# Package: quantMSImageR (via r-universe)

November 25, 2026

**Title** Processing and Quantification of Targeted Mass Spectrometry Imaging Data

**Version** 1.0.0

**Description** Implements tools for processing and quantifying targeted DESI-MRM mass spectrometry imaging (MSI) datasets, extending the Cardinal package. Includes signal-to-noise filtering against background pixels, tissue/background separation, per-feature ion images, quantile heatmaps, quantification against on-slide or on-tissue calibration standards, and batch export of per-feature text images for external viewers.

**License** MIT + file LICENSE

**URL** https://mjs-708.github.io/quantMSImageR, https://github.com/MJS-708/quantMSImageR

**BugReports** https://github.com/MJS-708/quantMSImageR/issues

**biocViews** Software, MassSpectrometry, ImagingMassSpectrometry, Metabolomics, Lipidomics, QualityControl, Normalization, Visualization

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**Depends** R (>= 4.5.0), Cardinal (>= 3.6.2)

**Imports** methods, stats, utils, grDevices, grid, chemCal (>= 0.2.3), viridis (>= 0.6.5), ComplexHeatmap (>= 2.20.0), circlize (>= 0.4.15), pracma (>= 2.4.0), rmarkdown (>= 2.20), purrr (>= 1.0.0), CardinalIO (>= 1.0.0), ProtGenerics (>= 1.34.0), dplyr (>= 1.1.4), ggplot2 (>= 4.0.0), tibble (>= 3.2.1), tidyr (>= 1.3.1), patchwork (>= 1.2.0), yaml (>= 2.3.0)

**Suggests** testthat (>= 3.0.0), knitr (>= 1.40), BiocStyle (>= 2.28.0), rstudioapi (>= 0.15.0), DT (>= 0.33), htmltools (>= 0.5.0), magick (>= 2.8.4), openxlsx (>= 4.2.0), matter (>= 2.0.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Collate** 'setClasses.R' 'accessors.R' 'alignFeatures.R' 'applySNR.R' 'back2NA.R' 'bindPanels.R' 'blockCentre.R' 'buildFeatureMeta.R' 'combineMSIs.R' 'contributionHm.R' 'createCalCurve.R' 'createMSIDataMatrix.R' 'generateTxtImages.R' 'imageR.R' 'int2SNR.R' 'int2conc.R' 'int2response.R' 'join_ion_library.R' 'labelROIs.R' 'palettes.R' 'plotCalCoverage.R' 'quantMSImageR-package.R' 'quantileHm.R' 'readMRM.R' 'removeBlankMzs.R' 'runExample.R' 'runStudy.R' 'selectTissuePixels.R' 'stitchAcquisitions.R' 'summariseCalLevels.R' 'trimMSI.R' 'validate.R' 'zero2NA.R'

**Config/roxygen2/version** 8.1.1

**Config/pak/sysreqs** cmake make libicu-dev libpng-dev libuv1-dev perl

**Repository** Bioconductor 3.24

**Date/Publication** 2026-11-23 22:56:44 UTC

**RemoteUrl** https://github.com/bioc/quantMSImageR

**RemoteRef** RELEASE_3_24

**RemoteSha** 13269b8f129c6b7ea4e3e062daa88a8695479463

# Contents

alignFeatures . . . 3
applySNR,quant_MSImagingExperiment-method . . . 5
back2NA,quant_MSImagingExperiment-method . . . 6
bindPanels . . . 7
blockCentre . . . 9
buildFeatureMeta . . . 10
calibrationInfo-class . . . 12
combineMSIs,MSImagingExperiment-method . . . 13
contributionHm . . . 14
createCalCurve,quant_MSImagingExperiment-method . . . 18
createMSIDataMatrix,quant_MSImagingExperiment-method . . . 20
generateTxtImages . . . 21
imageR,quant_MSImagingExperiment-method . . . 25
int2conc,quant_MSImagingExperiment-method . . . 27
int2response,quant_MSImagingExperiment-method . . . 28
int2SNR,quant_MSImagingExperiment-method . . . 31
labelROIs . . . 33
plotCalCoverage,quant_MSImagingExperiment-method . . . 35
print.quantValidation . . . 37
quant_MSImagingExperiment . . . 37
quantileHm . . . 39
quantMSImageR-accessors . . . 42
quantPalettes . . . 44
readMRM . . . 45
removeBlankMzs,quant_MSImagingExperiment-method . . . 48
runExample . . . 48

runStudy . . . . . 50
selectTissuePixels . . . . . 51
stitchAcquisitions . . . . . 53
summariseCalLevels,quant_MSImagingExperiment-method . . . . . 55
tissueInfo-class . . . . . 56
trimMSI . . . . . 57
validateConfig . . . . . 58
zero2NA,quant_MSImagingExperiment-method . . . . . 59

**Index** **61**

---

`alignFeatures` *Align two MSI objects to their common features*

---

## Description

Cardinal's `cbind` (used by `combineMSIs`) requires the `mz` key column to match exactly across objects. When acquisitions come from different instrument methods or ion-library versions, features with the same display name can carry different `mz` values, causing `cbind` to fail with a "non-matching key columns" error.

## Usage

```
alignFeatures(
  obj1,
  obj2,
  feature_match = c("transition", "name"),
  mz_tolerance = 0.4
)
```

## Arguments

| | |
|---|---|
| `obj1` | An `MSImagingExperiment` – a `quant_MSImagingExperiment` is one. Its feature set and `mz` values are used as the reference, and its `fData()` must carry a `name` column. |
| `obj2` | An `MSImagingExperiment` to align against `obj1`, likewise carrying `fData()$name`. |
| `feature_match` | Character. `"transition"` (default) verifies that same-named features share a precursor and product m/z; `"name"` matches on the display name alone, without checking what it refers to. |
| `mz_tolerance` | Numeric. Half-width in Da within which two precursor or product m/z values are taken to be the same (default `0.4`, i.e. nominal mass). MRM selects Q1 and Q3 at unit resolution, so two panels can record the same channel as 308.17 and 308.2; at a tighter tolerance this check would reject them as different transitions. See readMRM. |

## Details

`alignFeatures` resolves this by:

1. Subsetting both objects to the *intersection* of feature names.
2. Reordering `obj2` to match `obj1`'s feature order.
3. Forcing `obj2`'s `mz` values to equal `obj1`'s, which permits Cardinal's key check to pass after features have been matched by name. `combineMSIs` then restores `fData` from `obj1`, so all feature metadata originates from the first object.

Features are paired by display name, and by default that pairing is then *verified* against the transition each name refers to: precursor and product m/z must agree to within `mz_tolerance`. Two methods can reuse a name while differing in precursor ion, product ion or transition definition, and because this function rewrites the `mz` key, an unverified pairing would silently merge different measurements into one feature. A disagreement is therefore an error naming the features involved.

`feature_match = "name"` restores name-only matching for objects that carry no precursor/product metadata. It is not the default: it cannot detect the failure above, and this function is called automatically inside `generateTxtImages`.

The function is called automatically inside `generateTxtImages` before every cross-sample `combineMSIs`, so mixed-method studies (e.g. 1 Hz + 2 Hz acquisitions) are handled transparently. It is also exported so that it can be used directly in custom scripts or Rmd reports whenever two `quant_MSImagingExperiment` objects need to be combined.

## Value

A named list with two elements:

`obj1` Subset of `obj1` containing only common features.

`obj2` Subset of `obj2` reordered and `mz`-keyed to match `obj1`, ready for `combineMSIs`.

## See Also

`combineMSIs`, `generateTxtImages`

Other combining acquisitions: `bindPanels()`, `combineMSIs,MSImagingExperiment-method`, `stitchAcquisitions()`

## Examples

```
p1 <- system.file("extdata", "example.raw", "section01.RDS",
                  package = "quantMSImageR")
p2 <- system.file("extdata", "example.raw", "section02.RDS",
                  package = "quantMSImageR")
al <- alignFeatures(readRDS(p1), readRDS(p2))
```

`applySNR,quant_MSImagingExperiment-method`
*Apply SNR mask to intensity values*

## Description

Sets pixels to `NA` in `val_slot` wherever the `snr` spectra slot is `NA`. Typically called after `int2SNR()`: pixels that did not pass the SNR threshold (stored as `NA` in the `snr` slot) are suppressed in the intensity slot so they are excluded from downstream analysis and visualisation.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
applySNR(MSIobject, val_slot = "intensity", ...)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` object containing both `val_slot` and an `snr` spectra slot (populated by `int2SNR()`). |
| `val_slot` | Character. Name of the intensity slot to mask (default `"intensity"`). |
| `...` | Additional arguments (currently unused). |

## Value

The input object with sub-threshold pixels set to `NA` in `val_slot`.

## Destructive

This modifies `val_slot` in place rather than adding a new layer. Keep a copy of the object if the pre-masking values are needed afterwards.

## See Also

`int2SNR()`

Other filtering: `back2NA,quant_MSImagingExperiment-method`, `int2SNR,quant_MSImagingExperiment-method`, `int2response,quant_MSImagingExperiment-method`, `zero2NA,quant_MSImagingExperiment-method`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- int2SNR(obj, snr_thresh = 3)
obj <- applySNR(obj, val_slot = "intensity")
```

back2NA,quant_MSImagingExperiment-method
*Set background pixel intensities to NA*

## Description

Replaces the spectral values of background pixels with NA in a chosen spectra slot. Tissue pixels are left unchanged.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
back2NA(
  MSIobject,
  val_slot = "intensity",
  pixel_header = "sample_name",
  background = "background_pixels",
  sample_type = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment object whose pData() contains a column identifying pixel type. |
| val_slot | Character. Name of the spectra slot to modify (default "intensity"). |
| pixel_header | Character. Column of pData() holding the pixel-type labels (default "sample_name", the imaging convention). |
| background | Character. Value in pixel_header that labels background pixels (default "background_pixels"). The historical "noise_pixels" / "Noise" labels are matched too, so masks made with earlier versions keep working. |
| sample_type | Deprecated alias for pixel_header, kept for backward compatibility. When supplied it overrides pixel_header. |
| ... | Additional arguments (currently unused). |

## Value

The input quant_MSImagingExperiment with background pixels set to NA in val_slot. Returns the object unchanged if no background pixels are found.

## Destructive

This modifies val_slot in place rather than adding a new layer. Keep a copy of the object if the original values are needed afterwards.

## See Also

Other filtering: applySNR,quant_MSImagingExperiment-method, int2SNR,quant_MSImagingExperiment-method, int2response,quant_MSImagingExperiment-method, zero2NA,quant_MSImagingExperiment-method

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- back2NA(obj)
```

---

`bindPanels` *Merge two MSI objects of the same tissue by coordinate-matched rbind*

---

## Description

Combines two `MSImagingExperiment` objects acquired on the *same physical sample* into one whose feature set spans both inputs. Pixels are matched by `(x, y)` coordinates rather than by row order, so the two acquisitions can have different pixel counts – a raster acquired at a faster scan rate can, for instance, lose its last column.

## Usage

```
bindPanels(
  obj1,
  obj2,
  label = NULL,
  feature_match = c("transition", "name"),
  mz_tolerance = 0.4
)
```

## Arguments

| | |
|---|---|
| `obj1`, `obj2` | `MSImagingExperiment` (or `quant_MSImagingExperiment`) objects of the same physical tissue. Both must carry `x` and `y` in `pData(.)`. |
| `label` | Character. Optional. If supplied, written to `pData(result)$run` so the merged object combines cleanly downstream. |
| `feature_match` | Character. How a feature name appearing in both inputs is treated: `"transition"` (default) keeps `obj1`'s copy only after confirming both refer to the same precursor and product m/z, and errors otherwise; `"name"` keeps `obj1`'s copy on the strength of the name alone. |
| `mz_tolerance` | Numeric. Half-width in Da within which two precursor or product m/z values count as the same (default `0.4`, i.e. nominal mass – MRM selects Q1 and Q3 at unit resolution). See `readMRM()`. |

## Details

The common use is pairing a positive- and a negative-mode acquisition of the same section, but any two panels of the same physical area work.

## Value

A `quant_MSImagingExperiment` with:

- Features = obj1's features followed by obj2's features (feature names duplicated in both are kept once, with obj1's values).
- Pixels = intersection of (x, y) grids.
- Each pixel carries real intensities from both inputs.
- Every spectra layer both inputs carry – `response`, `snr` and calibrated amounts as well as `intensity` – together with `obj1`'s experiment metadata and calibration/tissue slots.

## Which pixels are matched

The `(x, y)` coordinates are the pixel indices `readMRM()` assigns, counted from 1 within each acquisition; the absolute stage position is not kept. Matching on them is only correct when both rasters start at the same stage position with the same pixel size. Two acquisitions that start elsewhere, or whose sample was moved between them, still match index for index and are merged without warning onto pixels that are not the same place on the tissue. Check that the two stage programs agree before merging.

Pixels present in only one input are dropped. Pixels present in both keep real intensities from both inputs, but they remain two separate acquisitions: taken at different times, the later one possibly on a surface the first had already desorbed, and aligned only as well as the stage allows. That supports comparing tissue regions across panels. Pixel-level colocalisation between panels carries all of those effects with it – treat it with caution and prefer features from a single acquisition. The report section can be switched off with `output: colocalisation: False` in the `runStudy()` config.

## Processed inputs

Both inputs must carry the same set of spectra layers, since a layer present on only one side cannot be filled in for the other half of the features. Bind panels at the same stage of processing: either both raw, or both after the same steps.

