink(tmp)
}
```

`readVisiumHDMASE` *Read 10x Genomics Visium HD Spatial Data*

Description

Load a 10x Genomics Visium HD Space Ranger output directory into a MultiAssaySpatialExperiment.

Usage

```
readVisiumHDMASE(
  data_dir,
  sample_id = NULL,
  bin_size = c("008", "002", "016"),
  load_boundaries = FALSE,
  images = TRUE,
  load_images = FALSE,
  min_area = NULL
)
```

Arguments

`data_dir`	Character. Path to Space Ranger output directory.
`sample_id`	Character. Optional sample identifier (default: directory name).
`bin_size`	Character vector. Bin sizes to load: "002", "008", "016" (default: "008").
`load_boundaries`	Logical. Whether to load cell segmentation boundaries (default: FALSE).
`images`	Logical. Whether to load image metadata (default: TRUE).
`load_images`	Logical. Whether to load image data (default: FALSE).
`min_area`	Numeric. Minimum polygon area for filtering (default: NULL).

Details

Visium HD provides binned expression at 2 µm, 8 µm, and 16 µm resolution (codes `"002"`, `"008"`, `"016"`). Each requested bin size becomes a separate assay in the returned object.

Value

A MultiAssaySpatialExperiment with one experiment per requested bin size.

See Also

readVisiumMASE, readXeniumMASE

Examples

```
## Not run:
## Requires a Visium HD Space Ranger output directory
mase <- readVisiumHDMASE("path/to/visium_hd", bin_size = "008")
names(mase)

## End(Not run)
```

---

readVisiumMASE *Read 10x Genomics Visium Spatial Data*

Description

Load a 10x Genomics Visium Space Ranger output directory into a MultiAssaySpatialExperiment.

Usage

```
readVisiumMASE(
  data_dir,
  sample_id = NULL,
  type = c("HDF5", "sparse"),
  data = c("filtered", "raw"),
  images = TRUE,
  load_images = FALSE,
  unit = c("pixel", "micron"),
  min_area = NULL,
  block_size = NULL
)
```

Arguments

data_dir	Character. Path to Space Ranger output directory.
sample_id	Character. Optional sample identifier (default: directory name).
type	Character. Matrix format: "HDF5" or "sparse" (default: "HDF5").
data	Character. Data type: "filtered" or "raw" (default: "filtered").
images	Logical. Whether to load image metadata (default: TRUE).
load_images	Logical. Whether to load image data (default: FALSE).

unit	Character. Coordinate units: "pixel" or "micron" (default: "pixel").
min_area	Numeric. Minimum polygon area for filtering (default: NULL).
block_size	Numeric. Block size for chunked reading (default: 100MB).

Details

Reads spot-by-gene counts, tissue positions, spot geometries, and optional H&E image metadata (images). Coordinates may be returned in pixels or microns (unit). For Visium HD data, use readVisiumHDMASE.

Value

A MultiAssaySpatialExperiment object.

See Also

readVisiumHDMASE, readXeniumMASE

Examples

```
## Not run:
## Requires a Space Ranger output directory (see package tests for mock layout)
mase <- readVisiumMASE("path/to/visium")
names(mase)

## End(Not run)
```

---

readXeniumMASE *Read 10x Genomics Xenium Spatial Data*

Description

Load a 10x Genomics Xenium output directory into a MultiAssaySpatialExperiment.

Usage

```
readXeniumMASE(
  data_dir,
  sample_id = NULL,
  segmentations = c("cell", "nucleus", "both"),
  images = TRUE,
  load_images = FALSE,
  add_transcripts = FALSE,
  min_area = NULL,
  min_phred = 20,
  block_size = NULL
)
```

Arguments

data_dir	Character. Path to Xenium output directory.
sample_id	Character. Optional sample identifier (default: directory name).
segmentations	Character. Segmentation type: "cell", "nucleus", or "both" (default: "cell").
images	Logical. Whether to load image metadata (default: TRUE).
load_images	Logical. Whether to load image data (default: FALSE).
add_transcripts	Logical. Whether to include transcript data (default: FALSE).
min_area	Numeric. Minimum polygon area for filtering (default: NULL).
min_phred	Integer. Minimum phred score for transcripts (default: 20).
block_size	Numeric. Block size for chunked reading (default: 100MB).

Details

Reads cell-by-gene counts, cell coordinates, optional segmentation polygons (segmentations), and optionally transcript-level coordinates (add_transcripts). Image metadata is included when images = TRUE; set load_images = TRUE to load raster data.

Value

A MultiAssaySpatialExperiment object with expression, coordinates, optional shapes, and spatialMap linkage.

See Also

readVisiumMASE, readVisiumHDMASE, readCosMxMASE, readMERSCOPEMASE

Examples

