Package 'opm'

July 2, 2014

Version 1.1.0

Date 2014-04-20

Title Analysing Phenotype Microarray and Growth Curve Data

Author Markus Goeker, with contributions by Benjamin Hofner, Lea A.I. Vaas, Johannes Sikorski, Nora Buddruhs and Anne Fiebig

Maintainer Markus Goeker <markus.goeker@dsmz.de>

Depends R (>= 3.0.0)

Imports methods, lattice, parallel, boot, hwriter, plotrix,Ckmeans.1d.dp, pkgutils (>= 0.6.0), yaml (>= 2.1.7), splines,mgcv, grofit (>= 1.1), rjson (>= 0.2.12), nlme, DBI

Suggests optparse, gplots, opmdata (>= 0.4.0), testthat, multcomp,KEGGREST, pathview, mboostDevel (>= 2.3.0), gridExtra

Description Tools for analysing OmniLog(R) and MicroStation(TM) phenotype microarray (PM) data as produced by the devices distributed by BIOLOG Inc. as well as similar kinds of data such as growth curves. Major facilities are plotting data,accurately estimating curve parameters, comparing and discretising data, creating phylogenetic formats and reports for taxonomic journals, drawing the PM analysis results in biochemical pathway graphs optionally including genome annotations, running multiple comparisons of means, easy interaction with powerful feature-selection approaches,integrating metadata, using the YAML format for the storage of data and metadata, batch conversion of large numbers of files,and database I/O. (The suggested mboostDevel is from R-Forge and not necessary.)

License GPL (>= 2)

URL http://opm.dsmz.de/

MailingList http://lists.r-forge.r-project.org/mailman/listinfo/opm-support

Collate 'substrate-info.R' 'well-map.R' 'plate-map.R' 'imports.R'
'package.R' 'constants.R' 'classes.R' 'auxiliary.R' 'kmeans.R'
'naming.R' 'metadata.R' 'getter.R' 'conversion.R'
'combination.R' 'aggregation.R' 'discretization.R' 'plotting.R'
'io.R' 'phylogeny.R' 'data.R' 'testing.R' 'multcomp.R' 'splinefit.R' 'dbio.R'

LazyData yes

LazyDataCompression bzip2

Repository CRAN

Repository/R-Forge/Project opm

Repository/R-Forge/Revision 602

Repository/R-Forge/DateTimeStamp 2014-04-20 04:46:43

Date/Publication 2014-04-22 07:23:32

NeedsCompilation no

R topics documented:

`aggregated` *Get aggregated data*

Description

Get the aggregated kinetic data or the aggregation settings used. (See `do_aggr` for generating aggregated data.)

Usage

```
  ## S4 method for signature 'MOPMX'
aggr_settings(object, join = NULL)
  ## S4 method for signature 'OPMA'
aggr_settings(object, join = NULL)
  ## S4 method for signature 'OPMS'
aggr_settings(object, join = NULL)

  ## S4 method for signature 'MOPMX'
aggregated(object, ...)
  ## S4 method for signature 'OPMA'
aggregated(object, subset = NULL, ci = TRUE,
    trim = c("no", "full", "medium"), full = FALSE, in.parens = TRUE,
    max = opm_opt("max.chars"), ...)
  ## S4 method for signature 'OPMS'
aggregated(object, ...)
```

Arguments

`object`	`OPMA`, `OPMS` or `MOPMX` object.
`subset`	Character vector. If not `NULL`, restrict to this or these parameter(s). See `param_names` for the possible values.
`ci`	Logical scalar. Include the estimates of confidence intervals (CIs) in the output?
`trim`	Character scalar. Parameter estimates from intrinsically negative reactions (i.e., no respiration) are sometimes biologically unreasonable because they are too large or too small, and some corrections might be appropriate. **no** No modification. **full** Negative lambda estimates are set to zero. **medium** Lambda estimates larger than `hours(object)` (i.e., the maximum time value observed) are set to that value. Negative lambda estimates smaller than `-hours(object)` are set to this value (i.e., the negative maximum time). **full** Like 'medium', but all negative values are set to zero, which is a less moderate treatment. Currently the other parameters are not checked, and all `NA` values, if any, also remain unchanged.
`full`	Logical scalar passed to `wells`. This and the following arguments affect the column names of the resulting matrix.
`in.parens`	Logical scalar also passed to that function.
`max`	Numeric scalar also passed to that function.
`join`	Empty or character scalar. If empty, a list is returned; a nested list in the case of `OPMS` objects with one contained list per plate. Otherwise this nested list is converted to a matrix or data frame, depending on the value of `join`. The following values yield a matrix in character mode and differ in how they would convert non-scalar values in a matrix in list mode, if encountered:

json Converts to a JSON string.

yaml Converts to a YAML string.

rcode Converts by deparsing, yielding an R code string.

All other values of `join` are passed as `what` argument to the `collect` method form the **pkgutils** package, with `dataframe` and `keep.unnamed` set to `TRUE` but `stringsAsFactors` to `FALSE`.

`...` Optional arguments passed between the methods or to `wells`.

Details

Note that the conversion of the settings list to a matrix or data frame might not work for all combinations of `object` and `join`, mainly because the options entry can hold arbitrary content. For similar conversions of the metadata, see the `OPMX` methods of `to_metadata`.

Value

`aggregated` yields a numeric matrix of aggregated values (a.k.a. the curve parameters). If bootstrapping was used, their CIs are included. The columns represent the wells, the rows the estimated parameters and their CIs.

`aggr_settings` returns a named list if `join` is empty. Other values yield a matrix or data frame (or an error). See the description of the argument above and the examples below for further details.

See Also

Other getter-functions: `anyDuplicated`, `contains`, `csv_data`, `dim`, `disc_settings`, `discretized`, `duplicated`, `has_aggr`, `has_disc`, `hours`, `max`, `measurements`, `minmax`, `seq`, `subset`, `thin_out`, `well`

Examples

```
# 'OPMA' methods
# Get full matrix
(x <- aggregated(vaas_1))[, 1:3]
stopifnot(is.matrix(x), dim(x) == c(12, 96))
(y <- aggregated(vaas_1, full = TRUE))[, 1:3] # full names
stopifnot(x == y, nchar(colnames(x)) < nchar(colnames(y)))
# Subsetting
(x <- aggregated(vaas_1, "lambda"))[, 1:3]
stopifnot(is.matrix(x), dim(x) == c(3, 96), any(x < 0))
# Now with lambda correction
(x <- aggregated(vaas_1, "lambda", trim = "full"))[, 1:3]
stopifnot(is.matrix(x), dim(x) == c(3, 96), !any(x < 0))

# settings
(x <- aggr_settings(vaas_1)) # yields named list
stopifnot(is.list(x), !is.null(names(x)))
(x <- aggr_settings(vaas_1, join = "json")) # yields a matrix
stopifnot(is.matrix(x), is.character(x), nrow(x) == 1)

# 'OPMS' methods
```

```
summary(x <- aggregated(vaas_4)) # => one matrix per OPM object
stopifnot(is.list(x), length(x) == length(vaas_4), sapply(x, is.matrix))

# settings
summary(x <- aggr_settings(vaas_4)) # list of named lists, one per plate
stopifnot(is.list(x), length(x) == length(vaas_4), sapply(x, is.list))
(x <- aggr_settings(vaas_4, join = "yaml")) # matrix, one row per plate
stopifnot(is.matrix(x), is.character(x), nrow(x) == 4)
```

---

annotated *Create vector or matrix with substrate annotation*

Description

These methods create vectors or matrices that include numeric values (selected parameter estimates or opm_mcp results) as well as an annotation of the according substrates.

Usage

```
  ## S4 method for signature 'MOPMX'
annotated(object, what = "kegg", how = "ids",
    output = opm_opt("curve.param"), lmap = NULL, sep = opm_opt("min.mode"),
    conc = FALSE)
  ## S4 method for signature 'OPMA'
annotated(object, what = "kegg", how = "ids",
    output = opm_opt("curve.param"), lmap = NULL, sep = NULL, conc = FALSE)
  ## S4 method for signature 'OPMD'
annotated(object, what = "kegg", how = "ids",
    output = opm_opt("curve.param"), lmap = NULL, sep = NULL, conc = FALSE)
  ## S4 method for signature 'OPMS'
annotated(object, what = "kegg", how = "ids",
    output = opm_opt("curve.param"), lmap = NULL, sep = opm_opt("min.mode"),
    conc = FALSE)
  ## S4 method for signature 'OPM_MCP_OUT'
annotated(object, what = "kegg", how = "ids",
    output = c("full", "plain"), lmap = NULL, sep = NULL, conc = FALSE)
  ## S4 method for signature 'opm_glht'
annotated(object, what = "kegg", how = "ids",
    output = "numeric", lmap = NULL, sep = opm_opt("comb.value.join"),
    conc = FALSE)
```

Arguments

object	An object of the classes opm_glht or OPM_MCP_OUT as created by opm_mcp, OPMA, OPMD or OPMS.
what	Character scalar indicating the kind of annotation to use. Passed as eponymous argument to substrate_info.

how — Character scalar. Indicating how the annotation is inserted. Currently 'ids', 'values' and 'data.frame' are supported. See below for details.

output — For the OPMA and OPMS methods, the estimated parameter of interest (see param_names). For the opm_glht method, either a numeric scalar or one of the following character scalars:

numeric Return the coefficients.

upwards Return FALSE, NA, or TRUE indicating whether the coefficients are significantly smaller or larger than, or insignificantly different from the cutoff given by opm_opt("threshold"). This is calculated from the confidence intervals stored in object.

downwards Return TRUE, NA, or FALSE indicating whether the coefficients are significantly smaller or larger than, or insignificantly different from the default cutoff.

smaller Return TRUE or FALSE indicating whether or not the coefficients are significantly smaller than the default cutoff.

larger Return TRUE or FALSE indicating whether or not the coefficients are significantly larger than the default cutoff.

equal Return TRUE or FALSE indicating whether or not the coefficients are insignificantly different from the default cutoff. Note that 'insignificantly different' does not mean 'significantly equal'!

different Return TRUE or FALSE indicating whether or not the coefficients are significantly different from the default cutoff.

Alternatively, character scalars such as '!75.0', '<100', '>150', '=85.0', ',3.0' or ''11.5' can be provided, with the first character translated to the corresponding meaning in the list above (with ',' for 'downwards' and '" for 'upwards') and the remaining string coerced to the cutoff to be used. If a numeric scalar is provided, it is used as cutoff in conjunction with the 'different' mode described above.

lmap — Vector to be used for mapping the created logical values, if any. See map_values and the examples below for details. If NULL, ignored. Also ignored if numeric instead of logical values are created.

sep — For the opm_glht method, the single character that has been used as eponymous argument in the call to opm_mcp. Necessary to unambiguously match substrate names within contrast names. For the OPMS method, a numeric scalar working like the cutoff argument of listing. Has only an effect if discretised values are chosen (and are available).

conc — Logical scalar indicating whether concentration information should be added to the output, either as a further matrix column or as attribute, depending on how. Note that concentration information might be partially or entirely NA, depending on the wells in use.

Details

All methods use substrate_info for translating substrate names to IDs. The methods differ only in the way numeric and logical values are generated.

The OPMA methods simply chooses a certain parameter. The OPMD method can also return discretised values and optionally translates them using lmap.

The OPMS method returns the averages of the selected parameter estimates over all contained plates. It is an error to select discretised values instead if they are not available for all plates. If otherwise, the discretised values are aggregated as indicated by the sep argument.

The opm_glht method makes only sense if each coefficient estimated by opm_mcp can be linked to a single substrate. This is usually **only** possible for the 'Dunnett' and 'Pairs' type of contrast if applied to the wells. Typical applications are the comparison of a single control well to a series of other wells and the comparison of all or a subset of the wells between two metadata-defined groups. See opm_mcp for details.

Because the current implementation of the opm_glht method attempts to identify the substrates within the names of the estimated coefficients (differences of means), some care must be taken when translating well coordinates to substrate names in the call to opm_mcp. Substrate IDs cannot be identified if they are abbreviated, i.e. a low value of the max argument passed to wells is used, and not accompanied by the well coordinates, i.e. if the in.parens argument is set to FALSE.

In the case of the 'Pairs' type of contrasts, some problems can be avoided by setting the 'comb.value.join' entry of opm_opt to another value. The same value must be used in the calls to opm_mcp and annotated, however.

Value

For how = "ids", a numeric or logical vector whose names are the IDs of the respective substrates in the database as chosen by what.

For how = "values", a numeric matrix containing the chosen computed values as first column together with data obtained via web service associated with the chosen database, in analogy to the download argument of substrate_info but after conversion to a numeric matrix. This option is not available for all values of what and requires additional libraries. See substrate_info for details.

The first column name of the matrix is like the 'value' entry returned by param_names in 'reserved.md.names' mode. The second one is 'Concentration' if conc is TRUE. Depending on the subsequent analysis, it might be necessary to convert the matrix to a data frame, to convert the column names to syntactical names (see make.names from the **base** package), or to remove all rows and columns with missing values.

For how = "data.frame", much like how = "values", but the numeric matrix is converted to a data frame, the column names are made syntactic, and all essentially binary (zero/one) columns are converted to factors.

See Also

Other multcomp-functions: opm_mcp

Examples

```
## OPMD and OPMS methods

# default settings
head(x <- annotated(vaas_1))
stopifnot(is.numeric(x), x > 0, !is.null(names(x)))
head(y <- annotated(vaas_4)) # this averages per well over all plates
stopifnot(is.numeric(x), y > 0, identical(names(y), names(x)))
```

```
# AUC instead of maximum height
head(y <- annotated(vaas_1, output = param_names()[4]))
stopifnot(y > x, identical(names(y), names(x)))

# generation of logical vectors
head(y <- annotated(vaas_4, output = param_names("disc.name")))
stopifnot(is.logical(y), identical(names(y), names(x)))

# mapping of logical vectors: FALSE => 1, NA => 2, TRUE => 3
head(y <- annotated(vaas_4, output = param_names("disc.name"), lmap = 1:3))
stopifnot(is.numeric(y), y > 0, identical(names(y), names(x)))

## 'opm_glht' method

if (require("multcomp")) {

# generation of 'opm_ghlt' test object
y <- opm_mcp(vaas_4[, , 1:4], model = ~ J(Well, Species),
  m.type = "aov", linfct = c(Pairs.Well = 1), full = FALSE)

# generation of numerical vector
head(y.ann <- annotated(y))
stopifnot(is.numeric(y.ann), !is.null(names(y.ann)))

# generation of logical vector indicating whether the compared groups behave
# equally (= insignificantly different) regarding the considered substrates
head(y.ann.eq <- annotated(y, output = "equal", lmap = 1:3))
stopifnot(y.ann.eq > 0, identical(names(y.ann.eq), names(x)[1:4]))

# generation of numeric vector with FALSE => 1, NA => 2 and TRUE => 3 and
# substrate names after translation of relevant characters to Greek letters
head(y.ann.eq <- annotated(y, output = "equal", what = "greek", lmap = 1:3))
stopifnot(is.numeric(y.ann.eq), !is.null(names(y.ann.eq)))

}
```

`as.data.frame` *Create data frame*

Description

These `as.data.frame` methods create a data frame from aggregated and discretised values in a manner distinct from `extract`. `flatten` converts into a 'flat' data frame, including all measurements in a single column (suitable, e.g., for **lattice**).

Usage

```
  ## S4 method for signature 'MOPMX'
as.data.frame(x, row.names = NULL,
```

```
    optional = FALSE, sep = "_", csv.data = TRUE, settings = TRUE,
    include = FALSE, ..., stringsAsFactors = default.stringsAsFactors())
  ## S4 method for signature 'OPM'
as.data.frame(x, row.names = NULL,
    optional = FALSE, sep = "_", csv.data = TRUE, settings = TRUE,
    include = FALSE, ..., stringsAsFactors = default.stringsAsFactors())
  ## S4 method for signature 'OPMA'
as.data.frame(x, row.names = NULL,
    optional = FALSE, sep = "_", csv.data = TRUE, settings = TRUE,
    include = FALSE, ..., stringsAsFactors = default.stringsAsFactors())
  ## S4 method for signature 'OPMD'
as.data.frame(x, row.names = NULL,
    optional = FALSE, sep = "_", csv.data = TRUE, settings = TRUE,
    include = FALSE, ..., stringsAsFactors = default.stringsAsFactors())
  ## S4 method for signature 'OPMS'
as.data.frame(x, row.names = NULL,
    optional = FALSE, sep = "_", csv.data = TRUE, settings = TRUE,
    include = FALSE, ..., stringsAsFactors = default.stringsAsFactors())
  ## S4 method for signature 'kegg_compound'
as.data.frame(x, row.names = NULL,
    optional = TRUE, ..., stringsAsFactors = FALSE)
  ## S4 method for signature 'kegg_compounds'
as.data.frame(x, row.names = NULL,
    optional = TRUE, ..., stringsAsFactors = FALSE)

  ## S4 method for signature 'MOPMX'
flatten(object, include = NULL, fixed = list(),
    factors = FALSE, ...)
  ## S4 method for signature 'OPM'
flatten(object, include = NULL, fixed = list(),
    factors = TRUE, exact = TRUE, strict = TRUE, full = TRUE,
    numbers = FALSE, ...)
  ## S4 method for signature 'OPMS'
flatten(object, include = NULL, fixed = list(),
    ...)
```

Arguments

x	Object of class OPM, its child classes, or OPMS or MOPMX. If an OPMS object, for the as.data.frame method its elements must either all be OPM or all be OPMA or all be OPMD objects. If a MOPMX object, its elements must be conforming OPMS or either OPM, OPMA or OPMS objects. There are as.data.frame methods for some of the objects created by substrate_info, too.
row.names	Optional vector for use as row names of the resulting data frame. Here, it is not recommended to try to set row names explicitly.
optional	Logical scalar passed to the list and matrix methods of as.data.frame.

sep	Character scalar used as word separator in column names. Set this to NULL or an empty vector to turn off character replacement in column names.
csv.data	Logical scalar indicating whether the csv_data entries that identify the plate shall be included.
settings	Logical scalar indicating whether the aggr_settings and disc_settings entries, if available, shall be included.
stringsAsFactors	Logical scalar passed to the list and matrix methods of as.data.frame.
object	OPM or OPMS object (or list).
include	For flatten, either NULL, character vector, list or formula. If not empty, include this meta-information in the data frame, replicated in each row. Otherwise it converted to a list and passed to metadata. See there for details. For as.data.frame, if empty or FALSE, ignored. If TRUE, all metadata are included using to_metadata. If otherwise and non-empty, metadata selected using extract_columns are included.
fixed	NULL or list. If not NULL, include these items in the data frame, replicated in each row.
factors	Logical scalar. See the stringsAsFactors argument of data.frame and as.data.frame from the **base** package.
exact	Logical scalar. Passed to metadata.
strict	Logical scalar. Passed to metadata.
full	Logical scalar. Replace well coordinates by full names?
numbers	Logical scalar. Use numbers instead of well names? This is *not* recommended for must usages.
...	Optional other arguments passed to wells, or from the OPMS to the OPM method, or to the list and matrix methods of as.data.frame.

Details

The as.data.frame methods for OPMX objects are mainly intended to produce objects that can easily be written to CSV files, for instance using write.table from the **utils** package. There are no **opm** methods other than batch_opm (which can write such files) that make use of the created kind of objects. In particular, they cannot be input again into **opm**.

The following entries are contained in the generated data frame:

- Optionally the csv_data entries that identify the plate.
- The names of the wells. Always included.
- For OPMA objects (and OPMS objects that contain them as well as MOPMX objects that contain such OPMA or OPMS objects), always the aggregated data (curve parameters), one column for each point estimate, upper and lower confidence interval of each parameter.
- For OPMA objects (and OPMS objects that contain them as well as MOPMX objects that contain such OPMA or OPMS objects), optionally the used aggregation settings, one column per entry, except for the 'options' entry (which is not a scalar). The column names are prefixed with "Aggr" followed by sep. If sep is empty, opm_opt("comb.key.join") is used.

- For OPMD objects (and OPMS objects that contain them as well as MOPMX objects that contain such OPMD or OPMS objects), always one column with the discretised data.
- For OPMD objects (and OPMS objects that contain them as well as MOPMX objects that contain such OPMD or OPMS objects), optionally the used discretisation settings, one column per entry, except for the 'options' entry (which is not a scalar). The column names are prefixed with "Disc" followed by sep. If sep is empty, opm_opt("comb.key.join") is used.

The limits of using CSV as output format already show up in this list, and in general we recommend to generate YAML or JSON output instead.

For the as.data.frame methods of the other classes, see substrate_info.

In the data frame returned by flatten, column names are unchecked (not converted to variable names). The three last columns are coding for time, well and value, with the exact spelling of the column names given by param_names.

The OPMS method yields an additional column for the plate, the exact spelling of its name also being available via param_names. This column contains the position of each plate within object.

The MOPMX method yields a another additional column for the plate type. There is currently no safeguard against having several OPMX objects of the same plate type within a MOPMX object.

Value

The as.data.frame methods create a data frame with one row for each combination of well and plate.

The flatten methods create a data frame with one row for each combination of time point, well and plate.