## See Also

`combineMSIs()`, `generateTxtImages()`

Other combining acquisitions: `alignFeatures()`, `combineMSIs,MSImagingExperiment-method`, `stitchAcquisitions()`

## Examples

```
# Both inputs must be the SAME physical area measured twice, so the example
# splits one section into two disjoint "panels" rather than using two
# different sections -- binding unrelated sections by coordinate would
# silently pair pixels that are not the same piece of tissue.
```

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- readRDS(p)
panel_a <- obj[1:4, ]
panel_b <- obj[5:nrow(fData(obj)), ]

merged <- bindPanels(panel_a, panel_b, label = "SampleA_1")
nrow(fData(merged))   # features from both panels
```

---

`blockCentre` *Remove a blocking factor's offset from an MSI object*

---

## Description

Divides every pixel of a feature by a value computed within its own block – the patient, donor, sex or slide the sample belongs to – so that samples from different blocks can be compared without the block's own level getting in the way.

## Usage

```
blockCentre(
  MSIobject,
  block,
  mode = c("center", "baseline", "off"),
  groups = NULL,
  baseline = NULL,
  val_slot = "intensity",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `MSImagingExperiment` (or `quant_MSImagingExperiment`). |
| `block` | Character/factor vector, one entry per PIXEL, naming the block that pixel's sample belongs to. |
| `mode` | `"center"` (default), `"baseline"` or `"off"`. `"off"` returns the object unchanged. |
| `groups` | Character vector, one entry per pixel, giving each pixel's group. Required when `mode = "baseline"`. |
| `baseline` | The value of `groups` that is the reference. Required when `mode = "baseline"`. |
| `val_slot` | Name of the `spectraData` slot to correct (default `"intensity"`). |
| `verbose` | Report what was done (default `TRUE`). |

## Details

The correction is multiplicative on intensity, which is additive in log space: a block-level gain (one patient's sections ionising better, one slide run on a different day) is removed, while genuine differences in how far a treatment moves a feature survive. That survival is the point – a patient who responds twice as strongly as another should still look twice as strong afterwards.

Two references are available. `"center"` divides by the block's own geometric mean, which needs no privileged sample and is steadier because it pools every sample in the block. `"baseline"` divides by the geometric mean of the block's `baseline` group, so each value reads directly as a fold change against its own control; it is more interpretable and noisier, since all of the reference's noise propagates into every other sample of that block. A block with no `baseline` sample falls back to `"center"` for that block alone, and says so.

## Value

`MSIobject` with the chosen slot divided through by its per-feature, per-block reference. Non-positive and missing values are left as they are: they carry no ratio.

## See Also

Other workflow: `generateTxtImages()`, `print.quantValidation()`, `runExample()`, `runStudy()`, `validateConfig()`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- readRDS(p)

# Two blocks, taken across the pixels so the example needs nothing else.
blk <- rep(c("A", "B"), length.out = ncol(obj))
centred <- blockCentre(obj, block = blk, mode = "center")

# The reference is a geometric mean, so that is what lands on 1: each
# feature of each block is now expressed relative to its own level.
ints <- as.matrix(spectraData(centred)[["intensity"]])
geo  <- function(x) exp(mean(log(x[is.finite(x) & x > 0])))
round(apply(ints[, blk == "A"], 1L, geo), 2)
```

---

`buildFeatureMeta` *Build per-feature metadata by joining an ion library on m/z*

---

## Description

Joins each row of `fData(combined)` to the ion library on the (`precursor_mz`, `product_mz`) pair, within `mz_tolerance`. Robust to feature renames in the YAML (`features.rename`) because m/z is stable across name changes.

## Usage

```
buildFeatureMeta(
  combined,
  ion_lib_meta,
  mz_tolerance = 0.4,
  ambiguity = c("combine", "error", "warn", "nearest"),
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| combined | An MSI object whose fData has columns name, precursor_mz and product_mz. Multi-transition features stored with " \|\| "-joined m/z strings are matched on the first token. |
| ion_lib_meta | A data.frame of ion library metadata (must contain precursor_mz and product_mz). Pass NULL to short-circuit and return NULL. |
| mz_tolerance | Numeric. Half-width in Da within which a feature's precursor and product – both, not either – are taken to be the library's. Default 0.4, nominal-mass matching, because MRM selects Q1 and Q3 at unit resolution. See readMRM(). |
| ambiguity | Character. What to do when a feature matches more than one library entry: "combine" (default, name it for all of them joined with " \|\| "), "error", "warn" or "nearest". See readMRM(). |
| verbose | Logical. Emit a message() reporting how many features matched. Default TRUE. |

## Details

Matching uses the same rule as readMRM(), so the two cannot disagree about which library entry a feature is. They previously built their m/z keys independently – one rounding to whole numbers, the other not – and a feature matching several library entries silently took the first one's annotation.

Used by runStudy.R (via inst/runStudy.R) and runExample() to build a per-feature metadata frame the heatmap report can consume – e.g. for splitting / colour-bar annotation by an ion-library column such as Met-1.

## Value

A data.frame of metadata, one row per feature in fData(combined), with row names set to the feature name and a name column appended. Unmatched features become a row of NAs. Returns NULL when ion_lib_meta is NULL.

## See Also

quantileHm(), generateTxtImages()

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- readRDS(p)
lib <- read.csv(system.file("extdata", "example_ion_library.csv",
                            package = "quantMSImageR"), check.names = FALSE)
fm <- buildFeatureMeta(obj, lib)
```

---

`calibrationInfo-class` *Calibration metadata for an imaging experiment*

---

## Description

Holds everything needed to turn a measured response into an estimated amount: the calibration design, the summarised response at each level, the fitted per-analyte models, and their fit quality. It is carried in the `calibrationInfo` slot of a quant_MSImagingExperiment.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | A `calibrationInfo` object. |

## Details

The slots are filled in workflow order and are empty (zero-row data frames, empty list) until the corresponding step has run:

`cal_metadata` `data.frame`. The calibration design supplied by the user: one row per (spot, analyte), with columns `identifier`, `analyte`, `amount_pg` and `level`. Set by `summariseCalLevels()`.

`cal_response_data` `data.frame`. Summarised response per calibration level, with columns `analyte`, `pg_perpixel`, `response_perpixel` and `level`. Set by `summariseCalLevels()`.

`cal_list` `list`. One `stats::lm` per analyte, of `response_perpixel ~ pg_perpixel`, **named by analyte** so that `int2conc()` can match models to features by `fData(x)$name`. Set by `createCalCurve()`.

`r2_df` `data.frame`. One row per analyte, with `feature` and `r2`; `int2conc()` adds an `out_of_range` column giving the proportion of pixels extrapolated beyond the calibrated range. Set by `createCalCurve()`.

Invariants checked by the validity method: `cal_list` must be named if it is non-empty, and every name must appear in `r2_df$feature` when `r2_df` is populated. An unnamed `cal_list` is accepted only when empty – matching models to features positionally would silently apply one analyte's curve to another.

Use the accessors (`calibrationModels()`, `calibrationDiagnostics()`, `calibrationLevels()`, `calibrationMetadata()`) rather than `@`.

### Value

An object of class `calibrationInfo`.

### Slots

`cal_metadata` `data.frame` of the calibration design.

`cal_list` Named `list` of fitted `lm` models, one per analyte.

`cal_response_data` `data.frame` of summarised response per level.

`r2_df` `data.frame` of fit quality per analyte.

### See Also

quant_MSImagingExperiment, `createCalCurve()`, quantMSImageR-accessors

Other calibration: `createCalCurve,quant_MSImagingExperiment-method`, `int2conc,quant_MSImagingExperiment-me`
`plotCalCoverage,quant_MSImagingExperiment-method`, `summariseCalLevels,quant_MSImagingExperiment-metho`

### Examples

```
# Empty, as attached to a freshly coerced object
ci <- calibrationInfo()
ci

# Populated by the calibration workflow
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))
cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
cal <- createCalCurve(cal, cal_type = "cal")
calibrationDiagnostics(cal)
```

---

## `combineMSIs,MSImagingExperiment-method`

*Combine MSI experiments across acquisitions*

---

### Description

Concatenates two or more acquisitions pixel-wise into a single object, so that a whole study can be filtered, summarised and plotted together. Each input keeps its own `run`, which is what downstream per-sample summaries group on.

### Usage

```
## S4 method for signature 'MSImagingExperiment'
combineMSIs(MSIobject, ...)
```

### Arguments

| | |
|---|---|
| MSIobject | An MSImagingExperiment; the first acquisition, whose fData() defines the shared feature axis. |
| ... | Further MSImagingExperiment objects to combine. All must share the feature axis of MSIobject – use alignFeatures() first if they do not. A pData() column missing from some objects is added to them as NA. |

### Value

A single quant_MSImagingExperiment holding every input's pixels, with the shared fData() restored and one run level per acquisition.

### See Also

Other combining acquisitions: alignFeatures(), bindPanels(), stitchAcquisitions()

### Examples

```
p1 <- system.file("extdata", "example.raw", "section01.RDS",
                  package = "quantMSImageR")
p2 <- system.file("extdata", "example.raw", "section02.RDS",
                  package = "quantMSImageR")
combined <- combineMSIs(readRDS(p1), readRDS(p2))
```

---

## contributionHm *Group-mean heatmap with per-sample contribution as opacity*

---

### Description

The same heatmap as quantileHm() – same orientation, same group and feature colour bars, same square cells – with only the cell fill changed. Each cell encodes two things at once:

### Usage

```
contributionHm(
  MSIobject,
  quant_val = 0.5,
  scale = c("shared", "feature"),
  heatmap_order = NA,
  heatmap_labs = NA,
  feature_split = NULL,
  feature_split_name = "Pathway",
  group_split_name = "Group",
  cell_size = 8,
  max_aspect = 1.5,
  cell_border = "white",
```

```
  fontsize = 8,
  palette = c("heatmap2", "heatmap0"),
  group_palette = "hat",
  feature_palette = "reading",
  alpha_floor = 0.6,
  saturate = 0.04,
  na_col = "grey88",
  sample_block = NULL,
  sample_block_name = "Region",
  block_palette = "Set 2"
)
```

## Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment object. |
| quant_val | Numeric in (0, 1). Quantile of pixel intensities summarised per feature per sample (default 0.5, the median). |
| scale | Character. How the colour scale is set. "shared" (default) puts every feature on ONE scale, capped at a quantile of all the group means, so a strongly separating feature is vivid and a flat one is pale – colours mean the same thing in every column and features can be compared with each other. "feature" normalises each feature by its own extreme, so every column fills the ramp regardless of how far it actually moved. That shows the pattern within a feature and makes comparison BETWEEN features meaningless – a feature varying by a few percent looks exactly like one that doubles. Use "shared" to ask which features changed most, and "feature" to read each feature's own profile.<br>The study report also accepts "both" in parameters$heatmap_scale, which draws each quantile section twice, once on each scale. That is a report-level setting; this function draws one heatmap and takes only "shared" or "feature". |
| heatmap_order | Character vector of run names in the desired column order. Must match values in pData(MSIobject)$run. Defaults to NA (use discovery order). |
| heatmap_labs | Character vector of display labels, one per entry in heatmap_order. Used to create column-split groups. Defaults to NA (no splitting). |
| feature_split | Optional character/factor vector of length equal to the number of features, used to split the heatmap columns into labelled groups. Typically derived from an ion library metadata column (e.g. Met-1). Defaults to NULL (no splitting). |
| feature_split_name | Display name for the feature-group annotation legend (e.g. "Met-1", "Pathway"). Defaults to "Pathway". |
| group_split_name | Display name for the sample-group annotation legend (e.g. "Treatment", "Group"). Defaults to "Group". |
| cell_size | Numeric. Side of a heatmap cell, in millimetres (default 8). Giving the body an absolute size is what makes cells square; leaving it to ComplexHeatmap stretches them to fill the device, which produces very oblong cells when there are far more features than samples. Set to NA to restore the fill-the-device behaviour. |

`max_aspect` Numeric >= 1. Largest cell width-to-height ratio allowed when one dimension has many more entries than the other (default `1.5`). Cells stay square until the counts differ by more than four-fold; beyond that the shorter dimension's cells are widened up to this ratio so the plotting area is not reduced to a sliver.

`cell_border` Colour for the line drawn around every cell (default `"white"`), which separates neighbouring cells instead of letting equal colours merge into one block. `NA` draws no border.

`fontsize` Numeric. Point size for the sample and feature labels and the annotation names (default `8`). `ComplexHeatmap` defaults to 12, which crowds the panel once feature names are long enough to need rotating.

`palette` Character. Diverging colour ramp for the z-scores: `"heatmap2"` (default, blue-white-red) or `"heatmap0"`. See `quantPalettes()`.

`group_palette, feature_palette`
Character. Qualitative palettes for the sample-group and feature-group colour bars (defaults `"hat"` and `"reading"`). Any `grDevices::hcl.colors()` palette name also works.

`alpha_floor` Numeric in [0, 1). Opacity given to a sample that did not contribute (default `0.6`). Below roughly this a non-contributing sample starts to look like missing data.

`saturate` Numeric in (0, 0.5). Fraction of cells allowed to saturate at the colour limit (default `0.04`, i.e. 4%).

`na_col` Colour for features that cannot be scored – no variance across samples, or missing values (default `"grey88"`).

`sample_block` Optional character/factor vector, one entry per sample, dividing the rows into blocks – one per region of interest, typically. Rows are split by it, and **each block is scored against itself**: both the z-score and the group mean are computed within a block, so the hue says how a group sits among the other samples' regions of that type rather than restating that one region type is brighter than another. A block with one sample has nothing to score against and is drawn in `na_col`. Defaults to `NULL` (one block, scored across all samples).

`sample_block_name`
Display name for the block annotation legend (default `"Region"`).

`block_palette` Character. Qualitative palette for the block colour bar (default `"Set 2"`), kept distinct from `group_palette`.

## Details

**Hue** the sample's **group** mean for that feature – what the group did.

**Opacity** that **sample's** contribution to the group mean – who made it do that.

A plain group-mean heatmap cannot show that a group effect rests on one replicate. Here it is visible directly: a uniformly solid block is a real group effect, whereas one opaque tile among faded ones means the mean is carried by a single sample.

### Value

A `ComplexHeatmap::Heatmap` (rows = samples, columns = features), carrying the opacity key in `attr(x, "contribution_legend")`. Pass that to `ComplexHeatmap::draw(annotation_legend_list = )` so the opacity channel is labelled; the report does this.

### How the two channels are computed

Values are the per-sample quantile of pixel intensities, as in `quantileHm()`, then z-scored **per feature across all samples**, not within group – so a cell reads "high or low for this feature" rather than "abundant or not". Features with zero variance, and missing values, stay `NA` and are drawn in `na_col`; they are never imputed to the row mean.

The group mean of those z-scores is the hue. The contribution is `z * sign(group mean)`, so positive means the sample moved with its group and zero or negative means it did not; only the positive part drives opacity, rescaled onto `[alpha_floor, 1]`. `alpha_floor` exists because a nearly transparent tile reads as missing data rather than as weak agreement.

### Why there are two colour limits

The hue is a group **mean**, and averaging $n$ replicates shrinks it by roughly $\sqrt{n}$. Reusing the per-sample z limit for the group mean therefore produces a washed-out panel that no amount of alpha tuning fixes. Both limits are computed from the quantity actually drawn, each as the `1 - saturate` quantile of its own absolute values, so a similar small fraction of cells saturates on each channel and the bulk of the data gets the full ramp. Expect the two to differ by a factor of two or more. Note this differs from `quantileHm()`, which clips to a fixed [-1, 1].

Both limits above are single numbers covering the whole panel, which is what lets one column be compared with another. `scale = "feature"` replaces them with a limit per feature, so each column is normalised by its own extreme and fills the ramp whatever its actual spread. That is the same reading `quantileHm()` gives, but keeping the group-mean hue and the contribution opacity: use it to look along a single feature, never across features.

### Comparing features with each other

One cap or many. That is the whole of it.

`scale = "shared"` computes a single limit for the panel and draws every column on it, so a feature whose group mean reaches the cap saturates and one at a fifth of it stays pale. The pale column is pale because it moved less, and that difference is the information: features can be ranked against each other by how far their groups separate, in units of each feature's own spread.

`scale = "feature"` gives every column its own limit, its own extreme, so each fills the ramp whatever it actually did. Nothing is pale, so nothing can be ranked. A feature with no group effect at all looks exactly as vivid as the strongest one in the study – its largest group mean might be 0.05, and it is still drawn at full intensity. Use it to read one feature's profile across samples; never to argue that a feature matters.

What is being compared under `"shared"` is **effect size, not fold change**. The hue is a group separation measured in standard deviations of that feature, so a feature that doubles but varies wildly between replicates can sit below one that moves 20% in every sample. For magnitude, read the fold-change table instead. The two orderings are both correct and answer different questions.

For a balanced two-group design the group-mean z-score is bounded at $\pm 1$: perfectly separated groups with no within-group scatter put every sample at $z = \pm 1$, so the group means land on the bound, and identical groups put them on 0. A cap of 0.5 therefore means the strongest feature in the panel reaches about half the separation the design can show.

Two limits on the comparison, both worth remembering. The standard deviation is estimated from the samples in hand, so with a handful per group those estimates are noisy and the ranking is coarse – read it as "which features stand out", not as a precise order. And the deviation is pooled across groups, so a large separation inflates its own denominator; the scale compresses at the top and is not linear in effect size there.

## See Also

`quantileHm()` for the per-sample view of the same matrix.

Other visualisation: `imageR`, `quant_MSImagingExperiment-method`, `quantPalettes()`, `quantileHm()`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
hm <- contributionHm(obj, quant_val = 0.5,
                     heatmap_order = "section01", heatmap_labs = "A")
```