```
## Not run:
## Requires a Xenium output directory (see package tests for mock layout)
mase <- readXeniumMASE("path/to/xenium")
names(mase)

## End(Not run)
```

---

shapesToLabels *Rasterize polygon shapes to a label image*

Description

Burn shape geometries into a label raster. When template is NULL, an empty raster is created from the bounding box of x.

Usage

```
shapesToLabels(x, template = NULL, field = "instance_id", background = NA, ...)
```

Arguments

x	An **sf** object or data.frame with polygon geometries.
template	Optional SpatRaster defining extent and resolution.
field	Character; attribute column used as label values (default "instance_id").
background	Numeric background value (default NA).
...	Additional arguments passed to terra::rasterize.

Value

A SpatRaster label image.

See Also

labelsToShapes, spatialShapes

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
    mat <- matrix(c(1L, 1L, 2L, 2L), 2, 2)
    polys <- labelsToShapes(mat)
    polys$instance_id <- polys[["lyr.1"]]
    shapesToLabels(polys, field = "instance_id", background = 0L)
}
```

---

spatial-accessors *Access spatial element slots*

Description

Generic functions to access spatialImages, spatialLabels, spatialPoints, spatialShapes, and spatialMap slots of MultiAssaySpatialExperiment. The imgData generic is imported from **SpatialExperiment**.

Usage

```
spatialImages(x)

spatialLabels(x)

spatialPoints(x)

spatialShapes(x)
```

```
spatialMap(x)

spatialImages(x) <- value

spatialLabels(x) <- value

spatialPoints(x) <- value

spatialShapes(x) <- value

spatialMap(x) <- value

spatialImage(x, i)

spatialLabel(x, i)

spatialPoint(x, i)

spatialShape(x, i)

spatialImageNames(x)

spatialLabelNames(x)

spatialPointNames(x)

spatialShapeNames(x)

spatialImage(x, i) <- value

spatialLabel(x, i) <- value

spatialPoint(x, i) <- value

spatialShape(x, i) <- value
```

Arguments

x	A MultiAssaySpatialExperiment object
value	Replacement value for the element
i	Index (character or integer) for the element name

Details

Slot accessors (`spatialImages`, `spatialLabels`, etc.) return the full element list. Single-element accessors (`spatialImage`, `spatialLabel`, etc.) take an index `i` (character or integer) and return the element. Name accessors return character vectors of element names. Replacement functions for slots take a full list; single- element replacements take an index and the new value.

For single-element accessors, a character index with no matching name (e.g., `spatialImage(x, "nonexistent")`) returns `NULL`. Integer indices that are out of bounds raise an error.

Value

For accessors: the slot or element contents. For replacement functions: the modified object.

Author(s)

Patrick Aboyoun

See Also

MultiAssaySpatialExperiment, RasterLayerList, PointsLayerList, ShapesLayerList

Examples

```
mat <- matrix(rnorm(20), 5, 4, dimnames = list(paste0("G", 1:5), paste0("obs", 1:4)))
expList <- ExperimentList(rna = mat)
cd <- S4Vectors::DataFrame(sample = "specimen_1", row.names = "specimen_1")
sm <- S4Vectors::DataFrame(
    assay = factor(rep("rna", 4), "rna"),
    primary = rep("specimen_1", 4),
    colname = paste0("obs", 1:4))
mase <- MultiAssaySpatialExperiment(
    experiments = expList,
    colData = cd,
    sampleMap = sm,
    images = RasterLayerList(brightfield = matrix(1:12, 3, 4))
)
spatialImageNames(mase)
spatialImage(mase, 1L)
spatialImage(mase, "brightfield")
spatialImages(mase) <- RasterLayerList(fluorescent = matrix(20:31, 3, 4))
spatialImageNames(mase)
```

---

spatialJoin *Spatial join between spatial layer tables*

Description

Join two spatial `DataFrame` layers on a spatial predicate. The `DataFrame` method uses **sf** attribute joins via `st_join`.

