# Package: SomaDataIO (via r-universe)

March 9, 2025

**Type** Package

**Title** Input/Output 'SomaScan' Data

**Version** 6.2.0

**Description** Load and export 'SomaScan' data via the 'Standard BioTools, Inc.' structured text file called an ADAT ('*.adat'). For file format see <https://github.com/SomaLogic/SomaLogic-Data/blob/main/README.md>. The package also exports auxiliary functions for manipulating, wrangling, and extracting relevant information from an ADAT object once in memory.

**License** MIT + file LICENSE

**URL** https://somalogic.github.io/SomaDataIO/, https://somalogic.com

**BugReports** https://github.com/SomaLogic/SomaDataIO/issues

**Depends** R (>= 4.1.0)

**Imports** cli, dplyr (>= 1.0.6), lifecycle (>= 1.0.0), magrittr (>= 2.0.1), methods, readxl (>= 1.3.1), tibble (>= 3.1.2), tidyr (>= 1.1.3)

**Suggests** Biobase, ggplot2, knitr, purrr, recipes, rlang, rmarkdown, spelling, testthat (>= 3.0.0), usethis (>= 2.0.1), withr

**VignetteBuilder** knitr

**Copyright** Standard BioTools, Inc. 2025

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**LazyDataCompression** xz

**LazyLoad** true

**Config/testthat/edition** 3

**Config/Needs/website** tidyverse/tidytemplate

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Stu Field [aut] (<https://orcid.org/0000-0002-1024-5859>),
Caleb Scheidel [cre], Standard BioTools, Inc. [cph, fnd]

**Maintainer** Caleb Scheidel <calebjscheidel@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-02-07 00:40:19 UTC

**Config/pak/sysreqs** libicu-dev

# Contents

adat-helpers *Helpers to Extract Information from an ADAT*

## Description

Retrieve elements of the HEADER attribute of a soma_adat object:

getAdatVersion() determines the the ADAT version number from a parsed ADAT header.

getSomaScanVersion() determines the original SomaScan assay version that generated RFU measurements within a soma_adat object.

checkSomaScanVersion() determines if the version of is a recognized version of SomaScan.

Table of SomaScan assay versions:

| Version | Commercial Name | Size |
|---|---|---|
| V4 | 5k | 5284 |
| v4.1 | 7k | 7596 |
| v5.0 | 11k | 11083 |

getSignalSpace() determines the current signal space of the RFU values, which may differ from the original SomaScan signal space if the data have been lifted. See lift_adat() and vignette("lifting-and-bridging" package = "SomaDataIO").

getSomaScanLiftCCC() accesses the lifting Concordance Correlation Coefficients between various SomaScan versions. For more about CCC metrics see lift_adat().

## Usage

```
getAdatVersion(x)

getSomaScanVersion(adat)

getSignalSpace(adat)

checkSomaScanVersion(ver)

getSomaScanLiftCCC(matrix = c("plasma", "serum"))
```

## Arguments

| | |
|---|---|
| x | Either a soma_adat object with intact attributes or the attributes themselves of a soma_adat object. |
| adat | A soma_adat object (with intact attributes), typically created using read_adat(). |
| ver | character(1). The SomaScan version as a string. **Note:** the "v"-prefix is case *in*sensitive. |
| matrix | Character. A string of (usually) either "serum" or "plasma". |

## Value

`\link[=getAdatVersion]{getAdatVersion()}`
: The key-value of the `Version` as a string.

`\link[=getSomaScanVersion]{getSomaScanVersion()}`
: The key-value of the `AssayVersion` as a string.

`\link[=getSignalSpace]{getSignalSpace()}`
: The key-value of the `SignalSpace` as a string.

`\link[=checkSomaScanVersion]{checkSomaScanVersion()}`
: Returns `NULL` (invisibly) if checks pass.

`\link[=getSomaScanLiftCCC]{getSomaScanLiftCCC()}`
: Returns a tibble of either the `serum` or `plasma` CCC between various versions of the SomaScan assay.

## Author(s)

Stu Field

## References

Lin, Lawrence I-Kuei. 1989. A Concordance Correlation Coefficient to Evaluate Reproducibility. **Biometrics**. 45:255-268.

## Examples

```
getAdatVersion(example_data)

attr(example_data, "Header.Meta")$HEADER$Version <- "99.9"
getAdatVersion(example_data)

ver <- getSomaScanVersion(example_data)
ver

rfu_space <- getSignalSpace(example_data)
rfu_space

is.null(checkSomaScanVersion(ver))

# plasma (default)
getSomaScanLiftCCC()

# serum
getSomaScanLiftCCC("serum")
```

adat2eSet *Convert ADAT to ExpressionSet Object*

## Description

Utility to convert a SomaLogic soma_adat object to an ExpressionSet object via the **Biobase** package from **Bioconductor**: https://www.bioconductor.org/packages/release/bioc/html/Biobase.html.

## Usage

```
adat2eSet(adat)
```

## Arguments

| | |
|---|---|
| adat | A soma_adat class object as read into the R environment using read_adat(). |

## Details

The **Biobase** package is required and must be installed from **Bioconductor** via the following at the R console:

```
if (!requireNamespace("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}
BiocManager::install("Biobase", version = remotes::bioc_version())
```

## Value

A Bioconductor object of class ExpressionSet.

## Author(s)

Stu Field

## References

https://bioconductor.org/install/

## See Also

Other eSet: pivotExpressionSet()

## Examples

```
eSet <- adat2eSet(example_data)
class(eSet)
eSet

ft <- Biobase::exprs(eSet)
head(ft[, 1:10L], 10L)
```

---

addAttributes *Add Attributes to* soma_adat *Objects*

---

## Description

Adds a set of attributes, typically "Header.Meta" and "Col.Meta", to a data.frame, tibble, soma_adat or similar tabular object. Existing attributes data are *not* over-written. Typically untouched are:

- names
- class
- row.names

## Usage

```
addAttributes(data, new.atts)
```

## Arguments

| | |
|---|---|
| data | The *receiving* data.frame object for new attributes. |
| new.atts | A *named* list object containing new attributes to add to the existing ones. |

## Value

A data frame object corresponding to data but with the attributes of new.atts grafted on to it. Existing attribute names are *not* over-written.

## Author(s)

Stu Field

## See Also

attr(), setdiff()

addClass *Add a Class to an Object*

## Description

Utility to add (prepend) a class(es) to existing objects.

## Usage

```
addClass(x, class)
```

## Arguments

| | |
|---|---|
| x | The object to receive new class(es). |
| class | Character. The name of additional class(es). |

## Value

An object with new classes.

## Author(s)

Stu Field

## See Also

class(), typeof(), structure()

## Examples

```
class(iris)

addClass(iris, "new") |> class()

addClass(iris, c("A", "B")) |> class()    # 2 classes

addClass(iris, c("A", "data.frame")) |> class()    # no duplicates

addClass(iris, c("data.frame", "A")) |> class()    # re-orders if exists
```

`calc_eLOD` *Calculate Estimated Limit of Detection (eLOD)*

## Description

Calculate the estimated limit of detection (eLOD) for SOMAmer reagent analytes in the provided input data. The input data should be filtered to include only buffer samples desired for eLOD calculation.

## Usage

```
calc_eLOD(data)
```

## Arguments

| | |
|---|---|
| `data` | A `soma_adat`, `data.frame`, or `tibble` object including SeqId columns (`seq.xxxxx.xx`) containing RFU values. |

## Details

eLOD is calculated using the following steps:

1. For each SOMAmer, the median and adjusted median absolute deviation ($MAD_{Adjusted}$) are calculated, where

$$MAD_{Adjusted} = 1.4826 * MAD$$

   The 1.4826 is a set constant used to adjust the MAD to be reflective of the standard deviation of the normal distribution.

2. For each SOMAmer, calculate

$$eLOD = median + 3.3 * MAD_{Adjusted}$$

Note: The eLOD is useful for non-core matrices, including cell lysate and CSF, but should be used carefully for evaluating background signal in plasma and serum.

## Value

A `tibble` object with 2 columns: SeqId and eLOD.