---

# createCalCurve,quant_MSImagingExperiment-method

*Fit per-analyte calibration models*

---

## Description

Fits a linear response-versus-amount model for each standard, which `int2conc()` later inverts to turn a measured response into an estimated amount per pixel.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
createCalCurve(
  MSIobject,
  cal_type,
  level = "level",
  background = "background",
  weighting = c("1/x", "none", "1/x2")
)
```

## Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment whose cal_response_data has been populated by summariseCalLevels(). |
| cal_type | Character, required. "cal" for standards deposited onto the slide, "std_addition" for standard addition on tissue. See Details. |
| level | Character. Column of cal_response_data holding the level labels (default "level"). |
| background | Character. Value of level marking the background level subtracted when cal_type = "std_addition". |
| weighting | Character. Regression weights: "1/x" (default), "1/x2" or "none". See the Weighting section. |

## Details

cal_type has no default and must be chosen explicitly, because the two modes apply materially different processing.

cal_type = "cal" fits the summarised response directly against the amount deposited, appropriate for standards spotted onto the slide beside the tissue (on-slide / external calibration).

cal_type = "std_addition" is classical **standard addition**, for standards deposited onto tissue where endogenous analyte is already present. It:

1. fits an unweighted response ~ deposited amount across all levels;
2. takes the x-intercept of that line – the amount at which predicted response is zero – as an estimate of the endogenous amount already in the tissue (this value is negative when endogenous signal is present);
3. subtracts it from every deposited amount, which shifts the x-axis so it expresses **total** amount present (endogenous + added);
4. drops the rows at the background level and refits on that shifted axis.

## Value

The input object with cal_list (one lm per standard, response versus amount in pg per pixel), r2_df (fit R-squared per standard) and the calibration metadata populated.

## Weighting

Calibration response in MSI is usually heteroscedastic – absolute scatter grows with amount – so an unweighted fit lets the top standards dominate and the low end, where most tissue pixels sit, is fitted worst. weighting therefore defaults to "1/x"; "1/x2" weights the low end harder still, and "none" is ordinary least squares.

A zero (blank) level has no 1/x weight. It is given the weight of the lowest positive level rather than a weight derived from a substituted tiny amount, which would let a single blank determine the whole regression.

Weighting is an empirical model choice, not a property of the data. Check it against residual structure, back-calculated accuracy at each level and replicate precision rather than against R-squared – see calibrationDiagnostics() and plotCalCoverage().

## See Also

Other calibration: calibrationInfo-class, int2conc,quant_MSImagingExperiment-method, plotCalCoverage,quant_MSImagingExperiment-method, summariseCalLevels,quant_MSImagingExperiment-metho

## Examples

```
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))
cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
cal <- createCalCurve(cal, cal_type = "cal")
```

---

# createMSIDataMatrix,quant_MSImagingExperiment-method

*Create a feature-by-sample MSI data matrix*

---

## Description

Flattens the pixel-level object into tabular form for downstream statistics: one row per pixel, or one row per region of interest when `roi_header` names a grouping column.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
createMSIDataMatrix(
  MSIobject,
  val_slot = "intensity",
  inputNA = TRUE,
  roi_header = NA
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` object. |
| `val_slot` | Character. Spectra slot to tabulate (default `"intensity"`). |
| `inputNA` | Logical. Convert zeros in the matrix to `NA` (default `TRUE`). |
| `roi_header` | Character. Column of `pData()` identifying regions of interest to average over. `NA` (the default) skips the ROI average and keeps one row per pixel. |

## Value

The input object with its `tissueInfo` slot populated: `all_pixel_matrix` (one row per pixel, one column per feature), `roi_average_matrix` (one row per ROI, when `roi_header` is given) and the accompanying sample/ROI metadata.

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- createMSIDataMatrix(obj, val_slot = "intensity", roi_header = NA)
```

---

generateTxtImages *Load, process and optionally export per-feature text-image matrices*

---

## Description

Reads one or more DESI-MRM acquisitions, applies SNR filtering and tissue/background separation, and writes per-feature text matrices that can be imported by imaging software (e.g. MassLynx QuanOptimise overlay tools). Setting `output_txt = FALSE` runs all processing but skips file writing, which is useful when the caller only needs the processed MSI objects (e.g. for HTML report rendering).

## Usage

```
generateTxtImages(
  fns,
  data_path,
  image_dir,
  lib_ion_path,
  snr_thresh = 0,
  thresh = 20,
  perc = 97,
  rot_clockwise = 0,
  average_method = "median",
  output_txt = TRUE,
  exclude = NULL,
  rename = NULL,
  ratios = NULL,
  output_ratios = TRUE,
  snr_overrides = NULL,
  is_name = NULL,
  is_norm_header = "IS_norm",
  is_mode = "line",
  is_window = 15,
  remove_IS = TRUE,
  type_header = "Type",
  rois = FALSE,
  stitch_overlap = c("error", "first", "last")
)
```

## Arguments

| | |
|---|---|
| fns | Character vector of acquisition names (without .raw suffix) for single-polarity studies. For dual-polarity studies pass a list where each element is either a plain string (single acquisition) or a named list with fields pos, neg, label, and optionally prefix_pos/prefix_neg (see bindPanels()). runStudy.R builds this list automatically from the YAML samples section.<br>A tissue acquired in pieces (top and bottom, say) is one element with a pieces field instead of pos/neg: a list of areas, each with its own pos and/or neg. Each piece's panels are bound, and the pieces are then placed by stage position with stitchAcquisitions() and become one run. |
| data_path | Path to the folder that contains the .raw acquisition directories. |
| image_dir | Root output directory. One sub-directory per acquisition is created inside it. |
| lib_ion_path | Full path to the MRM ion-library CSV. Must contain columns: transition_id, precursor_mz, product_mz, collision_eV, cone_V, Polarity, Type. |
| snr_thresh | Numeric scalar **or vector**. One or more minimum signal-to-noise thresholds; pixels below each threshold are set to NA. When a vector is supplied, one complete output directory tree is created per threshold value. A value of 0 skips SNR filtering entirely *and* removes the requirement for a tissue_pixels.csv mask – handy for a smoke-test pass before ROIs are drawn (default 0). |
| thresh | Numeric, one value from 0 to 100. Cold-spot percentile for the _hs<perc>_cs<thresh> .txt folders: after the hot-spot cap, values below this percentile are set to 0. Like perc, it is computed per sample over that sample's non-missing values – after SNR filtering, the tissue pixels that passed – and it does not touch the HTML report (default 20). |
| perc | Numeric, one value from 0 to 100. Hot-spot percentile for the same folders: values above it are capped to it (default 97). The report's images are capped by output: ion_images: percentile: instead. Keep thresh below perc: once the cap has made the top values equal, a floor at or above it removes either nothing or everything under the cap. |
| rot_clockwise | One of 0, 1, 2 or 3: the number of quarter turns clockwise applied to every .txt matrix with pracma::rot90(). Not degrees – 180 degrees is 2. rot90() counts turns modulo 4, so 180 would mean no rotation at all, and is refused. The report is never rotated (default 0). |
| average_method | Character, "mean" or "median". Statistic used to summarise the noise pixel vector in int2SNR(). Applied consistently to every acquisition (default "median"). Where a median noise level is only the imputed value (see int2SNR()), one warning for the whole call lists the samples affected; it has class quantMSImageR_imputed_noise_study. |
| output_txt | Logical. When FALSE, processing runs but no files are written to disk (default TRUE). |
| exclude | Character vector of feature names to drop before processing. Names must match fData()$name exactly (default NULL – keep all). |
| rename | Named character vector or list mapping old feature names to new display names, e.g. c("old name" = "new name"). Applied to all combined objects after loading; the new names appear in file names and the heatmap (default NULL – no renaming). |

`ratios` Optional list of metabolite ratio pairs. Each element must be a named list with fields:

- `num` Name of the numerator feature (matches `fData()$name` **after** any `rename` overrides are applied).
- `den` Name of the denominator feature.
- `label` *(optional)* Output filename prefix. Defaults to `"<num>_over_<den>"`.

Ratio images (numerator / denominator, pixel-wise) are written to the same SNR-filtered subdirectories as the individual ion images, immediately after them. Pixels where the denominator is zero or either ion is `NA` are set to `NA` (default `NULL` – no ratios).

`output_ratios` Logical. When `FALSE`, ratio txt files are not written even if `ratios` pairs are defined. Has no effect when `output_txt` is `FALSE` (default `TRUE`).

`snr_overrides` Optional named map of feature name to a feature-specific SNR threshold. Keys may be original ion-library names **or** post-`rename` display names. Listed features use their own threshold in every report; all other features use the global `snr_thresh`. Ignored for any report whose global threshold is `0` (the mask-free smoke-test pass). Default `NULL` (all features use `snr_thresh`).

`is_name` Character. Value identifying the internal standard in `fData()$feature_type` – that is, the ion library's `Type` column, typically `"IS"`. It is **not** a transition name. When supplied, each acquisition is normalised with `int2response()` before SNR filtering, and every downstream step – SNR, background masking, images and ratios – works on the resulting `response` layer instead of `intensity`. `NULL` or `"None"` (the default) skips normalisation.

`is_norm_header` Character. Ion-library column mapping each analyte to the standard that normalises it, needed only when `is_name` matches more than one feature (default `"IS_norm"`). See `int2response()`.

`is_mode` Character. Level at which the internal standard is summarised: `"line"` (default), `"sample"`, `"pixel"` or `"window"` (a rolling median over `is_window` consecutive pixels along the acquisition line). See `int2response()`.

`is_window` Integer. Number of consecutive pixels averaged when `is_mode = "window"` (default `15`).

`remove_IS` Logical. Drop the internal-standard feature after normalising (default `TRUE`).

`type_header` Character. Ion-library column holding the feature type, passed to `readMRM()` and matched by `is_name` (default `"Type"`).

`rois` Regions of interest drawn with `labelROIs()`. `TRUE` reads `roi_labels.csv` from each acquisition's `.raw` folder and adds `roi_label` and `roi_id` to `pData()`: the region's name and its numbered identifier (`airway_01`), `"unassigned"` for tissue outside every region and `NA` for background. Acquisitions without the file are entirely `"unassigned"`. `"auto"` is `TRUE` when any acquisition has the file; `FALSE` (default) ignores them. Regions already on an object – a `section:` RDS saved with `roi_label`/`roi_id`, as the bundled example sections are – are kept as they are, but `"auto"` cannot see them before loading, so switch regions on with `TRUE` for a study built from such sections.