Usage

```
spatialJoin(x, y, join = st_intersects, sparse = TRUE, ...)
```

Arguments

x	A DataFrame with spatial data.
y	A spatial table to join against.
join	Spatial predicate function (default st_intersects).
sparse	Logical; when TRUE, request a sparse pair-table result where supported by methods (default TRUE for the generic signature; the DataFrame method materializes via st_join).
...	Additional arguments for methods.

Value

Join result. The DataFrame method returns an sf object from st_join.

Author(s)

Patrick Aboyoun

See Also

spatialMatch for first-match indexing without attaching attributes.

Examples

```
library(sf)
pts <- S4Vectors::DataFrame(
    spot = c("A", "B"),
    geometry = c("POINT (1 1)", "POINT (5 5)"))
cells <- S4Vectors::DataFrame(
    cell = c("C1", "C2"),
    geometry = c(
        "POLYGON ((0 0, 3 0, 3 3, 0 3, 0 0))",
        "POLYGON ((4 4, 6 4, 6 6, 4 6, 4 4))"))
spatialJoin(pts, cells)
```

SpatialLayerList-class

Spatial layer list objects

Description

The SpatialLayerList class is a virtual base extending SimpleList. Concrete subclasses RasterLayerList, PointsLayerList, and ShapesLayerList provide typed containers for raster data (images, labels), point coordinates, and shape geometries.

Details

All layers must be named and have unique names. Element-type validity is enforced by each concrete class: RasterLayerList requires `dim()` with length >= 2 (e.g., `matrix`, `array`, `DelayedArray`, `StoredSpatialImage`); PointsLayerList requires `DataFrame` elements with at least 2 columns (typically x, y) and often an instance identifier; ShapesLayerList requires `DataFrame` elements (with **sf** a geometry column may be present).

Value

The constructors `RasterLayerList()`, `PointsLayerList()`, and `ShapesLayerList()` return an object of the corresponding class. Coercion via `as()` returns the requested `*LayerList`, and the `show()` method returns `NULL` invisibly.

Constructors

`RasterLayerList(...)`**:** Creates a RasterLayerList. Accepts named arguments, a single named list, or a single List. Each element must have `dim()` with length >= 2.

`PointsLayerList(...)`**:** Creates a PointsLayerList. Accepts named `DataFrame` elements or a single list/List. Each element must have at least 2 columns.

`ShapesLayerList(...)`**:** Creates a ShapesLayerList. Accepts named `DataFrame` or `data.frame` elements (coerced to `DataFrame`); or a single list/List.

Coercion

`as(x, "RasterLayerList")`, `as(x, "PointsLayerList")`, `as(x, "ShapesLayerList")`**:** Coerce `list` or List to the target class. For `ShapesLayerList`, `data.frame` elements are converted to `DataFrame`.

Displaying

The `show()` method prints the class name, length, and for each layer: index, name, element class, and dimensions (raster) or row x col (points/shapes).

Note

Terminology alignment with spatialdata:

MASE uses the "Layer" suffix (`PointsLayerList`, `ShapesLayerList`, `RasterLayerList`) rather than "Element" to avoid naming collisions with R base classes and Bioconductor conventions. The Python spatialdata package uses "element" as an internal discriminator (e.g., `element_type` column, `sdata.points`, `sdata.shapes`) but does not define List container classes.

MASE's `element_type` column in `spatialMap` corresponds to spatialdata's element type discriminator, routing to the appropriate spatial slot: `"points"` → `PointsLayerList`, `"shapes"` → `ShapesLayerList`. See `?MultiAssaySpatialExperiment` for details on the `spatialMap` linkage model.