## Author(s)

Caleb Scheidel, Christopher Dimapasok

## Examples

```
# filter data frame using vector of SampleId controls
df <- withr::with_seed(101, {
  data.frame(
    SampleType = rep(c("Sample", "Buffer"), each = 10),
    SampleId = paste0("Sample_", 1:20),
    seq.20.1.100 = runif(20, 1, 100),
    seq.21.1.100 = runif(20, 1, 100),
    seq.22.2.100 = runif(20, 1, 100)
  )
})
sample_ids <- paste0("Sample_", 11:20)
selected_samples <- df |> filter(SampleId %in% sample_ids)

selected_elod <- calc_eLOD(selected_samples)
head(selected_elod)
## Not run:
# filter `soma_adat` object to buffer samples
buffer_samples <- example_data |> filter(SampleType == "Buffer")

# calculate eLOD
buffer_elod <- calc_eLOD(buffer_samples)
head(buffer_elod)

# use eLOD to calculate signal to noise ratio of samples
samples_median <- example_data |> dplyr::filter(SampleType == "Sample") |>
  dplyr::summarise(across(starts_with("seq"), median, .names = "median_{col}")) |>
  tidyr::pivot_longer(starts_with("median_"), names_to = "SeqId",
                      values_to = "median_signal") |>
  dplyr::mutate(SeqId = gsub("median_seq", "seq", SeqId))

# analytes with signal to noise > 2
ratios <- samples_median |>
  dplyr::mutate(signal_to_noise = median_signal / buffer_elod$eLOD) |>
  dplyr::filter(signal_to_noise > 2) |>
  dplyr::arrange(desc(signal_to_noise))

head(ratios)

## End(Not run)
```

---

`cleanNames` *Clean Up Character String*

---

## Description

Often the names, particularly within `soma_adat` objects, are messy due to varying inputs, this function attempts to remedy this by removing the following:

- trailing/leading/internal whitespace

- non-alphanumeric strings (except underscores)
- duplicated internal dots (. .), (. . .), etc.
- SomaScan normalization scale factor format

### Usage

```
cleanNames(x)
```

### Arguments

| | |
|---|---|
| x | Character. String to clean up. |

### Value

A cleaned up character string.

### Author(s)

Stu Field

### See Also

trimws(), gsub(), sub()

### Examples

```
cleanNames("    sdkfj...sdlkfj.sdfii4994### ")

cleanNames("Hyb..Scale")
```

---

## Col.Meta *Analyte Annotations, Col.Meta, and Row Info*

---

### Description

In a standard SomaLogic ADAT, the section of information that sits directly above the measurement data (RFU data matrix) is the column meta data (Col.Meta), which contains detailed information and annotations about the analytes, SeqId()s, and their targets. See section below for further information about available fields and their descriptions. Use getAnalyteInfo() to obtain an object containing this information for programmatic analyses, and use getMeta() to obtain the column names representing the row-specific meta data about the samples (see section below).

### Col Meta (Analyte Annotations)

Information describing the *analytes* is found to the above the data matrix in a standard SomaLogic ADAT. This information may consist of the any or all of the following:

| Field | Description | Examp |
|---|---|---|
| SeqId | SomaLogic sequence identifier | 2182-5 |
| SeqidVersion | Version of SOMAmer sequence | 2 |
| SomaId | Target identifier, of the form SLnnnnnn (8 characters in length) | SL0003 |
| TargetFullName | Target name curated for consistency with UniProt name | Comple |
| Target | SomaLogic Target Name | C4b |
| UniProt | UniProt identifier(s) | P0C0L4 |
| EntrezGeneID | Entrez Gene Identifier(s) | 720 721 |
| EntrezGeneSymbol | Entrez Gene Symbol names | C4A C4 |
| Organism | Protein Source Organism | Human |
| Units | Relative Fluorescence Units | RFU |
| Type | SOMAmer target type | Protein |
| Dilution | Dilution mix assignment | 0.01% |
| PlateScale_Reference | PlateScale reference value | 1378.85 |
| CalReference | Calibration sample reference value | 1378.85 |
| medNormRef_ReferenceRFU | Median normalization reference value | 490.342 |
| Cal_V4_<YY>_<SSS>_<PPP> | Calibration scale factor (for given Year_Study_Plate) | 0.64 |
| ColCheck | QC acceptance criteria across all plates/sets | PASS |
| QcReference_<LLLLL> | QC sample reference value (for given QC lot) | PASS |
| CalQcRatio_V4_<YY>_<SSS>_<PPP> | Post calibration median QC ratio to reference (for given Year_Study_Plate) | 1.04 |

**Row Meta (Sample Annotations)**

Information describing the *samples* is typically found to the left of the data matrix in a standard SomaLogic ADAT. This information may consist of clinical information provided by the client, or run-specific diagnostic information included for assay quality control. Below are some examples of what may be present in this section:

| Field | Description | Examples |
|---|---|---|
| PlateId | Plate identifier | V4-18-004_001, V4-18-004_002 |
| ScannerID | Scanner used to analyze slide | SG12064173, SG14374437 |
| PlatePosition | Location on 96 well plate (A1-H12) | A1, H12 |
| SlideId | Agilent slide barcode | 2.58E+11 |
| Subarray | Agilent subarray (1 – 8) | 1,8 |
| SampleId | 1st form is Subject Identifier, 2nd form (calibrators, buffers) | 2031 |
| SampleType | 1st form for clinical samples (Sample), 2nd form as above | Sample, QC, Calibrator, Buffer |
| PercentDilution | Highest concentration the SOMAmer dilution groups | 20 |
| SampleMatrix | Sample matrix | Plasma-PPT |
| Barcode | 1D Barcode of aliquot | S622225 |
| Barcode2d | 2D Barcode of aliquot | 1.91E+08 |
| SampleNotes | Assay team sample observation | Cloudy, Low sample volume, Reddis |
| SampleDescription | Supplemental sample information | Plasma QC 1 |
| AssayNotes | Assay team run observation | Beads aspirated, Leak/Hole, Smear |
| TimePoint | Sample time point | Baseline |
| ExtIdentifier | Primary key for Subarray | EXID40000000032037 |
| SsfExtId | Primary key for sample | EID102733 |
| SampleGroup | Sample group | A, B |

| | | |
|---|---|---|
| SiteId | Collection site | SomaLogic, Covance |
| TubeUniqueID | Unique tube identifier | 1.12E+11 |
| CLI | Cohort definition identifier | CLI6006F001 |
| HybControlNormScale | Hybridization control scale factor | 0.948304 |
| RowCheck | Normalization acceptance criteria for all row scale factors | PASS, FLAG |
| NormScale_0_5 | Median signal normalization scale factor (0.5% mix) | 1.02718 |
| NormScale_0_005 | Median signal normalization scale factor (0.005% mix) | 1.119754 |
| NormScale_20 | Median signal normalization scale factor (20% mix) | 0.996148 |

## Examples

```
# Annotations/Col.Meta
tbl <- getAnalyteInfo(example_data)
tbl

# Row/sample Meta
r_m <- getMeta(example_data)
head(r_m)

# Normalization Scale Factors
grep("NormScale", r_m, value = TRUE)

# adat subset
example_data[1:3, head(r_m)]
```

---

`diffAdats` *Diff Two ADAT Objects*

---

## Description

Diff tool for the differences between two `soma_adat` objects. When diffs of the table *values* are interrogated, **only** the intersect of the column meta data or feature data is considered

## Usage

```
diffAdats(adat1, adat2, tolerance = 1e-06)
```

## Arguments

| | |
|---|---|
| `adat1`, `adat2` | Two `soma_adat` objects to compare. |
| `tolerance` | Numeric > `0`. Differences smaller than tolerance are not triggered. See `all.equal()`. |

## Value

`NULL`, invisibly. Called for side effects.

## Note

Only diffs of the column name *intersect* are reported.