`stitch_overlap` What `stitchAcquisitions()` should do when two pieces of one tissue claim the same position: `"error"` (default), or `"first"` / `"last"` to keep that piece's pixels where they collide.

## Value

Invisibly returns a named list:

`combined` Raw combined object (background pixels retained).

`combined_snr` SNR-filtered intensity for the **first** threshold in `snr_thresh`; sub-threshold pixels `NA`. Provided for backward compatibility.

`combined_snr_list` Named list of SNR-filtered objects, one per threshold in `snr_thresh`, named `"snr<value>"` (e.g. `"snr3"`).

`combined_NAbackground` Raw intensity with background set to `NA`.

`rois` Logical: whether region-of-interest labels were attached.

## Text-image output

With `output_txt = TRUE`, each sample gets these folders under `image_dir/<label>/`, each holding one tab-separated matrix per feature with no header and no coordinates (rows are y and columns x, before `rot_clockwise`). `<t>` is the SNR threshold, `<p>` is `perc` and `<c>` is `thresh`:

`intensity_raw` Measured values; background pixels 0. No SNR filter.

`intensity_SNRfiltered<t>` Measured values; background pixels and tissue pixels below SNR `<t>` are 0.

`response_SNRfiltered<t>_NORM` The same, rescaled to 0-100 within the sample.

`intensity_SNRfiltered<t>_hs<p>_cs<c>_removal` The SNR-filtered values capped at the sample's `<p>`th percentile, then values below its `<c>`th percentile set to 0. The closest to the report's ion images, which are capped but not floored.

`response_SNRfiltered<t>_hs<p>_cs<c>_NORM` The `_removal` matrix rescaled to 0-100 within the sample.

`response_SNRfiltered<t>_hs<p>_cs<c>_COMBINED` The `response_SNRfiltered<t>_NORM` matrix – **no** hot- or cold-spot step, despite the name – with cell `[1, 1]` replaced by the feature's largest value across the whole study, in measured units.

Missing values are written as 0.

Each matrix covers the sample's tissue bounding box, not the whole acquisition: `trimMSI()` drops every row and column that is entirely background, wherever it lies. Redrawing a tissue mask can therefore change a matrix's size and origin, and a registration made on the earlier matrices (an affine transform onto an H&E image, say) no longer lines up. To keep a frame, keep at least one tissue-labelled pixel in each of its rows and columns.

## See Also

`int2SNR()`, `applySNR()`, `back2NA()`, `imageR()`

Other workflow: `blockCentre()`, `print.quantValidation()`, `runExample()`, `runStudy()`, `validateConfig()`

## Examples

```
# Section-mode run on the bundled synthetic data (no tissue mask needed at
# snr_thresh = 0), returning the processed objects without writing files.
fns <- list(list(neg = "example", section = "section01", label = "A"))
res <- generateTxtImages(
  fns          = fns,
  data_path    = system.file("extdata", package = "quantMSImageR"),
  image_dir    = tempfile(),
  lib_ion_path = system.file("extdata", "example_ion_library.csv",
                             package = "quantMSImageR"),
  snr_thresh   = 0,
  output_txt   = FALSE
)
```

---

# imageR,quant_MSImagingExperiment-method

*Draw an ion image for one feature*

---

## Description

Renders a single feature's spatial distribution as a `ggplot`. Because the result is an ordinary `ggplot`, the layout, theme and scales can be modified with the usual `+` syntax – for example `+ facet_wrap(~ sample, ncol = 3)` to arrange several acquisitions in a grid.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
imageR(
  MSIobject,
  val_slot = "intensity",
  value = "response %",
  scale = "suppress",
  threshold = 1,
  sample_lab = "sample_ID",
  pixels = NA,
  percentile = 99,
  feat_ind = 1,
  perc_scale = FALSE,
  blank_back = TRUE,
  aspect_ratio = 1,
  text_image = FALSE,
  palette = c("heatmap0", "viridis"),
  raster = FALSE
)
```

## Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment object. |
| val_slot | Character. Spectra slot to image (default "intensity"). |
| value | Character. Legend label describing what the values represent (e.g. "pg/mm2" for a calibrated layer). |
| scale | Character. Colour-scale treatment: "suppress" (cap at percentile), "histogram" (contrast-enhance per image) or "sqrt". |
| threshold | Numeric. Lowest percentile removed from the colour scale, to stop residual background dominating it (default 1). |
| sample_lab | Character. Column of pData(MSIobject) used to label and facet the images (default "sample_ID"). |
| pixels | Character. Value of pData(MSIobject)$sample_type selecting which pixels to plot; NA (the default) plots all. |
| percentile | Numeric. Percentile at which the colour scale is capped when scale = "suppress" (default 99). |
| feat_ind | Integer. Row index of the feature in fData(MSIobject) to image (default 1). |
| perc_scale | Logical. Express the colour scale as a percentage of the maximum (TRUE) rather than raw values (FALSE, the default). |
| blank_back | Logical; when TRUE background/zero pixels are drawn transparent. |
| aspect_ratio | numeric plot aspect ratio (default 1). |
| text_image | Logical; when TRUE return the image as a numeric matrix (for text export) rather than a ggplot. |
| palette | Character. Colour scale for the image: "heatmap0" (the default, matching the YAML colours: ion_image: default) or "viridis" (perceptually uniform and colour-blind safe). See quantPalettes(). |
| raster | Logical. Draw the pixels with geom_raster() rather than geom_tile(). Much faster on a large acquisition – one image instead of a rectangle per pixel – and valid because MSI pixels lie on a regular lattice. Default FALSE, which keeps the original rendering. |

## Value

A ggplot object, or a numeric matrix when text_image = TRUE.

## See Also

Other visualisation: contributionHm(), quantPalettes(), quantileHm()

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
gg <- imageR(obj, feat_ind = 1, sample_lab = "run", scale = "suppress")
```

`int2conc,quant_MSImagingExperiment-method`
*Convert response to calibrated amount estimates*

## Description

Inverts each analyte's calibration model to turn a measured response into an **estimated amount per pixel**. Features without a calibration model are dropped.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
int2conc(
  MSIobject,
  val_slot = "intensity",
  pixel_header = "sample_name",
  pixels = "tissue_pixels",
  max_out_of_range = 0.1
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` carrying calibration models fitted by `createCalCurve()`. |
| `val_slot` | Character. Spectra slot holding the measured response (default `"intensity"`). |
| `pixel_header` | Character. Column of `pData()` holding the pixel-type labels (default `"sample_name"`). Pass `pixel_header = "sample_type"` for a calibration acquisition. |
| `pixels` | Character. Label(s) in `pixel_header` marking the pixels to quantify (default `"tissue_pixels"`, the imaging convention). Pass `pixels = "Tissue"` for a calibration acquisition. |
| `max_out_of_range` | Numeric in `[0, 1]`. Largest proportion of a feature's pixels allowed to fall outside the calibrated range, i.e. to be extrapolated rather than interpolated (default `0.1`, so at least 90% of pixels must be interpolated between real standards). Exceeding it is an **error**, naming the features and pointing at `plotCalCoverage()`: either the standards need to bracket the tissue, or the extrapolation has to be accepted explicitly by raising this value (`1` accepts any). Any extrapolation at all is reported by `message()`; this controls only the threshold at which that becomes fatal. |

## Details

Mass spectrometry imaging measures analyte response at each pixel rather than analyte amount directly. The values produced here are calibrated *amount* estimates (pg per pixel, and pg per mm-squared where the pixel size is known), not volumetric concentrations. They are conditional on the calibration design, matrix matching, acquisition conditions and the validity of the fitted model – see `plotCalCoverage()` for the coverage check that should accompany them.

## Value

The object subset to `pixels`, with two spectra slots added: `pg_pixel` (estimated amount per pixel) and, when `experimentData(MSIobject)$pixelSize` is available, `pg_mm2` (estimated amount per unit area). Features with no calibration model are removed, and `r2_df` gains an `out_of_range` column giving the proportion of each feature's pixels that were extrapolated beyond the calibrated range.

## Areal conversion

`pg_pixel` is inverted directly from the calibration model. `pg_mm2` is then derived from the pixel size recorded in the acquisition metadata:

$$pg/mm^2 = (pg/pixel) \times (1000/s)^2$$

where `s` is `experimentData(MSIobject)$pixelSize`, **expected in micrometres**, and the factor 1000 converts micrometres to millimetres. A single scalar is assumed, i.e. square pixels of side `s`; rectangular pixels are not currently supported, and supplying `s` in millimetres would inflate the result by a factor of 10^6. When `pixelSize` is missing or not a positive number the `pg_mm2` slot is omitted and a warning is issued.

## See Also

Other calibration: `calibrationInfo-class`, `createCalCurve,quant_MSImagingExperiment-method`, `plotCalCoverage,quant_MSImagingExperiment-method`, `summariseCalLevels,quant_MSImagingExperiment-metho`

## Examples

```
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))
cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
cal <- createCalCurve(cal, cal_type = "cal")

# A calibration acquisition labels its pixels in `sample_type`, so the
# imaging defaults are overridden here
cal <- int2conc(cal, pixel_header = "sample_type", pixels = "Tissue")
```

---

`int2response,quant_MSImagingExperiment-method`

*Normalise pixel intensities to internal-standard response*

---

## Description

Divides each feature's intensity by an internal standard measured in the same pixel, line or sample, which suppresses drift and much of the local variation in ionisation efficiency.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
int2response(
  MSIobject,
  val_slot = "intensity",
  normalisation = c("internal_standard", "within_feature"),
  IS_name = NULL,
  is_norm_header = "IS_norm",
  mode = c("line", "sample", "pixel", "window"),
  window = 15,
  remove_IS = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment object. |
| val_slot | Character. Spectra slot to normalise (default "intensity"). |
| normalisation | Character. "internal_standard" (default) or "within_feature". See Details. |
| IS_name | Character. Value identifying the internal standard in the feature-type column of fData(MSIobject) – the ion library's Type column, typically "IS". This is a type label, not a transition name: passing a feature name will not match. Required when normalisation = "internal_standard"; a value that matches no feature is an error, since falling back to a different normalisation would silently change the analytical method. |
| is_norm_header | Character. Feature-metadata column holding the analyte-to-standard mapping (default "IS_norm"). It is consulted **only** when IS_name matches more than one feature, so "None" (or NULL) is correct for the common case of a single standard shared by every analyte. |
| mode | Character. Level at which the standard is summarised:<br>"line" Median across a whole acquisition line (the default). A *line* is one horizontal raster row – constant y, varying x – which is the order DESI acquires in, so it is also the axis along which source drift accumulates.<br>"sample" Median across the whole acquisition.<br>"pixel" The standard in that pixel alone – responsive to local suppression, but carries the standard's own shot noise.<br>"window" Rolling median over window consecutive pixels along the same horizontal row, ordered by x. A compromise between "pixel" and "line": it smooths the standard's own shot noise while still tracking drift across the row. The window is truncated at the ends of a row rather than wrapping, so pixels from different rows are never mixed. |
| window | Integer. Number of consecutive pixels averaged when mode = "window" (default 15). Ignored for the other modes. |
| remove_IS | Logical. Drop the internal-standard features from the returned object (default TRUE). |

... Additional arguments (currently unused).

## Details

`normalisation` states what is being done, so that no argument value has to be read as an instruction:

`"internal_standard"` Divide by the standard named in `IS_name`. This is the quantitative path.

`"within_feature"` Divide each feature by its own summary at the chosen `mode`. This removes between-line or between-sample scale differences within a feature, but it is *not* internal-standard normalisation and does not make features comparable to one another.

To leave values untouched, do not call this function. There is deliberately no "none" value here: skipping a step is the caller's decision, not a mode of the step. The YAML workflow exposes `normalisation: none` for that, and simply does not call this function.

## Value

The input object with a `response` spectra slot holding the normalised values.

## Multiple internal standards

When `IS_name` matches exactly one feature, every analyte is normalised to it. When it matches several – a panel carrying one class-specific standard per lipid class, for instance – an explicit analyte-to-standard mapping is required, because dividing by several standards at once has no defined meaning. Supply it as an ion-library column (named by `is_norm_header`, default `"IS_norm"`) whose value on each analyte row is the transition name of the standard that normalises it. Any analyte left unmapped is an error rather than a silent fallback.

With a single standard the column is never read, so `is_norm_header = "None"` is the right setting for a panel where every analyte shares one standard.