See Also

utils::write.table stats::reshape pkgutils::flatten

Other conversion-functions: extract, extract_columns, merge, oapply, opmx, plates, rep, rev, sort, split, to_yaml, unique

Examples

```
## OPMD method of as.data.frame()
summary(x <- as.data.frame(vaas_1))
stopifnot(is.data.frame(x), nrow(x) == 96)

## OPMS method of as.data.frame()
summary(x <- as.data.frame(vaas_4[, , 1:10]))
stopifnot(is.data.frame(x), nrow(x) == 10 * 4)

## OPM method of flatten()
# distinct numbers of columns due to distinct selection settings
head(x <- flatten(vaas_1))
stopifnot(is.data.frame(x), identical(dim(x), c(36864L, 3L)))
head(x <- flatten(vaas_1, fixed = "TEST", include = "Strain"))
stopifnot(is.data.frame(x), identical(dim(x), c(36864L, 5L)))

## OPMS method of flatten()
```

```
# distinct numbers of columns due to distinct selection settings
head(x <- flatten(vaas_4[, , 1:10]))
stopifnot(is.data.frame(x), identical(dim(x), c(15360L, 4L)))
head(x <- flatten(vaas_4[, , 1:10], fixed = "TEST", include = ~ Strain))
stopifnot(is.data.frame(x), identical(dim(x), c(15360L, 6L)))
```

---

batch_opm *Batch-convert* PM *data*

Description

Batch-convert from OmniLog® CSV (or previous **opm** YAML or JSON) to **opm** YAML (or JSON). It is possible to add metadata to each set of raw data and to aggregate the curves; these additional data will then be included in the output files.

Usage

```
batch_opm(names, md.args = NULL, aggr.args = NULL,
  force.aggr = FALSE, disc.args = NULL,
  force.disc = FALSE, gen.iii = opm_opt("gen.iii"),
  device = "mypdf", dev.args = NULL, plot.args = NULL,
  csv.args = NULL,
  table.args = list(sep = "\t", row.names = FALSE), ...,
  proc = 1L, outdir = "", overwrite = "no",
  output = c("yaml", "json", "csv", "xyplot", "levelplot", "split", "clean"),
  combine.into = NULL, verbose = TRUE, demo = FALSE)
```

Arguments

md.args	If not NULL but a list, passed as arguments to include_metadata with the data read from each individual file as additional argument 'object'. If NULL, metadata are not included (but may already be present in the case of YAML input).
aggr.args	If not NULL but a list, passed as arguments to do_aggr with the data read from each individual file as additional argument object. If NULL, aggregation takes not place (but aggregated data may already be present in case of YAML input).
force.aggr	Logical scalar. If FALSE, do not aggregate already aggregated data (which can be present in YAML input).
disc.args	If not NULL but a list, passed as arguments to do_disc with the data read from each individual file as additional argument object. If NULL, discretisation takes not place (but discretised data may already be present in case of YAML input).
force.disc	Logical scalar. If FALSE, do not discretise already discretised data (which can be present in YAML input).
device	Character scalar describing the graphics device used for outputting plots. See Devices from the **grDevices** package and mypdf from the **pkgutils** package for possible values. The extension of the output files is created from the device name after a few adaptations (such as converting postscript to ps).

`dev.args` List. Passed as additional arguments to `device`.

`plot.args` List. Passed as additional arguments to the plotting function used.

`csv.args` If not `NULL` but a list, used for specifying ways to use `csv_data` entries directly as `metadata`. The list can contain character vectors used for selecting and optionally renaming CSV entries or functions that can be applied to an entire data frame containing all CSV entries. Note that this argument has nothing to do with `csv` output.

`table.args` Passed to `write.table` from the **utils** package if `output` is set to `csv`. Do not confuse this with `csv.args`.

`...` Optional arguments passed to `batch_process` in addition to `verbose` and `demo`. Note that `out.ext`, `fun` and `fun.args` are set automatically. Alternatively, these are parameters passed to `batch_collect`.

`output` Character scalar determining the main output mode.

clean Apply `clean_filenames`.

csv Create CSV files, by default one per input file.

json Create JSON files, by default one per input file.

levelplot Create graphics files, by default one per input file, containing the output of `level_plot`.

split Split multiple-plate new style or old style CSV files with `split_files`.

yaml Create YAML files, by default one per input file.

xyplot Create graphics files, by default one per input file, containing the output of `xy_plot`.

`combine.into` Empty or character scalar modifying the output mode unless it is 'clean' or 'split'. If non-empty, causes the creation of a single output file named per plate type encountered in the input, instead of one per input file (the default). Thus `combine.into` should be given as a template string for `sprintf` from the **base** package with one placeholder for the plate-type, and without a file extension.

`names` Character vector with names of files in one of the formats accepted by `read_single_opm`, or names of directories containing such files, or both; or convertible to such a vector. See the `include` argument of `read_opm` and `explode_dir` for how to select subsets from the input files or directories.

`gen.iii` Logical or character scalar. If `TRUE`, invoke `gen_iii` on each plate. This is automatically done with CSV input if the plate type is given as OTH (which is usually the case for plates run in ID mode). If a character scalar, it is used as the `to` argument of `gen_iii` to set other plate types unless it is empty.

`demo` Logical scalar. Do not read files, but print a vector with the names of the files that would be (attempted to) read, and return them invisibly?

`proc` Integer scalar. The number of processes to spawn. Cannot be set to more than 1 core if running under Windows. See the `cores` argument of `do_aggr` for details.

`outdir` Character vector. Directories in which to place the output files. If `NULL` or only containing empty strings, the directory of each input file is used.

`overwrite` Character scalar. If 'yes', conversion is always tried if `infile` exists and is not empty. If 'no', conversion is not tried if `outfile` exists and is not empty. If 'older', conversion is tried if `outfile` does not exist or is empty or is older than `infile` (with respect to the modification time).

`verbose` Logical scalar. Print conversion and success/failure information?

Details

This function is for batch-converting many files; for writing a single object to a YAML file (or string), see `to_yaml`.

A YAML document can comprise *scalars* (single values of some type), *sequences* (ordered collections of some values, without names) and *mappings* (collections assigning a name to each value), in a variety of combinations (e.g., mappings of sequences). The output of `batch_opm` is one YAML document *per plate* which represents a mapping with the following components (key-value pairs):

metadata Arbitrarily nested mapping of arbitrary metadata entries. Empty if no metadata have been added.

csv_data Non-nested mapping containing the OmniLog® run information read from the input CSV file (character scalars) together with the measurements. The most important entry is most likely the plate type.

measurements A mapping whose values are sequences of floating-point numbers of the same length and in the appropriate order. The keys are 'hours', which refers to the time points, and the well coordinates, ranging between 'A01' and 'H12'.

aggregated A mapping, only present if curve parameters have been estimated. Its keys correspond to those of 'measurements' with the exception of 'hours'. The values are themselves mappings, whose keys indicate the respective curve parameter and whether this is the point estimate or the upper or lower confidence interval. The values of these secondary mappings are floating-point numbers.

aggr_settings A mapping, only present if curve parameters have been estimated. Its keys are 'software', 'version' and 'options'. The value of the former two is a character scalar. The value of 'options' is an arbitrarily nested mapping with arbitrary content.

discretized A mapping, only present if curve parameters have been estimated and also discretised. Its keys correspond to those of 'measurements' with the exception of 'hours'. The values are logical scalars.

disc_settings A mapping, only present if curve parameters have been estimated and also discretised. Its keys are 'software', 'version' and 'options'. The value of the former two is a character scalar. The value of 'options' is an arbitrarily nested mapping with arbitrary content.

Details of the contents should be obvious from the documentation of the classes of the objects from which the YAML output is generated. In the case of YAML input with several plates per file, `batch_opm` generates YAML output files containing a sequence of mappings as described above, one per plate, to keep a 1:1 relationship between input and output files.

Attempting to generate YAML from input data with a wrong character encoding might cause R to crash or hang. This problem was observed with CSV files that were generated on a distinct operating system and contained special characters such as German umlauts. It is then necessary to explicitly (and correctly) specify the encoding used in these files; see the 'file.encoding' option of `opm_opt` for how to do this.

JSON, which is almost a subset of YAML, can also be generated, but has more restrictions. It is only recommended if a YAML parser is unavailable. It is also more delicate regarding the encoding of character strings.

When inputting YAML files generated with the help of the **yaml** package (on which the **opm** implementation is based), or JSON files generated with the help of the **rjson** package, using other programming languages, a potential problem is that they, and YAML in general, lack a native representation of `NA` values. Such entries are likely to be misunderstood as 'NA' character scalars (if the **json** package or the **yaml** package prior to version 2.1.7 are used) or as `.na`, `.na.real`, `.na.logical` or `.na.character` character scalars (if more recent versions of the **yaml** package are used). Input functions in other programming languages should conduct according conversions. **opm** translates these values when converting a list to a `OPM` object.

See `as.data.frame` regarding the generated CSV.

Value

The function invisibly returns a matrix which describes each attempted file conversion. See `batch_process` for details.

References

http://www.yaml.org/

http://www.json.org/

http://www.biolog.com/

See Also

utils::read.csv yaml::yaml.load_file grDevices::Devices

pkgutils::mypdf

Other io-functions: `batch_collect`, `batch_process`, `clean_filenames`, `collect_template`, `explode_dir`, `file_pattern`, `glob_to_regex`, `read_opm`, `read_single_opm`, `split_files`, `to_metadata`

Examples

```
test.files <- opm_files("omnilog")
if (length(test.files) > 0) { # if the files are found
  num.files <- length(list.files(outdir <- tempdir()))
  x <- batch_opm(test.files[1], outdir = outdir)
  stopifnot(length(list.files(outdir)) == num.files + 1, is.matrix(x))
  stopifnot(file.exists(x[, "outfile"]))
  stopifnot(test.files[1] == x[, "infile"])
  unlink(x[, "outfile"])
} else {
  warning("opm example data files not found")
}
```

boccuto_et_al *Autism cell-line example data set*

Description

Example data set for analysing phenotype microarray data that were not recorded with an OmniLog® instrument.

Format

`boccuto_et_al` is a `MOPMX` object with four `OPMX` objects of the dimension 35 x 1 x 96 as elements. The plate types are 'CUSTOM:PM-M01' to 'CUSTOM:PM-M04'. The well assignment of these is fully identical to their non-custom counterparts, but separate plate types are nevertheless useful here to avoid comparing apples and oranges, as the scale of the measurements is totally different from OmniLog units.

Details

The data set is identical to the supplement of Boccuto *et al.* (2013) except for the following differences:

- The measurements are not logarithmised.
- The negative controls are contained, hence the plates are complete.
- The individuals N1 to N15 are missing, hence the data set is reduced to autism-spectrum disorder patients and control group.

The data are point measurements, so a call to `do_aggr` would just copy the data. Two of the metadata entries are important, 'individual' for identifying the cell culture and 'group' for assigning the individuals to patients and control group, respectively.

Note

Information provided by C.E. Schwartz and colleagues additional to the supplement of Boccuto *et al.* (2013) is gratefully acknowledged.

References

Boccuto, L., Chen, C.-F., Pittman, A.R., Skinner, C.D., McCartney, H.J., Jones, K., Bochner, B.R., Stevenson, R.E., Schwartz, C.E. 2013. Decreased tryptophan metabolim in patients with autism spectrum disorder. *Molecular Autism* **4**: 16.

Schwartz, C.E., pers. comm.

Examples

```
## Not run:

# Calling this yielded a variable 'boccuto_et_al' containing the data. The
# opm package must be loaded beforehand using library().
data(boccuto_et_al)

# Pseudo-aggregate the data (use a copy of each point measurement as
# maximum-height value).
boccuto_et_al <- do_aggr(boccuto_et_al)

## End(Not run)
# Copy the well maps of the pre-defined counterparts.
register_plate(`CUSTOM:PM-M01` = wells(plate = "PM-M01"))
register_plate(`CUSTOM:PM-M02` = wells(plate = "PM-M02"))
register_plate(`CUSTOM:PM-M03` = wells(plate = "PM-M03"))
register_plate(`CUSTOM:PM-M04` = wells(plate = "PM-M04"))
# Now the data would be ready for analysis.
```

---

c *Combination and addition of plates*

Description

Combine an OPMX or MOPMX object with other objects.

Usage

```
  ## S4 method for signature 'ANY,MOPMX'
e1 + e2
  ## S4 method for signature 'MOPMX,ANY'
e1 + e2
  ## S4 method for signature 'MOPMX,OPMX'
e1 + e2
  ## S4 method for signature 'OPM,MOPMX'
e1 + e2
  ## S4 method for signature 'OPM,OPM'
e1 + e2
  ## S4 method for signature 'OPM,OPMS'
e1 + e2
  ## S4 method for signature 'OPM,list'
e1 + e2
  ## S4 method for signature 'OPMS,MOPMX'
e1 + e2
  ## S4 method for signature 'OPMS,OPM'
e1 + e2
  ## S4 method for signature 'OPMS,OPMS'
e1 + e2
```

```
  ## S4 method for signature 'OPMS,list'
e1 + e2

  ## S4 method for signature 'MOPMX'
c(x, ..., recursive = FALSE)
  ## S4 method for signature 'OPMX'
c(x, ..., recursive = FALSE)
```

Arguments

x	OPMX or MOPMX object.
...	Other R objects.
recursive	Logical scalar. See c from the **base** package.
e1	OPMX object. If e2 is a MOPMX object, anything that can be converted with as to that class.
e2	OPMX object, or list. If e1 is a MOPMX object, anything that can be converted with as to that class.

Value

The OPMX method of c creates an OPMS object if possible, otherwise a list, or an OPM object (if ... is not given and x is such an object). Similarly, the MOPMX method of c creates a MOPMX object is possible and a list otherwise.

If successful, + yields an OPMS object that contains the plates from both e1 and e2, but it raises an error if the plates cannot be combined.

See Also

base::c

Other combination-functions: $<-, [<-, [[<-, opms

Examples

```
# Adding nothing
dim(x <- c(vaas_1))
stopifnot(identical(x, vaas_1))
dim(x <- c(vaas_4))
stopifnot(identical(x, vaas_4))

# Not particularly useful: adding identical plates!
dim(x <- c(vaas_1, vaas_1)) # yields a two-plate OPMS object
stopifnot(identical(dim(x), c(2L, dim(vaas_1))))

# Also not particularly useful: adding partially identical plates!
dim(x <- c(vaas_4, vaas_1))
stopifnot(identical(dim(x), c(5L, dim(vaas_1))))

# The following examples do not show particularly useful additions, as the
# plates are either entirely or partially identical. Note the changes in the
```

```
# dimensions.

# OPM+OPM method
dim(x <- vaas_1 + vaas_1)
stopifnot(identical(dim(x), c(2L, dim(vaas_1))))

# OPM+OPMS method
dim(x <- vaas_1 + vaas_4)
stopifnot(identical(dim(x), c(5L, dim(vaas_1))))

# OPM+list method
dim(x <- vaas_1 + list(vaas_1, vaas_1))
stopifnot(identical(dim(x), c(3L, dim(vaas_1))))

# OPMS+OPMS method
dim(x <- vaas_4 + vaas_4)
stopifnot(identical(dim(x), c(8L, dim(vaas_4)[-1L])))

# OPMS+OPM method
dim(x <- vaas_4 + vaas_1)
stopifnot(identical(dim(x), c(5L, dim(vaas_1))))

# OPMS+list method
dim(x <- vaas_4 + list(vaas_1))
stopifnot(identical(dim(x), c(5L, dim(vaas_1))))
```

`ci_plot` *Plot point estimates with* CIs

Description

Draw point estimates with their confidence intervals. Used for comparing aggregated values together with their confidence intervals between plates. This method can in most cases **not** be applied to entire plates but to selected wells only.

Usage

```
  ## S4 method for signature 'OPMS'
ci_plot(object, as.labels,
    subset = opm_opt("curve.param"), ...)
  ## S4 method for signature 'data.frame'
ci_plot(object, rowname.sep = " ",
    prop.offset = 0.04, align = "center", col = "blue", na.action = "warn",
    draw.legend = TRUE, legend.field = c(1, 1), x = "topleft", xpd = TRUE,
    vline = 0, split.at = param_names("split.at"), crr = 0.75, ...)
```

Arguments

object	OPMS object or (rarely) a data frame. If an OPMS object, it is in most cases necessary to restrict the plates to at most about one dozen wells. See [for how to achieve this. The data frame method is not normally directly called by an **opm** user but via the OPMS method, unless it is used after extract was applied to a data frame for calculating point estimates and confidence intervals from groups of observations. See there for details. Otherwise, the data frame should be as exported by the OPMS method of extract with ci set to TRUE. There must be a column named param_names("split.at") followed by columns with only numeric values. Columns before that split column, if any, are used for grouping. The rows must entirely comprise triplets representing (i) the point estimate, (ii) the lower and (iii) the upper confidence interval.
as.labels	List. Metadata to be joined and used to draw a legend. Ignored if NULL.
subset	Character scalar. The parameter to plot. Only a single one can be selected. See param_names for the options.
rowname.sep	Character scalar. Used when joining explanatory columns into row labels of the plots.
prop.offset	Numeric scalar. A proportional offset that is added to the vertical range of the panels (after determining the maximum range among all panels to ensure consistency within the plot).
align	Character scalar. How to apply the offset; in addition to the default, 'left' and 'right' are possible.
col	Character scalar. Colour to be used.
na.action	Character scalar. What to do if a confidence interval contains NA values; one of 'ignore', 'warn' and 'error'.
draw.legend	Logical scalar. Ignored if there are no explanatory columns.
legend.field	Two-element numeric vector. Indicates the panel in which the legend is drawn. Subsequent arguments work then relative to this panel. If legend.field has less then two fields, the number of panels is set to 1 (the entire plot), and the legend is drawn relative to that.
x	Legend position, passed to legend from the **graphics** package. Ignored unless draw.legend is TRUE.
xpd	Logical scalar. Also passed to that function.
vline	Numeric scalar with the position on the y-axis of a vertical line to be drawn. Ignored if NULL.
crr	Numeric scalar (column-row-ratio) interpreted as number of columns per number of rows. Determines the arrangement of the subplots.
...	Optional arguments passed to legend, or arguments passed from the OPMS method to the data frame method.
split.at	Character vector. See extract.

Details

The default placement of the legend is currently not necessarily very useful. When plotting entire PM plates, the 'mar' parameter of par most likely would need to be set to a lower value, but it is recommended to plot only subsets of plates, i.e. selected wells.

Value

Character vector describing the plot's legend, returned invisibly.

References

Vaas LAI, Sikorski J, Michael V, Goeker M, Klenk H-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846.

See Also

graphics::plot

Other plotting-functions: heat_map, level_plot, parallelplot, radial_plot, summary, xy_plot

Examples

```
x <- ci_plot(vaas_4[, , 1:3], as.labels = list("Species", "Strain"),
  subset = "A", x = "bottomright", legend.field = NULL)
# note that the values on the y axes are drawn to scale
x
stopifnot(is.character(x), identical(length(x), 4L))
# ... and that the return value contains the legend (even if it is not drawn)

## See also the examples for the data-frame method of extract().
```

---

collect_template *Input metadata*

Description

Either collect a metadata template from OmniLog® CSV comments assisting in later on adding metadata using include_metadata or create a data frame holding potential OPM or OPMS object metadata.

Usage

```
  ## S4 method for signature 'MOPMX'
collect_template(object, outfile = NULL,
    sep = "\t", previous = outfile, md.args = list(),
    selection = opm_opt("csv.selection"), add.cols = NULL, normalize = FALSE,
    instrument = NULL, ..., demo = FALSE)
  ## S4 method for signature 'OPM'
```

```
collect_template(object, outfile = NULL,
    sep = "\t", previous = outfile, md.args = list(),
    selection = opm_opt("csv.selection"), add.cols = NULL, normalize = FALSE,
    instrument = NULL, ..., demo = FALSE)
  ## S4 method for signature 'OPMS'
collect_template(object, outfile = NULL,
    sep = "\t", previous = outfile, md.args = list(),
    selection = opm_opt("csv.selection"), add.cols = NULL, normalize = FALSE,
    instrument = NULL, ..., demo = FALSE)
  ## S4 method for signature 'character'
collect_template(object, outfile = NULL,
    sep = "\t", previous = outfile, md.args = list(),
    selection = opm_opt("csv.selection"), add.cols = NULL, normalize = FALSE,
    instrument = NULL, include = list(), ..., demo = FALSE)

  ## S4 method for signature 'ANY'
to_metadata(object, stringsAsFactors = FALSE,
    optional = TRUE, sep = "\t", strip.white = FALSE, ...)
  ## S4 method for signature 'MOPMX'
to_metadata(object, stringsAsFactors = FALSE,
    optional = TRUE, sep = "\t", strip.white = FALSE, ...)
  ## S4 method for signature 'WMD'
to_metadata(object, stringsAsFactors = FALSE,
    optional = TRUE, sep = "\t", strip.white = FALSE, ...)
  ## S4 method for signature 'WMDS'
to_metadata(object, stringsAsFactors = FALSE,
    optional = TRUE, sep = "\t", strip.white = FALSE, ...)
  ## S4 method for signature 'character'
to_metadata(object, stringsAsFactors = FALSE,
    optional = TRUE, sep = "\t", strip.white = NA, ...)
```

Arguments

object	Character vector or OPM, OPMS or MOPMX object. If a character vector is provided to collect_template, it acts like the names argument of read_opm. That is, if it is a directory name, this is automatically scanned for all CSV and YAML files it contains (unless restrictions with patterns are made). One can also provide file names, or a mixture of file and directory names. Regarding the supported input file formats, see read_single_opm. The OPM, OPMS and MOPMX methods collect a data frame from their input object. to_metadata needs the name of an input file (unnamed character scalar), or any object convertible to a data frame. Might also be WMD or OPMS object. If a named character vector with more than a single element, it is used as the first row of the resulting data frame. This behaviour is mainly intended for using this function after a call to the OPM method of csv_data.
outfile	Character scalar. Ignored if NULL, empty or empty string. Otherwise, interpreted as the name of a CSV output file. If metadata have already been collected in an older file with the same name, old metadata will be kept, identifiers for novel

files will be included, their so far empty entries set to NA. Users who wish to keep the old version can use two distinct names for novel and old files; see previous.

sep — Character scalar. CSV field separator for outfile and for the input file of to_metadata (which ignores the argument unless object is interpreted as input file).

previous — Ignored if empty. Otherwise passed to to_metadata. If it is a file name different from outfile, it is an error if the file does not exist.

md.args — List of other arguments passed to the 'to_metadata' methods.

add.cols — Optional character vector with the names of columns to be added to the result, or NULL. If not empty, names of columns to be added, initially containing NA.

selection — Elements to be extracted from the CSV comments contained in each file. Character vector passed to csv_data.

normalize — Logical scalar also passed to csv_data.

instrument — Logical scalar or scalar convertible to integer, or empty. Ignored if empty. If logical and TRUE, opm_opt("machine.id") is inserted as additional column. Otherwise, instrument is used directly.

include — File inclusion pattern (or generator for a pattern). Passed to batch_collect.