Other intentional differences: MASE uses `spatialPoints()`, `spatialShapes()`, etc. (with "spatial" prefix) as S4 generic accessors, while spatialdata uses bare attribute access (`sdata.points`, `sdata.shapes`). This follows Bioconductor S4 conventions and avoids conflicts with base R functions.

Author(s)

Patrick Aboyoun

See Also

- MultiAssaySpatialExperiment for usage in multi-assay context
- SimpleList for the base class

Examples

```
RasterLayerList()
RasterLayerList(img = matrix(1:12, 3, 4))

pts <- S4Vectors::DataFrame(x = c(1, 2), y = c(3, 4))
PointsLayerList(coords = pts)

ShapesLayerList()
df <- S4Vectors::DataFrame(id = 1:2, value = c("a", "b"))
ShapesLayerList(regions = df)
```

---

spatialMatch *Spatial match for DataFrame*

Description

For each row of `x`, find the first matching row in `table` based on a spatial relationship. Analogous to `match(x, table)` from **S4Vectors**: returns an integer vector of positions in `table`.

Arguments

`x`	A `DataFrame` with coordinate columns representing point locations.
`table`	A `DataFrame` with a geometry column representing spatial regions.
`...`	Additional arguments passed to methods. Common arguments: `coords` Character vector of length 2 naming the coordinate columns in `x`. `geom` Character; name of the geometry column in `table` (default `"geometry"`). `join` A spatial join function (default `sf::st_intersects`). Must accept two geometry arguments and return either an sgbp list or an integer vector.

Details

The `DataFrame` method builds point geometries from the coordinate columns of `x`, applies the `join` function against the geometry column of `table`, and returns the first match per row.

Downstream packages can define methods for their own `DataFrame` subclasses to push the spatial join to an alternative backend (e.g., SQL-based spatial joins, Arrow compute, in-memory acceleration).

Value

An integer vector of length `nrow(x)`. Each element is the row position in `table` of the first spatial match, or `NA` if no match is found.

Author(s)

Patrick Aboyoun

See Also

`spatialOverlaps` for the predicate analogue (like `is.na`).

Examples

```
library(sf)
pts <- S4Vectors::DataFrame(x = c(1, 5, 10), y = c(1, 5, 10))
shapes <- S4Vectors::DataFrame(
    geometry = st_as_sfc(c(
        "POLYGON((0 0, 6 0, 6 6, 0 6, 0 0))",
        "POLYGON((7 7, 12 7, 12 12, 7 12, 7 7))")))
spatialMatch(pts, shapes, coords = c("x", "y"))
## 1  1  2
```

`spatialOverlaps` *Spatial overlap predicate for DataFrame*

Description

Test which rows of a `DataFrame` spatially intersect a region. Analogous to `is.na` or `duplicated` from **S4Vectors**: returns a logical vector that can be used for subsetting via `x[spatialOverlaps(x, y, ...), ]`.

Arguments

`x`	A `DataFrame` with spatial data (coordinate columns or a geometry column).
`y`	A spatial region to test against. Typically an `sfc`, `sfg`, or similar geometry object from **sf**.
`...`	Additional arguments passed to methods. Common arguments: `coords` Character vector of length 2 naming the coordinate columns in `x` (e.g., `c("x", "y")`). When provided, point geometries are constructed from these columns and tested for intersection with `y`. `geom` Character; name of the geometry column in `x` (default `"geometry"`). Used when `coords` is not provided.

Details

The DataFrame method uses **sf** functions (st_as_sfc, st_intersects). Downstream packages can define methods for their own DataFrame subclasses to use alternative backends (e.g., SQL databases, Arrow, HDF5).

Two modes are supported based on the arguments:

Coordinate mode When coords is provided, point geometries are constructed from those columns and tested against y.

Geometry mode When coords is NULL (the default), the column named by geom is used directly.

Value

A logical vector of length nrow(x). TRUE for rows whose spatial data intersects y.