## See Also

Other filtering: applySNR,quant_MSImagingExperiment-method, back2NA,quant_MSImagingExperiment-method, int2SNR,quant_MSImagingExperiment-method, zero2NA,quant_MSImagingExperiment-method

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
# No internal standard in the example panel: normalise each feature to
# itself, per acquisition line.
obj <- int2response(obj, val_slot = "intensity",
                    normalisation = "within_feature")
```

`int2SNR,quant_MSImagingExperiment-method`
*Calculate background-referenced signal-to-noise ratios*

## Description

Expresses each tissue pixel's response as a ratio to the same feature's response in the background pixels of the same acquisition.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
int2SNR(
  MSIobject,
  val_slot = "intensity",
  pixel_header = "sample_name",
  background = "background_pixels",
  tissue = "tissue_pixels",
  snr_thresh = 3,
  average = c("median", "mean"),
  snr_overrides = NULL,
  no_background = c("error", "all_na", "copy"),
  noise = NULL,
  sample_type = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` whose `pData()` carries a background label in the `pixel_header` column. |
| `val_slot` | Character. Spectra slot holding the measured response (default `"intensity"`; use `"response"` after `int2response()`). |
| `pixel_header` | Character. Column of `pData()` holding the pixel-type labels (default `"sample_name"`, the imaging convention). |
| `background` | Character. Value in `pixel_header` marking the background (non-tissue) pixels used to estimate the noise level (default `"background_pixels"`). The historical `"noise_pixels"` / `"Noise"` labels are matched too, so masks made with earlier versions keep working. |
| `tissue` | Character. Value in `pixel_header` marking the tissue pixels SNR is calculated for (default `"tissue_pixels"`). |
| `snr_thresh` | Global minimum SNR to accept (below this value SNR = NA). Applied to every feature unless overridden by `snr_overrides`. |
| `average` | Character, `"median"` (default) or `"mean"`: statistic used to summarise the background-pixel vector per feature. |

| | |
|---|---|
| snr_overrides | Optional named numeric vector mapping feature name (as in fData(MSIobject)$name) to a feature-specific SNR threshold. A feature listed here uses its own threshold instead of snr_thresh; features not listed fall back to snr_thresh. Default NULL (all features use the global threshold). |
| no_background | Character. What to do when pixel_header contains no background pixels, so there is nothing to reference the signal against: "error" (default), "all_na" (create the slot, all NA) or "copy" (copy val_slot into the snr slot – note the slot then holds measured response, not a ratio). "error" and "all_na" apply only when snr_thresh > 0; with snr_thresh = 0 no filtering was requested, so the values are copied through regardless. |
| noise | Deprecated alias for background, kept for backward compatibility. When supplied it overrides background. |
| sample_type | Deprecated alias for pixel_header, kept for backward compatibility. When supplied it overrides pixel_header. |
| ... | Additional arguments (currently unused). |

## Details

For feature f and tissue pixel p, with background-pixel set B:

$$SNR_{f,p} = I_{f,p} / \text{summary}_{b \in B}(I_{f,b})$$

where summary is the mean or median selected by average. This is a background-referenced signal ratio, not the classical analytical definition based on the standard deviation of the noise.

median is the default because a background region that clips part of the tissue, or contains a spot of carryover, shifts a mean far more than a median. Zero background values are treated as missing and imputed at one tenth of the smallest non-zero background value for that feature, so a feature whose background is entirely zero or NA is skipped and its snr stays NA.

When **more than half** of a feature's background pixels are zero – common in DESI-MRM, where many transitions read exactly zero off tissue – the median *is* that imputed value. The noise level is then set by the imputation rule rather than by the background, and snr_thresh = t passes any pixel above t / 10 times the dimmest non-zero background value: at t = 10, anything at least as bright as the dimmest background pixel. int2SNR() warns when this happens, naming the features (the warning has class quantMSImageR_imputed_noise). average = "mean" is computed from the background's own values, to which the imputed zeros add almost nothing, and gives a stricter noise level for such features. Results are unchanged by the warning; it says how much the filter is doing.

Run this **after** internal-standard normalisation (int2response()) when one is used. A threshold of 3 is a detection-level criterion; a stricter threshold is advisable before converting a feature to calibrated amounts.

## Value

The input object with an additional snr spectra slot. Values below the selected threshold, and values outside the pixels labelled tissue, are stored as NA in that slot. No existing slot is modified.

## See Also

Other filtering: applySNR,quant_MSImagingExperiment-method, back2NA,quant_MSImagingExperiment-method, int2response,quant_MSImagingExperiment-method, zero2NA,quant_MSImagingExperiment-method

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")

# The defaults match the imaging convention used throughout the package
obj <- int2SNR(obj, snr_thresh = 3)
names(spectraData(obj))
```

---

# labelROIs *Label regions of interest within the tissue of an acquisition*

---

## Description

Run after selectTissuePixels(). Shows the ion image of a chosen transition across the acquisition's **tissue pixels only**, lets you draw a region – an airway, a vessel, parenchyma – and asks what it is. Repeat for as many regions as you need; the study report then compares them (see the roi: block of the config template).

## Usage

```
labelROIs(
  name,
  data_path,
  lib_ion_path,
  feature = NULL,
  enhance = "histogram",
  overwrite = FALSE,
  copy_to = NULL,
  preview = TRUE
)
```

## Arguments

| | |
|---|---|
| name | Character. Acquisition name, without the .raw suffix. |
| data_path | Character. Path to the folder that contains the .raw acquisition directory. |
| lib_ion_path | Character. Full path to the MRM ion-library CSV. |
| feature | Integer index or character name of the transition whose ion image to draw on. When NULL (default), all features are displayed first and you are prompted to choose one. |

| | |
|---|---|
| enhance | Character. Contrast enhancement passed to `Cardinal::image()`. Default `"histogram"`. |
| overwrite | Logical. `TRUE` starts a new `roi_labels.csv`, keeping the existing one as `roi_labels_<date-time>.csv` `FALSE` (default) adds to it. |
| copy_to | Character. Other acquisitions over the same area (the other panels of this sample) to copy the finished regions to. An acquisition that already has regions is skipped unless `overwrite = TRUE`. Default `NULL`. |
| preview | Logical. Draw the saved regions over the tissue when done. Default `TRUE`. |

## Details

```
#> Draw a region on the image, then close the window.
#> Label for this region (used in this study: airway, vessel): airway
#> Saved as airway_03.
#> Add another region? [y/n]: y
```

## Value

Invisibly, the saved regions: a data frame with columns `x`, `y`, `roi_label` and `roi_id`, one row per labelled pixel.

## Labels and numbers

Several regions can share a label: a section has many airways. Each is numbered as it is saved (`airway_01`, `airway_02`, ...), so the report can treat them as independent regions or pool them per sample. The prompt lists the labels already used anywhere in the study (every `roi_labels.csv` beside this one in `data_path`), so the same thing is called the same name in every acquisition. Type the label only; the number is added for you.

A pixel belongs to at most one region. A region drawn over part of an earlier one takes those pixels, and an earlier region left with no pixels is dropped. Tissue outside every region is `unassigned`, which the report can keep or leave out.

## The file

Regions are saved as `roi_labels.csv` in the acquisition's `.raw` folder, one per acquisition, beside `tissue_pixels.csv` – which is not touched. It lists the labelled pixels by `x`, `y`, `roi_label` and `roi_id`, and is written after every region, so nothing is lost if the session ends early. Running `labelROIs()` again adds regions to the file; `overwrite = TRUE` starts again, keeping the old file as a dated backup.

Panels acquired over the same area share their pixel grid, so the regions drawn on one apply to the others: `copy_to` writes the same file into their folders. Regions are matched by pixel position, as the tissue mask is.

## See Also

`selectTissuePixels()`, `runStudy()`

Other acquisition: `readMRM()`, `removeBlankMzs`, `quant_MSImagingExperiment-method`, `selectTissuePixels()`, `trimMSI()`

## Examples

```
# Requires a graphics device and user input, so it only runs interactively.
if (interactive()) {
  # Once per acquisition, after its tissue mask:
  selectTissuePixels("my_acquisition", data_path = "path/to/raw",
                     lib_ion_path = "path/to/ion_library.csv")
  labelROIs("my_acquisition", data_path = "path/to/raw",
            lib_ion_path = "path/to/ion_library.csv",
            copy_to = c("my_acquisition_pos01", "my_acquisition_pos02"))
}
```

---

# plotCalCoverage,quant_MSImagingExperiment-method

*Check that measured pixels fall within the calibrated range*

---

## Description

Converting a response into a calibrated amount is only trustworthy where the standards actually constrain the fit. A calibration whose levels sit above or below the signal measured in tissue is extrapolation, not quantification. `int2conc()` reports the proportion of extrapolated pixels per feature, but without an explicit coverage check that is still easy to overlook – and the plot shows *where* the mismatch is, not just how much of it there is.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
plotCalCoverage(
  MSIobject,
  features = NULL,
  val_slot = "intensity",
  conc_slot = "pg_pixel",
  log_base = 2,
  bins = 40,
  cal_colour = "red",
  cal_alpha = 0.4,
  cal_linetype = "dotted",
  ...
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` that has been through `createCalCurve()` and `int2conc()`, so it carries both the fitted models (see `calibrationModels()`) and a calibrated-amount slot. |
| `features` | Features to plot: a character vector of names as in `fData(MSIobject)$name`, or numeric row indices. Defaults to every feature that has a calibration model. |

| | |
|---|---|
| val_slot | Character. Spectra slot holding the measured response (default "intensity"). |
| conc_slot | Character. Spectra slot holding the calibrated amount estimate written by int2conc(). Defaults to "pg_pixel"; objects calibrated with earlier versions carry "conc - pg/pixel", which is matched automatically. |
| log_base | Numeric. Base for both axes (default 2). |
| bins | Integer. Number of histogram bins for the pixel distribution (default 40). |
| cal_colour | Colour used for the standards and the level markers (default "red"). |
| cal_alpha | Numeric. Opacity of the calibration-level marker lines (default 0.4), kept faint so they read as a reference grid rather than competing with the data. |
| cal_linetype | Line type for the calibration-level markers (default "dotted"). |
| ... | Additional arguments (currently unused). |

## Details

This function plots the two together on log axes so the overlap can be judged directly. The lower panel is the calibration itself: the standard spots and the fitted model that int2conc() inverts. The upper panel is the distribution of the object's own pixels on the same x-axis, with the calibration levels marked. When the histogram sits inside the span of those marks, every converted pixel is interpolated between real standards.

The standards are read back from each model's own model frame (stats::model.frame) rather than from cal_response_data. This matters for cal_type = "std_addition", where createCalCurve() shifts the amount axis by the estimated endogenous amount before fitting: the stored response data is on the unshifted axis, so plotting it against the fitted line would put the points and the model on different scales. The model frame always holds the points the line was actually fitted to.

The fitted model is linear in amount, so on log axes it is drawn as a curve evaluated across the plotted range rather than as a straight line. Any part of it predicting a non-positive response has no log-scale counterpart and is dropped.

## Value

A patchwork object: the pixel histogram stacked above the calibration curve, sharing an x-axis. Printing it draws the figure; it can also be further modified with ggplot2 layers.

## See Also

Other calibration: calibrationInfo-class, createCalCurve,quant_MSImagingExperiment-method, int2conc,quant_MSImagingExperiment-method, summariseCalLevels,quant_MSImagingExperiment-method

## Examples

```
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))
cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
cal <- createCalCurve(cal, cal_type = "cal")
```

```
cal <- int2conc(cal, pixel_header = "sample_type", pixels = "Tissue")

plotCalCoverage(cal, val_slot = "intensity")
```

---

`print.quantValidation` *Print a validation result*

---

## Description

Print a validation result

## Usage

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

## Arguments

| | |
|---|---|
| `x` | A `quantValidation` object. |
| `...` | Ignored. |

## Value

`x`, invisibly.

## See Also

Other workflow: `blockCentre()`, `generateTxtImages()`, `runExample()`, `runStudy()`, `validateConfig()`

---

`quant_MSImagingExperiment`
*Quantifiable MS imaging experiment*

---

## Description

Extends `Cardinal::MSImagingExperiment` with the metadata `quantMSImageR` needs to filter against background pixels and to convert a measured response into an estimated amount. Nearly every function in the package takes one of these, so a Cardinal object is normally coerced on the way in.

## Details

The object is **features x pixels**: one row per MRM transition, one column per pixel. Derived layers are added as additional spectra slots rather than overwriting `intensity`, so the raw values survive the whole workflow:

`intensity` Raw measured response, as read from the acquisition.

`response` Internal-standard-normalised, added by `int2response()`.

`snr` Background-referenced signal-to-noise, added by `int2SNR()`.

`pg_pixel`, `pg_mm2` Calibrated amount estimates, added by `int2conc()`.

Inherited structure comes from Cardinal: `fData()` for feature metadata (`mz`, `name`, transition m/z, ion-library columns), `pData()` for pixel metadata (`x`, `y`, `run`, and the tissue mask in `sample_name`) and `experimentData()` for acquisition metadata – `pixelSize` in particular, without which `int2conc()` cannot produce `pg_mm2`.

Several acquisitions are held in a single object, distinguished by the `run` column of `pData()`; this is what `combineMSIs()` produces and what the per-sample summaries group on.

## Value

An object of class `quant_MSImagingExperiment`, extending `Cardinal::MSImagingExperiment` with `calibrationInfo` and `tissueInfo` slots.

## Slots

`calibrationInfo` A calibrationInfo object.

`tissueInfo` A tissueInfo object.