... — Other arguments passed to batch_collect, or to read.delim or as.data.frame.

demo — Logical scalar. Run in 'demo' mode? Also passed to batch_collect. If TRUE, file input and output would be omitted and only the respective file names shown.

stringsAsFactors — Logical scalar passed to as.data.frame.

optional — Logical scalar passed to as.data.frame or used after negation as 'check.names' argument of read.delim.

strip.white — Logical scalar. For the file-name method, passed to read.delim (and set to TRUE if it is NA). It is often advisable to set this to FALSE if CSV input is done for a later call to collect_template. For a character vector not interpreted as file name, set to FALSE if NA.

Details

The character method batch-collects templates for meta-information from files and optionally add these data as novel rows to previously collected data. It writes the collected template to a file for use with an external editor, and/or creates a data frame for editing the data directly in R with the edit function.

The to_metadata character method reads metadata from an input file and is only a thin wrapper for read.delim but contains some useful adaptations (such as **not** converting strings to factors, and not modifying column names). The default method reads metadata from an object convertible to a data frame and is only a thin wrapper of as.data.frame but contains the same useful adaptations as the file-name method.

The WMD and OPMS methods create a data frame from the contained metadata, where necessary converting nested metadata entries to data-frame columns of mode 'list'. The number of rows of the resulting data frame corresponds to the length of object, the number of columns to the size of the set created from all valid names at the top level of the metadata entries.

Value

to_metadata yields a data frame. Regarding collect_template, in the case of the character method, a data frame, returned invisibly if outfile is given; if demo is TRUE, a character vector of file names instead, returned invisibly. The OPM method returns a data frame with one row and the number of columns equal to the sum of the lengths of selection and add.cols. The OPM method returns such a data frame with one row per contained plate.

References

http://www.biolog.com/

See Also

base::default.stringsAsFactors base::as.data.frame

utils::edit utils::read.delim

Other io-functions: batch_collect, batch_opm, batch_process, clean_filenames, explode_dir, file_pattern, glob_to_regex, read_opm, read_single_opm, split_files

Examples

```
## collect_template()

# Character method
test.files <- opm_files("omnilog")
if (length(test.files) > 0) { # if the files are found

  # Without writing to a file
  (x <- collect_template(test.files))
  stopifnot(is.data.frame(x), identical(x[, "File"], test.files))
  # now proceed with e.g.
  # x <- edit(x)

  # Write to file
  outfile <- tempfile()
  stopifnot(!file.exists(outfile))
  # This results in a CSV outfile which could be used as a starting point
  # for including the metadata of interest together with the plate
  # identifiers in a single file. include_metadata() can then be used to
  # integrate the metadata in OPM, OPMA or OPMS objects.
  x <- collect_template(test.files, outfile = outfile)
  stopifnot(file.exists(outfile))
  unlink(outfile)
} else {
  warning("test files not found")
}

# OPM method
(x <- collect_template(vaas_1)) # => data frame, one row per plate
stopifnot(dim(x) == c(1, 3))
(x <- collect_template(vaas_1, instrument = TRUE))
```

```
stopifnot(dim(x) == c(1, 4))
(x <- collect_template(vaas_1, add.cols = c("A", "B")))
stopifnot(dim(x) == c(1, 5)) # => data frame with more columns
# see include_metadata() for how to use this to add metadata information

# OPMS method
(x <- collect_template(vaas_4)) # => data frame, one row per plate
stopifnot(identical(dim(x), c(4L, 3L)))
(x <- collect_template(vaas_4, add.cols = c("A", "B")))
stopifnot(identical(dim(x), c(4L, 5L))) # => data frame with more columns
# again see include_metadata() for how to use this to add metadata
# information

## to_metadata()

# Character method
(x <- to_metadata(list(a = 7:8, `b c` = letters[1:2])))
tmpfile <- tempfile()
write.table(x, tmpfile, row.names = FALSE, sep = "\t")
(x1 <- read.delim(tmpfile)) # comparison with base R function
(x2 <- to_metadata(tmpfile))
stopifnot(identical(names(x2), names(x)), !identical(names(x1), names(x)))

# Default method
x <- list(a = 7:8, `b c` = letters[1:2])
(x1 <- as.data.frame(x))
(x2 <- to_metadata(x))
stopifnot(!identical(names(x), names(x1)), identical(names(x), names(x2)))

# WMD method
(x <- to_metadata(vaas_1)) # one row per OPM object
stopifnot(is.data.frame(x), nrow(x) == length(vaas_1), ncol(x) > 0)

# OPMS method
(x <- to_metadata(vaas_4)) # one row per OPM object
stopifnot(is.data.frame(x), nrow(x) == length(vaas_4), ncol(x) > 0)
copy <- vaas_4
metadata(copy) <- x
stopifnot(identical(copy, vaas_4))
# ... this works only in the special case of non-nested metadata that
# have the same set of entries in all OPMS elements
```

`csv_data` *Information from input* CSV *file*

Description

Information about the plate as originally read from the input CSV file (see `read_opm` and `read_single_opm` for reading such files).

Usage

```
  ## S4 method for signature 'MOPMX'
csv_data(object, ...)
  ## S4 method for signature 'OPM'
csv_data(object,
    keys = character(), strict = TRUE,
    what = c("select", "filename", "setup_time", "position", "other"),
    normalize = FALSE)
  ## S4 method for signature 'OPMS'
csv_data(object, ...)
```

Arguments

object	OPM, OPMS or MOPMX object.
keys	Character vector (or other objects usable as vector index). An optional subselection. If empty (the default), all CSV data are returned. By default it is an error to select non-existing items. Ignored unless what is 'select'.
strict	Logical scalar indicating whether or not it is an error if keys are not found. Ignored unless what is 'select'.
what	Character scalar specifying a subset of the data. If 'select', use keys and strict. If 'other', select all CSV entries that have no special meaning (it makes sense to include only these in the metadata, see the examples). Otherwise a shortcut for one of the more important CSV entries.
normalize	Logical scalar indicating whether plate position and setup time entries (if selected) should be normalised. This should always work for the plate positions, but for the setup times it depends on the values for the opm_opt keys time.fmt and time.zone (see also merge). For other entries, normalisation means replacing backslashes by slashes.
...	Optional arguments passed between the methods.

Details

It is easy to copy the CSV data to the metadata; see the examples section. Editing of the CSV data has deliberately not been implemented into **opm**, but the metadata can be modified using a plethora of methods, even manually with edit.

Value

For the OPM method, a named character vector (unnamed character scalar in the case of filename, setup_time and filename and if what is not 'select'). For the other methods either a named character vector (if the selection yields a single entry) or a character matrix with one row per plate. Missing entries in one of the plates yield NA in the output.

See Also

base::strptime

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, dim, disc_settings, discretized, duplicated, has_aggr, has_disc, hours, max, measurements, minmax, seq, subset, thin_out, well

Examples

```
## 'OPM' method

(x <- csv_data(vaas_1, "Setup Time"))
stopifnot(identical(x, c(`Setup Time` = "8/30/2010 1:53:08 PM")))
# compare this to 'what = "setup_time"'; here, names are kept
(y <- csv_data(vaas_1, "Setup Time", normalize = TRUE))
stopifnot(!is.na(y), y != x, names(y) == names(x))

(x <- csv_data(vaas_1, what = "filename")) # one file name (of course)
stopifnot(is.character(x), length(x) == 1L)

(x <- csv_data(vaas_1, what = "position")) # single position (of course)
(y <- csv_data(vaas_1, what = "position", normalize = TRUE))
stopifnot(x == " 7-B", y == "07-B") # four characters when normalised

(x <- csv_data(vaas_1, what = "setup_time")) # single setup time (of course)
(y <- csv_data(vaas_1, what = "setup_time", normalize = TRUE))
stopifnot(length(x) == 1, x != y) # converted to canonical data/time format
# WARNING: It is unlikely that all OmniLog output has the setup-time format
# defined by default in opm_opt("time.fmt")

## 'OPMS' method

(x <- csv_data(vaas_4, "Setup Time")) # one setup time per plate
stopifnot(is.character(x), length(x) == 4)

(x <- csv_data(vaas_4, what = "filename"))  # one file name per plate
stopifnot(is.character(x), length(x) == 4L)

(x <- csv_data(vaas_4, what = "position")) # one position per plate
stopifnot(is.character(x), length(x) == length(vaas_4))

(x <- csv_data(vaas_4, what = "setup_time")) # one setup time per plate
(y <- csv_data(vaas_4, what = "setup_time", normalize = TRUE))
stopifnot(length(x) == 4, x != y) # converted to canonical data/time format
# see the warning above

## Useful application: copying selected CSV data to the metadata

x <- vaas_4
# this appends the CSV data after conversion to a suitable data frame
metadata(x, -1) <- to_metadata(csv_data(x, what = "other"))
to_metadata(x)
stopifnot(sapply(metadata(x), length) > sapply(metadata(vaas_4), length))
```

dim *Get dimensions*

Description

Get the dimensions of the measurements of an OPM object, or get the dimensions of an OPMS object, or the number of plates stored in an OPMX object, or the indexes of all these plates.

Usage

```
  ## S4 method for signature 'OPM'
dim(x)
  ## S4 method for signature 'OPMS'
dim(x)

  ## S4 method for signature 'WMD'
length(x)
  ## S4 method for signature 'WMDS'
length(x)

  ## S4 method for signature 'WMD'
seq(...)
  ## S4 method for signature 'WMDS'
seq(...)
```

Arguments

x	OPMX object.
...	OPMS objects. Several ones can be provided, but all but the first one are ignored. For reasons of comparability, the OPM method of seq deliberately results in an error.

Details

Note that dim cannot be used to determine the correspondence of the time points between all plates as it reports only the time points of the first plate. Instead the OPMS method of hours must be used.

seq yields the indexes of all plates contained in an OPMS object. This is mainly useful for looping over such objects. See [for a loop-construct usage example, and note that oapply is also available.

Value

For the OPM method of dim, a two-element numeric vector (number of time points and number of wells). For the OPMS method, a numeric vector with (i) the number of contained OPM objects, and (ii) and (iii) the dimensions of the first plate. length returns an integer scalar. This seq method yields an integer vector (starting with 1 and at least of length 2).

See Also

base::dim base::length base::seq

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, disc_settings, discretized, duplicated, has_aggr, has_disc, hours, max, measurements, minmax, subset, thin_out, well

Examples

```
# OPM methods
(x <- dim(vaas_1))
stopifnot(identical(x, c(384L, 96L)))
(x <- length(vaas_1))
stopifnot(identical(x, 1L)) # 1 plate contained
(x <- try(seq(vaas_1), silent = TRUE)) # deliberately yields an error
stopifnot(inherits(x, "try-error"))

# OPMS methods
(x <- dim(vaas_4)) # 2nd value needs not be correct for all plates
stopifnot(identical(x, c(4L, 384L, 96L)))
(x <- length(vaas_4))
stopifnot(identical(x, 4L)) # 4 plates contained
(x <- seq(vaas_4))
stopifnot(identical(x, 1:4)) # indexes for 4 plates
(y <- seq(vaas_4, letters, LETTERS)) # other arguments are ignored
stopifnot(identical(x, y))
```

---

discrete *Discretisation functions*

Description

These are the helper functions called by do_disc (which is the function normally applied by an **opm** user). discrete converts continuous numeric characters to discrete ones. best_cutoff determines the best cutoff for dividing a numeric matrix into two categories by minimising within-group discrepancies. That is, for each combination of row group and column maximise the number of contained elements that are in the category in which most of the elements within this combination of row group and column are located.

Usage

```
  ## S4 method for signature 'matrix,character'
best_cutoff(x, y, ...)
  ## S4 method for signature 'matrix,factor'
best_cutoff(x, y,
    combined = TRUE, lower = min(x, na.rm = TRUE),
    upper = max(x, na.rm = TRUE), all = FALSE)

  ## S4 method for signature 'array'
```

```
discrete(x, ...)
  ## S4 method for signature 'data.frame'
discrete(x, ..., as.labels = NULL,
    sep = " ")
  ## S4 method for signature 'numeric'
discrete(x, range, gap = FALSE,
    output = c("character", "integer", "logical", "factor", "numeric"),
    middle.na = TRUE, states = 32L, ...)
```

Arguments

x	Numeric vector or array object convertible to a numeric vector. The data-frame method first calls extract, restricting the columns to the numeric ones. best_cutoff only accepts a numeric matrix.
range	If a numeric vector, in non-gap mode (see next argument) the assumed real range of the data; must contain all elements of x, but can be much wider. In gap mode, it must, in contrast, lie within the range of x. If range is set to TRUE, the empirical range of x is used in non-gap mode. In gap mode, the range is determined using run_kmeans with the number of clusters set to 3 and then applying borders to the result. The number of clusters is set to 2 if range is FALSE in gap mode.
gap	Logical scalar. If TRUE, always convert to binary or ternary characters, ignoring states. range then indicates a partial range of x within which character conversion is ambiguous and has to be treated as either missing information or intermediate character state, depending on middle.na. If FALSE (the default), apply an equal-width-intervals discretisation with the widths determined from the number of requested states and range.
output	String determining the output mode: 'character', 'integer', 'logical', 'factor', or 'numeric'. 'numeric' simply returns x, but performs the range checks. One cannot combine 'logical' with TRUE values for both gap and middle.na.
middle.na	Logical scalar. Only relevant in gap mode. In that case, if TRUE, the middle value yields NA (uncertain whether negative or positive). If FALSE, the middle value lies between the left and the right one (i.e., a third character state meaning 'weak'). This is simply coded as 0-1-2 and thus cannot be combined with 'logical' as output setting.
states	Integer or character vector. Ignored in gap mode and if output is not 'character'. Otherwise, the possible values are • a single-element character vector, which is split into its elements; • a multiple-element character vector which is used directly; • an integer vector indicating the elements to pick from the default character states. In the latter case, a single integer is interpreted as the upper bound of an integer vector starting at 1.
as.labels	Vector of data-frame indexes. See extract. (If given, this argument must be named.)

sep	Character scalar. See extract. (If given, this argument must be named.)
y	Factor or character vector indicating group affiliations. Its length must correspond to the number of rows of x.
combined	Logical scalar. If TRUE, determine a single threshold for the entire matrix. If FALSE, determine one threshold for each group of rows of x that corresponds to a level of y.
lower	Numeric scalar. Lower bound for the cutoff values to test.
upper	Numeric scalar. Upper bound for the cutoff values to test.
all	Logical scalar. If TRUE, calculate the score for all possible cutoffs for x. This is slow and is only useful for plotting complete optimisation curves.
...	Optional arguments passed between the methods or, if requested, to run_kmeans (except object and k, see there).

Details

One of the uses of discrete is to create character data suitable for phylogenetic studies with programs such as PAUP* and RAXML. These accept only discrete characters with at most 32 states, coded as 0 to 9 followed by A to V. For the full export one additionally needs phylo_data. The matrix method is just a wrapper that takes care of the matrix dimensions, and the data-frame method is a wrapper for that method.

The term 'character' as used here has no direct connection to the eponymous mode or class of R. Rather, the term is borrowed from taxonomic classification in biology, where, technically, a single 'character' is stored in one column of a data matrix if each organism is stored in one row. Characters are the *quasi-independent units* of evolution on the one hand and of phylogenetic reconstruction (and thus taxonomic classification) on the other hand.

The scoring function to be maximised by best_cutoff is calculated as follows. All values in x are divided into those larger then the cutoff and those at most large as the cutoff. For each combination of group and matrix column the frequencies of the two categories are determined, and the respective larger ones are summed up over all combinations. This value is then divided by the frequency over the entire matrix of the more frequent of the two categories. This is done to avoid trivial solutions with minimal and maximal cutoffs, causing all values to be placed in the same category.

Value

discrete generates a double, integer, character or logical vector or factor, depending on output. For the matrix method, a matrix composed of a vector as produced by the numeric method, the original dimensions and the original dimnames attributes of x.

If combined is TRUE, best_cutoff yields either a matrix or a vector: If all is TRUE, a two-column matrix with (i) the cutoffs examined and (ii) the resulting scores. If all is FALSE, a vector with the entries 'maximum' (the best cutoff) and 'objective' (the score it achieved). If combined is FALSE, either a list of matrices or a matrix. If all is TRUE, a list of matrices structures like the single matrix returned if combined is TRUE. If all is FALSE, a matrix with two columns called 'maximum' 'objective', and one row per level of y.

References

Dougherty, J., Kohavi, R., Sahami, M. 1995 Supervised and unsupervised discretisation of continuous features. In: Prieditis, A., Russell, S. (eds.) *Machine Learning: Proceedings of the fifth international conference*.

Ventura, D., Martinez, T. R. 1995 An empirical comparison of discretisation methods. *Proceedings of the Tenth International Symposium on Computer and Information Sciences*, p. 443–450.

Wiley, E. O., Lieberman, B. S. 2011 *Phylogenetics: Theory and Practice of Phylogenetic Systematics*. Hoboken, New Jersey: Wiley-Blackwell.

Bunuel, L. 1972 *Le charme discret de la bourgeoisie.* France/Spain, 96 min.

See Also

base::cut stats::optimize

Other discretization-functions: do_disc

Examples

```
# Treat everything between 3.4 and 4.5 as ambiguous
(x <- discrete(1:5, range = c(3.5, 4.5), gap = TRUE))
stopifnot(x == c("0", "0", "0", "?", "1"))

# Treat everything between 3.4 and 4.5 as intermediate
(x <- discrete(1:5, range = c(3.5, 4.5), gap = TRUE, middle.na = FALSE))
stopifnot(x == c("0", "0", "0", "1", "2"))

# Boring example: real and possible range as well as the number of states
# to code the data have a 1:1 relationship
(x <- discrete(1:5, range = c(1, 5), states = 5))
stopifnot(identical(x, as.character(0:4)))

# Now fit the data into a potential range twice as large, and at the
# beginning of it
(x <- discrete(1:5, range = c(1, 10), states = 5))
stopifnot(identical(x, as.character(c(0, 0, 1, 1, 2))))

# Matrix and data-frame methods
x <- matrix(as.numeric(1:10), ncol = 2)
(y <- discrete(x, range = c(3.4, 4.5), gap = TRUE))
stopifnot(identical(dim(x), dim(y)))
(yy <- discrete(as.data.frame(x), range = c(3.4, 4.5), gap = TRUE))
stopifnot(y == yy)

# K-means based discretisation of PM data (prefer do_disc() for this)
x <- extract(vaas_4, as.labels = list("Species", "Strain"),
  in.parens = FALSE)
(y <- discrete(x, range = TRUE, gap = TRUE))[, 1:3]
stopifnot(c("0", "?", "1") %in% y)

## best_cutoff()
x <- matrix(c(5:2, 1:2, 7:8), ncol = 2)
```

```
grps <- c("a", "a", "b", "b")

# combined optimisation
(y <- best_cutoff(x, grps))
stopifnot(is.numeric(y), length(y) == 2) # two-element numeric vector
stopifnot(y[["maximum"]] < 4, y[["maximum"]] > 3, y[["objective"]] == 2)
plot(best_cutoff(x, grps, all = TRUE), type = "l")

# separate optimisation
(y <- best_cutoff(x, grps, combined = FALSE))
stopifnot(is.matrix(y), dim(y) == c(2, 2)) # numeric matrix
stopifnot(y["a", "objective"] == 2, y["b", "objective"] == 2)
(y <- best_cutoff(x, grps, combined = FALSE, all = TRUE))
plot(y$a, type = "l")
plot(y$b, type = "l")
```

`discretized` *Get discretised data*

Description

Get the discretised kinetic data or the discretisation settings used. (See `do_disc` for generating discretised data.)

Usage

```
  ## S4 method for signature 'MOPMX'
disc_settings(object, join = NULL)
  ## S4 method for signature 'OPMD'
disc_settings(object, join = NULL)
  ## S4 method for signature 'OPMS'
disc_settings(object, join = NULL)

  ## S4 method for signature 'MOPMX'
discretized(object, ...)
  ## S4 method for signature 'OPMD'
discretized(object, full = FALSE, in.parens = TRUE,
    max = opm_opt("max.chars"), ...)
  ## S4 method for signature 'OPMS'
discretized(object, ...)
```

Arguments

`object`	`OPMD`, `OPMS` or `MOPMX` object.
`full`	Logical scalar passed to `wells`. This and the following arguments affect the names of the resulting vector.
`in.parens`	Logical scalar also passed to that function.
`max`	Numeric scalar also passed to that function.

join	Empty or character scalar. Works like the eponymous argument of aggr_settings; see there for details.
...	Optional arguments passed between the methods or to wells.