Author(s)

Patrick Aboyoun

See Also

spatialMatch for the relational analogue (like match).

Examples

```
library(sf)
pts <- S4Vectors::DataFrame(x = c(1, 5, 10), y = c(1, 5, 10))
polygon <- st_as_sfc("POLYGON((0 0, 6 0, 6 6, 0 6, 0 0))")
spatialOverlaps(pts, polygon, coords = c("x", "y"))
## TRUE  TRUE FALSE
```

Index

∗ **classes**
MultiAssaySpatialExperiment-class, 9
SpatialLayerList-class, 31
∗ **methods**
MultiAssaySpatialExperiment-class, 9
MultiAssaySpatialExperiment-SpatialExperiment, 13
MultiAssaySpatialExperiment-subset, 15
SpatialLayerList-class, 31
[,MultiAssaySpatialExperiment,ANY,ANY,ANY-method *(MultiAssaySpatialExperiment-subset)*, 15

addImg,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13
aggregateByRegion, 3, *4*
aggregateByRegion,MultiAssaySpatialExperiment-method *(aggregateByRegion)*, 3
annotateWithRegions, *3*, 4
annotateWithRegions,MultiAssaySpatialExperiment-method *(annotateWithRegions)*, 4
assay, *3*

buildSpatialMap, 6, *17*

c,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-combine)*, 12
cbind,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-combine)*, 12
checkSpatialMap, 7
coerce,MultiAssaySpatialExperiment,SpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13
coerce,SpatialExperiment,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13

DataFrame, *6, 7*, *10*, *17*, *19*, *23*, *30, 31*, *33, 34*

ExperimentList, *10*, *17*

getImg,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13

imgData,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-class)*, 9
imgData<-,MultiAssaySpatialExperiment,ANY-method *(MultiAssaySpatialExperiment-class)*, 9
imgRaster,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13
imgSource,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13

labelsToShapes, 8, *28*
List, *32*
listToMap, *6, 7*

mirrorImg,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13
molecules,MultiAssaySpatialExperiment-method *(MultiAssaySpatialExperiment-SpatialExperiment)*, 13
MultiAssayExperiment, *9, 10*
MultiAssaySpatialExperiment, *3*, *5*, *7*, *9*, *12–16*, *18*, *22–30*, *33*
MultiAssaySpatialExperiment *(MultiAssaySpatialExperiment-class)*, 9

MultiAssaySpatialExperiment-class, 9
MultiAssaySpatialExperiment-combine, *11*, 12, *16*
MultiAssaySpatialExperiment-SpatialExperiment (MultiAssaySpatialExperiment-SpatialExperiment), 13
MultiAssaySpatialExperiment-subset, *11*, *12*, 15

PointsLayerList, *7*, *9–11*, *17*, *30–32*
PointsLayerList (SpatialLayerList-class), 31
PointsLayerList-class (SpatialLayerList-class), 31
prepMASE, *7*, 16
prepMultiAssay, *16*, *17*

RasterLayerList, *8–11*, *17*, *30–32*
RasterLayerList (SpatialLayerList-class), 31
RasterLayerList-class (SpatialLayerList-class), 31
readCosMxMASE, 18, *23*, *27*
readCSVForMASE, 19
readCSVForMASE,character-method (readCSVForMASE), 19
readGeoJSONForMASE, 20
readGeoJSONForMASE,character-method (readGeoJSONForMASE), 20
readGeoParquetForMASE, 20
readGeoParquetForMASE,character-method (readGeoParquetForMASE), 20
readHDF5ForMASE, 21
readHDF5ForMASE,character,character-method (readHDF5ForMASE), 21
readMERSCOPEMASE, *19*, 22, *27*
readParquetForMASE, 23
readParquetForMASE,character-method (readParquetForMASE), 23
readVisiumHDMASE, 24, *26*, *27*
readVisiumMASE, 25, *25*, *27*
readXeniumMASE, *19*, *23*, *25*, 26, *26*
rmvImg,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-SpatialExperiment), 13
rotateImg,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-SpatialExperiment), 13