## Author(s)

Stu Field

## Examples

```
# subset `example_data` for speed
# all SeqIds from 2000 -> 2999
seqs <- grep("^seq\\.2[0-9]{3}", names(example_data), value = TRUE)
ex_data_small <- head(example_data[, c(getMeta(example_data), seqs)], 10L)
dim(ex_data_small)

# no diff to itself
diffAdats(ex_data_small, ex_data_small)

# remove random column
rm <- withr::with_seed(123, sample(1:ncol(ex_data_small), 1))
diffAdats(ex_data_small, ex_data_small[, -rm])

# randomly shuffle Subarray
diffAdats(ex_data_small, dplyr::mutate(ex_data_small, Subarray = sample(Subarray)))

# modify 2 RFUs randomly
new <- ex_data_small
new[5L, c(rm, rm + 1L)] <- 999
diffAdats(ex_data_small, new)
```

---

# getAnalyteInfo *Get Analyte Annotation Information*

---

## Description

Uses the `Col.Meta` attribute (analyte annotation data that appears above the protein measurements in the `*.adat` text file) of a `soma_adat` object, adds the `AptName` column key, conducts a few sanity checks, and generates a "lookup table" of analyte data that can be used for simple manipulation and indexing of analyte annotation information. Most importantly, the analyte column names of the `soma_adat` (e.g. `seq.XXXX.XX`) become the `AptName` column of the lookup table and represents the key index between the table and `soma_adat` from which it comes.

## Usage

```
getAnalyteInfo(adat)

getTargetNames(tbl)

getFeatureData(adat)
```

## Arguments

| | |
|---|---|
| adat | A soma_adat object (with intact attributes), typically created using read_adat(). |
| tbl | A tibble object containing analyte target annotation information. This is usually the result of a call to getAnalyteInfo(). |

## Value

A tibble object with columns corresponding to the column meta data entries in the soma_adat. One row per analyte.

## Functions

- getTargetNames(): creates a lookup table (or dictionary) as a named list object of AptNames and Target names in key-value pairs. This is a convenient tool to quickly access a TargetName given the AptName in which the key-value pairs map the seq.XXXX.XX to its corresponding TargetName in tbl. This structure which provides a convenient auto-completion mechanism at the command line or for generating plot titles.
- getFeatureData(): **[Superseded]**. Please now use getAnalyteInfo().

## Author(s)

Stu Field

## See Also

getAnalytes(), is_intact_attr(), read_adat()

## Examples

```
# Get Aptamer table
anno_tbl <- getAnalyteInfo(example_data)
anno_tbl

# Use `dplyr::group_by()`
dplyr::tally(dplyr::group_by(anno_tbl, Dilution))  # print summary by dilution

# Columns containing "Target"
anno_tbl |>
  dplyr::select(dplyr::contains("Target"))

# Rows of "Target" starting with MMP
anno_tbl |>
  dplyr::filter(grepl("^MMP", Target))

# Target names
tg <- getTargetNames(anno_tbl)

# how to use for plotting
feats <- sample(anno_tbl$AptName, 6)
op <- par(mfrow = c(2, 3))
```

```
sapply(feats, function(.x) plot(1:10, main = tg[[.x]]))
par(op)
```

---

`getAnalytes` *Get Analytes*

---

## Description

Return the feature names (i.e. the column names for SOMAmer reagent analytes) from a `soma_adat`. S3 methods also exist for these classes:

```
#> [1] getAnalytes.character  getAnalytes.data.frame getAnalytes.default
#> [4] getAnalytes.list       getAnalytes.matrix     getAnalytes.recipe
#> [7] getAnalytes.soma_adat
#> see '?methods' for accessing help and source code
```

`getMeta()` returns the inverse, a character vector of string names of *non*-analyte feature columns/variables, which typically correspond to the clinical ("meta") data variables. S3 methods exist for these classes:

```
#> [1] getMeta.character  getMeta.data.frame getMeta.default    getMeta.list
#> [5] getMeta.matrix     getMeta.soma_adat
#> see '?methods' for accessing help and source code
```

## Usage

```
getAnalytes(x, n = FALSE, rm.controls = FALSE)

getMeta(x, n = FALSE)

getFeatures(x, n = FALSE, rm.controls = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | Typically a `soma_adat` class object created using `read_adat()`. |
| `n` | Logical. Return an integer corresponding to the *length* of the features? |
| `rm.controls` | Logical. Should all control and non-human analytes (e.g. `HybControls`, `Non-Human`, `Non-Biotin`, `Spuriomer`) be removed from the returned value? |

## Value

`getAnalytes()`: a character vector of ADAT feature ("analyte") names.

`getMeta()`: a character vector of ADAT clinical ("meta") data names.

For both, if `n = TRUE`, an integer corresponding to the **length** of the character vector.

### Functions

- `getFeatures()`: **[Superseded]**. Please now use `getAnalytes()`.

### Author(s)

Stu Field

### See Also

`is.apt()`

### Examples

```
# RFU feature variables
apts <- getAnalytes(example_data)
head(apts)
getAnalytes(example_data, n = TRUE)

# vector string
bb <- getAnalytes(names(example_data))
all.equal(apts, bb)

# create some control sequences
# ~~~~~~~~~ Spuriomer ~~~ HybControl ~~~
apts2 <- c("seq.2053.2", "seq.2171.12", head(apts))
apts2
no_crtl <- getAnalytes(apts2, rm.controls = TRUE)
no_crtl
setdiff(apts2, no_crtl)

# clinical variables
mvec <- getMeta(example_data)
head(mvec, 10)
getMeta(example_data, n = TRUE)

# test 'data.frame' and 'character' S3 methods are identical
identical(getMeta(example_data), getMeta(names(example_data))) # TRUE
```

---

## groupGenerics *Group Generics for* soma_adat *Class Objects*

---

### Description

S3 group generic methods to apply group specific prototype functions to the RFU data **only** of `soma_adat` objects. The clinical meta data are *not* transformed and remain unmodified in the returned object (`Math()` and `Ops()`) or are ignored for the `Summary()` group. See `groupGeneric()`.

## Usage

```
## S3 method for class 'soma_adat'
Math(x, ...)

antilog(x, base = 10)

## S3 method for class 'soma_adat'
Ops(e1, e2 = NULL)

## S3 method for class 'soma_adat'
Summary(..., na.rm = FALSE)

## S3 method for class 'soma_adat'
e1 == e2
```

## Arguments

| | |
|---|---|
| x | The soma_adat class object to perform the transformation. |
| ... | Additional arguments passed to the various group generics as appropriate. |
| base | A positive or complex number: the base with respect to which logarithms are computed. |
| e1, e2 | Objects. |
| na.rm | Logical. Should missing values be removed? |

## Value

A soma_adat object with the same dimensions of the input object with the feature columns transformed by the specified generic.

## Functions

- antilog(): performs the inverse or anti-log transform for a numeric vector of soma_adat object. **note:** default is base = 10, which differs from the log() default base *e*.
- Ops(soma_adat): performs binary mathematical operations on class soma_adat. See Ops().
- Summary(soma_adat): performs summary calculations on class soma_adat. See Summary().
- == : compares left- and right-hand sides of the operator *unless* the RHS is also a soma_adat, in which case diffAdats() is invoked.

## Math

Group members:

```
#>  [1] "abs"      "acos"     "acosh"    "asin"     "asinh"    "atan"
#>  [7] "atanh"    "ceiling"  "cos"      "cosh"     "cospi"    "cummax"
#> [13] "cummin"   "cumprod"  "cumsum"   "digamma"  "exp"      "expm1"
#> [19] "floor"    "gamma"    "lgamma"   "log"      "log10"    "log1p"
#> [25] "log2"     "sign"     "sin"      "sinh"     "sinpi"    "sqrt"
#> [31] "tan"      "tanh"     "tanpi"    "trigamma" "trunc"
```

Commonly used generics of this group include:

- log(), log10(), log2(), antilog(), abs(), sign(), floor(), sqrt(), exp()

## Ops

Group members:

```
#>  [1] "+"   "-"   "*"   "^"   "%%"  "%/%" "/"   "=="  ">"   "<"   "!="  "<="
#> [13] ">="
```

Note that for the `==` method if the RHS is also a soma_adat, diffAdats() is invoked which compares LHS vs. RHS. Commonly used generics of this group include:

- +, -, *, /, ^, ==, >, <

## Summary

Group members:

```
#> [1] "all"   "any"   "max"   "min"   "prod"  "range" "sum"
```

Commonly used generics of this group include:

- max(), min(), range(), sum(), any()

## Author(s)

Stu Field

## See Also

groupGeneric(), getGroupMembers(), getGroup()

## Examples

```
# subset `example_data` for speed
# all SeqIds from 2000 -> 2999
seqs <- grep("^seq\\.2[0-9]{3}", names(example_data), value = TRUE)
ex_data_small <- head(example_data[, c(getMeta(example_data), seqs)], 10L)
dim(ex_data_small)

ex_data_small$seq.2991.9

# Math Generics:
# -------------
# log-transformation
a <- log(ex_data_small)
a$seq.2991.9

b <- log10(ex_data_small)
b$seq.2991.9
```

```
isTRUE(all.equal(b, log(ex_data_small, base = 10)))

# floor
c <- floor(ex_data_small)
c$seq.2991.9

# square-root
d <- sqrt(ex_data_small)
d$seq.2991.9

# rounding
e <- round(ex_data_small)
e$seq.2991.9

# inverse log
antilog(1:4)

alog <- antilog(b)
all.equal(ex_data_small, alog)    # return `b` -> linear space

# Ops Generics:
# -------------
plus1 <- ex_data_small + 1
times2 <- ex_data_small * 2

sq <- ex_data_small^2
all.equal(sqrt(sq), ex_data_small)

gt100k <- ex_data_small > 100000
gt100k

ex_data_small == ex_data_small   # invokes diffAdats()

# Summary Generics:
# -------------
sum(ex_data_small)

any(ex_data_small < 100)  # low RFU analytes

sum(ex_data_small < 100)  # how many

min(ex_data_small)

min(ex_data_small, 0)

max(ex_data_small)

max(ex_data_small, 1e+7)

range(ex_data_small)
```

`is_intact_attr` *Are Attributes Intact?*

## Description

This function runs a series of checks to determine if a `soma_adat` object has a complete set of attributes. If not, this indicates that the object has been modified since the initial `read_adat()` call. Checks for the presence of both "Header.Meta" and "Col.Meta" in the attribute names. These entries are added during the `read_adat()` call. Specifically, within these sections it also checks for the presence of the following entries:

**"Header.Meta" section:** "HEADER", "COL_DATA", and "ROW_DATA"

**"Col.Meta" section:** "SeqId", "Target", "Units", and "Dilution"

If any of the above they are altered or missing, `FALSE` is returned.

`is.intact.attributes()` is **[Superseded]**. It remains for backward compatibility and may be removed in the future. You are encouraged to shift your code to `is_intact_attr()`.

## Usage

```
is_intact_attr(adat, verbose = interactive())

is.intact.attributes(adat, verbose = interactive())
```

## Arguments

| | |
|---|---|
| `adat` | A `soma_adat` object to query. |
| `verbose` | Logical. Should diagnostic information about failures be printed to the console? If the default, see `interactive()`, is invoked, only messages via direct calls are triggered. This prohibits messages generated deep in the call stack from bubbling up to the user. |

## Value

Logical. `TRUE` if all checks pass, otherwise `FALSE`.

## See Also

`attributes()`

## Examples

```
# checking attributes
my_adat <- example_data
is_intact_attr(my_adat)           # TRUE
is_intact_attr(my_adat[, -303L])   # doesn't break atts; TRUE
attributes(my_adat)$Col.Meta$Target <- NULL    # break attributes
is_intact_attr(my_adat)  # FALSE (Target missing)
```

## is_seqFormat *Test* AptName *Format*

### Description

Test whether an object is in the new seq.XXXX.XX format.

### Usage

```
is_seqFormat(x)
```

### Arguments

| | |
|---|---|
| x | The object to be tested. |

### Value

A logical indicating whether x contains AptNames consistent with the new format, beginning with a seq. prefix.

### Author(s)

Stu Field, Eduardo Tabacman

### Examples

```
# character S3 method
is_seqFormat(names(example_data))   # no; meta data not ^seq.
is_seqFormat(tail(names(example_data), -20L))   # yes

# soma_adat S3 method
is_seqFormat(example_data)
```

## lift_adat *Lift an ADAT Between Assay Versions*

### Description

The SomaScan platform continually improves its technical processes between assay versions. The primary change of interest is content expansion, and other protocol changes may be implemented including: changing reagents, liquid handling equipment, and well volumes.

Table of SomaScan assay versions:

| Version | Commercial Name | Size |
|---|---|---|
| V4 | 5k | 5284 |
| v4.1 | 7k | 7596 |

| v5.0 | 11k | 11083 |
|---|---|---|

However, for a given analyte, these technical upgrades can result in minute measurement signal differences, requiring a calibration (aka "lifting" or "bridging") to bring RFUs into a comparable signal space. This is accomplished by applying an analyte-specific scalar, a linear transformation, to each analyte RFU measurement (column). If you have an annotations file (`*.xlsx`) and wish to examine the bridging scalars themselves, please see `read_annotations()`.

Lifting between SomaScan versions no longer requires an annotations file containing lifting scalars. We now enable users to pass a `bridge` parameter, indicating the direction of the bridge. For example, to "lift" between `11k` -> `7k`, you *must* be acting on SomaScan data in `11k` RFU space and would pass `bridge = "11k_to_7k"`. Likewise, `7k` -> `5k` requires `bridge = "7k_to_5k"`. Lastly, you may also lift directly from `11k` -> `5k` (aka "double-bridge") with `bridge = "11k_to_5k"`. See below for all options for the `bridge` argument.

## Usage

```
lift_adat(
  adat,
 bridge = c("11k_to_7k", "11k_to_5k", "7k_to_11k", "7k_to_5k", "5k_to_11k", "5k_to_7k"),
  anno.tbl = deprecated()
)

is_lifted(adat)
```

## Arguments

| | |
|---|---|
| `adat` | A `soma_adat` object (with intact attributes), typically created using `read_adat()`. |
| `bridge` | The direction of the lift (i.e. bridge). |
| `anno.tbl` | **[Deprecated]**. Please now use the `bridge` argument. |

## Details

Matched samples across assay versions are used to calculate bridging scalars. For each analyte, this scalar is computed as the ratio of population *medians* across assay versions. Please see the lifting vignette `vignette("lifting-and-bridging", package = "SomaDataIO")` for more details.

## Value

`lift_adat()`: A "lifted" `soma_adat` object corresponding to the scaling requested in the `bridge` parameter. RFU values are rounded to 1 decimal place to match standard SomaScan delivery format.

`is_lifted()`: Logical. Whether the RFU values in a `soma_adat` have been lifted from its original signal space to a new signal space.

### Lin's CCC

The Lin's Concordance Correlation Coefficient (CCC) is calculated by computing the correlation between post-lift RFU values and the RFU values generated on the original SomaScan version. This CCC estimate is a measure of how well an analyte can be bridged across SomaScan versions. See `vignette("lifting-and-bridging", package = "SomaDataIO")`. As with the lifting scalars, if you have an annotations file you may view the analyte-specific CCC values via `read_annotations()`. Alternatively, `getSomaScanLiftCCC()` retrieves these values from an internal object for both `"serum"` and `"plasma"`.

### Analyte Setdiff

- Newer versions of SomaScan typically have additional content, i.e. new reagents added to the multi-plex assay that bind to additional proteins. When lifting *to* a previous SomaScan version, new reagents that do *not* exist in the "earlier" assay version assay are scaled by 1.0, and thus maintained, unmodified in the returned object. Users may need to drop these columns in order to combine these data with a previous study from an earlier SomaScan version, e.g. with `collapseAdats()`.
- In the inverse scenario, lifting "forward" *from* a previous, lower-plex version, there will be extra reference values that are unnecessary to perform the lift, and a warning is triggered. The resulting data consists of RFU data in the "new" signal space, but with fewer analytes than would otherwise be expected (e.g. `11k` space with only 5284 analytes; see example below).

### References

Lin, Lawrence I-Kuei. 1989. A Concordance Correlation Coefficient to Evaluate Reproducibility. **Biometrics**. 45:255-268.

### Examples