Value

Logical vector or matrix in the case of discretized, named list in the case of disc_settings. See the examples for details.

See Also

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, dim, duplicated, has_aggr, has_disc, hours, max, measurements, minmax, seq, subset, thin_out, well

Examples

```
# 'OPM' methods
(x <- discretized(vaas_1))[1:3] # => logical vector
stopifnot(is.logical(x), !is.matrix(x), length(x) == dim(x)[2L])
stopifnot(names(x) == colnames(aggregated(vaas_1)))
(x <- discretized(vaas_1, full = TRUE))[1:3] # => with full names
stopifnot(names(x) == colnames(aggregated(vaas_1, full = TRUE)))

# settings
(x <- disc_settings(vaas_1)) # => named list
stopifnot(is.list(x), !is.null(names(x)))
(x <- disc_settings(vaas_1, join = "yaml")) # matrix, one row per plate
stopifnot(is.matrix(x), is.character(x), nrow(x) == 1)

# 'OPMS' methods
(x <- discretized(vaas_4))[, 1:3] # => logical matrix
stopifnot(is.logical(x), is.matrix(x), ncol(x) == dim(x)[2L])
stopifnot(colnames(x) == colnames(aggregated(vaas_1)))

# settings
summary(x <- disc_settings(vaas_4)) # => list of named lists, one per plate
stopifnot(is.list(x), is.null(names(x)), length(x) == length(vaas_4))
stopifnot(duplicated(x)[-1])
(x <- disc_settings(vaas_4, join = "json")) # matrix, one row per plate
stopifnot(is.matrix(x), is.character(x), nrow(x) == 4)
```

---

do_aggr *Aggregate kinetics using curve-parameter estimation*

Description

Aggregate the kinetic data using curve-parameter estimation, i.e. infer parameters from the kinetic data stored in an OPM object using either the **grofit** package or the built-in method. Optionally include the aggregated values in a novel OPMA object together with previously collected information.

Usage

```
  ## S4 method for signature 'MOPMX'
do_aggr(object, ...)
  ## S4 method for signature 'OPM'
do_aggr(object, boot = 100L, verbose = FALSE,
    cores = 1L, options = list(), method = "grofit", plain = FALSE)
  ## S4 method for signature 'OPMS'
do_aggr(object, ...)
  ## S4 method for signature 'matrix'
do_aggr(object, what = c("AUC", "A"),
    boot = 100L, ci = 0.95, as.pe = "median", ci.type = "norm",
    time.pos = 1L, transposed = FALSE, raw = FALSE, ...)
```

Arguments

object	OPM, OPMS or MOPMX object or matrix as output by measurements, i.e. with the time points in the first columns and the measurements in the remaining columns (there must be at least two). For deviations from this scheme see time.pos and transposed.
boot	Integer scalar. Number of bootstrap replicates used to estimate 95-percent confidence intervals (CIs) for the parameters. Set this to zero to omit bootstrapping, resulting in NA entries for the CIs. Note that under the default settings of the matrix method for as.pe, bootstrapping is also necessary to obtain the point estimate.
verbose	Logical scalar. Print progress messages?
cores	Integer scalar. Number of cores to use. Setting this to a value > 1 requires that mclapply from the **parallel** package can be run with more than 1 core, which is impossible under Windows. The cores argument has no effect if opm-fast is chosen (see below).
options	List. For its use in **grofit** mode, see grofit.control in that package. The boot and verbose settings, as the most important ones, are added separately (see above). The verbose mode is not very useful in parallel processing. With method "spline.fit", options can be specified using the function set_spline_options.
method	Character scalar. The aggregation method to use. Currently only the following methods are supported: **splines** Fit various splines (smoothing splines and P-splines from **mgcv** and smoothing splines via smooth.spline) to PM data. Recommended. **grofit** The grofit function in the eponymous package, with spline fitting as default. **opm-fast** The native, faster parameter estimation implemented in the matrix method. This will only yield two of the four parameters, the area under the curve and the maximum height. The area under the curve is estimated as the sum of the areas given by the trapezoids defined by each pair of adjacent time points. The maximum height is just the result of max. By default the median of the bootstrap values is used as point estimate. For details see as.pe.

plain	Logical scalar. If TRUE, only the aggregated values are returned (as a matrix, for details see below). Otherwise they are integrated in an OPMA object together with object.
what	Character scalar. Which parameter to estimate. Currently only two are supported.
ci	Confidence interval to use in the output. Ignored if boot is not positive.
as.pe	Character scalar determining what to output as the point estimate. Either median, mean or pe; the first two calculate the point estimate from the bootstrapping replicates, the third one use the point estimate from the raw data. If boot is 0, as.pe is reset to pe, if necessary, and a warning is issued.
ci.type	Character scalar determining the way the confidence intervals are calculated. Either norm, basic or perc; see boot.ci from the **boot** package for details.
time.pos	Character or integer scalar indicating the position of the column (or row, see next argument) with the time points.
transposed	Character or integer scalar indicating whether the matrix is transposed compared to the default.
raw	Logical scalar. Return the raw bootstrapping result without CI estimation and construction of the usually resulting matrix?
...	Optional arguments passed between the methods or to boot from the eponymous package.

Details

Behaviour is special if the plate_type is one of those that have to be explicitly set using gen_iii and there is just one point measurement. Because this behaviour is usual for plates measured either in Generation-III (identification) mode or on a MicroStation™, the point estimate is simply regarded as 'A' parameter (maximum height) and all other parameters are set to NA.

The OPMS method just applies the OPM method to each contained plate in turn; there are no interdependencies.

Examples with plain = TRUE are not given, as only the return value is different: Let x be the normal result of do_aggr(). The matrix returned if plain is TRUE could then be received using aggregated(x), whereas the 'method' and the 'settings' attributes could be obtained as components of the list returned by aggr_settings(x).

The matrix method quickly estimates the curve parameters AUC (area under the curve) or A (maximum height). This is not normally directly called by an **opm** user but via the other do_aggr methods.

The aggregated values can be queried for using has_aggr and received using aggregated.

Value

If plain is FALSE, an OPMA object. Otherwise a numeric matrix of the same structure than the one returned by aggregated but with an additional 'settings' attribute containing the (potentially modified) list proved via the settings argument, and a 'method' attribute corresponding to the method argument.

The matrix method returns a numeric matrix with three rows (point estimate, lower and upper CI) and as many columns as data columns (or rows) in `object`. If `raw` is `TRUE`, it returns an object of the class 'boot'.

References

Brisbin, I. L., Collins, C. T., White, G. C., McCallum, D. A. 1986 A new paradigm for the analysis and interpretation of growth data: the shape of things to come. *The Auk* **104**, 552–553.

Efron, B. 1979 Bootstrap methods: another look at the jackknife. *Annals of Statistics* **7**, 1–26.

Kahm, M., Hasenbrink, G., Lichtenberg-Frate, H., Ludwig, J., Kschischo, M. grofit: Fitting biological growth curves with R. *Journal of Statistical Software* **33**, 1–21.

Vaas, L. A. I., Sikorski, J., Michael, V., Goeker, M., Klenk H.-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846.

See Also

grofit::grofit

Examples

```
# OPM method

# Run a fast estimate of A and AUC without bootstrapping
copy <- do_aggr(vaas_1, method = "opm-fast", boot = 0,
  options = list(as.pe = "pe"))
aggr_settings(vaas_1)
aggr_settings(copy)
stopifnot(has_aggr(vaas_1), has_aggr(copy))

# Compare the results to the ones precomputed with grofit
# (1) A
a.grofit <- aggregated(vaas_1, "A", ci = FALSE)
a.fast <- aggregated(copy, "A", ci = FALSE)
plot(a.grofit, a.fast)
stopifnot(cor.test(a.fast, a.grofit)$estimate > 0.999)
# (2) AUC
auc.grofit <- aggregated(vaas_1, "AUC", ci = FALSE)
auc.fast <- aggregated(copy, "AUC", ci = FALSE)
plot(auc.grofit, auc.fast)
stopifnot(cor.test(auc.fast, auc.grofit)$estimate > 0.999)

## Not run:  # Without confidence interval (CI) estimation
  x <- do_aggr(vaas_1, boot = 0, verbose = TRUE)
  aggr_settings(x)
  aggregated(x)

  # Calculate CIs with 100 bootstrap (BS) replicates, using 4 cores
  # (do not try to use > 1 core on Windows)
  x <- do_aggr(vaas_1, boot = 100, verbose = TRUE, cores = 4)
  aggr_settings(x)
```

```
  aggregated(x)

## End(Not run)

# matrix method
(x <- do_aggr(measurements(vaas_1)))[, 1:3]
stopifnot(identical(dim(x), c(3L, 96L)))
```

---

do_disc *Discretise curve parameters*

Description

Discretise the aggregated kinetic data, i.e. infer discrete values from the curve parameters stored in an OPMA or OPMS object. Here, only discretisation into positive, negative and ambiguous reactions is supported, and by default based on the 'maximum height' curve parameter (which is biologically reasonable though).

Usage

```
  ## S4 method for signature 'MOPMX'
do_disc(object, ...)
  ## S4 method for signature 'OPMA'
do_disc(object, cutoff, groups = FALSE,
    plain = FALSE, subset = opm_opt("disc.param"), unify = FALSE)
  ## S4 method for signature 'OPMS'
do_disc(object, cutoff = TRUE, groups = FALSE,
    plain = FALSE, subset = opm_opt("disc.param"), unify = !length(cutoff),
    ...)
```

Arguments

object	OPMA, OPMS or MOPMX object.
cutoff	Determines the discretisation approach. If non-empty, passed as range argument to discrete (with gap set to TRUE), thus triggering discretisation using either k-means partitioning or one or two predefined thresholds. If empty (e.g., NULL), a discretisation cutoff is determined using best_cutoff, which is only possible for OPMS objects.
groups	List, NULL or character vector passed as 'as.labels' argument to extract, or logical scalar. In that case, if TRUE, groups are automatically created with one plate per group. If FALSE, grouping is not used, i.e. there is only a single group containing all plates. Note that if cutoff is empty and groups is TRUE, an error is raised since best_cutoff needs groups with more than a single element. The groups argument has no effect on OPMA objects.
plain	Logical scalar indicating whether or not an OPMD or OPMS object should be created.

subset — Character scalar passed to extract. It is recommended to use the maximum height (currently called 'A').

unify — Logical or numeric scalar indicating whether results should be unified per group. This works by choosing the most frequent value (mode) if its frequency is above a given threshold and NA otherwise. (The same approach is used by listing and phylo_data.)

If unify is a logical scalar, NA triggers unification using 1 as threshold, i.e. all ambiguities are codes as NA. Using TRUE turns on unification with the default threshold given by opm_opt("min.mode"), whereas FALSE turns unification off.

If unify is a numeric scalar, values below or equal to zero turn unification off. Values above zero are directly used as unification threshold, thus values above 1 or numeric NA make no sense (cause an error).

See 'Details' below on the potential consequences of unification. In the disc_settings entries, an according 'unified' entry will report the threshold used, with -1 indicating no unification.

The unify argument has no effect on OPMA objects (because they represent a single group with a single member).

... — Optional arguments passed between the methods or to extract. The latter is only relevant for certain settings of groups, see above.

Details

If unify is set to FALSE, the discretisation results are always consistent (in the sense described for the OPMD class) with the discretised parameter. If unify is set to TRUE this cannot be guaranteed any more. To enforce consistency, use opm_opt(strict.OPMD = TRUE).

The discretised values can be queried for using has_disc and received using discretized.

Value

If plain is FALSE, an OPMD or OPMS object. Otherwise a logical vector whose length corresponds to the number of wells in object with an additional 'settings' attribute describing the run. The vector and its attribute would correspond to the discretized and disc_settings entries of a resulting OPMD object, respectively.

See Also

Other discretization-functions: best_cutoff, discrete

Examples

```
## OPMA method

# arbitrary threshold, no ambiguity
summary(x <- do_disc(vaas_1, cutoff = 100))
stopifnot(has_disc(x), dim(x) == dim(vaas_1), !is.na(discretized(x)))
(y <- disc_settings(x))  # stored discretisation settings
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists
```

```
# arbitrary thresholds, allowing intermediate ('weak') reactions
summary(x <- do_disc(vaas_1, cutoff = c(75, 125)))
# the intermediate reactions are coded as NA
stopifnot(has_disc(x), dim(x) == dim(vaas_1), any(is.na(discretized(x))))
(y <- disc_settings(x)) # stored discretisation settings
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists

# using k-means, two categories, no intermediate ('weak') reactions
summary(x <- do_disc(vaas_1, cutoff = FALSE))
stopifnot(has_disc(x), dim(x) == dim(vaas_1), !is.na(discretized(x)))
(y <- disc_settings(x)) # stored discretisation settings
stopifnot(identical(y$method, "kmeans"))
stopifnot(is.list(y), is.list(y$options)) # named lists

# using k-means, now allowing intermediate ('weak') reactions
summary(x <- do_disc(vaas_1, cutoff = TRUE))
stopifnot(has_disc(x), dim(x) == dim(vaas_1), any(discretized(x)))
(y <- disc_settings(x)) # stored discretisation settings
stopifnot(identical(y$method, "kmeans"))
stopifnot(is.list(y), is.list(y$options)) # named lists


## OPMS method

# arbitrary threshold, no ambiguity, no groups
x <- do_disc(vaas_4, cutoff = 100)
stopifnot(has_disc(x), dim(x) == dim(vaas_4), !is.na(discretized(x)))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists

# arbitrary threshold, no ambiguity, no groups, with unification
x <- do_disc(vaas_4, cutoff = 100, unify = TRUE)
stopifnot(has_disc(x), dim(x) == dim(vaas_4))
stopifnot(any(is.na(discretized(x)))) # NAs caused by unification
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# all plates made uniform (makes not much sense)

# arbitrary threshold, no ambiguity, with groups, 1 plate per group
x <- do_disc(vaas_4, cutoff = 100, groups = TRUE)
stopifnot(has_disc(x), dim(x) == dim(vaas_4), !is.na(discretized(x)))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# here, the plate numbers yield the group names

# arbitrary threshold, no ambiguity, with specified groups
x <- do_disc(vaas_4, cutoff = 100, groups = "Species")
stopifnot(has_disc(x), dim(x) == dim(vaas_4), !is.na(discretized(x)))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
```

```
stopifnot(identical(y$method, "direct"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# now, groups are from the metadata (but played no role)

# using k-means, no ambiguity, with specified groups
x <- do_disc(vaas_4, cutoff = FALSE, groups = "Species")
stopifnot(has_disc(x), dim(x) == dim(vaas_4), !is.na(discretized(x)))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "kmeans"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# grouping by species, discretised separately

# same, with unification
x <- do_disc(vaas_4, cutoff = FALSE, groups = "Species", unify = TRUE)
stopifnot(has_disc(x), dim(x) == dim(vaas_4))
stopifnot(any(is.na(discretized(x)))) # NAs caused by unification
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "kmeans"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# grouping by species, discretised separately, then made uniform

# using best_cutoff(), groups defined by species affiliation (makes not
# much sense and by default yields warnings with these data)
x <- do_disc(vaas_4, cutoff = NULL, groups = "Species")
stopifnot(has_disc(x), dim(x) == dim(vaas_4), any(is.na(discretized(x))))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "best-cutoff"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# groups as above, 2 strains per species, but some additional entries

# using best_cutoff(), single group for all plates (makes even less sense
# and by default also yields warnings with these data)
x <- do_disc(vaas_4, cutoff = NULL, groups = FALSE)
stopifnot(has_disc(x), dim(x) == dim(vaas_4), any(is.na(discretized(x))))
(y <- disc_settings(x)[[1]]) # stored discretisation settings
stopifnot(identical(y$method, "best-cutoff"))
stopifnot(is.list(y), is.list(y$options)) # named lists
# no subgroups, all 4 data sets in one group, and some additional entries
```

`duplicated` *Determine duplicated plates*

Description

Check whether some, or duplicated, `OPM` objects are contained within an `OPMS` object, or whether `OPMX` objects are contained within an `MOPMX` object. For reasons of consistency, the `OPM` methods always returns `FALSE` or `0`. Alternatively, query `OPMX` objects with other such objects.

Usage

```
  ## S4 method for signature 'MOPMX,ANY'
anyDuplicated(x, incomparables, ...)
  ## S4 method for signature 'MOPMX,missing'
anyDuplicated(x, incomparables,
    ...)
  ## S4 method for signature 'OPM,ANY'
anyDuplicated(x, incomparables, ...)
  ## S4 method for signature 'OPM,missing'
anyDuplicated(x, incomparables, ...)
  ## S4 method for signature 'OPMS,ANY'
anyDuplicated(x, incomparables, ...)
  ## S4 method for signature 'OPMS,missing'
anyDuplicated(x, incomparables, ...)

  ## S4 method for signature 'MOPMX,MOPMX'
contains(object, other, ...)
  ## S4 method for signature 'MOPMX,OPMX'
contains(object, other, ...)
  ## S4 method for signature 'OPM,OPM'
contains(object, other, ...)
  ## S4 method for signature 'OPM,OPMS'
contains(object, other, ...)
  ## S4 method for signature 'OPMS,OPM'
contains(object, other, ...)
  ## S4 method for signature 'OPMS,OPMS'
contains(object, other, ...)
  ## S4 method for signature 'OPMX,MOPMX'
contains(object, other, ...)

  ## S4 method for signature 'MOPMX,ANY'
duplicated(x, incomparables,
    what = c("all", "plate.type", "metadata"), exact = TRUE, strict = FALSE,
    ...)
  ## S4 method for signature 'MOPMX,missing'
duplicated(x, incomparables, ...)
  ## S4 method for signature 'OPM,ANY'
duplicated(x, incomparables, ...)
  ## S4 method for signature 'OPM,missing'
duplicated(x, incomparables, ...)
  ## S4 method for signature 'OPMS,ANY'
duplicated(x, incomparables,
    what = c("all", "csv", "metadata"), exact = TRUE, strict = FALSE, ...)
  ## S4 method for signature 'OPMS,missing'
duplicated(x, incomparables, ...)
```

Arguments

x	OPMX or MOPMX object.
incomparables	Vector passed to duplicated from the **base** package. By default this is FALSE.
what	Indicating which parts of x should be compared. If a character scalar, the following entries are special: **all** Compares entire OPM objects. **csv** Compares the CSV data entries setup_time and position (see csv_data). Not for MOPMX objects. **metadata** Compares the entire metadata content. **plate.type** Compares the plate types (only for MOPMX objects). If what does not match any of these, or is not a character scalar at all, it is passed as key argument to metadata, and the resulting metadata subsets are compared. The following two arguments are only relevant in this case.
exact	Logical scalar passed to metadata.
strict	Logical scalar passed to metadata.
...	Optional arguments passed to duplicated from the **base** package. For contains, optional arguments passed to identical from the **base** package, allowing for fine-control of identity.
object	OPMX object.
other	For the OPMX method of contains, an OPMX object used as query.

Details

The OPMS and OPM methods of contains test, for instance, whether an OPM object is contained in an OPMS object. The length of the resulting logical vector is the length of other.

Value

Logical vector in the case of duplicated, integer scalar in the case of anyDuplicated. 0 if no values are duplicated, the index of the first or last (depending on fromLast) duplicated object otherwise. contains returns a logical vector.

See Also

base::duplicated base::anyDuplicated base::identical

Other getter-functions: aggr_settings, aggregated, csv_data, dim, disc_settings, discretized, has_aggr, has_disc, hours, max, measurements, minmax, seq, subset, thin_out, well

Examples

```
# 'OPM' methods
(x <- duplicated(vaas_1)) # 1 element => nothing duplicated
stopifnot(identical(x, FALSE))

(x <- anyDuplicated(vaas_1))
stopifnot(identical(x, 0L)) # no complete plate duplicated
```

```
(x <- anyDuplicated(vaas_1, what = list("Strain", "Species")))
stopifnot(identical(x, 0L)) # no organisms duplicated

# 'OPMS' methods
stopifnot(!duplicated(vaas_4)) # => no complete plates duplicated
stopifnot(!duplicated(vaas_4, what = list("Species", "Strain")))
# => no organisms duplicated
stopifnot(duplicated(vaas_4, what = "Species") == rep(c(FALSE, TRUE), 2))
# => species duplicated
x <- vaas_4[c(1, 1)] # => complete plate duplicated
stopifnot(c(FALSE, TRUE) == duplicated(x))

stopifnot(identical(anyDuplicated(vaas_4), 0L))
stopifnot(identical(anyDuplicated(vaas_4, what = list("Strain")), 0L))
# => no strains duplicated
stopifnot(identical(anyDuplicated(vaas_4, what = list("Species")), 2L))
# => species duplicated
x <- vaas_4[c(1, 1)] # complete plate duplicated
stopifnot(identical(anyDuplicated(x), 2L))

## contains: 'OPMS'/'OPM' methods
(x <- contains(vaas_4, vaas_4[3])) # single one contained
stopifnot(length(x) == 1, x)
(x <- contains(vaas_4, vaas_4))
stopifnot(length(x) == 4, x) # all contained
(x <- contains(vaas_4[3], vaas_4)) # one of four contained
stopifnot(length(x) == 4, sum(x) == 1)
stopifnot(contains(vaas_4[3], vaas_4[3])) # identical OPM objects
stopifnot(!contains(vaas_4[3], vaas_4[2])) # non-identical OPM objects
```

`explode_dir` *Batch processing of files*

Description

Batch-collect information from a series of input files or batch-convert data from input files to data in output files. Alternatively, turn a mixed file/directory list into a list of files or create a regular expression matching certain file extensions, or convert a wildcard pattern to a regular expression. These functions are not normally directly called by an **opm** user but by the other IO functions of the package such as `collect_template` or `batch_opm`. One can use their `demo` argument directly for testing the results of the applied file name patterns.