## See Also

calibrationInfo, tissueInfo, quantMSImageR-accessors, `combineMSIs()`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")

# Cardinal object in, quant_MSImagingExperiment out
obj <- as(readRDS(p), "quant_MSImagingExperiment")

dim(obj)                      # features x pixels
names(spectraData(obj))       # available layers
head(as.data.frame(pData(obj)), 3)

# Derived layers accumulate alongside intensity
obj <- int2SNR(obj, snr_thresh = 3)
names(spectraData(obj))
```

quantileHm *Quantile heatmap of MSI features across samples*

## Description

For each feature, computes the nth quantile of pixel intensities within each acquisition run, then z-scores the resulting per-feature profile across samples (clipped to [-1, 1]) and renders a `ComplexHeatmap::Heatmap`.

## Usage

```
quantileHm(
  MSIobject,
  quant_val,
  heatmap_order = NA,
  heatmap_labs = NA,
  feature_split = NULL,
  feature_split_name = "Pathway",
  group_split_name = "Group",
  sample_block = NULL,
  sample_block_name = "Region",
  block_palette = "Set 2",
  cell_size = 8,
  max_aspect = 1.5,
  cell_border = "white",
  fontsize = 8,
  palette = c("heatmap2", "heatmap0"),
  na_col = "grey88",
  group_palette = "hat",
  feature_palette = "reading",
  row_split = NULL,
  row_split_name = NULL,
  column_split_name = NULL
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` object. |
| `quant_val` | Numeric in (0, 1). Quantile to summarise per feature per sample (e.g. `0.95` for the 95th percentile). |
| `heatmap_order` | Character vector of run names in the desired column order. Must match values in `pData(MSIobject)$run`. Defaults to `NA` (use discovery order). |
| `heatmap_labs` | Character vector of display labels, one per entry in `heatmap_order`. Used to create column-split groups. Defaults to `NA` (no splitting). |
| `feature_split` | Optional character/factor vector of length equal to the number of features, used to split the heatmap columns into labelled groups. Typically derived from an ion library metadata column (e.g. `Met-1`). Defaults to `NULL` (no splitting). |

feature_split_name
: Display name for the feature-group annotation legend (e.g. "Met-1", "Pathway"). Defaults to "Pathway".

group_split_name
: Display name for the sample-group annotation legend (e.g. "Treatment", "Group"). Defaults to "Group".

sample_block
: Optional character/factor vector, one entry per sample, dividing the rows into blocks – one per region of interest, typically. Rows are split by it, and **each block is z-scored against itself**, so a colour says where a sample sits among the others of that block rather than among every row. Without it, a panel holding airways and parenchyma would mostly report that the two tissues differ, which is rarely the question. A block with only one sample has no spread to score against and is drawn in na_col. Defaults to NULL (one block, scored across all samples).

sample_block_name
: Display name for the block annotation legend (default "Region").

block_palette
: Character. Qualitative palette for the block colour bar (default "Set 2"), kept distinct from group_palette so a region is not mistaken for a group.

cell_size
: Numeric. Side of a heatmap cell, in millimetres (default 8). Giving the body an absolute size is what makes cells square; leaving it to ComplexHeatmap stretches them to fill the device, which produces very oblong cells when there are far more features than samples. Set to NA to restore the fill-the-device behaviour.

max_aspect
: Numeric >= 1. Largest cell width-to-height ratio allowed when one dimension has many more entries than the other (default 1.5). Cells stay square until the counts differ by more than four-fold; beyond that the shorter dimension's cells are widened up to this ratio so the plotting area is not reduced to a sliver.

cell_border
: Colour for the line drawn around every cell (default "white"), which separates neighbouring cells instead of letting equal colours merge into one block. NA draws no border.

fontsize
: Numeric. Point size for the sample and feature labels and the annotation names (default 8). ComplexHeatmap defaults to 12, which crowds the panel once feature names are long enough to need rotating.

palette
: Character. Diverging colour ramp for the z-scores: "heatmap2" (default, blue-white-red) or "heatmap0". See quantPalettes().

na_col
: Colour for features that cannot be scored – no variance across samples, or missing values (default "grey88"). The same default as contributionHm(), so an unscoreable feature looks identical in both.

group_palette, feature_palette
: Character. Qualitative palettes for the sample-group and feature-group colour bars (defaults "hat" and "reading"). Any grDevices::hcl.colors() palette name also works.

row_split, row_split_name, column_split_name
: Deprecated aliases for feature_split, feature_split_name and group_split_name. They were named for the axis each grouping landed on before samples and features swapped axes; supplying them still works.

## Details

**Rows are samples and columns are features**, in the order given by `heatmap_order`. Studies normally have more samples than features, so this puts the long dimension vertically where there is room for it and keeps sample names horizontally readable.

## Value

A `ComplexHeatmap::Heatmap` object (rows = samples, columns = features), with an absolutely-sized body unless `cell_size` is `NA`.

## What the colour means

Each feature is z-scored **on its own**, across samples: the column is centred on that feature's mean and divided by its own standard deviation. The colour therefore says where a sample sits *within that feature's* range, and the ramp is anchored at 0 so white is the feature's mean.

**Colours are not comparable between features.** Because every column is scaled by its own spread, a feature whose samples differ by a few percent fills the same blue-to-red range as one that doubles. Reading across a row to compare two features is the one thing this heatmap will not support. Use it to ask "which samples are high for this feature", not "which feature changed most" – for the latter, see the fold-change table, or `contributionHm()`, which puts every feature on one shared scale.

The z-scores are clipped to `[-1, 1]` and the ramp spans exactly that, so any sample more than one standard deviation from its feature's mean is drawn at full intensity. With a handful of samples per group a clear separation saturates readily, which is deliberate – the panel is meant to show the pattern, not the magnitude – but it does mean two saturated cells can sit at very different z-scores.

A feature with no variance across samples, and a feature that is missing, both have no z-score: there is no spread to place a sample within. They are drawn in `na_col` rather than given a position on the ramp, matching `contributionHm()`. In particular neither is drawn as low, so a single-coloured grey column means "cannot be scored", not "depleted everywhere".

## Why there is no scale argument here

`contributionHm()` takes `scale = "shared"` or `"feature"`, choosing whether features can be compared with each other. This function has no such option, and the reason is structural rather than an omission.

Comparability needs the coloured quantity to still carry cross-feature information when the colour scale is applied. Here it does not. Every column is z-scored to mean 0 and standard deviation 1 *before* anything is drawn, so a feature whose samples differ by two percent and one that doubles arrive at the ramp with the same spread. The fixed [-1, 1] limits are shared by every column, but sharing limits cannot restore what standardising already removed – the flattening happens first.

`contributionHm()` escapes this because it colours the group **mean** of those z-scores, which is not renormalised per feature: it sits near 0 when the groups do not separate and approaches 1 when they separate cleanly, and that range survives to the colour scale. A shared cap on it therefore means something.

So: this heatmap for one feature's profile across samples, and `contributionHm(scale = "shared")` when the question is which features separate the groups. Adding `scale` here would offer a comparison the arithmetic cannot support.

## See Also

Other visualisation: contributionHm(), imageR, quant_MSImagingExperiment-method, quantPalettes()

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
hm <- quantileHm(obj, quant_val = 0.5,
                  heatmap_order = "section01", heatmap_labs = "A")
```

---

# quantMSImageR-accessors

*Accessors for calibration and tissue metadata*

---

## Description

Extract and replace the contents of a quant_MSImagingExperiment's calibrationInfo and tissueInfo slots. Prefer these to reaching into the slots with @: the internal representation may change, but these functions will not.

## Usage

```
calibrationData(x)

## S4 method for signature 'quant_MSImagingExperiment'
calibrationData(x)

calibrationData(x) <- value

## S4 replacement method for signature 'quant_MSImagingExperiment'
calibrationData(x) <- value

calibrationModels(x)

## S4 method for signature 'quant_MSImagingExperiment'
calibrationModels(x)

calibrationModels(x) <- value

## S4 replacement method for signature 'quant_MSImagingExperiment'
calibrationModels(x) <- value

calibrationDiagnostics(x)

## S4 method for signature 'quant_MSImagingExperiment'
```

```
calibrationDiagnostics(x)

calibrationR2(x)

## S4 method for signature 'quant_MSImagingExperiment'
calibrationR2(x)

calibrationLevels(x)

## S4 method for signature 'quant_MSImagingExperiment'
calibrationLevels(x)

calibrationMetadata(x)

## S4 method for signature 'quant_MSImagingExperiment'
calibrationMetadata(x)

tissueData(x)

## S4 method for signature 'quant_MSImagingExperiment'
tissueData(x)

tissueMatrix(x, which = c("roi", "pixel"))

## S4 method for signature 'quant_MSImagingExperiment'
tissueMatrix(x, which = c("roi", "pixel"))
```

## Arguments

| | |
|---|---|
| x | A quant_MSImagingExperiment object. |
| value | Replacement value. |
| which | For tissueMatrix(), whether to return the per-ROI average matrix ("roi", the default) or the per-pixel matrix ("pixel"). |

## Details

Note the naming: calibrationInfo() and tissueInfo() are the *class constructors*, so the slot accessors are calibrationData() and tissueData().

The calibration accessors follow the workflow. summariseCalLevels() populates calibrationLevels(), createCalCurve() fills calibrationModels() and calibrationR2(), and int2conc() reads the models back out. calibrationData() moves the whole block at once, which is what carrying fitted models from a standards acquisition to a study object requires.

## Value

calibrationModels() a named list of lm objects, one per calibrated analyte. calibrationDiagnostics() a data frame of fit quality, one row per analyte: feature, r2, and an out_of_range column once int2conc() has run. calibrationR2() the feature/r2 columns of that table only. calibrationLevels()

the summarised response per calibration level. `calibrationMetadata()` the calibration design supplied by the user. `calibrationData()` the whole `calibrationInfo` object. `tissueMatrix()` a data frame of features by pixels or by ROI. The replacement forms return the modified object.

## Examples

```
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))

cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
cal <- createCalCurve(cal, cal_type = "cal")

names(calibrationModels(cal))
calibrationDiagnostics(cal)
head(calibrationLevels(cal))

# Carry the fitted models onto a study acquisition
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
study <- as(readRDS(p), "quant_MSImagingExperiment")
calibrationData(study) <- calibrationData(cal)
```

---

`quantPalettes` *Colour palettes used by quantMSImageR*

---

## Description

The palettes available to `imageR()`, `quantileHm()` and the metadata colour bars, returned as named character vectors of hex colours.

## Usage

```
quantPalettes(name = NULL, n = NULL)
```

## Arguments

| | |
|---|---|
| `name` | Optional palette name. When `NULL` (default) the whole list is returned. |
| `n` | Optional number of colours. Qualitative palettes are truncated, continuous ones are interpolated. `NULL` (default) returns the palette at its native length. |

## Details

Four of these – `heatmap0`, `heatmap2`, `hat` and `reading` – are taken from the `ltc` package by Loukas Theodosiou (https://github.com/loukesio/ltc-color-palettes), which is MIT licensed. `heatmap2` and `hat` are re-ordered relative to ltc – see below – and `hat` substitutes a grey for ltc's black; the remaining colours are unchanged. They are reproduced here rather than depended on: `ltc` brings in 34 recursive dependencies, which is a large amount of installation surface for four colour vectors in a package that is otherwise light.

Which palette suits what:

`viridis` Sequential, perceptually uniform and colour-blind safe. The default for ion images, where the quantity is one-directional.

`heatmap0` Nine-colour sequential ramp running dark blue to teal to sand to red. An alternative for ion images.

`heatmap2` Five-colour diverging ramp, blue to white to red. Suits z-scores, where the midpoint is meaningful.

`hat`, `reading` Qualitative, for categorical metadata such as study group or pathway class. `hat` carries 10 well-separated hues, re-sequenced from ltc's ordering so that the first few are maximally distinct, with ltc's black replaced by a neutral grey placed last; `reading` is a softer 8-colour set.

## Value

A named list of character vectors, or a single character vector of hex colours when `name` is given.

## See Also

Other visualisation: `contributionHm()`, `imageR`, `quant_MSImagingExperiment-method`, `quantileHm()`

## Examples

```
names(quantPalettes())
quantPalettes("heatmap2")
quantPalettes("hat", n = 3)
```

---

`readMRM` *Read a Waters DESI-MRM acquisition into an MSImagingExperiment*

---

## Description

Reads one or more analyte text files from `<.raw>/imaging/`, matches transitions to the ion library on rounded (precursor, product) m/z, and constructs a Cardinal::MSImagingExperiment. Handles both the single-file case (one combined `Analyte 1.txt`, the modern QuanOptimise default) and the multi-file case (older exports that split transitions across several `Analyte N.txt` files) transparently.

## Usage

```
readMRM(
  name,
  folder,
  lib_ion_path,
  overwrite = TRUE,
  type_header = "Type",
  is_norm_header = "IS_norm",
  mz_tolerance = 0.4,
  ambiguity = c("combine", "error", "warn", "nearest")
)
```

## Arguments

| | |
|---|---|
| name | Acquisition name (without the `.raw` suffix). |
| folder | Path to the directory that contains the `<name>.raw/` folder. |
| lib_ion_path | Full path to the ion-library CSV. Must contain columns `transition_id`, `precursor_mz`, `product_mz`, `collision_eV`, `cone_V`, `Polarity`, `Type` (`"Analyte"` for in-tissue analytes, `"IS"` for internal standards). Additional metadata columns are preserved by downstream code via `buildFeatureMeta()`.<br>Measured transitions are matched to it on precursor and product m/z within `mz_tolerance`; a transition the library does not describe is kept, named by its instrument index and typed `"Unknown"`. |
| overwrite | Logical. When `TRUE` (default) the raw text files are re-parsed; when `FALSE` and a cached `MSImagingExperiment.rds` exists inside the `.raw` folder, it is returned instead. |
| type_header | Character. Name of the ion-library column holding the feature type – the values that mark internal standards versus analytes (default `"Type"`). Its contents become `fData()$feature_type`, which is what `int2response()` matches `IS_name` against. |
| is_norm_header | Character. Name of the optional ion-library column that maps each analyte to the internal standard normalising it (default `"IS_norm"`). Carried through to `fData()` when present, so that `int2response()` can use it when a panel has more than one standard. `"None"` (or `NULL`), or simply omitting the column, is correct for a single-standard panel. |
| mz_tolerance | Numeric. Half-width, in Da, within which a measured precursor and product are taken to be the library's. **Both** must agree; a row matching on precursor alone is not a match.<br>The default `0.4` is nominal-mass matching, because that is what the instrument does: a triple quadrupole running MRM selects Q1 and Q3 at unit resolution, so a product ion written as `308.1` and one written as `308.3` are the same measurement and no acquisition can separate them. Matching more tightly than the instrument resolves invents a distinction that is not in the data, and makes annotation depend on how many decimal places somebody typed into the library.<br>The consequence is that genuinely isobaric compounds now collide – which is correct, and is what `ambiguity = "combine"` is for. Tighten it only for a high-resolution method where Q3 really is selective. |

`ambiguity` Character. What to do when a measured transition matches more than one library entry within `mz_tolerance`, which m/z alone cannot resolve. Isomers sharing a nominal precursor and product are the usual cause, and at unit resolution they are indistinguishable by definition.

`"combine"` (default) names the feature for every entry it matched, joined with `" || "` – `"LTC4 || 14_15-LTC4"`. The joined name says the measurement is one of these, which is what the data supports; naming it for one of them asserts more than was measured. Annotation columns come from the closest entry. `"error"` refuses, which is right when a collision means the library is wrong rather than the chemistry ambiguous; `"warn"` takes the closest and says so; `"nearest"` takes the closest quietly.

Two library entries for the *same* compound are redundancy, not ambiguity. `"combine"` reports those in its message so they can be removed from the library rather than carried as a joined name forever.

## Details

Results are cached to `<.raw>/MSImagingExperiment.rds`; pass `overwrite = TRUE` (the default) to re-parse from the raw text files.

## Value

An `MSImagingExperiment` with feature metadata joined to the ion library, ready to pass into `generateTxtImages()` or `selectTissuePixels()`. `x` and `y` are grid indices counted from the acquisition's own first pixel; `pData()` also carries `x_stage` and `y_stage`, the stage position of each pixel in mm, which `stitchAcquisitions()` uses to place pieces of one tissue. A cached object written by an earlier version lacks those two columns until it is re-read with `overwrite = TRUE`.

## See Also

Other acquisition: `labelROIs()`, `removeBlankMzs`, `quant_MSImagingExperiment-method`, `selectTissuePixels()`, `trimMSI()`

## Examples

```
# Return a previously parsed acquisition from its cached .rds
folder <- system.file("extdata", package = "quantMSImageR")
lib <- system.file("extdata", "ion_library_pos04.csv",
                   package = "quantMSImageR")
obj <- readMRM("pos04_test", folder = folder, lib_ion_path = lib,
               overwrite = FALSE)
```

## removeBlankMzs,quant_MSImagingExperiment-method *Remove features without observed signal*

### Description

Drops any feature whose intensities are entirely zero or NA – typically a transition present in the acquisition method but never detected, or a panel row padded in when acquisitions were aligned. Reporting these alongside real measurements is how empty transitions reach a figure.

### Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
removeBlankMzs(MSIobject)
```

### Arguments

| | |
|---|---|
| MSIobject | A quant_MSImagingExperiment object. |

### Value

The input object with features carrying no data removed.