```
# `example_data` is SomaScan (V4, 5k)
adat <- head(example_data, 3L)
dim(adat)

getSomaScanVersion(adat)

getSignalSpace(adat)

# perform 'lift'
lift_11k <- lift_adat(adat, "5k_to_11k")  # warning

is_lifted(lift_11k)

dim(lift_11k)

# attributes updated to reflect the 'lift'
attr(lift_11k, "Header")$HEADER$SignalSpace

attr(lift_11k, "Header")$HEADER$ProcessSteps
```

loadAdatsAsList *Load ADAT files as a list*

## Description

Load a series of ADATs and return a list of soma_adat objects, one for each ADAT file. collapseAdats() concatenates a list of ADATs from loadAdatsAsList(), while maintaining the relevant attribute entries (mainly the HEADER element). This makes writing out the final object possible without the loss of HEADER information.

## Usage

```
loadAdatsAsList(files, collapse = FALSE, verbose = interactive(), ...)

collapseAdats(x)
```

## Arguments

| | |
|---|---|
| files | A character string of files to load. |
| collapse | Logical. Should the resulting list of ADATs be collapsed into a single ADAT object? |
| verbose | Logical. Should the function call be run in *verbose* mode. |
| ... | Additional arguments passed to read_adat(). |
| x | A list of soma_adat class objects returned from loadAdatsAsList(). |

## Details

**Note 1:** The default behavior is to "vertically bind" (rbind()) on the *intersect* of the column variables, with unique columns silently dropped.

**Note 2:** If "vertically binding" on the column *union* is desired, use dplyr::bind_rows(), however this results in NAs in non-intersecting columns. For many files with little variable intersection, a sparse RFU-matrix will result (and will likely break ADAT attributes):

```
adats <- loadAdatsAsList(files)
union_adat <- dplyr::bind_rows(adats, .id = "SourceFile")
```

## Value

A list of ADATs named by files, each a soma_adat object corresponding to an individual file in files. For collapseAdats(), a single, collapsed soma_adat object.

## Author(s)

Stu Field

## See Also

read_adat()

Other IO: parseHeader(), read_adat(), soma_adat, write_adat()

## Examples

```
# only 1 file in directory
dir(system.file("extdata", package = "SomaDataIO"))

files <- system.file("extdata", package = "SomaDataIO") |>
  dir(pattern = "[.]adat$", full.names = TRUE) |> rev()

adats <- loadAdatsAsList(files)
class(adats)

# collapse into 1 ADAT
collapsed <- collapseAdats(adats)
class(collapsed)

# Alternatively use `collapse = TRUE`

  loadAdatsAsList(files, collapse = TRUE)
```

---

# merge_clin *Merge Clinical Data into SomaScan*

---

## Description

Occasionally, additional clinical data is obtained *after* samples have been submitted to SomaLogic, or even after 'SomaScan' results have been delivered. This requires the new clinical variables, i.e. non-proteomic, data to be merged with 'SomaScan' data into a "new" ADAT prior to analysis. merge_clin() easily merges such clinical variables into an existing soma_adat object and is a simple wrapper around dplyr::left_join().

## Usage

```
merge_clin(x, clin_data, by = NULL, by_class = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | A soma_adat object (with intact attributes), typically created using read_adat(). |
| clin_data | One of 2 options:<br>• a data frame containing clinical variables to merge into x, or<br>• a path to a file, typically a *.csv, containing clinical variables to merge into x. |

| | |
|---|---|
| by | A character vector of variables to join by. See dplyr::left_join() for more details. |
| by_class | If clin_data is a file path, a named character vector of the variable and its class. This ensures the "by-key" is compatible for the join. For example, c(SampleId = "character"). See read.table() for details about its colClasses argument, and also the examples below. |
| ... | Additional parameters passed to dplyr::left_join(). |

## Details

This functionality also exists as a command-line tool (R script) contained in merge_clin.R that lives in the cli/merge system file directory. Please see:

- dir(system.file("cli/merge", package = "SomaDataIO"), full.names = TRUE)
- vignette("cli-merge-tool", package = "SomaDataIO")

## Value

A soma_adat with new clinical variables merged.

## Author(s)

Stu Field

## See Also

dplyr::left_join()

## Examples

```
# retrieve clinical data
clin_file <- system.file("cli/merge", "meta.csv",
                         package = "SomaDataIO",
                         mustWork = TRUE)
clin_file

# view clinical data to be merged:
# 1) `group`
# 2) `newvar`
clin_df <- read.csv(clin_file, colClasses = c(SampleId = "character"))
clin_df

# create mini-adat
apts <- withr::with_seed(123, sample(getAnalytes(example_data), 2L))
adat <- head(example_data, 9L) |>   # 9 x 2
  dplyr::select(SampleId, all_of(apts))

# merge clinical variables
merged <- merge_clin(adat, clin_df, by = "SampleId")
merged
```

```
# Alternative syntax:
#   1) pass file path
#   2) merge on different variable names
#   3) convert join type on-the-fly
clin_file2 <- system.file("cli/merge", "meta2.csv",
                          package = "SomaDataIO",
                          mustWork = TRUE)

id_type <- typeof(adat$SampleId)
merged2 <- merge_clin(adat, clin_file2,                # file path
                      by = c(SampleId = "ClinKey"),    # join on 2 variables
                      by_class = c(ClinKey = id_type)) # match types
merged2
```

---

`params` *Common Parameters in* **SomaDataIO**

---

## Description

The parameters below are commonly used throughout the **SomaDataIO** package.

## Arguments

| | |
|---|---|
| `adat` | A `soma_adat` object (with intact attributes), typically created using `read_adat()`. |
| `x` | A `soma_adat` object (with intact attributes), typically created using `read_adat()`. |
| `matrix` | Character. A string of (usually) either `"serum"` or `"plasma"`. |

## Value

A `soma_adat` class object.

---

`parseHeader` *SomaLogic ADAT parser*

---

## Description

Parses the header section of an ADAT file.

## Usage

```
parseHeader(file)
```

## Arguments

| | |
|---|---|
| `file` | Character. The elaborated path and file name of the `*.adat` file to be loaded into an R workspace environment. |

## Value

A list of relevant file information required by `read_adat()` in order to complete loading the ADAT file, including:

| | |
|---|---|
| `Header.Meta` | list of notes and other information about the adat |
| `Col.Meta` | list of vectors that contain the column meta data about individual analytes, includes information about the target name and calibration and QC ratios |
| `file_specs` | list of values of the file parsing specifications |
| `row_meta` | character vector of the clinical variables; assay information that is included in the adat output along with the RFU data |

## Author(s)

Stu Field

## See Also

Other IO: `loadAdatsAsList()`, `read_adat()`, `soma_adat`, `write_adat()`

## Examples

```
f <- system.file("extdata", "example_data10.adat",
                 package = "SomaDataIO", mustWork = TRUE)
header <- parseHeader(f)
names(header)

header$Header.Meta

header$file_specs

header$row_meta

head(as.data.frame(header$Col.Meta))
```

---

# pivotExpressionSet *Convert to Long Format*

---

## Description

Utility to convert an `ExpressionSet` class object from the "wide" data format to the "long" format via `tidyr::pivot_longer()`. The **Biobase** package is required for this function.

## Usage

```
pivotExpressionSet(eSet)

meltExpressionSet(eSet)
```

### Arguments

| | |
|---|---|
| eSet | An ExpressionSet class object, created using adat2eSet(). |

### Value

A tibble consisting of the long format conversion of an ExpressionSet object.

### Functions

- meltExpressionSet(): **[Superseded]**. Please now use pivotExpressionSet().