Usage

```
explode_dir(names, include = NULL, exclude = NULL,
  ignore.case = TRUE, wildcard = TRUE, recursive = TRUE,
  missing.error = TRUE, remove.dups = TRUE)

batch_collect(names, fun, fun.args = list(), proc = 1L,
```

```
    ..., use.names = TRUE, simplify = FALSE, demo = FALSE)

  batch_process(names, out.ext, io.fun, fun.args = list(),
    proc = 1L, outdir = NULL,
    overwrite = c("yes", "older", "no"), in.ext = "any",
    compressed = TRUE,
    literally = inherits(in.ext, "AsIs"), ...,
    verbose = TRUE, demo = FALSE)

  file_pattern(type = c("both", "csv", "yaml", "json", "yorj", "any", "empty"),
    compressed = TRUE, literally = inherits(type, "AsIs"))

  glob_to_regex(object)

  ## S3 method for class 'character'
 glob_to_regex(object)

  ## S3 method for class 'factor'
 glob_to_regex(object)
```

Arguments

names	Character vector containing file names or directories, or convertible to such.
object	Character vector or factor.
include	If a character scalar, used as regular expression or wildcard (see the wildcard argument) for selecting from the input files. If NULL, ignored. If a list, used as arguments of file_pattern and its result used as regular expression. Note that selection is done **after** expanding the directory names to file names.
exclude	Like include, but for excluding matching input files. Note that exclusion is done **after** applying include.
ignore.case	Logical scalar. Ignore differences between uppercase and lowercase when using include and exclude? Has no effect for NULL values for include or exclude, respectively.
wildcard	Logical scalar. Are include and exclude wildcards (as used by UNIX shells) that first need to be concerted to regular expressions? Has no effect if lists are used for include or exclude, respectively. See below for details on such wildcards (a.k.a. globbing patterns).
recursive	Logical scalar. Traverse directories recursively and also consider all subdirectories? See list.files from the **base** package for details.
missing.error	Logical scalar. If a file/directory does not exist, raise an error or only a warning?
remove.dups	Logical scalar. Remove duplicates from names? Note that if requested this is done **before** expanding the names of directories, if any.
fun	Collecting function. Should use the file name as first argument.
fun.args	Optional list of arguments to fun or io.fun.
...	Optional further arguments passed to explode_dir.

proc	Integer scalar. The number of processes to spawn. Cannot be set to more than 1 core if running under Windows. See the cores argument of do_aggr for details.
simplify	Logical scalar. Should the resulting list be simplified to a vector or matrix if possible?
use.names	Logical scalar. Should names be used for naming the elements of the result?
out.ext	Character scalar. The extension of the output file names (without the dot).
outdir	Character vector. Directories in which to place the output files. If NULL or only containing empty strings, the directory of each input file is used.
in.ext	Character scalar. Passed through file_pattern, then used for the replacement of old file extensions with new ones.
type	Character scalar indicating the file types to be matched by extension. For historical reasons, both means either CSV or YAML *or* JSON, whereas yorj means either YAML or JSON. Alternatively, directly the extension or extensions, or a list of file names (not NA).
compressed	Logical scalar. Shall compressed files also be matched? This affects the returned pattern as well as the pattern used for extracting file extensions from complete file names (if literally is TRUE).
literally	Logical scalar. Interpret type literally? This also allows for vectors with more than a single element, as well as the extraction of file extensions from file names.
demo	Logical scalar. In the case of batch_process, if TRUE do not convert files, but print the attempted input file-output file conversions and invisibly return a matrix with input files in the first and output files in the second column? For the other functions, the effect is equivalent.
io.fun	Conversion function. Should accept infile and outfile as the first two arguments.
overwrite	Character scalar. If 'yes', conversion is always tried if infile exists and is not empty. If 'no', conversion is not tried if outfile exists and is not empty. If 'older', conversion is tried if outfile does not exist or is empty or is older than infile (with respect to the modification time).
verbose	Logical scalar. Print conversion and success/failure information?

Details

Other functions that call explode_dir have a demo argument which, if set to TRUE, caused the respective function to do no real work but print the names of the files that it would process in normal running mode.

glob_to_regex changes a shell globbing wildcard into a regular expression. This is just a slightly extended version of glob2rx from the **utils** package, but more conversion steps might need to be added here in the future. Particularly explode_dir and the IO functions calling that function internally use glob_to_regex. Some hints when using globbing patterns are given in the following.

The here used globbing search patterns contain only two special characters, '?' and '*', and are thus more easy to master than regular expressions. '?' matches a single arbitrary character, whereas '*' matches zero to an arbitrary number of arbitrary characters. Some examples:

a?c Matches abc, axc, a c etc. but not abbc, abbbc, ac etc.

a*c Matches abc, abbc, ac etc. but not abd etc.

ab* Matches abc, abcdefg, abXYZ etc. but not acdefg etc.

?bc Matches abc, Xbc, bc etc. but not aabc, abbc, bc etc.

Despite their simplicity, globbing patterns are often sufficient for selecting file names.

Value

`explode_dir` returns a character vector (which would be empty if all existing files, if any, had been unselected).

`batch_collect` returns a list, potentially simplified to a vector, depending on the output of `fun` and the value of `simplify`. See also `demo`.

In normal mode, `batch_process` creates an invisibly returned character matrix in which each row corresponds to a named character vector with the keys `infile`, `outfile`, `before` and `after`. The latter two describe the result of the action(s) before and after attempting to convert `infile` to `outfile`. The `after` entry is the empty string if no conversion was tried (see `overwrite`), `ok` if conversion was successful and a message describing the problems otherwise. For the results of the `demo` mode see above.

`file_pattern` yields a character scalar, holding a regular expression. `glob_to_regex` yields a vector of regular expressions.

See Also

base::list.files base::Sys.glob utils::glob2rx base::regex

Other io-functions: `batch_opm`, `clean_filenames`, `collect_template`, `read_opm`, `read_single_opm`, `split_files`, `to_metadata`

Examples

```
# explode_dir()
# Example with temporary directory
td <- tempdir()
tf <- tempfile()
(x <- explode_dir(td))
write(letters, tf)
(y <- explode_dir(td))
stopifnot(length(y) > length(x))
unlink(tf)
(y <- explode_dir(td))
stopifnot(length(y) == length(x))

# Example with R installation directory
(x <- explode_dir(R.home(), include = "*/doc/html/*"))
(y <- explode_dir(R.home(), include = "*/doc/html/*", exclude = "*.html"))
stopifnot(length(x) == 0L || length(x) > length(y))

# batch_collect()
# Read the first line from each of the OPM test data set files
f <- opm_files("testdata")
```

```
if (length(f) > 0) { # if the files are found
  x <- batch_collect(f, fun = readLines, fun.args = list(n = 1L))
  # yields a list with the input files as names and the result from each
  # file as values (exactly one line)
  stopifnot(is.list(x), identical(names(x), f))
  stopifnot(sapply(x, is.character), sapply(x, length) == 1L)
} else {
  warning("test files not found")
}
# For serious tasks, consider to first try the function in 'demo' mode.

# batch_process()
# Read the first line from each of the OPM test data set files and store it
# in temporary files
pf <- function(infile, outfile) write(readLines(infile, n = 1), outfile)
infiles <- opm_files("testdata")
if (length(infiles) > 0) { # if the files are found
  x <- batch_process(infiles, out.ext = "tmp", io.fun = pf,
    outdir = tempdir())
  stopifnot(is.matrix(x), identical(x[, 1], infiles))
  stopifnot(file.exists(x[, 2]))
  unlink(x[, 2])
} else {
  warning("test files not found")
}
# For serious tasks, consider to first try the function in 'demo' mode.

# file_pattern()
(x <- file_pattern())
(y <- file_pattern(type = "csv", compressed = FALSE))
stopifnot(nchar(x) > nchar(y))
# constructing pattern from existing files
(files <- list.files(pattern = "[.]"))
(x <- file_pattern(I(files))) # I() causes 'literally' to be TRUE
stopifnot(grepl(x, files, ignore.case = TRUE))

# glob_to_regex()
x <- "*what glob2rx() can't handle because a '+' is included*"
(y <- glob_to_regex(x))
(z <- glob2rx(x))
stopifnot(!identical(y, z))
# Factor method
(z <- glob_to_regex(as.factor(x)))
stopifnot(identical(as.factor(y), z))
```

`extract` *Extract aggregated values and/or metadata*

Description

Extract selected aggregated and/or discretised values into common matrix or data frame. The `extract` data-frame method conducts normalisation and/or computes normalised point-estimates and respective confidence intervals for user-defined experimental groups. It is mainly a helper function for `ci_plot`. `extract_columns` extracts only selected metadata entries for use as additional columns in a data frame or (after joining) as character vector with labels.

Usage

```
  ## S4 method for signature 'MOPMX'
extract(object, as.labels,
    subset = opm_opt("curve.param"), ci = FALSE, trim = "full",
    dataframe = FALSE, as.groups = NULL, ...)
  ## S4 method for signature 'OPMS'
extract(object, as.labels,
    subset = opm_opt("curve.param"), ci = FALSE, trim = "full",
    dataframe = FALSE, as.groups = NULL, sep = " ", dups = "warn",
    exact = TRUE, strict = TRUE, full = TRUE, max = 10000L, ...)
  ## S4 method for signature 'data.frame'
extract(object, as.groups = TRUE,
    norm.per = c("row", "column", "none"), norm.by = TRUE, subtract = TRUE,
    direct = inherits(norm.by, "AsIs"), dups = c("warn", "error", "ignore"),
    split.at = param_names("split.at"))

  ## S4 method for signature 'WMD'
extract_columns(object, what, join = FALSE,
    sep = " ", dups = c("warn", "error", "ignore"), factors = TRUE,
    exact = TRUE, strict = TRUE)
  ## S4 method for signature 'WMDS'
extract_columns(object, what, join = FALSE,
    sep = " ", dups = c("warn", "error", "ignore"), factors = TRUE,
    exact = TRUE, strict = TRUE)
  ## S4 method for signature 'data.frame'
extract_columns(object, what,
    as.labels = NULL, as.groups = NULL, sep = opm_opt("comb.value.join"),
    factors = is.list(what), direct = is.list(what) || inherits(what, "AsIs"))
```

Arguments

`object`	`OPMS` object, `MOPMX` object or data frame, for `extract` with one column named as indicated by `split.at` (default given by `param_names("split.at")`), columns with factor variables before that column and columns with numeric vectors after that column. For `extract_columns` optionally an `OPM` object.
`as.labels`	List, character vector or formula indicating the metadata to be joined and used as row names (if `dataframe` is `FALSE`) or additional columns (if otherwise). Ignored if `NULL`. If a `as.labels` is a formula and `dataframe` is `TRUE`, the pseudo-function `J` within the formula can be used to trigger combination of factors immediately

after selecting them as data-frame columns, much like `as.groups`.

`subset` Character vector. The parameter(s) to put in the matrix. One of the values of `param_names()`. Alternatively, if it is `param_names("disc.name")`, discretised data are returned, and `ci` is ignored..

`ci` Logical scalar. Also return the confidence intervals?

`trim` Character scalar. See `aggregated` for details.

`dataframe` Logical scalar. Return data frame or matrix?

`as.groups` For the `OPMS` method, a list, character vector or formula indicating the metadata to be joined and either used as 'row.groups' attribute of the output matrix or as additional columns of the output data frame. See `heat_map` for its usage. Ignored if empty.

If a `as.groups` is a formula and `dataframe` is `TRUE`, the pseudo-function `J` within the formula can be used to trigger combination of factors immediately after selecting them as data-frame columns, much like `as.labels`.

If `as.groups` is a logical scalar, `TRUE` yields a trivial group that contains all elements, `FALSE` yields one group per element, and `NA` yields an error. The column name in which this factor is placed if `dataframe` is `TRUE` is determined using `opm_opt("group.name")`.

For the data-frame method, a logical, character or numeric vector indicating according to which columns (before the `split.at` column) the data should be aggregated by calculating means and confidence intervals. If `FALSE`, such an aggregation does not take place. If `TRUE`, all those columns are used for grouping.

`sep` Character scalar. Used as separator between the distinct metadata entries if these are to be pasted together. `extract_columns` ignores this unless `join` is `TRUE`. The data-frame method always joins the data unless `what` is a list.

`dups` Character scalar specifying what to do in the case of duplicate labels: either 'warn', 'error' or 'ignore'. Ignored unless `join` is `TRUE` and if `object` is an `OPM` object. For the data-frame method of `extract`, a character scalar defining the action to conduct if `as.groups` contains duplicates.

`exact` Logical scalar. Passed to `metadata`.

`strict` Logical scalar. Also passed to `metadata`.

`full` Logical scalar indicating whether full substrate names shall be used. This is passed to `wells`, but in contrast to what `flatten` is doing the argument here refers to the generation of the column names.

`max` Numeric scalar. Passed to `wells`.

`...` Optional other arguments passed to `wells`.

`norm.per` Character scalar indicating the presence and direction of a normalisation step.

none No normalisation.

row Normalisation per row. By default, this would subtract the mean of each plate from each of its values (over all wells of that plate).

column Normalisation per column. By default, this would subtract the mean of each well from each of its values (over all plates in which this well is present).

This step can further by modified by the next three arguments.

`norm.by` Vector indicating which wells (columns) or plates (rows) are used to calculate means used for the normalisation. By default, the mean is calculated over all rows or columns if normalisation is requested using `norm.per`. But if `direct` is `TRUE`, `norm.by` is directly interpreted as numeric vector used for normalisation.

`direct` Logical scalar. For `extract`, indicating how to use `norm.by`. See there for details. For `extract_columns`, indicating whether to extract column names directly, or search for columns of one to several given classes.

`subtract` Logical scalar indicating whether normalisation (if any) is done by subtracting or dividing.

`split.at` Character vector defining alternative names of the column at which the data frame shall be divided. Exactly one must match.

`what` For the `OPMS` method, a list of metadata keys to consider, or single such key; passed to `metadata`. A formula is also possible; see there for details. A peculiarity of `extract_columns` is that including `J` as a pseudo-function call in the formula triggers the combination of metadata entries to new factors immediately after selecting them, as long as `join` is `FALSE`.

For the data-frame method, just the names of the columns to extract, or their indexes, as vector, if `direct` is `TRUE`. Alternatively, the name of the class to extract from the data frame to form the matrix values.

In the 'direct' mode, `what` can also be a named list of vectors used for indexing. In that case a data frame is returned that contains the columns from `object` together with new columns that result from pasting the selected columns together. If `what` is named, its names are used as the new column names. Otherwise each name is created by joining the respective value within `what` with the `"comb.key.join"` entry of `opm_opt` as separator.

`join` Logical scalar. Join each row together to yield a character vector? Otherwise it is just attempted to construct a data frame.

`factors` Logical scalar determining whether strings should be converted to factors. Note that this would only affect newly created data-frame columns.

Details

`extract_columns` is not normally directly called by an **opm** user because `extract` is available, which uses this function, but can be used for testing the applied metadata selections beforehand.

The `extract_columns` data-frame method is partially trivial (extract the selected columns and join them to form a character vector or new data-frame columns), partially more useful (extract columns with data of a specified class).

Not all `MOPMX` objects are suitable for `extract`. The call will be successful if only `OPMS` objects are contained, i.e. `OPM` objects are forbidden. But even if successful it might result in `NA` values within the resulting matrix or data frame. This may cause methods that call `extract` to fail. `NA` values will not occur if the set of row names created using `as.labels` is equal between the distinct elements of `object`. The also holds if `dataframe` is `TRUE`, even though in that case row names are only temporarily created.

Duplicate combinations of row and columns names currently cause the MOPMX methods to skip all of them except the last one if dataframe is FALSE. This should mainly effect substrates that occur in plates of distinct plate types.

Similarly, duplicate row names will cause the skipping of all but the last one. This can be circumvented by using an as.labels argument that yields unique row names. If as.labels is empty, the MOPMX method of extract will create potentially unique row names from the names if these are present but from the plate types if the 'names' attribute is NULL. This will not be done, and rows will neither be skipped nor reordered, if dataframe is TRUE.

Otherwise row names and names of substrate columns will be reordered (sorted). The created 'row.groups' attribute, if any, will be adapted accordingly. If dataframe is TRUE, the placement of the columns created by as.groups will also be as usual, but duplicates, if any, will be removed.

Value

Numeric matrix or data frame from extract; always a data frame for the data-frame method with the same column structure as object and, if grouping was used, a triplet structure of the rows, as indicated in the new split.at column: (i) group mean, (ii) lower and (iii) upper boundary of the group confidence interval. The data could then be visualised using ci_plot. See the examples.

For the OPMS method of extract_columns, a data frame or character vector, depending on the join argument. The data-frame method of extract_columns returns a character vector or a data frame, too, but depending on the what argument.

Author(s)

Lea A.I. Vaas, Markus Goeker

See Also

aggregated for the extraction of aggregated values from a single OPMA objects.

boot::norm base::data.frame base::as.data.frame base::matrix base::as.matrix base::cbind

Other conversion-functions: as.data.frame, flatten, merge, oapply, opmx, plates, rep, rev, sort, split, to_yaml, unique

Examples

```
## 'OPMS' method
opm_opt("curve.param") # default parameter

# generate matrix (containing the parameter given above)
(x <- extract(vaas_4, as.labels = list("Species", "Strain")))[, 1:3]
stopifnot(is.matrix(x), dim(x) == c(4, 96), is.numeric(x))
# using a formula also works
(y <- extract(vaas_4, as.labels = ~ Species + Strain))[, 1:3]
stopifnot(identical(x, y))

# generate data frame
(x <- extract(vaas_4, as.labels = list("Species", "Strain"),
  dataframe = TRUE))[, 1:3]
stopifnot(is.data.frame(x), dim(x) == c(4, 99))
```

```
# using a formula
(y <- extract(vaas_4, as.labels = ~ Species + Strain,
  dataframe = TRUE))[, 1:3]
stopifnot(identical(x, y))
# using a formula, with joining into new columns
(y <- extract(vaas_4, as.labels = ~ J(Species + Strain),
  dataframe = TRUE))[, 1:3]
stopifnot(identical(x, y[, -3]))

# put all parameters in a single data frame
x <- lapply(param_names(), function(name) extract(vaas_4, subset = name,
  as.labels = list("Species", "Strain"), dataframe = TRUE))
x <- do.call(rbind, x)

# get discretised data
(x <- extract(vaas_4, subset = param_names("disc.name"),
  as.labels = list("Strain")))[, 1:3]
stopifnot(is.matrix(x), identical(dim(x), c(4L, 96L)), is.logical(x))

## data-frame method

# extract data from OPMS-object as primary data frame
# second call to extract() then applied to this one
(x <- extract(vaas_4, as.labels = list("Species", "Strain"),
  dataframe = TRUE))[, 1:3]

# no normalisation, but grouping for 'Species'
y <- extract(x, as.groups = "Species", norm.per = "none")
# plotting using ci_plot()
ci_plot(y[, c(1:6, 12)], legend.field = NULL, x = 350, y = 1)

# normalisation by plate means
y <- extract(x, as.groups = "Species", norm.per = "row")
# plotting using ci_plot()
ci_plot(y[, c(1:6, 12)], legend.field = NULL, x = 130, y = 1)

# normalisation by well means
y <- extract(x, as.groups = "Species", norm.per = "column")
# plotting using ci_plot()
ci_plot(y[, c(1:6, 12)], legend.field = NULL, x = 20, y = 1)

# normalisation by subtraction of the well means of well A10 only
y <- extract(x, as.groups = "Species", norm.per = "row", norm.by = 10,
  subtract = TRUE)
# plotting using ci_plot()
ci_plot(y[, c(1:6, 12)], legend.field = NULL, x = 0, y = 0)

## extract_columns()

# 'OPMS' method

# Create data frame
(x <- extract_columns(vaas_4, what = list("Species", "Strain")))
```

```
stopifnot(is.data.frame(x), dim(x) == c(4, 2))
(y <- extract_columns(vaas_4, what = ~ Species + Strain))
stopifnot(identical(x, y)) # same result using a formula
(y <- extract_columns(vaas_4, what = ~ J(Species + Strain)))
stopifnot(is.data.frame(y), dim(y) == c(4, 3)) # additional column created
stopifnot(identical(x, y[, -3]))
(x <- extract_columns(vaas_4, what = TRUE)) # use logical scalar
stopifnot(is.data.frame(x), dim(x) == c(4, 1))
(y <- extract_columns(vaas_4, what = FALSE))
stopifnot(is.data.frame(y), dim(y) == c(4, 1), !all(y[, 1] == x[, 1]))

# Create a character vector
(x <- extract_columns(vaas_4, what = list("Species", "Strain"), join = TRUE))
stopifnot(is.character(x), length(x) == 4L)
(x <- try(extract_columns(vaas_4, what = list("Species"), join = TRUE,
  dups = "error"), silent = TRUE)) # duplicates yield error
stopifnot(inherits(x, "try-error"))
(x <- try(extract_columns(vaas_4, what = list("Species"), join = TRUE,
  dups = "warn"), silent = TRUE)) # duplicates yield warning only
stopifnot(is.character(x), length(x) == 4L)

# data-frame method, 'direct' running mode
x <- data.frame(a = 1:26, b = letters, c = LETTERS)
(y <- extract_columns(x, I(c("a", "b")), sep = "-"))
stopifnot(grepl("^\\s*\\d+-[a-z]$", y)) # pasted columns 'a' and 'b'

# data-frame method, using class name
(y <- extract_columns(x, as.labels = "b", what = "integer", as.groups = "c"))
stopifnot(is.matrix(y), dim(y) == c(26, 1), rownames(y) == x$b)
stopifnot(identical(attr(y, "row.groups"), x$c))
```

`find_substrate` *Identify substrates or positions*

Description

Identify the names of substrates as used in the stored plate annotations, or identify the positions of substrates, i.e. the plate(s) and well(s) in which they occur. Exact or error-tolerant matching can be used, as well as globbing and regular-expression matching.