### See Also

Other acquisition: labelROIs(), readMRM(), selectTissuePixels(), trimMSI()

### Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- removeBlankMzs(obj)
```

## runExample *Run the quantMSImageR example study*

### Description

Executes the full DESI-MRM workflow on a small synthetic dataset bundled with the package. The dataset contains two samples – SampleA (circular tissue, sections 01-03) and SampleB (square tissue, sections 04-06) – acquired on a panel of eight oxylipin features (seven analytes + one internal standard, all negative-ion mode).

## Usage

```
runExample(
  render_report = TRUE,
  output_txt = FALSE,
  snr_thresh = 3,
  shapes = NULL,
  calibrate = TRUE
)
```

## Arguments

| | |
|---|---|
| render_report | Logical. Render and open the HTML report in the RStudio viewer or default browser? Requires pandoc. Default TRUE. |
| output_txt | Logical. Write per-feature .txt image matrices to a temporary directory? Default FALSE. |
| snr_thresh | Numeric. SNR threshold passed to generateTxtImages(). Default 3. |
| shapes | Character vector of tissue shapes to include – any of "circle", "square". NULL (the default) uses both, i.e. the full two-group example (group A circle + group B square). |
| calibrate | Logical. Also run the quantification demo on the bundled synthetic calibration standards (summariseCalLevels() -> createCalCurve() -> int2conc())? In a real study this step is driven by the YAML calibration: block. Default TRUE. |

## Details

Distinct tissue shapes per sample make the rendered ion images visually easy to distinguish, and the bundled ion library carries a Met-1 column (COX, LOX, IS) that drives the row-split colour bar in the report's main heatmap and the diagonal colour blocks in the correlation triangles.

All file paths are resolved automatically via system.file(), so the example works on any machine without editing a YAML.

## Value

Invisibly returns the list produced by generateTxtImages(), with an added calibrated element when calibrate = TRUE: the study sections with their tissue pixels converted to pg_pixel and pg_mm2 layers, restricted to the analytes that have a standard. When render_report = TRUE a report element gives the path to the rendered HTML.

## See Also

Other workflow: blockCentre(), generateTxtImages(), print.quantValidation(), runStudy(), validateConfig()

## Examples

```
# The workflow itself, without rendering or opening the HTML report

res <- runExample(render_report = FALSE, shapes = "circle")
```

```
names(res)


# The full demo: renders the report and opens it in the viewer/browser
if (interactive()) {
  res <- runExample()
  res$report        # path to the rendered HTML
}
```

---

runStudy *Run a full DESI-MRM study from a YAML configuration*

---

## Description

Executes the whole workflow for a study described by a single YAML file: loads every acquisition, applies the tissue masks, runs SNR filtering at each requested threshold, optionally builds calibration models and converts tissue pixels to estimated amounts, writes per-feature `.txt` images, and renders one HTML report per SNR threshold.

## Usage

```
runStudy(config_file)
```

## Arguments

| | |
|---|---|
| `config_file` | Path to the study YAML configuration. |

## Details

The configuration format is documented by the template shipped with the package:

```
file.show(system.file("config_template.yaml", package = "quantMSImageR"))
```

A study can also be run from the command line, which uses this function internally:

```
Rscript -e 'quantMSImageR::runStudy("config.yaml")'
```

## Value

Invisibly, the list produced by `generateTxtImages()` for the study, with a `calibrated` element added when the configuration's `calibration:` block is enabled. Called for its side effects: `.txt` images, the calibrated `.RDS` and the HTML reports written under the configured output paths.

## See Also

Other workflow: `blockCentre()`, `generateTxtImages()`, `print.quantValidation()`, `runExample()`, `validateConfig()`

## Examples

```
# A minimal study driven from a config, using the bundled synthetic data.
# snr_thresh = 0 skips SNR filtering and the tissue-mask requirement, and
# both outputs are switched off, so this runs without writing a report.

out <- file.path(tempdir(), "qmsi_study")
dir.create(out, showWarnings = FALSE)

cfg <- list(
  study = "example_study",
  paths = list(
    data_path    = system.file("extdata", package = "quantMSImageR"),
    out_path     = out,
    image_dir    = file.path(out, "images"),
    lib_ion_path = system.file("extdata", "example_ion_library.csv",
                               package = "quantMSImageR")
  ),
  samples = list(
    list(neg = "example", sections = list("section01"),
         run_id = "S1", label = "A")
  ),
  parameters = list(snr_thresh = 0),
  output = list(render_report = FALSE, output_txt = FALSE)
)

cfg_file <- file.path(out, "config.yaml")
yaml::write_yaml(cfg, cfg_file)

res <- runStudy(cfg_file)
names(res)
```

---

`selectTissuePixels` *Interactively select tissue pixels for an acquisition*

---

## Description

Loads a DESI-MRM acquisition, displays ion images of all features so the user can identify which best discriminates tissue from background, then opens an interactive ROI-selection window on a chosen feature. The resulting logical mask is saved as `tissue_pixels.csv` inside the acquisition's `.raw` folder, where `generateTxtImages()` will find it automatically.

## Usage

```
selectTissuePixels(
  name,
  data_path,
  lib_ion_path,
```

```
  feature = NULL,
  enhance = "histogram",
  overwrite = FALSE,
  preview = TRUE
)
```

## Arguments

| | |
|---|---|
| name | Character. Acquisition name, without the .raw suffix (e.g. "14Dec_AntibodyStudy_1"). |
| data_path | Character. Path to the folder that contains the .raw acquisition directory. |
| lib_ion_path | Character. Full path to the MRM ion-library CSV. |
| feature | Integer index or character name of the feature to use for tissue selection. When NULL (default), all features are displayed first and the user is prompted to choose one. |
| enhance | Character. Contrast enhancement passed to Cardinal::image(). Default "histogram". |
| overwrite | Logical. If TRUE, an existing tissue_pixels.csv will be overwritten. Default FALSE. |
| preview | Logical. Draw the saved mask (tissue versus background over the pixel coordinates) once selection finishes, so it can be checked before the workflow uses it. Default TRUE. |

## Details

This function must be run **before** the YAML workflow for each acquisition that does not yet have a tissue_pixels.csv. It requires an interactive R session with a graphics device (i.e. not inside Rscript --no-save).

## Value

Invisibly returns a data frame with columns x, y, tissue_pixels and background_pixels (logical), one row per pixel, in acquisition order.

## See Also

Other acquisition: labelROIs(), readMRM(), removeBlankMzs, quant_MSImagingExperiment-method, trimMSI()

## Examples

```
# Requires a graphics device and user input, so it only runs interactively.
if (interactive()) {
  # Step 1 (once per acquisition, before running the study):
  selectTissuePixels(
    name         = "my_acquisition",
    data_path    = "path/to/raw",
    lib_ion_path = "path/to/ion_library.csv"
  )

  # Step 2: run the full workflow from the study config
```

```
  runStudy("path/to/study.yaml")
}
```

---

`stitchAcquisitions` *Stitch acquisitions that are pieces of one tissue into a single sample*

---

## Description

A tissue larger than the stage range is sometimes acquired in two passes – top then bottom – producing two `.raw` folders that are *one* sample. This places their pixels on one grid and makes them a single run.

## Usage

```
stitchAcquisitions(..., label = NULL, overlap = c("error", "first", "last"))
```

## Arguments

| | |
|---|---|
| `...` | Two or more `MSImagingExperiment` objects, or a single list of them. All must carry the same features and the same spectra layers. |
| `label` | Character. Run identifier for the stitched sample. Defaults to the first object's run. |
| `overlap` | What to do when two pieces claim the same grid position. `"error"` (default) refuses, because averaging or silently dropping a pixel invents data. `"first"` keeps the pixel from the earlier piece in the argument list and drops the later one's; `"last"` is the reverse. Either is a reasonable reading when passes deliberately share a strip, and the number dropped is reported. |

## Details

It is not the same operation as either neighbour, and picking the wrong one is quiet rather than loud:

`bindPanels()` two MRM *panels* over one tissue: different transitions, shared pixel grid. It **intersects** pixels on `(x, y)`, so given two spatial halves it keeps only coordinates present in both – nothing, if the halves do not overlap.

`combineMSIs()` several *samples* into one object, each keeping its own run identifier. It refuses duplicate run identifiers, precisely to stop two sections silently becoming one sample.

`stitchAcquisitions()` several *pieces of one sample* into one run. The union of pixels, one run identifier.

Why this matters beyond the picture: two halves of one tissue are not two replicates. Left as separate samples they count twice in every group summary – the group mean, the box/violin plots, any downstream model – and because the halves agree with each other they look like a reproducible effect rather than one measurement.

## Value

One object holding the union of the input pixels, with `run` set to `label` throughout and `x`, `y` on the common grid.

## Where the pieces go

Each acquisition numbers its pixels from its own first pixel, so every piece starts at `(1, 1)` and the indices alone cannot say where one piece lies relative to another. `readMRM()` also keeps the stage position of every pixel (`x_stage`, `y_stage`, in mm), and that is what places the pieces: a piece acquired below, above or beside another lands there, whatever order the pieces are given in. The origin is the smallest stage position of any piece.

- **Same pixel size.** Pieces must share a pixel size on each axis (within 2%); otherwise there is no common grid to put them on, and this stops.
- **Nearest grid position.** A piece whose origin falls between grid positions of the first – the stage moved a fraction of a pixel – is moved to the nearest one. The largest such shift is reported, and is at most half a pixel.
- **Gaps stay empty.** Stage space between two pieces is not filled in; no pixel is invented.
- **Overlap is an error.** Pixels from two pieces landing on the same grid position mean the acquisitions overlap, or are not pieces of one tissue. Averaging them or keeping one would invent a measurement, so this stops.

Objects without stage positions – cached by an earlier version of `readMRM()`, or built another way – keep their own `(x, y)`, and any pixel two of them share is the same error. Re-read the acquisitions with `readMRM(overwrite = TRUE)` to place them by stage position instead.

## Regions of interest

Pieces labelled with `labelROIs()` each number their regions from `_01`, so `airway_01` in the top piece and `airway_01` in the bottom piece are different airways. They are renumbered across the stitched sample, in piece order, so every region keeps an identifier of its own.

## See Also

`bindPanels()`, `combineMSIs()`, `readMRM()`

Other combining acquisitions: `alignFeatures()`, `bindPanels()`, `combineMSIs`, `MSImagingExperiment-method`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- readRDS(p)
# Split by pixel, then stitch back: one sample, all pixels, one run.
half1 <- obj[, 1:50]
half2 <- obj[, 51:ncol(obj)]
whole <- stitchAcquisitions(half1, half2, label = "section01")
ncol(whole) == ncol(obj)
```

summariseCalLevels,quant_MSImagingExperiment-method
*Summarise the response at each calibration level*

## Description

Averages the signal across the pixels of each calibration spot, giving one response value per standard per level. This is the input `createCalCurve()` fits its models to.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
summariseCalLevels(
  MSIobject,
  cal_metadata,
  val_slot = "intensity",
  cal_header = "sample_type",
  cal_label = "Cal",
  id = "identifier"
)
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` whose `pData()` carries the calibration labels and spot identifiers. |
| `cal_metadata` | Data frame of calibration metadata, with columns `analyte` (feature name as in `fData()$name`), `identifier` (maps to `pData()`), `amount_pg` (amount of standard deposited at each spot) and `level` (dilution level). A legacy `lipid` column is read as `analyte`. |
| `val_slot` | Character. Spectra slot to summarise (default `"intensity"`). |
| `cal_header` | Character. Column of `pData()` used to select calibration pixels (default `"sample_type"`). |
| `cal_label` | Character. Value of `cal_header` marking calibration pixels (default `"Cal"`). |
| `id` | Character. Column present in both `cal_metadata` and `pData()` that identifies a unique calibration spot (default `"identifier"`). |

## Details

Every spot is also summarised for dispersion, because a mean and a pixel count cannot show whether a calibration level was measured reproducibly. The returned table carries `sd_response` and `cv_response` alongside the mean, so an imprecise level is visible rather than averaged away.