### Author(s)

Stu Field

### See Also

Other eSet: adat2eSet()

### Examples

```
# subset into a reduced mini-ADAT object
# 10 samples (rows)
# 5 clinical variables and 3 features (cols)
sub_adat <- example_data[1:10, c(1:5, 35:37)]
ex_set   <- adat2eSet(sub_adat)

# convert ExpressionSet object to long format
adat_long <- pivotExpressionSet(ex_set)
```

---

## read_adat *Read (Load) SomaLogic ADATs*

---

### Description

The parse and load a *.adat file as a data.frame-like object into an R workspace environment. The class of the returned object is a soma_adat object.

read.adat() is **[Superseded]**. For backward compatibility it will likely never go away completely, but you are strongly encouraged to shift your code to use read_adat().

is.soma_adat() checks whether an object is of class soma_adat. See inherits().

### Usage

```
read_adat(file, debug = FALSE, verbose = getOption("verbose"), ...)

read.adat(file, debug = FALSE, verbose = getOption("verbose"), ...)

is.soma_adat(x)
```

## Arguments

| | |
|---|---|
| file | Character. The elaborated path and file name of the *.adat file to be loaded into an R workspace. |
| debug | Logical. Used for debugging and development of an ADAT that fails to load, particularly out-of-spec, poorly modified, or legacy ADATs. |
| verbose | Logical. Should the function call be run in *verbose* mode, printing relevant diagnostic call information to the console. |
| ... | Additional arguments passed ultimately to read.delim(), or additional arguments passed to either other S3 print or summary methods as required by those generics. |
| x | An R object to test. |

## Value

A data.frame-like object of class soma_adat consisting of SomaLogic RFU (feature) data and clinical meta data as columns, and samples as rows. Row names are labeled with the unique ID "SlideId_Subarray" concatenation. The sections of the ADAT header (e.g., "Header.Meta", "Col.Meta", ...) are stored as attributes (e.g. attributes(x)$Header.Meta).

Logical. Whether x inherits from class soma_adat.

## Author(s)

Stu Field

## See Also

read.delim()

Other IO: loadAdatsAsList(), parseHeader(), soma_adat, write_adat()

## Examples

```
# path to *.adat file
# replace with your file path
adat_path <- system.file("extdata", "example_data10.adat",
                         package = "SomaDataIO", mustWork = TRUE)
adat_path

my_adat <- read_adat(adat_path)

is.soma_adat(my_adat)
```

## read_annotations *Import a SomaLogic Annotations File*

### Description

Import a SomaLogic Annotations File

### Usage

```
read_annotations(file)
```

### Arguments

| | |
|---|---|
| file | A path to an annotations file location. This is a sanctioned, versioned file provided by Standard BioTools, Inc. and should be an *unmodified* `*.xlsx` file. |

### Value

A `tibble` containing analyte-specific annotations and related (e.g. lift/bridging) information, keyed on SomaLogic SeqId, the unique SomaScan analyte identifier.

### Examples

```
## Not run:
  # for example
  file <- "~/Downloads/SomaScan_11K_Annotated_Content.xlsx"
  anno_tbl <- read_annotations(file)

## End(Not run)
```

## rownames *Helpers for Working With Row Names*

### Description

Easily move row names to a column and vice-versa without the unwanted side-effects to object class and attributes. Drop-in replacement for `tibble::rownames_to_column()` and `tibble::column_to_rownames()` which can have undesired side-effects to complex object attributes. Does not import any external packages, modify the environment, or change the object (other than the desired column). When using `col2rn()`, if explicit row names exist, they are overwritten with a warning. `add_rowid()` does *not* affect row names, which differs from `tibble::rowid_to_column()`.

## Usage

```
rn2col(data, name = ".rn")

col2rn(data, name = ".rn")

has_rn(data)

rm_rn(data)

set_rn(data, value)

add_rowid(data, name = ".rowid")
```

## Arguments

| | |
|---|---|
| data | An object that inherits from class data.frame. Typically a soma_adat class object. |
| name | Character. The name of the column to move. |
| value | Character. The new set of names for the data frame. If duplicates exist they are modified on-the-fly via make.unique(). |

## Value

All functions attempt to return an object of the same class as the input with fully intact and unmodified attributes (aside from those required by the desired action). has_rn() returns a scalar logical.

## Functions

- rn2col(): moves the row names of data to an explicit column whether they are explicit or implicit.
- col2rn(): is the inverse of rn2col(). If row names exist, they will be overwritten (with warning).
- has_rn(): returns a boolean indicating whether the data frame has explicit row names assigned.
- rm_rn(): removes existing row names, leaving only "implicit" row names.
- set_rn(): sets (and overwrites) existing row names for data frames only.
- add_rowid(): adds a sequential integer row identifier; starting at 1:nrow(data). It does *not* remove existing row names currently, but may in the future (please code accordingly).

## Examples

```
df <- data.frame(a = 1:5, b = rnorm(5), row.names = LETTERS[1:5])
df
rn2col(df)              # default name is `.rn`
rn2col(df, "AptName")   # pass `name =`
```

```
# moving columns
df$mtcars <- sample(names(mtcars), 5)
col2rn(df, "mtcars")   # with a warning

# Move back and forth easily
# Leaves original object un-modified
identical(df, col2rn(rn2col(df)))

# add "id" column
add_rowid(mtcars)

# remove row names
has_rn(mtcars)
mtcars2 <- rm_rn(mtcars)
has_rn(mtcars2)
```

---

SeqId *Working with SomaLogic SeqIds*

---

## Description

The SeqId is the cornerstone used to uniquely identify SomaLogic analytes. SeqIds follow the format <Pool>-<Clone>_<Version>, for example "1234-56_7" can be represented as:

| Pool | Clone | Version |
|---|---|---|
| 1234 | 56 | 7 |

See **Details** below for the definition of each sub-unit. The <Pool>-<Clone> combination is sufficient to uniquely identify a specific analyte and therefore versions are no longer provided (though they may be present in legacy ADATs). The tools below enable users to extract, test, identify, compare, and manipulate SeqIds across assay runs and/or versions.

## Usage

```
getSeqId(x, trim.version = FALSE)

regexSeqId()

locateSeqId(x, trailing = TRUE)

seqid2apt(x)

apt2seqid(x)

is.apt(x)

is.SeqId(x)
```

```
matchSeqIds(x, y, order.by.x = TRUE)

getSeqIdMatches(x, y, show = FALSE)
```

## Arguments

| | |
|---|---|
| x | Character. A vector of strings, usually analyte/feature column names, AptNames, or SeqIds. For seqid2apt(), a vector *of* SeqIds. For apt2seqid(), a character vector *containing* SeqIds. For matchSeqIds(), a vector of pattern matches containing SeqIds. Can be AptNames with GeneIDs, the seq.XXXX format, or even "naked" SeqIds. |
| trim.version | Logical. Whether to remove the version number, i.e. "1234-56_7" -> "1234-56". Primarily for legacy ADATs. |
| trailing | Logical. Should the regular expression explicitly specify *trailing* SeqId pattern match, i.e. "regex$"? This is the most common case and the default. |
| y | Character. A second vector of AptNames containing SeqIds to match against those in contained in x. For matchSeqIds() these values are returned if there are matching elements. |
| order.by.x | Logical. Order the returned character string by the x (first) argument? |
| show | Logical. Return the data frame visibly? |

## Details

| | |
|---|---|
| **Pool:** | ties back to the original well during **SELEX** |
| **Clone:** | ties to the specific sequence within a pool |
| **Version:** | refers to custom modifications (optional/defunct) |

AptName a SeqId combined with a string, usually a GeneId- or seq.-prefix, for convenient, human-readable manipulation from within R.

## Value

getSeqId(): a character vector of SeqIds captured from a string.

regexSeqId(): a regular expression (regex) string pre-defined to match SomaLogic the SeqId pattern.

locateSeqId(): a data frame containing the start and stop integer positions for SeqId matches at each value of x.

seqid2apt(): a character vector with the seq.* prefix, i.e. the inverse of getSeqId().

apt2seqid(): a character vector of SeqIds. is.SeqId() will return TRUE for all elements.

is.apt(), is.SeqId(): Logical. TRUE or FALSE.

matchSeqIds(): a character string corresponding to values in y of the intersect of x and y. If no matches are found, character(0).

getSeqIdMatches(): a $nx2$ data frame, where n is the length of the intersect of the matching SeqIds. The data frame is named by the passed arguments, x and y.