Usage

```
  ## S4 method for signature 'MOPMX'
find_positions(object, ...)
  ## S4 method for signature 'OPM'
find_positions(object, type = NULL, ...)
  ## S4 method for signature 'OPMS'
find_positions(object, ...)
  ## S4 method for signature 'character'
```

```
find_positions(object, type = NULL, ...)
  ## S4 method for signature 'factor'
find_positions(object, ...)
  ## S4 method for signature 'list'
find_positions(object, ...)
  ## S4 method for signature 'substrate_match'
find_positions(object, ...)

  ## S4 method for signature 'character'
find_substrate(object,
    search = c("exact", "glob", "approx", "regex", "pmatch"), max.dev = 0.2)
  ## S4 method for signature 'factor'
find_substrate(object, ...)
```

Arguments

`object`	Query character vector or factor, when searching for positions alternatively a list, an S3 object of class 'substrate_match', an `OPM` or an `OPMS` object.
`type`	Ignored if empty or `FALSE`. Otherwise, passed to `plate_type` for normalisation and then used to restrict the positions to those in that plate. Changes the output object to a vector; see below for details. In the case of `OPMX` objects, this can be set to `TRUE`, causing the use of the plate of `object`.
`search`	Character scalar indicating the search mode. **exact** Query names must exactly match (parts of) the well annotations. **glob** Shell globbing is used. See `glob_to_regex` for a description of globbing patterns. **approx** Approximate matching is used; the number or proportion of errors allowed is set using `max.dev`, and neither globbing or regular-expression matching is done in that case. **regex** Regular-expression matching is used. **pmatch** Uses `pmatch` from the **base** package. All matching is case-insensitive except for the `exact` and `pmatch` search modes.
`max.dev`	Numeric scalar indicating the maximum allowed deviation. If < 1, the proportion of characters that might deviate, otherwise their absolute number. It can also be a list; see the 'max.distance' argument of `agrep` in the **base** package for details. Has an effect only if 'approx' is chosen as search mode (see the `search` argument).
`...`	Optional arguments passed between the methods.

Details

When searching for positions, the query names must be written exactly as used in the stored plate annotations. To determine their spelling, use `find_substrate`. This spelling is not guaranteed to be stable between distinct **opm** releases.

Value

find_substrate returns an S3 object of class 'substrate_match'; basically a list of character vectors (empty if nothing was found), with duplicates removed and the rest sorted. The names of the list correspond to names.

The find_positions character method returns a list of character matrices (empty if nothing was found), with one row per position found, the plate name in the first column and the well name in the second. The names of this list correspond to names. The OPM and OPMS methods do the same, using their own substrates. The list and 'substrate_match' methods return lists of such lists.

See Also

base::grep base::agrep

Other naming-functions: gen_iii, listing, opm_files, param_names, plate_type, register_plate, select_colors, substrate_info, wells

Examples

```
## find_substrate()

# Note that 'exact' search matches parts of the names, whereas globbing
# matches entire strings if there are no wildcards (which wouldn't make much
# sense)
(x <- find_substrate("D-Glucose", search = "exact"))
(y <- find_substrate("D-Glucose", search = "glob"))
stopifnot(length(x[[1]]) > length(y[[1]]))

# 'pmatch' matching matches partially at the beginning and returns at most
# one match (the first one)
(y <- find_substrate("D-Glucose", search = "pmatch"))
stopifnot(length(x[[1]]) > length(y[[1]]))

# Now allowing mismatches
(z <- find_substrate("D-Glucose", search = "approx"))
stopifnot(length(z[[1]]) > length(x[[1]]))

# Factor method
(zz <- find_substrate(as.factor("D-Glucose"), search = "approx"))
stopifnot(identical(z, zz))

## find_positions()

# Character method; compare correct and misspelled substrate name
(x <- find_positions(c("D-Glucose", "D-Gloucose")))
stopifnot(length(x[[1]]) > length(x[[2]]))

# Factor method
(y <- find_positions(as.factor(c("D-Glucose", "D-Gloucose"))))
stopifnot(identical(y, x))

# Restrict to  a certain plate
(x <- find_positions(c("D-Glucose", "D-Gloucose"), type = "Gen III"))
```

```
stopifnot(is.character(x), any(is.na(x)), !all(is.na(x)))

# List method
x <- find_positions(find_substrate(c("D-Glucose", "D-Gloucose")))
x[[1]][1:3]
x[[2]]
stopifnot(length(x[[1]]) > length(x[[2]]))
```

---

has_aggr *Are aggregated or discretised data present?*

Description

Check whether aggregated or discretised data are present. (See do_aggr and do_disc for generating such data.) This always returns FALSE for the OPM class, but not necessarily for its child classes.

Usage

```
  ## S4 method for signature 'MOPMX'
has_aggr(object, ...)
  ## S4 method for signature 'OPM'
has_aggr(object)
  ## S4 method for signature 'OPMS'
has_aggr(object, ...)

  ## S4 method for signature 'MOPMX'
has_disc(object, ...)
  ## S4 method for signature 'OPM'
has_disc(object)
  ## S4 method for signature 'OPMS'
has_disc(object, ...)
```

Arguments

object	OPM, OPMS or MOPMX object.
...	Optional arguments passed between the methods.

Value

Logical vector, one element per plate.

See Also

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, dim, disc_settings, discretized, duplicated, hours, max, measurements, minmax, seq, subset, thin_out, well

Examples

```
stopifnot(has_aggr(vaas_1), has_disc(vaas_1)) # OPM methods
stopifnot(has_aggr(vaas_4), has_disc(vaas_4)) # OPMS methods
```

---

heat_map *Heat map*

Description

A wrapper for `heatmap` from the **stats** package and `heatmap.2` from the **gplots** package with some adaptations likely to be useful for OmniLog® data. The data-frame and `OPMS` methods extract a numeric matrix from a given data frame or `OPMS` object and pass the result to the matrix method.

Usage

```
  ## S4 method for signature 'MOPMX'
heat_map(object, as.labels,
    subset = opm_opt("curve.param"), as.groups = NULL, sep = " ",
    extract.args = list(), ...)
  ## S4 method for signature 'OPMS'
heat_map(object, as.labels,
    subset = opm_opt("curve.param"), as.groups = NULL, sep = " ",
    extract.args = list(), ...)
  ## S4 method for signature 'data.frame'
heat_map(object, as.labels,
    as.groups = NULL, sep = " ", ...)
  ## S4 method for signature 'matrix'
heat_map(object,
    hclustfun = "ward", distfun = "euclidean", scale = "none",
    r.groups = "row.groups", r.col = opm_opt("colors"),
    c.groups = "col.groups", c.col = opm_opt("colors"),
    magnif = 4, cexRow = magnif[1L] / sqrt(nrow(object)),
    cexCol = magnif[length(magnif)] / sqrt(ncol(object)),
    borders = c(0.55, 0.75),
    margins = if (use.fun[1L] == "gplots")
      c(borders[1L] * cexCol * max(nchar(colnames(object))),
      borders[length(borders)] * cexRow * max(nchar(rownames(object))))
    else
      c(5, 5),
    col = opm_opt("heatmap.colors"), asqr = FALSE, log1 = FALSE, lmap = 1L:3L,
    abbrev = c("none", "row", "column", "both"),
    ...,
    use.fun = c("gplots", "stats"))
```

Arguments

object	Matrix, data frame or OPMS or MOPMX object. The matrix method is mainly designed for curve-parameter matrices as created by extract but can be used with any numeric matrix. If a data frame, it must contain at least one column with numeric data. Not all MOPMX objects are suitable for this function; see the remarks under extract.
as.labels	Character, numeric or logical vector indicating the positions of the columns to be joined and used as row labels. If NULL or empty, the row names of object are used. See extract for details.
as.groups	Character, numeric or logical vector indicating the positions of the columns to be joined and used as group indicators. If NULL or empty, groups are ignored.
sep	Character scalar determining how to join row and group names. See extract for details.
subset	Character scalar passed to the OPMS method of extract.
extract.args	Optional list of arguments passed to that method.
hclustfun	Determines the clustering method used. If a function, used directly. If a character scalar, used as the 'method' argument of hclust. If a list, passed as argument list to hclust.
distfun	Determines the distance method used. If a function, used directly. If a character scalar, used as the 'method' argument of dist. If a list, passed as argument list to dist.
scale	Character scalar. See heatmap for details. The default was changed to no rescaling because the curve parameters estimated from OmniLog® data have the same scale. If the relative changes per substrate are of interest, 'column' should be used.
r.groups	Determines the plotting of a colour bar indicating row groups. If NULL, ignored. If a function, applied to the row names of object; should then yield one group name for each row name. If a character scalar, the name of an attribute of object that contains the row group affiliations (ignored if this is not found). Otherwise, coerced to 'character' mode. Finally the groups are converted to a factor and used for selecting from r.col.
r.col	Character vector of colour names used by r.groups. Ignored if that is NULL.
c.groups	Determines the plotting of a colour bar indicating column groups. If NULL, ignored. If a function, applied to the column names of object; should then yield one group name for each column name. If a character scalar, the name of an attribute of object that contains the column group affiliations (ignored if this is not found). Otherwise, coerced to 'character' mode. Finally the groups are converted to a factor and used for selecting from c.col.
c.col	Character vector of colour names used by c.groups. Ignored if that is NULL.
magnif	Numeric vector. Factor(s) used per default by cexRow and cexCol.
cexRow	Magnification of the row labels.
cexCol	Magnification of the column labels.
borders	Numeric vector. Factor(s) used per default by margin and cexCol.

margins	Two-element numeric vector determining the relative size of the margin (i) at the bottom and (ii) at the left.
col	Character vector containing the proper heat map colours.
asqr	Logical scalar indicating whether the data should be treated with the arcsine-square root transformation. This usually only makes sense for proportion data and cannot be used in conjunction with the log1 argument set to TRUE. If NA, percentages are assumed.
log1	Logical scalar indicating whether log1p should be used for transforming the data prior to plotting.
lmap	Numeric scalar with at least three elements, or empty. If empty, ignored. Otherwise used for mapping logical values to numeric values. See map_values for details. Ignored if the data are not logical.
abbrev	Character scalar indicating whether row or column shall be abbreviated before plotting. Note that abbreviation is done by shortening words and ending them with a dot, so there is no guarantee that a certain maximum length will be obtained.
...	Optional arguments passed to heatmap or heatmap.2. Note that some defaults of heatmap.2 are overwritten even though this is not transparent from the argument list of heat_map. If set explicitly, the default heatmap.2 behaviour is restored. ... also represents all arguments passed from the OPMS or data-frame methods to the matrix method.
use.fun	Character scalar. If gplots, it is attempted to load the **gplots** package and use its heatmap.2 function (the default). If this fails, a warning is issued, and heatmap from the **stats** package (the default) is called instead.

Value

A list as output by heatmap or heatmap.2 with the additional entries rowColMap or colColMap giving the mapping(s) of group names to colours as named character vector(s), if this feature was used.

See Also

stats::heatmap gplots::heatmap.2

Other plotting-functions: ci_plot, level_plot, parallelplot, radial_plot, summary, xy_plot

Examples

```
# Matrix method (usually unnecessary, see below)
x <- extract(vaas_4, as.labels = list("Strain"),
  as.groups = list("Species"))
hm <- heat_map(x)
stopifnot(identical(metadata(vaas_4, "Species"), names(hm$rowColMap)))

# 'OPMS' method (more convenient)
hm.2 <- heat_map(vaas_4, as.labels = "Strain", as.groups = "Species")
stopifnot(identical(hm[-3], hm.2[-3]))
```

```
# Data-frame method
x <- extract(vaas_4, as.labels = list("Species", "Strain"), dataframe = TRUE)
hm <- heat_map(x, as.labels = "Strain", as.groups = "Species")
stopifnot(identical(metadata(vaas_4, "Species"), names(hm$rowColMap)))
```

`html_args` HTML *formatting and output label generation*

Description

These are helper functions for `phylo_data` allowing for either the easy fine-tuning of the generated HTML output or for the conversions of strings to safe phylogenetic taxon labels.

Usage

```
html_args(character.states = c(`negative reaction` = "-", `weak reaction` = "w",
  `positive reaction` = "+"),
  multiple.sep = "/", organisms.start = "Organisms: ",
  states.start = "Symbols: ", legend.dot = TRUE,
  legend.sep.1 = ", ", legend.sep.2 = "; ",
  table.summary = "character matrix", no.html = TRUE,
  greek.letters = TRUE, css.file = opm_opt("css.file"),
  embed.css = FALSE, ...)

safe_labels(x, format, enclose = TRUE, pad = FALSE,
  comment = FALSE)
```

Arguments

`character.states`
: Character vector used for mapping integers to the elements in the corresponding position. It is also used in conjunction with its names to create the table legend. The default value is useful for data of mode 'logical', mapping `FALSE`, `NA` and `TRUE`, in this order. Data of this kind are by default internally converted to an according integer vector.

`multiple.sep`
: Character scalar used for joining multiple-state characters together.

`organisms.start`
: Character scalar prepended to the organism part of the table legend. Ignored if empty.

`states.start`
: Character scalar prepended to the character-states part of the table legend. Ignored if empty.

`legend.dot`
: Logical scalar indicating whether or not a dot shall be appended to the table-legend entries.

`legend.sep.1`
: Character scalar used for the first pass of joining the table-legend entries together.

`legend.sep.2` Character scalar used for the second pass of joining the table-legend entries together.

`table.summary` Character scalar inserted as 'summary' attribute of the resulting HTML table.

`no.html` Logical scalar indicating whether substrate names should be cleaned from characters that might interfere with HTML code. Setting this to `FALSE` might yield invalid HTML.

`greek.letters` Logical scalar indicating whether or not letters between 'a' and 'e' within substrate names should be converted to the corresponding Greek letters. This is done after the cleaning step, if any.

`css.file` Character vector indicating the name of one to several CSS files to link or embed. Empty strings and empty vectors are ignored. If `embed.css` is `FALSE` it is no error if the file does not exist, but the page will then probably not be displayed as intended.

Under Windows it is recommended to convert a file name `f` using `normalizePath(f, winslash = "/")` before linking it.

`embed.css` Logical scalar indicating whether or not CSS files shall be embedded, not linked.

`...` Optional other arguments available for inserting user-defined HTML content. Currently the following ones (in their order of insertion) are not ignored, and can even be provided several times:

meta Used as (additional) 'meta' entries within the HTML head.

headline Override the use of the `title` argument as headline (placed above the table legend). An empty argument would turn it off.

prepend List or character vector to be inserted before the table legend. Lists are converted recursively. List names will be converted to 'title' and 'class' attributes (if missing, names are inferred from the nesting level; see `opm_opt`, entry `html.class`). Names of other vectors, if any, are converted to 'title' and 'span' attributes. Character vectors are converted using `safe_labels` unless they inherit from `AsIs` (see `I` from the **base** package).

insert As above, but inserted between the legend and the table.

append As above, but inserted after the table.

`x` Character vector or convertible to such.

`format` Character scalar. See `phylo_data`.

`enclose` Logical scalar. See `phylo_data` and the description of `comment`.

`pad` Logical scalar. Bring labels to the same number of characters by appending spaces? Has no effect for PHYLIP and HTML output format.

`comment` Logical scalar. If `TRUE`, comments as used in the respective format will be produced. PHYLIP and EPF do not accept comments and will yield an error. If `enclose` is `TRUE`, the comment-enclosing characters are appended and prepended to the vector, otherwise to each string separately.

Details

These functions are not normally called directly by an **opm** user but by `phylo_data`; see there for their usual application. The `phylo_data` methods for `OPMD_Listing` and `OPMS_Listing` objects do not support all HTML formatting options.

Label cleaning invokes either the replacement of disallowed characters or the enclosing of all labels in single quotes and the doubling of already existing single quotes, if any.

Value

List of HTML arguments or character vector with modified labels.

See Also

base::normalizePath base::I base::gsub

Other phylogeny-functions: phylo_data

Examples

```
# Some animals you might know
x <- c("Elephas maximus", "Loxodonta africana", "Giraffa camelopardalis")

(y <- safe_labels(x, "phylip"))
stopifnot(nchar(y) == 10L) # truncation

(y <- safe_labels(x, "epf"))
stopifnot(nchar(y) == nchar(x)) # changes in length unnecessary
(y <- safe_labels(x, "epf", pad = TRUE))
stopifnot(nchar(y) == 22) # padded to uniform length

(y <- safe_labels(x, "nexus", enclose = TRUE))
stopifnot(grepl("^'.*'$", y)) # all strings enclosed in sinqle quotes
```

---

include_metadata *Add or map metadata or edit them by hand*

Description

Either include metadata by mapping CSV data and column names in a data frame (optionally read from file), or modify meta-information stored together with the measurements by using a function or other kinds of mappings and return the objects otherwise unchanged, or invoke `edit` from the **utils** package for editing the metadata by hand.

Usage

```
  ## S4 method for signature 'MOPMX'
edit(name, ...)
  ## S4 method for signature 'WMDX'
edit(name, ...)

  ## S4 method for signature 'MOPMX'
include_metadata(object, ...)
  ## S4 method for signature 'OPM'
```

```
include_metadata(object, md,
    keys = opm_opt("csv.keys"), ...)
  ## S4 method for signature 'WMD'
include_metadata(object, md, keys, replace = FALSE,
    skip.failure = FALSE, remove.keys = TRUE, ...)
  ## S4 method for signature 'WMDS'
include_metadata(object, ...)

  ## S4 method for signature 'MOPMX,ANY'
map_metadata(object, mapping, ...)
  ## S4 method for signature 'MOPMX,missing'
map_metadata(object, mapping,
    values = TRUE, classes = "factor")
  ## S4 method for signature 'WMD,FOE'
map_metadata(object, mapping,
    values = parent.frame(), classes = NULL)
  ## S4 method for signature 'WMD,character'
map_metadata(object, mapping,
    values = TRUE, classes = "factor")
  ## S4 method for signature 'WMD,function'
map_metadata(object, mapping,
    values = TRUE, classes = "ANY", ...)
  ## S4 method for signature 'WMD,missing'
map_metadata(object, mapping,
    values = TRUE, classes = "factor")
  ## S4 method for signature 'WMDS,ANY'
map_metadata(object, mapping, ...)
  ## S4 method for signature 'WMDS,missing'
map_metadata(object, mapping,
    values = TRUE, classes = "factor")

  ## S4 method for signature 'list,formula'
map_values(object, mapping,
    coerce = parent.frame())
```

Arguments

object	OPM (WMD), OPMS (WMDS) or MOPMX object. For map_values, a list.
name	Like object, but for the edit method.
md	Data frame containing keys as column names, or name of file from which to read the data frame. Handled by to_metadata.
keys	Character vector.
replace	Logical scalar indicating whether the previous metadata, if any, shall be replaced by the novel ones, or whether these shall be appended.
skip.failure	Logical scalar. Do not stop with an error message if (unambiguous) selection is impossible but raise a warning only?
remove.keys	Logical scalar. When including md in the metadata, discard the keys columns?

`mapping` In most cases passed to `map_values`.

- If a function, this is just a wrapper for `rapply`, with `how` set to 'replace', if `values` is `TRUE`. It is applied to all non-list elements of `metadata`, which is traversed recursively.
- Alternatively, a character vector. `metadata_chars` can be used to create a template for such a vector.
- `mapping` can also be a formula; in that case, `metadata` is replaced by the according method of `map_values`. If the left side of the formula is missing, the entire metadata are replaced by the result, which is an error if the result is not a list.
- If `mapping` is missing, the behaviour is special; see the next two arguments.

The **opm** package augments `map_values` with a method for lists and formulae. For all other methods, see the **pkgutils** package.

`values` Mostly a logical scalar.

- For the function and character-vector methods, if `FALSE`, metadata names, not values, are mapped, and `classes` is ignored (names are always of class 'character').
- For the formula method, `values` is the enclosing environment used.
- If `mapping` is missing, setting `values` to `TRUE` causes all non-list entries that only comprise `NA` values to be removed.

`classes` Character vector or (for the character vector-based mapping) `TRUE`. For the mapping with a function or vector, this specifies the classes in addition to 'character' that are mapped (after converting to 'character' mode).

If `classes` is `TRUE`, `mapping` is treated as a mapping between class names, and the according conversions are applied. See the `coerce` argument of `map_values` for details.

If `mapping` is missing, `classes` specifies classes that are converted to character vectors.

`coerce` Character vector or `TRUE`. See the description of `map_values` in the **pkgutils** package for details.

`...` Optional arguments passed to `mapping` if it is a function, and from the `WMDS` method to the `WMD` method, or from `include_metadata` to `to_metadata`, or as additional arguments to `edit` from the **utils** package.

Details

The `WMDS` method applies the inclusion and mapping routines to all plates in turn and returns an `WMDS` object with accordingly modified metadata.