Problems in the calibration design are errors rather than quiet `NA`s: an analyte that is not in the object, an identifier that matches no pixel, a spot with no usable response, and duplicated identifiers that disagree about amount or level all stop the function. Each of these would otherwise produce a curve that looks fitted but is not anchored to the data it claims.

## Value

The input object with `cal_response_data` populated: one row per (standard, calibration spot) pair holding `response_perpixel` (the mean), `median_response`, `sd_response`, `cv_response`, `pixels`, `n_nonmissing` and `pg_perpixel`. Amounts are expressed in **pg per pixel**.

## See Also

Other calibration: `calibrationInfo-class`, `createCalCurve,quant_MSImagingExperiment-method`, `int2conc,quant_MSImagingExperiment-method`, `plotCalCoverage,quant_MSImagingExperiment-method`

## Examples

```
cal_dir <- system.file("extdata", "cal_example.raw", package = "quantMSImageR")
cal <- as(readRDS(file.path(cal_dir, "cal_MSI.RDS")),
          "quant_MSImagingExperiment")
cal_metadata <- read.csv(file.path(cal_dir, "calibration_metadata.csv"))
cal <- summariseCalLevels(cal, cal_metadata, val_slot = "intensity",
                          cal_label = "Cal", id = "identifier")
```

---

# tissueInfo-class *Tissue-level summaries for an imaging experiment*

---

## Description

Holds the tabular views of an imaging experiment produced by `createMSIDataMatrix()`: one row per pixel, and optionally one row per region of interest, together with the accompanying metadata. It is carried in the `tissueInfo` slot of a quant_MSImagingExperiment.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | A `tissueInfo` object. |

## Details

`all_pixel_matrix` `data.frame`, one row per pixel and one column per feature.

`roi_average_matrix` `data.frame`, one row per region of interest, populated only when `createMSIDataMatrix(roi_heade` is given.

`sample_metadata` `data.frame` describing the rows of the matrices above (sample, run, ROI identifier).

`feature_metadata` `data.frame` describing their columns.

All four are zero-row data frames until `createMSIDataMatrix()` has run. Access them with `tissueMatrix()` and `tissueData()` rather than `@`.

## Value

An object of class `tissueInfo`.

## Slots

`roi_average_matrix` `data.frame` of per-ROI averages.

`all_pixel_matrix` `data.frame` of per-pixel values.

`sample_metadata` `data.frame` describing the matrix rows.

`feature_metadata` `data.frame` describing the matrix columns.

## See Also

quant_MSImagingExperiment, `createMSIDataMatrix()`, quantMSImageR-accessors

## Examples

```
ti <- tissueInfo()

p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- createMSIDataMatrix(obj, val_slot = "intensity", roi_header = NA)
dim(tissueMatrix(obj, which = "pixel"))
```

---

# `trimMSI` *Remove all-background rows and columns from an MSI object*

---

## Description

Drops any x-column or y-row in which **every** pixel is labelled as `"background_pixels"` (or the historical `"noise_pixels"`) in `pData(MSI_data)$sample_name`. This trims the empty scan borders that arise from the Waters acquisition geometry without affecting tissue or mixed-content rows/columns.

## Usage

```
trimMSI(MSI_data)
```

## Arguments

| | |
|---|---|
| `MSI_data` | A `quant_MSImagingExperiment` object whose `pData()` contains columns `x`, `y`, and `sample_name` (populated by `makeFactor()`). |

### Details

The rule is applied to every row and column, not only the outer ones: an all-background row *between* two pieces of tissue is dropped too, closing the gap. What remains is set by the mask, so redrawing the tissue mask can change the size and origin of everything trimmed from it – including the `.txt` matrices written by `generateTxtImages()`.

### Value

A subset `quant_MSImagingExperiment` with the all-background rows and columns removed.

### See Also

Other acquisition: `labelROIs()`, `readMRM()`, `removeBlankMzs`, `quant_MSImagingExperiment-method`, `selectTissuePixels()`

### Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- trimMSI(obj)
```

---

## validateConfig *Validate a study YAML configuration*

---

### Description

Structural and cross-file checks on a study config before anything is processed: required blocks present, referenced files and ion-library columns actually exist, enumerated values spelled correctly, and the calibration and internal-standard settings coherent with each other.

### Usage

```
validateConfig(config, check_paths = TRUE)
```

### Arguments

| | |
|---|---|
| `config` | A config list, or a path to a study YAML file. |
| `check_paths` | Logical. Also check that the referenced data directory, ion library and calibration files exist on disk (default `TRUE`). Set `FALSE` to validate a config's structure away from its data. |

### Details

Every check runs, so one call reports every problem rather than stopping at the first. `runStudy()` calls this first and refuses to start if there are errors – these are mistakes that would otherwise surface as an obscure failure halfway through a long run, or worse, as a plausible-looking result computed from the wrong column.

## Value

A `quantValidation` object with `errors`, `warnings` and a `summary` data frame of every check.

## See Also

Other workflow: `blockCentre()`, `generateTxtImages()`, `print.quantValidation()`, `runExample()`, `runStudy()`

## Examples

```
# The shipped template is structurally valid; its placeholder paths are not
# meant to exist, so the path checks are skipped here.
cfg <- system.file("config_template.yaml", package = "quantMSImageR")
validateConfig(cfg, check_paths = FALSE)
```

---

# `zero2NA,quant_MSImagingExperiment-method`

*Replace zero intensities with NA*

---

## Description

Zeros in an MRM acquisition mean "nothing recorded" rather than "measured as zero", and left in place they bias means and compress colour scales. This converts them to `NA` so they are excluded from summaries.

## Usage

```
## S4 method for signature 'quant_MSImagingExperiment'
zero2NA(MSIobject, val_slot = "intensity")
```

## Arguments

| | |
|---|---|
| `MSIobject` | A `quant_MSImagingExperiment` object. |
| `val_slot` | Character. Spectra slot to modify (default `"intensity"`). |

## Value

The input object with zero values in `val_slot` replaced by `NA`.

## Destructive

This modifies `val_slot` in place rather than adding a new layer. Keep a copy of the object if the original values are needed afterwards.

## See Also

Other filtering: `applySNR,quant_MSImagingExperiment-method`, `back2NA,quant_MSImagingExperiment-method`, `int2SNR,quant_MSImagingExperiment-method`, `int2response,quant_MSImagingExperiment-method`

## Examples

```
p <- system.file("extdata", "example.raw", "section01.RDS",
                 package = "quantMSImageR")
obj <- as(readRDS(p), "quant_MSImagingExperiment")
obj <- zero2NA(obj, val_slot = "intensity")
```

# Index


∗ **accessors**
quantMSImageR-accessors, 42
∗ **acquisition**
labelROIs, 33
readMRM, 45
removeBlankMzs,quant_MSImagingExperiment-method, 48
selectTissuePixels, 51
trimMSI, 57
∗ **calibration**
calibrationInfo-class, 12
createCalCurve,quant_MSImagingExperiment-method, 18
int2conc,quant_MSImagingExperiment-method, 27
plotCalCoverage,quant_MSImagingExperiment-method, 35
summariseCalLevels,quant_MSImagingExperiment-method, 55
∗ **combining acquisitions**
alignFeatures, 3
bindPanels, 7
combineMSIs,MSImagingExperiment-method, 13
stitchAcquisitions, 53
∗ **filtering**
applySNR,quant_MSImagingExperiment-method, 5
back2NA,quant_MSImagingExperiment-method, 6
int2response,quant_MSImagingExperiment-method, 28
int2SNR,quant_MSImagingExperiment-method, 31
zero2NA,quant_MSImagingExperiment-method, 59
∗ **visualisation**
contributionHm, 14
imageR,quant_MSImagingExperiment-method, 25
quantileHm, 39
quantPalettes, 44
∗ **workflow**
blockCentre, 9
generateTxtImages, 21
print.quantValidation, 37
runExample, 48
runStudy, 50
validateConfig, 58

alignFeatures, 3
alignFeatures(), *8*, *14*, *54*
applySNR (applySNR,quant_MSImagingExperiment-method), 5
applySNR(), *24*
applySNR,quant_MSImagingExperiment-method, 5

back2NA (back2NA,quant_MSImagingExperiment-method), 6
back2NA(), *24*
back2NA,quant_MSImagingExperiment-method, 6
bindPanels, 7
bindPanels(), *4*, *14*, *22*, *53*, *54*
blockCentre, 9
blockCentre(), *24*, *37*, *49*, *50*, *59*
buildFeatureMeta, 10
buildFeatureMeta(), *46*

calibrationData (quantMSImageR-accessors), 42
calibrationData,quant_MSImagingExperiment-method (quantMSImageR-accessors), 42
calibrationData<- (quantMSImageR-accessors), 42

calibrationData<-,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationDiagnostics
(quantMSImageR-accessors), 42
calibrationDiagnostics(), 12, 19
calibrationDiagnostics,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationInfo, 38
calibrationInfo
(calibrationInfo-class), 12
calibrationInfo-class, 12
calibrationLevels
(quantMSImageR-accessors), 42
calibrationLevels(), 12
calibrationLevels,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationMetadata
(quantMSImageR-accessors), 42
calibrationMetadata(), 12
calibrationMetadata,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationModels
(quantMSImageR-accessors), 42
calibrationModels(), 12, 35
calibrationModels,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationModels<-
(quantMSImageR-accessors), 42
calibrationModels<-,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
calibrationR2
(quantMSImageR-accessors), 42
calibrationR2,quant_MSImagingExperiment-method
(quantMSImageR-accessors), 42
Cardinal::MSImagingExperiment, 45
combineMSIs, 3, 4
combineMSIs
(combineMSIs,MSImagingExperiment-method),
13
combineMSIs(), 8, 38, 53, 54
combineMSIs,MSImagingExperiment-method,
13
contributionHm, 14
contributionHm(), 26, 40–42, 45
createCalCurve
(createCalCurve,quant_MSImagingExperiment-method),
18
createCalCurve(), 12, 13, 27, 35, 36, 43, 55
createCalCurve,quant_MSImagingExperiment-method,
18
createMSIDataMatrix
(createMSIDataMatrix,quant_MSImagingExperiment-method),
20
createMSIDataMatrix(), 56, 57
createMSIDataMatrix,quant_MSImagingExperiment-method,
20

generateTxtImages, 4, 21
generateTxtImages(), 8, 10, 11, 37, 47,
49–51, 58, 59
grDevices::hcl.colors(), 16, 40

imageR
(imageR,quant_MSImagingExperiment-method),
25
imageR(), 24, 44
imageR,quant_MSImagingExperiment-method,
25
int2conc
(int2conc,quant_MSImagingExperiment-method),
27
int2conc(), 12, 18, 35, 36, 38, 43
int2conc,quant_MSImagingExperiment-method,
27
int2response
(int2response,quant_MSImagingExperiment-method),
28
int2response(), 23, 31, 32, 38, 46
int2response,quant_MSImagingExperiment-method,
28
int2SNR
(int2SNR,quant_MSImagingExperiment-method),
31
int2SNR(), 5, 22, 24, 38
int2SNR,quant_MSImagingExperiment-method,
31

labelROIs, 33
labelROIs(), 23, 47, 48, 52, 54, 58

plotCalCoverage
(plotCalCoverage,quant_MSImagingExperiment-method),
35
plotCalCoverage(), 19, 27
plotCalCoverage,quant_MSImagingExperiment-method,
35
print.quantValidation, 37

print.quantValidation(), *10*, *24*, *49, 50*, *59*

quant_MSImagingExperiment, *12, 13*, 37, *56, 57*
quant_MSImagingExperiment-class *(*quant_MSImagingExperiment*)*, 37
quantileHm, 39
quantileHm(), *11*, *14*, *17, 18*, *26*, *44, 45*
quantMSImageR-accessors, *13*, *38*, 42, *57*
quantPalettes, 44
quantPalettes(), *16*, *18*, *26*, *40*, *42*

readMRM, *3*, 45
readMRM(), *7, 8*, *11*, *23*, *34*, *48*, *52*, *54*, *58*
removeBlankMzs *(*removeBlankMzs,quant_MSImagingExperiment-method*)*, 48
removeBlankMzs,quant_MSImagingExperiment-method, 48
runExample, 48
runExample(), *10, 11*, *24*, *37*, *50*, *59*
runStudy, 50
runStudy(), *8*, *10*, *24*, *34*, *37*, *49*, *58, 59*

selectTissuePixels, 51
selectTissuePixels(), *33, 34*, *47, 48*, *58*
show,calibrationInfo-method *(*calibrationInfo-class*)*, 12
show,tissueInfo-method *(*tissueInfo-class*)*, 56
stitchAcquisitions, 53
stitchAcquisitions(), *4*, *8*, *14*, *22, 23*, *47*
summariseCalLevels *(*summariseCalLevels,quant_MSImagingExperiment-method*)*, 55
summariseCalLevels(), *12*, *19*, *43*
summariseCalLevels,quant_MSImagingExperiment-method, 55
system.file(), *49*

tissueData *(*quantMSImageR-accessors*)*, 42
tissueData(), *56*
tissueData,quant_MSImagingExperiment-method *(*quantMSImageR-accessors*)*, 42
tissueInfo, *38*
tissueInfo *(*tissueInfo-class*)*, 56
tissueInfo-class, 56
tissueMatrix *(*quantMSImageR-accessors*)*, 42
tissueMatrix(), *56*
tissueMatrix,quant_MSImagingExperiment-method *(*quantMSImageR-accessors*)*, 42
trimMSI, 57
trimMSI(), *24*, *34*, *47, 48*, *52*

validateConfig, 58
validateConfig(), *10*, *24*, *37*, *49, 50*

zero2NA *(*zero2NA,quant_MSImagingExperiment-method*)*, 59
zero2NA,quant_MSImagingExperiment-method, 59