## Functions

- `getSeqId()`: extracts/captures the the `SeqId` match from an analyte column identifier, i.e. column name of an ADAT loaded with `read_adat()`. Assumes the `SeqId` pattern occurs at the end of the string, which for the vast majority of cases will be true. For edge cases, see the `trailing` argument to `locateSeqId()`.
- `regexSeqId()`: generates a pre-formatted regular expression for matching of `SeqIds`. Note the *trailing* match, which is most commonly required, but `locateSeqId()` offers an alternative to mach *anywhere* in a string. Used internally in *many* utility functions
- `locateSeqId()`: generates a data frame of the positional `SeqId` matches. Specifically designed to facilitate `SeqId` extraction via `substr()`. Similar to `stringr::str_locate()`.
- `seqid2apt()`: converts a `SeqId` into anonymous-AptName format, i.e. `1234-56` -> `seq.1234.56`. Version numbers (`1234-56_ver`) are always trimmed when present.
- `apt2seqid()`: converts an anonymous-AptName into `SeqId` format, i.e. `seq.1234.56` -> `1234-56`. Version numbers (`seq.1234.56.ver`) are always trimmed when present.
- `is.apt()`: regular expression match to determine if a string *contains* a `SeqId`, and thus is probably an `AptName` format string. Both legacy `EntrezGeneSymbol-SeqId` combinations or newer so-called `"anonymous-AptNames"` formats (`seq.1234.45`) are matched.
- `is.SeqId()`: tests for `SeqId` format, i.e. values returned from `getSeqId()` will always return `TRUE`.
- `matchSeqIds()`: matches two character vectors on the basis of their intersecting `SeqIds`. Note that elements in `y` not containing a `SeqId` regular expression are silently dropped.
- `getSeqIdMatches()`: matches two character vectors on the basis of their intersecting *SeqIds* only (irrespective of the `GeneID`-prefix). This produces a two-column data frame which then can be used as to map between the two sets.

  The final order of the matches/rows is by the input corresponding to the *first* argument (`x`).

  By default the data frame is invisibly returned to avoid dumping excess output to the console (see the `show =` argument.)

## Author(s)

Stu Field

## See Also

`intersect()`

## Examples

```
x <- c("ABDC.3948.48.2", "3948.88",
       "3948.48.2", "3948-48_2", "3948.48.2",
       "3948-48_2", "3948-88",
       "My.Favorite.Apt.3948.88.9")

tibble::tibble(orig       = x,
               SeqId      = getSeqId(x),
               SeqId_trim = getSeqId(x, TRUE),
               AptName    = seqid2apt(SeqId))
```

```
# Logical Matching
is.apt("AGR2.4959.2") # TRUE
is.apt("seq.4959.2")  # TRUE
is.apt("4959-2")      # TRUE
is.apt("AGR2")        # FALSE


# SeqId Matching
x <- c("seq.4554.56", "seq.3714.49", "PlateId")
y <- c("Group", "3714-49", "Assay", "4554-56")
matchSeqIds(x, y)
matchSeqIds(x, y, order.by.x = FALSE)

# vector of features
feats <- getAnalytes(example_data)

match_df <- getSeqIdMatches(feats[1:100], feats[90:500])  # 11 overlapping
match_df

a <- utils::head(feats, 15)
b <- withr::with_seed(99, sample(getSeqId(a)))   # => SeqId & shuffle
(getSeqIdMatches(a, b))                          # sorted by first vector "a"
```

---

SomaDataIO-deprecated *Deprecated function(s) of the* **SomaDataIO** *package*

---

## Description

These functions have either been **[Superseded]** or **[Deprecated]** in the current version of **SomaDataIO** package. They may eventually be completely removed, so please re-code your scripts accordingly based on the suggestions below:

| Function | | Now Use |
|---|---|---|
| getSomamers() | **[Superseded]** | getAnalytes() |
| getSomamerData() | **[Superseded]** | getAnalyteInfo() |

## Details

Some badges you may see in **SomaDataIO**:

**[Superseded]**

**[Deprecated]**

**[Soft-deprecated]**

**[Stable]**

SomaScanObjects *Example Data and Objects*

## Description

The example_data object is intended to provide existing and prospective SomaLogic customers with example data to enable analysis preparation prior to receipt of SomaScan data, and also for those generally curious about the SomaScan data deliverable. It is **not** intended to be used as a control group for studies or provide any metrics for SomaScan data in general.

## Format

**example_data** a soma_adat parsed via read_adat() containing 192 samples (see below for breakdown of sample type). There are 5318 columns containing 5284 analyte features and 34 clinical meta data fields. These data have been pre-processed via the following steps:

- hybridization normalized (all samples)
- calibrators and buffers median normalized
- plate scaled
- calibrated
- Adaptive Normalization by Maximum Likelihood (ANML) of QC and clinical samples

**Note1:** The Age and Sex (M/F) fields contain simulated values designed to contain biological signal.

```
**Note2:** The `SampleType` column contains sample source/type information
and usually the `SampleType == Sample` represents the "client" samples.

**Note3:** The original source file can be found at
\url{https://github.com/SomaLogic/SomaLogic-Data}.
```

**ex_analytes** character string of the analyte features contained in the soma_adat object, derived from a call to getAnalytes().

**ex_anno_tbl** a lookup table corresponding to a transposed data frame of the "Col.Meta" attribute of an ADAT, with an index key field AptName included in column 1, derived from a call to getAnalyteInfo().

**ex_target_names** A lookup table mapping SeqId feature names -> target names contained in example_data. This object (or one like it) is convenient at the console via auto-complete for labeling and/or creating plot titles on the fly.

## Data Description

The example_data object contains a SomaScan V4 study from healthy normal individuals. The RFU measurements themselves and other identifiers have been altered to protect personally identifiable information (PII), but also retain underlying biological signal as much as possible. There are 192 total EDTA-plasma samples across two 96-well plate runs which are broken down by the following types:

- 170 clinical samples (client study samples)
- 10 calibrators (replicate controls for combining data across runs)
- 6 QC samples (replicate controls used to assess run quality)
- 6 Buffer samples (no protein controls)

### Data Processing

The standard V4 data normalization procedure for EDTA-plasma samples was applied to this dataset. For more details on the data standardization process see the Data Standardization and File Specification Technical Note. General details are outlined above.

### Source

https://github.com/SomaLogic/SomaLogic-Data

Standard BioTools, Inc.

### Examples

```
# S3 print method
example_data

# print header info
print(example_data, show_header = TRUE)

class(example_data)

# Features/Analytes
head(ex_analytes, 20L)

# Feature info table (annotations)
ex_anno_tbl

# Search via `filter()`
dplyr::filter(ex_anno_tbl, grepl("^MMP", Target))

# Lookup table -> targets
# MMP-9
ex_target_names$seq.2579.17

# gender hormone FSH
tapply(example_data$seq.3032.11, example_data$Sex, median)

# gender hormone LH
tapply(example_data$seq.2953.31, example_data$Sex, median)

# Target lookup
ex_target_names$seq.2953.31     # tab-completion at console

# Sample Type/Source
table(example_data$SampleType)
```

```
# Sex/Gender Variable
table(example_data$Sex)

# Age Variable
summary(example_data$Age)
```

---

soma_adat *The* soma_adat *Class and S3 Methods*

---

## Description

The soma_adat data structure is the primary internal R representation of SomaScan data. A soma_adat is automatically created via read_adat() when loading a *.adat text file. It consists of a data.frame-like object with leading columns as clinical variables and SomaScan RFU data as the remaining variables. Two main attributes corresponding to analyte and SomaScan run information contained in the *.adat file are added:

- Header.Meta: information about the SomaScan run, see parseHeader() or attr(x, "Header.Meta")
- Col.Meta: annotations information about the SomaScan reagents/analytes, see getAnalyteInfo() or attr(x, "Col.Meta")
- file_specs: parsing specifications for the ingested *.adat file
- row_meta: the names of the non-RFU fields. See getMeta().

See groupGenerics() for a details on Math(), Ops(), and Summary() methods that dispatch on class soma_adat.