scaleFactors,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-SpatialExperiment), 13
ShapesLayerList, *7*, *9–11*, *17*, *30–32*
ShapesLayerList (SpatialLayerList-class), 31
ShapesLayerList-class (SpatialLayerList-class), 31
shapesToLabels, *8*, 27
show,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
show,PointsLayerList-method (SpatialLayerList-class), 31
show,RasterLayerList-method (SpatialLayerList-class), 31
show,ShapesLayerList-method (SpatialLayerList-class), 31
SimpleList, *31*, *33*
spatial-accessors, 28
spatialCoords,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-SpatialExperiment), 13
spatialCoordsNames,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-SpatialExperiment), 13
SpatialExperiment, *13*, *14*
spatialImage (spatial-accessors), 28
spatialImage,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialImage<- (spatial-accessors), 28
spatialImage<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialImageNames (spatial-accessors), 28
spatialImageNames,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialImages (spatial-accessors), 28
spatialImages,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialImages<- (spatial-accessors), 28
spatialImages<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialJoin, 30

spatialJoin,DataFrame,DataFrame-method (spatialJoin), 30
spatialLabel (spatial-accessors), 28
spatialLabel,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialLabel<- (spatial-accessors), 28
spatialLabel<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialLabelNames (spatial-accessors), 28
spatialLabelNames,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialLabels, *8*
spatialLabels (spatial-accessors), 28
spatialLabels,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialLabels<- (spatial-accessors), 28
spatialLabels<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
SpatialLayerList, *31*
SpatialLayerList-class, 31
spatialMap (spatial-accessors), 28
spatialMap,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialMap<- (spatial-accessors), 28
spatialMap<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialMatch, *31*, 33, *35*
spatialMatch,DataFrame,DataFrame-method (spatialMatch), 33
spatialOverlaps, 34, *34*
spatialOverlaps,DataFrame-method (spatialOverlaps), 34
spatialPoint (spatial-accessors), 28
spatialPoint,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialPoint<- (spatial-accessors), 28
spatialPoint<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialPointNames (spatial-accessors), 28
spatialPointNames,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialPoints (spatial-accessors), 28
spatialPoints,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialPoints<- (spatial-accessors), 28
spatialPoints<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialShape (spatial-accessors), 28
spatialShape,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialShape<- (spatial-accessors), 28
spatialShape<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialShapeNames (spatial-accessors), 28
spatialShapeNames,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialShapes, *28*
spatialShapes (spatial-accessors), 28
spatialShapes,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
spatialShapes<- (spatial-accessors), 28
spatialShapes<-,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-class), 9
st_as_sfc, *5*
st_intersects, *5*, *31*
st_join, *5*, *30*
st_nearest_feature, *5*
subsetBy, *15*
subsetByAssay,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-subset), 15
subsetByBoundingBox (MultiAssaySpatialExperiment-subset), 15
subsetByBoundingBox,MultiAssaySpatialExperiment-method (MultiAssaySpatialExperiment-subset),

15
subsetByColData,MultiAssaySpatialExperiment,ANY-method
(MultiAssaySpatialExperiment-subset), 15
subsetByColData,MultiAssaySpatialExperiment,character-method
(MultiAssaySpatialExperiment-subset), 15
subsetByColumn, *15*
subsetByColumn,MultiAssaySpatialExperiment,ANY-method
(MultiAssaySpatialExperiment-subset), 15
subsetByPolygon, *3*, *5*
subsetByPolygon
(MultiAssaySpatialExperiment-subset), 15
subsetByPolygon,MultiAssaySpatialExperiment-method
(MultiAssaySpatialExperiment-subset), 15
subsetByRow,MultiAssaySpatialExperiment,ANY-method
(MultiAssaySpatialExperiment-subset), 15
subsetByRow,MultiAssaySpatialExperiment,List-method
(MultiAssaySpatialExperiment-subset), 15
subsetByRow,MultiAssaySpatialExperiment,list-method
(MultiAssaySpatialExperiment-subset), 15
SummarizedExperiment, *21*

terra::as.polygons, *8*