Calling `edit` will only work if `to_metadata` yields a data frame suitable for the `edit` method from the **utils** package. This usually means that the `metadata` must be rectangular, even though this is not enforced by the implementation of the `OPMX` classes. Entries missing in some elements of `name` should not present a problem, however. Values that remained `NA` would be removed before returning the result. The `MOPMX` method works by calling each element in turn (allowing for independent editing).

Value

Novel WMD or WMDS object with modified metadata.

See Also

utils::edit

Other metadata-functions: metadata, metadata_chars

Examples

```
## include_metadata()

(x <- collect_template(vaas_1, add.cols = "Location")) # generate data frame
x[1, "Location"] <- "Braunschweig" # insert additional information
copy <- include_metadata(vaas_1, x) # include the data in new OPM object
stopifnot(is.null(metadata(vaas_1, "Location")))
stopifnot(identical(metadata(copy, "Location"), "Braunschweig"))

## map_metadata()

# WMD methods

# WMD+function method
copy <- map_metadata(vaas_1, identity)
stopifnot(identical(copy, vaas_1))
copy <- map_metadata(vaas_1, identity, values = FALSE)
stopifnot(identical(copy, vaas_1))
copy <- map_metadata(vaas_1, function(x) paste(x, "!"), values = FALSE)
(x <- metadata_chars(vaas_1, values = FALSE))
(y <- metadata_chars(copy, values = FALSE))
stopifnot(identical(as.character(y), paste(x, "!")))

# WMD+character method: mapping a value
map <- metadata_chars(vaas_1)
map["First replicate"] <- "Rep. 1"
copy <- map_metadata(vaas_1, map)
stopifnot(identical(names(metadata(copy)), names(metadata(vaas_1))))
stopifnot(!identical(metadata(copy, "Experiment"),
  metadata(vaas_1, "Experiment")))

# WMD+character method: mapping a name
map <- metadata_chars(vaas_1, values = FALSE)
map["Plate number"] <- "Plate no."
copy <- map_metadata(vaas_1, map, values = FALSE)
stopifnot(!identical(names(metadata(copy)), names(metadata(vaas_1))))

# WMD+formula method
copy <- map_metadata(vaas_1, Organism ~ paste(Species, Strain))
(x <- setdiff(metadata_chars(copy), metadata_chars(vaas_1)))
stopifnot(length(x) == 1, x == "Escherichia coli DSM30083T")
stopifnot(identical(copy, # same result with expression
  map_metadata(vaas_1, expression(Organism <- paste(Species, Strain)))))
```

```
# WMD+missing method
(x <- metadata(map_metadata(vaas_1)))
stopifnot(identical(x, metadata(vaas_1))) # nothing to modify in that case

# WMDS method

# using a function
copy <- map_metadata(vaas_4, identity)
stopifnot(identical(copy, vaas_4))
copy <- map_metadata(vaas_4, identity, values = FALSE)
stopifnot(identical(copy, vaas_4))
copy <- map_metadata(vaas_4, function(x) paste(x, "!"), values = FALSE)
(x <- metadata_chars(vaas_4, values = FALSE))
(y <- metadata_chars(copy, values = FALSE))
stopifnot(identical(as.character(y), paste(x, "!")))

# using a character vector
map <- metadata_chars(vaas_4)
map["First replicate"] <- "Rep. 1"
copy <- map_metadata(vaas_4, map)
x <- metadata(vaas_4, "Experiment")
stopifnot(x == "First replicate")
y <- metadata(copy, "Experiment")
stopifnot(y == "Rep. 1")

# using a formula
copy <- map_metadata(vaas_4, Organism ~ paste(Species, Strain))
(x <- setdiff(metadata_chars(copy), metadata_chars(vaas_4)))
stopifnot(length(x) == 4) # one entry per plate

# 'mapping' missing
(x <- metadata(map_metadata(vaas_4)))
stopifnot(identical(x, metadata(vaas_4))) # nothing to modify in that case

## Not run:  ## edit metadata by hand
  x <- edit(vaas_4) # this would create a new object
  x <- edit(x) # overwrite x in 2nd editing step
  ## This will not necessarily work if the metadata are nested!

## End(Not run)

## List/formula method of map_values()
x <- list(a = 1:8, c = 9, d = 'x')
(y <- map_values(x, ~ a + c))
stopifnot(is.numeric(y), y == c(10:17))
(y <- map_values(x, b ~ a + c))
stopifnot(is.list(y), y$b == c(10:17))

# ...applied to a data frame
x <- data.frame(a = 1:5, b = 6:10)
(y <- map_values(x, c ~ a + b))
stopifnot(is.data.frame(y), dim(y) == c(5, 3))
```

```
(z <- map_values(x, ~ a + b))
stopifnot(identical(z, y$c))
# same effect with an expression
(z <- map_values(x, expression(c <- a + b)))
stopifnot(identical(z, y))
```

`level_plot` *Level plot*

Description

Level plot for OPM and OPMS objects using the function from the **lattice** package.

Usage

```
  ## S4 method for signature 'OPM'
level_plot(x, main = list(),
    colors = opm_opt("color.borders"), panel.headers = FALSE, cex = NULL,
    strip.fmt = list(), striptext.fmt = list(), legend.sep = " ",
    space = "Lab", bias = 0.5, num.colors = 200L, ...)
  ## S4 method for signature 'OPMS'
level_plot(x, main = list(),
    colors = opm_opt("color.borders"), panel.headers = TRUE, cex = NULL,
    strip.fmt = list(), striptext.fmt = list(), legend.sep = " ",
    space = "Lab", bias = 0.5, num.colors = 200L, ...)
```

Arguments

x	OPM or OPMS object.
main	The settings controlling the construction of the main title. Works like the main argument of xy_plot.
colors	Character vector indicating the colours (at least two).
panel.headers	NULL, logical scalar, expression or character vector. NULL and FALSE turn panel headers off. TRUE causes the panel headers to be constructed from the plate numbers or those metadata that were included by flatten (see there). Character vectors and expressions are directly used for the text within these panel headers. Currently ignored by the OPM method.
cex	Numeric scalar. Magnification of axis annotation. If NULL, automatically adapted to the number of wells (at least a good guess is made).
strip.fmt	List controlling the format of the description strip above each panel. For instance, the background colour is set using the bg key. For further details, see strip.custom from the **lattice** package. strip.fmt is ignored if panel.headers is FALSE and currently always ignored by the OPM method.
striptext.fmt	List controlling the format of the text within the strip above each panel. See xy_plot for details, which has an argument of the same name.

legend.sep	Character scalar. This works like the eponymous argument to flatten (see there); it is ignored unless metadata are chosen for constructing the panel headers.
space	Character scalar passed to colorRampPalette from the **grDevices** package. These and the following arguments are for fine-tuning the colour palette used for plotting.
bias	Numeric scalar also passed to colorRampPalette.
num.colors	Numeric scalar passed to the function returned by colorRampPalette.
...	Arguments that are passed to flatten.

Value

An object of class 'trellis'. See levelplot from the **lattice** package for details.

References

Jacobsen, J. S., Joyner, D. C., Borglin, S. E., Hazen, T. C., Arkin, A. P. et al. 2007 Visualization of growth curve data from phenotype microarray experiments. *11th International Conference on Information Visualization (IV07).* Zuerich, Switzerland, July 4-6 2007. Published by the IEEE Computer Society.

Sarkar, D. 2008 *Lattice: Multivariate Data Visualization with R.* New York: Springer, 265 p.

Vaas, L. A. I., Sikorski, J., Michael, V., Goeker, M., Klenk H.-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846.

See Also

lattice::levelplot grDevices::colorRampPalette

Other plotting-functions: ci_plot, heat_map, parallelplot, radial_plot, summary, xy_plot

Examples

```
# OPM method
level_plot(vaas_1, main = "Levelplot example")

# OPMS method
## Not run:
# headers include species and strain
level_plot(vaas_4, include = c("Species", "Strain"))

## End(Not run)
```

max *Overall or minimal maximum*

Description

Get the maximum of all wells or (a) specified one(s), or their smallest maximum. The OPMS method works by calling the OPM method on all plates and then determining the overall maximum or overall minimum of the maxima.

Usage

```
  ## S4 method for signature 'OPM'
max(x, ..., na.rm = FALSE)
  ## S4 method for signature 'OPMS'
max(x, ..., na.rm = FALSE)

  ## S4 method for signature 'OPM'
minmax(x, ..., na.rm = FALSE)
  ## S4 method for signature 'OPMS'
minmax(x, ..., na.rm = FALSE)
```

Arguments

x	OPM or OPMS object.
...	Coordinate of one to several wells. If missing, the maximum or smallest of all wells is returned. See well for details. If only as single well is selected, the result of minmax is actually identical to the one of max.
na.rm	Logical scalar. See max from the **base** package. Has no effect here because NA values are not allowed within the measurements.

Value

Numeric scalar.

See Also

base::max base::min

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, dim, disc_settings, discretized, duplicated, has_aggr, has_disc, hours, measurements, seq, subset, thin_out, well

Examples

```
# OPM method
(x <- max(vaas_1))
(y <- max(vaas_1, 1)) # this is the negative control
stopifnot(x > y) # i.e., some stronger reactions present
```

```
(x <- minmax(vaas_1))
stopifnot(max(vaas_1) > x) # obviously

# OPMS method
(x <- max(vaas_4))
(y <- max(vaas_4, 1)) # this is the negative control
stopifnot(x > y) # i.e., some stronger reactions present
(x <- minmax(vaas_4))
stopifnot(max(vaas_4) > x) # obviously
```

`measurements` *Stored measurements*

Description

Return the measurements, optionally only from selected wells and with or without the time points, or only the time points.

Usage

```
  ## S4 method for signature 'MOPMX'
hours(object, ...)
  ## S4 method for signature 'OPM'
hours(object,
    what = c("max", "all", "size", "summary", "interval", "minmax"))
  ## S4 method for signature 'OPMS'
hours(object, ...)

  ## S4 method for signature 'MOPMX'
measurements(object, ...)
  ## S4 method for signature 'OPM'
measurements(object, i)
  ## S4 method for signature 'OPMS'
measurements(object, ...)

  ## S4 method for signature 'MOPMX'
well(object, ...)
  ## S4 method for signature 'OPM'
well(object, i, drop = TRUE, use.names = TRUE)
  ## S4 method for signature 'OPMS'
well(object, ...)
```

Arguments

`object`	OPM, OPMS or MOPMX object.
`i`	Optional character or numeric vector with name(s) or position(s) of well(s). Wells are originally named 'A01' to 'H12' but might have been subset beforehand. `i` can also be a formula, allowing for sequences of well coordinates. See the examples.

drop	Logical scalar. If only a single well was selected, simplify it to a vector?
use.names	Logical scalar indicating whether the time points should be used as names for the measurements.
what	Character scalar determining the output mode as follows:
	all Numeric vector: all time points, in order.
	interval The difference between each pair of adjacent time points, NA if this is irregular or only one time point is left.
	max Numeric scalar: the largest time point.
	minmax Numeric scalar: the smallest maximum. For OPM objects this is apparently identical to 'max'.
	size Integer scalar: the number of time points.
	summary Display a summary.
...	Optional arguments passed between the methods.

Value

measurements returns a numeric matrix with column names indicating the well coordinate and a first column containing the time points. The other columns contain the values from each well. There is one row per time point. Column names are appropriately set, but not translated (as, e.g., to substrate names). It is possible to select wells, but the time points are always included as first column (in contrast to well). The i argument refers only to the remaining matrix.

Do not confuse well with wells. well yields a numeric matrix or vector, depending on i and drop. It will always ignore the time points as values, in contrast to measurements. But depending on use.names they would be inserted as names.

The return value of hours is dependent on the what argument; see there.

See Also

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, dim, disc_settings, discretized, duplicated, has_aggr, has_disc, max, minmax, seq, subset, thin_out

Examples

```
# 'OPM' methods

head(x <- measurements(vaas_1))[, 1:5] # => numeric matrix
stopifnot(is.matrix(x), is.numeric(x))
stopifnot(dim(x) == c(384, 97))
head(x <- measurements(vaas_1, "B03"))
stopifnot(is.matrix(x), is.numeric(x), dim(x) == c(384, 2))
head(y <- measurements(vaas_1, ~ B03)) # => same result with formula
stopifnot(identical(y, x))

head(x <- well(vaas_1, "B04")) # => numeric vector
stopifnot(is.numeric(x), length(x) == 384)
head(x <- well(vaas_1, c("B08", "C07"))) # => numeric matrix
stopifnot(is.matrix(x), dim(x) == c(384, 2))
```

```
# selecting adjacent wells is easer when using a formula
head(x <- well(vaas_1, c("B12", "C01", "C02")))
stopifnot(is.matrix(x), dim(x) == c(384, 3))
head(y <- well(vaas_1, ~ B12:C02)) # => same result
stopifnot(identical(x, y))

(x <- hours(vaas_1)) # the default is 'max'
stopifnot(identical(x, 95.75))
(x <- hours(vaas_1, "minmax"))
stopifnot(identical(x, 95.75))
(x <- hours(vaas_1, "summary"))
stopifnot(is.table(x))
(x <- hours(vaas_1, "interval"))
stopifnot(identical(x, 0.25))
(x <- hours(vaas_1, "size"))
stopifnot(identical(x, 384L))

# 'OPMS' methods

summary(x <- measurements(vaas_4)) # => list of numeric matrices
stopifnot(is.list(x), length(x) == length(vaas_4))
stopifnot(sapply(x, is.matrix), sapply(x, is.numeric))

head(x <- well(vaas_4, "B04"))[, 1:5] # => numeric matrix
stopifnot(is.matrix(x), dim(x) == c(4, 384))
head(y <- well(vaas_4, ~ B04))[, 1:5] # using a formula
stopifnot(identical(x, y)) # => same result

(x <- hours(vaas_4)) # all with the same overall running time
stopifnot(length(x) == 4, x == 95.75)
```

`merge` *Merge or split plates*

Description

Combine all plates in a single OPM object by treating them as originating from subsequent runs of the same experimental plate. Adjust the times accordingly. Alternatively, split plates according to the contained regular series of substrates, if any. The MOPMX method merges according to plate types, optionally including a novel element.

Usage

```
  ## S4 method for signature 'CMAT,ANY'
merge(x, y)
  ## S4 method for signature 'CMAT,factor'
merge(x, y)
  ## S4 method for signature 'CMAT,logical'
merge(x, y)
```

```
  ## S4 method for signature 'MOPMX,ANY'
merge(x, y)
  ## S4 method for signature 'MOPMX,missing'
merge(x, y)
  ## S4 method for signature 'OPM,OPM'
merge(x, y, sort.first = TRUE,
    parse = TRUE)
  ## S4 method for signature 'OPM,missing'
merge(x, y, sort.first = TRUE,
    parse = TRUE)
  ## S4 method for signature 'OPM,numeric'
merge(x, y, sort.first = TRUE,
    parse = TRUE)
  ## S4 method for signature 'OPMS,missing'
merge(x, y, sort.first = TRUE,
    parse = TRUE)
  ## S4 method for signature 'OPMS,numeric'
merge(x, y, sort.first = TRUE,
    parse = TRUE)

  ## S4 method for signature 'MOPMX,ANY,ANY'
split(x, f, drop)
  ## S4 method for signature 'MOPMX,ANY,missing'
split(x, f, drop)
  ## S4 method for signature 'MOPMX,factor,ANY'
split(x, f, drop)
  ## S4 method for signature 'MOPMX,factor,missing'
split(x, f, drop)
  ## S4 method for signature 'MOPMX,list,ANY'
split(x, f, drop)
  ## S4 method for signature 'MOPMX,list,missing'
split(x, f, drop)
  ## S4 method for signature 'OPM,ANY,missing'
split(x, f, drop)
  ## S4 method for signature 'OPM,factor,ANY'
split(x, f, drop)
  ## S4 method for signature 'OPM,factor,missing'
split(x, f, drop)
  ## S4 method for signature 'OPM,missing,ANY'
split(x, f, drop)
  ## S4 method for signature 'OPM,missing,missing'
split(x, f, drop)
  ## S4 method for signature 'OPMS,ANY,missing'
split(x, f, drop)
  ## S4 method for signature 'OPMS,factor,ANY'
split(x, f, drop)
  ## S4 method for signature 'OPMS,factor,missing'
split(x, f, drop)
```

```
  ## S4 method for signature 'OPMS,missing,ANY'
split(x, f, drop)
  ## S4 method for signature 'OPMS,missing,missing'
split(x, f, drop)
  ## S4 method for signature 'OPMX,ANY,ANY'
split(x, f, drop)
```

Arguments

x	OPMX or MOPMX object.
y	For the OPMS method a numeric vector indicating the time(s) (in hours) between two subsequent plates. Must be positive throughout, and its length should fit to the number of plates (e.g., either 1 or `length(x) - 1` would work). If missing, `0.25` is used. If x is an OPM object, a missing or numeric y argument causes merge to just return x because there is nothing to merge. But y can be an OPM object in that case, which, if compatible, will be merged with x. For the MOPMX method, the optional y can be any object that can be convert to the class of x using as.
sort.first	Logical scalar. Sort the plates according to their setup times before merging?
parse	Logical scalar. Ignored unless sort.first is TRUE. For sorting, parse the setup times using strptime from the **base** package? It is an error if this does not work, but see 'Details'.
f	For the OPMX methods, a factor or missing. If missing, the behaviour is special. Splitting is applied to the plates themselves and attempted according to the positions of substrates within series as revealed by substrate_info in 'concentration' mode. If a factor, f is used as in the default split method from the **base** package, yielding a list (MOPMX object) of single or multiple plates. If neither missing nor a factor, f is used as key argument of metadata. The resulting entries are pasted together per plate and converted to a factor used for splitting x. For the MOPMX methods, f is a factor, a list of factors, or an object suitable as metadata key. If a factor, it is directly used for splitting x. If a list of factors, the factor lengths must correspond to the lengths of the elements of x, in turn. Each element of x is then split separately, and the resulting MOPMX objects are reassigned, yielding a list with one MOPMX object per factor level. Factor levels that do not occur in some of the elements of x are dropped, with a warning, independent of drop argument. If f is neither a factor nor a list of factors, such a list of factors is generated from the metadata, with NULL results replaced by NA.
drop	Passed to [. The default is FALSE.

Details

This OPMS method of merge is intended for dealing with slowly growing or reacting organisms that need to be analysed with subsequent runs of the same plate in PM mode. Results obtained with

Geodermatophilus strains and Generation-III plates indicate that this works well in practice. See the references, and see the documentation of the `montero_et_al` data set in the **opmdata** package.

See the arguments `time.fmt` and `time.zone` of `opm_opt` for modifying the parsing of setup-time entries. If it does not work, additional time-string templates must be stored.

The `CMAT` method of `merge` is only for internal use.

The `split` methods with missing `f` are for splitting plates that contain series of substrate-usage assays as indicated in the full substrate names (mostly interpretable as concentrations). `OPMS` objects are generated that contain each replicate within the series in a separate plate and the replicate ID indicated in the metadata entry given by `opm_opt("series.key")`. This allows for comparisons between within-plate replicates.

Splitting according to substrate series will not work if these are not regular, i.e. the same substrates occur in each replicate. In such cases `x` will be returned, with a warning. Substrates without a replicate ('concentration') indicator would silently be skipped, however. The composition and order of the wells per pseudo-plate must be made uniform. This is done by enforcing well names and well ordering of the first replicate in all forthcoming replicates.

After a successful splitting, the numeric suffixes in the full well names make no sense any more, as each plate contains a constant set of such suffixes. The `no.num` argument of `wells` and the dependent methods can be used to remove the suffixes before displaying the full well names.

The `MOPMX` method for `merge` will raise an error if elements occur within `x` (and optionally `y`) that have the same plate type but cannot be combined any way because they contain distinct sets of wells. See the comments on combining plates into a `OPMS` object.

Value

The `OPMX` method of `merge` yields an `OPM` object. The `metadata` and `csv_data` will be taken from the first contained plate, but aggregated values, if any, will be dropped.

The `MOPMX` method for `merge` yields a `MOPMX` object with a potentially different number of elements.

The `split` methods yield either an `OPMS` or an `MOPMX` object; the `MOPMX` method for `split` yields a list of `MOPMX` objects.

References

Montero-Calasanz, M. d. C., Goeker, M., Poetter, G., Rohde, M., Sproeer, C., Schumann, P., Gorbushina, A. A., Klenk, H.-P. 2012 *Geodermatophilus arenarius* sp. nov., a xerophilic actinomycete isolated from Saharan desert sand in Chad. *Extremophiles* **16**, 903–909.

Montero-Calasanz, M. d. C., Goeker, M., Rohde, M., Schumann, P., Poetter, G., Sproeer, C., Gorbushina, A. A., Klenk, H.-P. 2013 *Geodermatophilus siccatus* sp. nov., isolated from arid sand of the Saharan desert in Chad. *Antonie van Leeuwenhoek* **103**, 449–456.