See reexports() for a details on re-exported S3 generics from other packages (mostly dplyr and tidyr) to enable S3 methods to be dispatched on class soma_adat.

Below is a list of *all* currently available S3 methods that dispatch on the soma_adat class:

```
#>  [1] [              [[             [[<-           [<-
#>  [5] ==             $              $<-            anti_join
#>  [9] arrange        count          filter         full_join
#> [13] getAdatVersion getAnalytes    getMeta        group_by
#> [17] inner_join     is_seqFormat   left_join      Math
#> [21] median         merge          mutate         Ops
#> [25] print          rename         right_join     row.names<-
#> [29] sample_frac    sample_n       select         semi_join
#> [33] separate       slice_sample   slice          summary
#> [37] Summary        transform      ungroup        unite
#> see '?methods' for accessing help and source code
```

The S3 print() method returns summary information parsed from the object attributes, if present, followed by a dispatch to the tibble::tibble() print method. Rownames are printed as the first column in the print method only.

The S3 summary() method returns the following for each column of the ADAT object containing SOMAmer data (clinical meta data is *excluded*):

- Target (if available)
- Minimum value
- 1st Quantile
- Median
- Mean
- 3rd Quantile
- Maximum value
- Standard deviation
- Median absolute deviation (`mad()`)
- Interquartile range (`IQR()`)

The S3 `Extract()` method is used for sub-setting a `soma_adat` object and relies heavily on the `[` method that maintains the `soma_adat` attributes intact *and* subsets the `Col.Meta` so that it is consistent with the newly created object.

S3 extraction via `$` is fully supported, however, as opposed to the `data.frame` method, partial matching is *not* allowed for class `soma_adat`.

S3 extraction via `[[` is supported, however, we restrict the usage of `[[` for `soma_adat`. Use only a numeric index (e.g. `1L`) or a character identifying the column (e.g. `"SampleID"`). Do not use `[[i,j]]` syntax with `[[`, use `[` instead. As with `$`, partial matching is *not* allowed.

S3 assignment via `[` is supported for class `soma_adat`.

S3 assignment via `$` is fully supported for class `soma_adat`.

S3 assignment via `[[` is supported for class `soma_adat`.

S3 `median()` is *not* currently supported for the `soma_adat` class, however a dispatch is in place to direct users to alternatives.

## Usage

```
## S3 method for class 'soma_adat'
print(x, show_header = FALSE, ...)

## S3 method for class 'soma_adat'
summary(object, tbl = NULL, digits = max(3L, getOption("digits") - 3L), ...)

## S3 method for class 'soma_adat'
x[i, j, drop = TRUE, ...]

## S3 method for class 'soma_adat'
x$name

## S3 method for class 'soma_adat'
x[[i, j, ..., exact = TRUE]]

## S3 replacement method for class 'soma_adat'
x[i, j, ...] <- value
```

```
## S3 replacement method for class 'soma_adat'
x$i, j, ... <- value

## S3 replacement method for class 'soma_adat'
x[[i, j, ...]] <- value

## S3 method for class 'soma_adat'
median(x, na.rm = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x, object | A soma_adat class object. |
| show_header | Logical. Should all the Header Data information be displayed instead of the data frame (tibble) object? |
| ... | Ignored. |
| tbl | An annotations table. If NULL (default), annotation information is extracted from the object itself (if possible). Alternatively, the result of a call to getAnalyteInfo(), from which Target names can be extracted. |
| digits | Integer. Used for number formatting with signif(). |
| i, j | Row and column indices respectively. If j is omitted, i is used as the column index. |
| drop | Coerce to a vector if fetching one column via tbl[, j]. Default FALSE, ignored when accessing a column via tbl[j]. |
| name | A name or a string. |
| exact | Ignored with a warning(). |
| value | A value to store in a row, column, range or cell. |
| na.rm | a logical value indicating whether NA values should be stripped before the computation proceeds. |

## Value

The set of S3 methods above return the soma_adat object with the corresponding S3 method applied.

## See Also

groupGenerics()

Other IO: loadAdatsAsList(), parseHeader(), read_adat(), write_adat()

## Examples

```
# S3 print method
example_data

# show the header info (no RFU data)
```

```
print(example_data, show_header = TRUE)

# S3 summary method
# MMP analytes (4)
mmps <- c("seq.2579.17", "seq.2788.55", "seq.2789.26", "seq.4925.54")
mmp_adat <- example_data[, c("Sex", mmps)]
summary(mmp_adat)

# Summarize by group
mmp_adat |>
  split(mmp_adat$Sex) |>
  lapply(summary)

# Alternatively pass annotations with Target info
anno <- getAnalyteInfo(mmp_adat)
summary(mmp_adat, tbl = anno)
```

---

`transform` *Scale Transform* `soma_adat` *Columns/Rows*

---

## Description

Scale the *i-th* row or column of a `soma_adat` object by the *i-th* element of a vector. Designed to facilitate linear transformations of *only* the analyte/RFU entries by scaling the data matrix. If scaling the analytes/RFU (columns), `v` *must* have `getAnalytes(adat, n = TRUE)` elements. If scaling the samples (rows), `v` *must* have `nrow(_data)` elements.

## Usage

```
## S3 method for class 'soma_adat'
transform(`_data`, v, dim = 2L, ...)
```

## Arguments

| | |
|---|---|
| `_data` | A `soma_adat` object. |
| `v` | A numeric vector of the appropriate length corresponding to `dim`. |
| `dim` | Integer. The dimension to apply elements of `v` to. `1` = rows; `2` = columns (default). |
| `...` | Currently not used but required by the S3 generic. |

## Details

Performs the following operations (quickly):

Columns:

$$M_{nxp} = A_{nxp} * diag(v)_{pxp}$$

Rows:

$$M_{nxp} = diag(v)_{nxn} * A_{nxp}$$

## Value

A modified value of _data with either the rows or columns linearly transformed by v.

## Note

This method in intentionally naive, and assumes the user has ordered v to match the columns/rows of _data appropriately. This must be done upstream.

## See Also

apply(), sweep()

## Examples

```
# simplified example of underlying operations
M <- matrix(1:12, ncol = 4)
M

v <- 1:4
M %*% diag(v)    # transform columns

v <- 1:3
diag(v) %*% M    # transform rows

# dummy ADAT example:
v    <- c(2, 0.5)     # double seq1; half seq2
adat <- data.frame(sample      = paste0("sample_", 1:3),
                   seq.1234.56 = c(1, 2, 3),
                   seq.9999.88 = c(4, 5, 6) * 10)
adat

# `soma_adat` to invoke S3 method dispatch
class(adat) <- c("soma_adat", "data.frame")
trans <- transform(adat, v)
data.frame(trans)
```

---

# write_adat *Write an ADAT to File*

---

## Description

One can write an existing modified internal ADAT (soma_adat R object) to an external file. However the ADAT object itself *must* have intact attributes, see is_intact_attr().

## Usage

```
write_adat(x, file)
```

## Arguments

| | |
|---|---|
| x | A soma_adat object (with intact attributes), typically created using read_adat(). |
| file | Character. File path where the object should be written. For example, extensions should be *.adat. |

## Details

The ADAT specification *no longer* requires Windows end of line (EOL) characters ("\r\n"). The current EOL spec is "\n" which is commonly used in POSIX systems, like MacOS and Linux. Since the EOL affects the resulting checksum, ADATs written on other systems generate slightly differing files. Standardizing to "\n" attempts to solve this issue. For reference, see the EOL encoding for operating systems below:

| Symbol | Platform | Character |
|---|---|---|
| LF | Linux | "\n" |
| CR | MacOS | "\r" |
| CRLF | DOS/Windows | "\r\n" |

## Value

Invisibly returns the input x.

## Author(s)

Stu Field

## See Also

read_adat(), is_intact_attr()

Other IO: loadAdatsAsList(), parseHeader(), read_adat(), soma_adat

## Examples

```
# trim to 1 sample for speed
adat_out <- head(example_data, 1L)

# attributes must(!) be intact to write
is_intact_attr(adat_out)

write_adat(adat_out, file = tempfile(fileext = ".adat"))
```

# Index