See Also

opmdata::montero_et_al

Other conversion-functions: `as.data.frame`, `extract`, `extract_columns`, `flatten`, `oapply`, `opmx`, `plates`, `rep`, `rev`, `sort`, `to_yaml`, `unique`

Examples

```
## merge: OPM methods
stopifnot(identical(merge(vaas_1, 0.5), vaas_1)) # nothing to merge
summary(x <- merge(vaas_1, vaas_1)) # biologically unreasonable!
stopifnot(is(x, "OPM"), dim(x) == c(2 * hours(vaas_1, "size"), 96))

## merge: OPMS methods
summary(x <- merge(vaas_4)) # biologically unreasonable for these data!
stopifnot(is(x, "OPM"), dim(x) == c(sum(hours(vaas_4, "size")), 96))

# See opmdata::montero_et_al for an object to which this can be sensibly
# applied. An according example is given in the montero_et_al documentation.

## split: OPM methods
(x <- split(vaas_1))
metadata(x, opm_opt("series.key"))
stopifnot(is(x, "OPMS"), dim(x) == c(2, hours(vaas_1, "size"), 1))
# only D-Serine is present as series, all other wells are skipped
# thus split is more useful when applied to other plate types such as "ECO"

(x <- split(vaas_1, "Species"))
stopifnot(is(x, "MOPMX"), length(x) == 1)

## split: OPMS methods
(x <- split(vaas_4))
metadata(x, opm_opt("series.key"))
stopifnot(is(x, "OPMS"), dim(x) == c(8, hours(vaas_4, "size")[1], 1))

(x <- split(vaas_4, "Species"))
stopifnot(is(x, "MOPMX"), length(x) == 2)

# Split into list of OPMS objects with the same overall measurement hours
x <- split(vaas_4, as.factor(hours(vaas_4)))
stopifnot(is(x, "MOPMX"), length(x) == 1, class(x[[1]]) == "OPMS")
# ... because the running times were actually already identical, the list
# contains only a single element.
```

`metadata` *Get metadata*

Description

Get meta-information stored together with the data or collect all 'character' entries from the meta-information stored together with the measurements. Optionally coerce data of other types.

Usage

```
  ## S4 method for signature 'MOPMX'
metadata(object, ...)
```

```
  ## S4 method for signature 'WMD'
metadata(object, key = NULL, exact = TRUE,
    strict = FALSE)
  ## S4 method for signature 'WMDS'
metadata(object, ...)

  ## S4 method for signature 'MOPMX'
metadata_chars(object, ...)
  ## S4 method for signature 'WMD'
metadata_chars(object, values = TRUE,
    classes = "factor")
  ## S4 method for signature 'WMDS'
metadata_chars(object, ...)
```

Arguments

object	WMD, WMDS or MOPMX object.
key	NULL, vector, factor or formula. • If NULL or otherwise empty, return all metadata. • If a non-empty list, treated as list of keys. Return value would be the list of corresponding metadata values. Here, character vectors of length > 1 can be used to query nested metadata lists. • If neither empty nor a list nor a formula (i.e. usually a character or numeric vector), key is treated as a single list key. Factors are converted to 'character' mode. • Formulae can also be used and are converted to a list or character or numeric vector using the rules described under 'Details'. • It is in general not recommended to use numeric vectors as key arguments, either directly or within a list or formula.
exact	Logical scalar. Use exact or partial matching of keys? Has no effect if key is empty.
strict	Logical scalar. Is it an error if a NULL value results from fetching a metadata key?
values	Logical scalar. If FALSE, metadata names, not values, are collected, and classes is ignored (names are always of class 'character' and need not be coerced).
classes	Character vector containing the names of classes that should also be collected (and coerced to 'character'), or TRUE. In that case, a mapping template for the classes themselves is returned. See the coerce argument of map_values for details.
...	Optional argument passed from the WMDS to the WMD method.

Details

The result of metadata_chars can be used to create a mapping for map_metadata. The WMDS method just applies the WMD method to all contained plates in turn.

Value

metadata generates a list (empty if metadata were not set or if partial selection using key did not result).

metadata_chars yields a character vector, sorted and made unique. Original names attributes, if any, are dropped and replaced by the character vector itself. (This might be convenient regarding its use with map_metadata.)

See Also

Other metadata-functions: edit, include_metadata, map_metadata, map_values, metadata.set,

Examples

```
# 'WMD' methods

(x <- metadata(vaas_1, "Strain"))
stopifnot(x == "DSM30083T")
(y <- metadata(vaas_1, ~ Strain)) # using a formula => same result
stopifnot(identical(x, y))

(x <- metadata_chars(vaas_1, values = FALSE))
stopifnot(names(x) == x) # mapping metadata keys to themselves
(x <- metadata_chars(vaas_1, values = TRUE))
stopifnot(names(x) == x) # mapping metadata values to themselves
# See map_metadata() for a potential usage of the metadata_chars() result

# 'WMDS' methods

(x <- metadata(vaas_4, "Strain"))
stopifnot(x == c("DSM18039", "DSM30083T", "DSM1707", "429SC1"))
(y <- metadata(vaas_4, ~ Strain)) # using a formula => same result
stopifnot(identical(x, y))

(x <- metadata_chars(vaas_4, values = TRUE)) # the values
(y <- metadata_chars(vaas_4, values = FALSE)) # the keys
stopifnot(length(x) > length(y))
```

---

metadata.set *Replace metadata*

Description

Set the meta-information stored together with the data. For most kinds of arguments the WMDS and MOPMX methods set the meta-information stored together with the measurements for all plates at once. But they can address the plates individually if value is a data frame, and they can address metadata keys individually if value is a formula.

Usage

```
  ## S4 replacement method for signature 'MOPMX,ANY,ANY'
metadata(object, key) <- value
  ## S4 replacement method for signature 'MOPMX,ANY,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'MOPMX,missing,ANY'
metadata(object, key) <- value
  ## S4 replacement method for signature 'MOPMX,missing,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,ANY,ANY'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,character,ANY'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,character,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,character,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,character,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,list,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,list,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,list,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,list,list'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,missing,FOE'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,missing,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,missing,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,missing,data.frame'
metadata(object,
    key) <- value
  ## S4 replacement method for signature 'WMD,missing,list'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,numeric,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,numeric,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,numeric,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMD,numeric,list'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,ANY,ANY'
metadata(object, key) <- value
```

```
  ## S4 replacement method for signature 'WMDS,ANY,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,ANY,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,ANY,data.frame'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,character,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,character,data.frame'
metadata(object,
    key) <- value
  ## S4 replacement method for signature 'WMDS,missing,FOE'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,missing,WMD'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,missing,WMDS'
metadata(object, key) <- value
  ## S4 replacement method for signature 'WMDS,missing,data.frame'
metadata(object,
    key) <- value
  ## S4 replacement method for signature 'WMDS,missing,list'
metadata(object, key) <- value
```

Arguments

object	WMD, WMDS or MOPMX object.
key	Missing, numeric scalar, character vector, factor, or list. • If missing, replace all metadata by value (unless value is a formula that specifies the key to replace). • If a numeric scalar, then if positive, prepend value to old metadata. If negative, append value to old metadata. If zero, replace old metadata entirely by value. • If a list, treated as list of keys; expect value to be a list of corresponding metadata values to be set. Names are replaced by the values of either list if they are missing. • If a character vector, used as key for setting/replacing this metadata entry to/by value. It is an error if key has zero length. If it contains more than one entry, a nested query is done. See [[from the **base** package for details. • The factor method calls the character method after converting key to mode 'character'.
value	Character vector, list, data frame, formula, WMD or WMDS object. • If key is a character vector, this can be arbitrary value(s) to be included in the metadata (if NULL, this metadata entry is deleted). • If key is otherwise, value must be list of values to be prepended, appended or set as metadata, either entirely or specifically, depending on key. • Formulae can also be used as value. In that case, the formula can specify the key to be replaced. See the examples below and map_values for details.

- If `object` is of class `WMDS`, `value` can be a data frame whose number of rows must be equal to the number of plates. Metadata to be set will then be selected from each individual row in turn and in input order. This works analogously if `value` is an `WMDS` object. The lengths of both objects must match. If `value` is a `WMD` object, its metadata entries will be recycled.
- If `object` is of class `WMD`, `value` cannot be of class `WMD`.

Details

This method can easily be used to copy (selected parts of) the `csv_data` to the metadata; see there for details.

`map_metadata` can also be used to modify metadata but it will return a novel object. See `edit` for manually modifying metadata.

Value

`value`.

See Also

Other metadata-functions: `edit`, `include_metadata`, `map_metadata`, `map_values`, `metadata`, `metadata_chars`

Examples

```
## WMD methods

# WMD/missing/list method
copy <- vaas_1
new.md <- list(Species = "Thermomicrobium roseum")
metadata(copy) <- new.md
stopifnot(identical(metadata(copy), new.md))

# WMD/missing/formula method (operates on previous entries!)
copy <- vaas_1
metadata(copy) <- Organism ~ paste(Species, Strain)
(x <- metadata(copy, "Organism"))
stopifnot(is.null(metadata(vaas_1, "Organism")), !is.null(x))

# WMD/numeric/list method
copy <- vaas_1
metadata(copy, 1) <- list(Authors = "Vaas et al.")
stopifnot(length(metadata(copy)) > length(metadata(vaas_1)))

# WMD/list/list method
copy <- vaas_1
stopifnot(identical(metadata(copy, "Species"), "Escherichia coli"))

# You can use this to translate the keys on-the-fly...
metadata(copy, list(Organism = "Species")) <- list(
  Organism = "Bacillus subtilis")
```

```
stopifnot(length(metadata(copy)) == length(metadata(vaas_1)))
stopifnot(identical(metadata(copy, "Species"), "Bacillus subtilis"))
stopifnot(is.null(metadata(copy, "Organism"))) # this was not set!

# ...but you need not
metadata(copy, list("Species")) <- list(Species = "Yersinia pestis")
stopifnot(length(metadata(copy)) == length(metadata(vaas_1)))
stopifnot(identical(metadata(copy, "Species"), "Yersinia pestis"))

# Names need not be duplicated
metadata(copy, list("Species")) <- list("Gen. sp.")
stopifnot(length(metadata(copy)) == length(metadata(vaas_1)))
stopifnot(identical(metadata(copy, "Species"), "Gen. sp."))

# ...but this would delete the entry because nothing would be found in
# 'value'
metadata(copy, list("Species")) <- list(Organism = "E. coli")
stopifnot(length(metadata(copy)) < length(metadata(vaas_1)))
stopifnot(is.null(metadata(copy, "Species")))

# ...this yields a general mechanism for metadata deletion by providing an
# empty list as 'value'.

# WMD/character/any method
copy <- vaas_1
metadata(copy, "Strain") <- "08/15"
stopifnot(length(metadata(copy)) == length(metadata(vaas_1)))
stopifnot(metadata(copy, "Strain") != metadata(vaas_1, "Strain"))

# WMD/factor/any method
metadata(copy, as.factor("Strain")) <- metadata(vaas_1, "Strain")
stopifnot(metadata(copy, "Strain") == metadata(vaas_1, "Strain"))

## WMDS methods

# WMDS/missing/list method
copy <- vaas_4
(metadata(copy) <- list(x = -99)) # will replace all of them
stopifnot(identical(unique(metadata(copy)), list(list(x = -99))))
metadata(copy[2]) <- list(x = 1) # will replace those of 2nd plate
stopifnot(identical(unique(metadata(copy)),
  list(list(x = -99), list(x = 1))))

# WMDS/missing/WMD method
(metadata(copy) <- vaas_1) # will also replace all of them
stopifnot(identical(unique(metadata(copy)), list(metadata(vaas_1))))

# WMDS/missing/formula method
copy <- vaas_4
metadata(copy) <- Organism ~ paste(Species, Strain)
(x <- metadata(copy, "Organism"))
stopifnot(length(x) == length(metadata(vaas_4, "Organism")) + 4)
```

```
# WMDS/ANY/ANY method
copy <- vaas_4
(metadata(copy, "Species") <- "Bacillus subtilis") # will set all of them
stopifnot(identical(unique(metadata(copy, "Species")), "Bacillus subtilis"))
stopifnot(!identical(metadata(copy), metadata(vaas_4)))
metadata(copy) <- vaas_4 # reset
metadata(copy)
stopifnot(identical(metadata(copy), metadata(vaas_4)))
(metadata(copy) <- vaas_1) # set everything to metadata of vaas_1
stopifnot(identical(unique(metadata(copy)), list(metadata(vaas_1))))

# WMDS/character/data frame method
copy <- vaas_4
(x <- data.frame(Type = grepl("T$", metadata(vaas_4, "Strain"))))
metadata(copy, "Type") <- x
# one-column data frames are simplified
stopifnot(identical(metadata(copy, "Type"), x$Type))
# if keys match, a partial selection of the data frame is used
(x <- cbind(x, Notype = !x$Type))
metadata(copy, "Type") <- x
stopifnot(identical(metadata(copy, "Type"), x$Type))
# if keys do not match, the entire data-frame rows are included
metadata(copy, "Type2") <- x
stopifnot(!identical(metadata(copy, "Type2"), x$Type))
```

OPM *Real classes of the* **opm** *package*

Description

Classes whose members can be generated and manipulated by an **opm** user: OPM, OPMA, OPMD, OPMS and MOPMX.

Details

OPM is an acronym for 'OmniLog® Phenotype Microarray'. This is the class for holding single-plate OmniLog® phenotype microarray data without aggregated values, but with information read from the original input CSV files as well as an additional arbitrary amount of arbitrarily organised metadata. Objects of this class are usually created by inputting files with `read_single_opm` or `read_opm`, not with a call to `new` or `as`.

OPM inherits from `WMD` and, hence, has all its methods.

Regarding the coercion of this class to other classes (see the `as` method from the **methods** package), consider the following:

- Conversion of its child classes to this class is straightforward, see the examples below.
- The coercion of this class (and its child classes) to a list (and vice versa) is only for expert users and relies on a mapping between slot names and keys in the list, i.e. the list must be appropriately named. For instance, this is the mechanism when reading from and writing to YAML, see `to_yaml`.

- Coercions to other data frames and matrices first coerce the `measurements` and then add the other slots as attributes.
- Methods such as `flatten` and `extract` might be way more appropriate for converting OPM objects.

OPMA is an acronym for 'OPM, aggregated'. This is the class for holding single-plate OmniLog® phenotype microarray data together with aggregated values. Objects of this class are usually created by calling `do_aggr` on an OPM object, or by inputting files with `read_single_opm` or `read_opm` if these files already contain aggregated data.

OPMA inherits from OPM and, hence, has all its methods.

OPMD is an acronym for 'OPM, discretised'. This is the class for holding single-plate OmniLog® phenotype microarray data together with aggregated **and** discretised values. Objects of this class are usually created by calling `do_disc` on an OPMA object, or by inputting files with `read_single_opm` or `read_opm` if these files already contain discretised data.

OPMD inherits from OPMA and, hence, has all its methods.

The discretised data are considered as 'consistent' with the curve parameter from which they have been estimated if no `FALSE` value corresponds to curve parameter larger than the curve parameter of any `TRUE` value; `NA` values are not considered when checking consistency. The `strict.OPMD` entry of `opm_opt` determines whether an error or only a warning is issued in the case of inconsistency.

OPMS is the class for holding multiple-plate OmniLog® phenotype microarray data with or without aggregated or discretised values. Regarding the name: OPMS is just the plural of OPM. Objects of this class are usually created by calling `opms` or other combination functions on OPM or derived objects, or by inputting files with `read_opm` if these files altogether contain more than a single plate. OPMS objects are not normally created with a call to `new` or `as`. The data may have been obtained from distinct organisms and/or replicates, but **must** correspond to the same plate type and **must** contain the same wells.

OPMS inherits from `WMDS` and, hence, has all its methods. As a rule, OPMS has the same methods as the OPM class, but adapted to a collection of more than one OPM object. Also, OPMS can hold OPMD and OPMA as well as OPM objects, even though this is not indicated for all its methods in this manual.

MOPMX is an object for holding OPMX objects with potentially multiple plate types. Regarding the name: the OPMS stands for 'multiple'. MOPMX objects are generated by `read_opm(convert = "grp")` and `opms(group = TRUE)`. They currently contain relatively few methods of their own and behave like lists. MOPMX objects can be created with `new` or `as` and then further manipulated; see the examples below.

OPM_MCP_OUT is a data-frame based class useful as intermediate result of `opm_mcp`. See there and its `annotated` method for usages.

See Also

methods::Methods methods::new

Other classes: `FOE`, `OPMA_DB`, `OPMD_DB`, `OPMX`, `OPM_DB`, `WMD`, `WMDS`, `WMDX`, `YAML_VIA_LIST`

Examples

```
## overview on the classes
```

```
showClass("OPM")
showClass("OPMA")
showClass("OPMD")
showClass("OPMS")
showClass("MOPMX")

## OPMX conversions with as()
showMethods("coerce", classes = c("OPM", "OPMA", "OPMD", "OPMS"))
data(vaas_1)
data(vaas_4)
(x <- as(vaas_1, "OPMA")) # drops the discretised data
stopifnot(has_disc(vaas_1), !has_disc(x))
(x <- as(vaas_1, "OPM")) # drops the aggregated data
stopifnot(has_aggr(vaas_1), !has_aggr(x))

## MOPMX creation and conversion
(x <- new("MOPMX")) # don't do this with the other classes
(x <- as(vaas_1, "MOPMX"))
(x <- as(vaas_4, "MOPMX"))
# conversion backwards is only possible as long as the MOPMX object contains
# only a single OPMX object
showMethods("coerce", classes = "MOPMX")
```

---

opm.package *The* **opm** *package*

Description

Package for analysing OmniLog® phenotype microarray (PM) data, as well as similar kinetic data such as growth curves.

Details

Here is a brief guideline for using this manual. In addition to this manual, tutorials (vignettes) are available together with the package, as well as code examples accessible via demo.

families All functions and methods belong to a family of functions and methods with similar purposes. The respective other family members are found in each 'See Also' entry.

classes Users normally will create at least one object of the class OPM or derived classes. All these classes store PM data; they differ in whether they also contain aggregated values (OPMA) or aggregated and discretised values (OPMD), and whether they contain more than a single plate of the same plate type (OPMS) or of potentially many different plate types (MOPMX). Example objects are available via vaas_1 and vaas_4.

input Most **opm** users will start by inputting data using read_opm, which create the appropriate objects. OmniLog® phenotype microarray data are structured in **plates**. Each plate has 12 x 8 **well** layout, and each well contains the respiration measurements on one substrate or inhibitor, or combination of substrates or inhibitors. For input example files, see opm_files.

alternatives In addition to PM data, kinetics from other kinds of kinetic information, such as growth curves, can be analysed. The method of choice for converting such data to the objects suitable for **opm** is `opmx`, which accepts a variety of data-frame formats.

global options Options affecting the default parameters of a number of **opm** functions can be set and queried for using `opm_opt`.

forbidden names Some names should be used with caution when annotating **opm** objects; see `param_names` for details.

YAML Input and output of YAML files is based on the **yaml** package. Up to **opm** version 0.7, this package was not required for the installation of **opm**. It is now mandatory to install one of the newer versions of **yaml** (>= v2.1.5). These are based on **libyaml** as parser instead of **Syck**, are faster and contain some bug fixes. The YAML-related functions of **opm** are `to_yaml` and `batch_opm`. Optionally, JSON code can be output, which uses a subset of the YAML format.

running time Computations on such high-dimensional data may take some time. The limiting steps are aggregating (curve-parameter estimation) and plotting many curves together. The former step can be conducted in parallel if `mclapply` from the **parallel** package can be run with more than 1 core (basically anywhere except for Windows). Moreover, the particularly time-consuming bootstrapping can usually be turned off. There is also a fast estimation method for the parameters 'area under the curve' and 'maximum height'. See `do_aggr` and the methods it refers to for details.

advanced plotting The **gplots** package is also not required for the installation of **opm** but can be used to draw more advanced heat maps. See `heat_map` and its accompanying methods for details. The other customised plotting functions of the package are contained in the same method family.

database I/O Working with relational and other databases is easily possible with **opm** provided that such databases exist, are correctly set up and accessible by the user. SQL code for setting up the suggested (extensible) scheme for a relational database comes with the package. See `opm_dbput` for details.

References

http://www.biolog.com/

Bochner, B. R., Gadzinski, P., Panomitros, E. 2001 Phenotype MicroArrays for high throughput phenotypic testing and assay of gene function. *Genome Research* **11**, 1246–1255 (http://dx.doi.org/10.1101/gr.186501).

Bochner, B. R. 2009 Global phenotypic characterization of bacteria. *FEMS Microbiological Reviews* **33**, 191–205.

http://opm.dsmz.de/

Vaas, L. A. I., Sikorski, J., Michael, V., Goeker, M., Klenk H.-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846 (http://dx.doi.org/10.1371/journal.pone.0034846).

Vaas, L. A. I., Sikorski, J., Hofner, B., Goeker, M., Klenk H.-P. 2013 opm: An R package for analysing OmniLog(R) Phenotype MicroArray Data. *Bioinformatics* **29**, 1823–1824 (http://dx.doi.org/10.1093/bioinformatics/btt291).

http://www.yaml.org/

http://www.json.org/

Examples

```
## Not run:  ## show the vignettes
  vignette("opm-tutorial")
  vignette("opm-substrates")
  vignette("opm-growth-curves")

## End(Not run)

## Not run:  ## demo of some I/O, plotting, text and table generation options

  # Beforehand, set 'my.csv.dir' to the name of a directory that contains
  # CSV files with input data (and *no* other kinds of CSV files) either
  # directly or within its subdirectories.
  setwd(my.csv.dir)
  demo("multiple-plate-types", package = "opm")

## End(Not run)

# the other demos require additional libraries to be installed
if (interactive())
  demo(package = "opm")

# list all classes, methods and functions exported by the package
ls("package:opm")
```

---

opms OPMS *constructor*

Description

Easily build OPMS (or MOPMX) objects.

Usage

```
opms(..., precomputed = TRUE, skip = FALSE,
  group = FALSE)
```

Arguments

...	One to several objects which are either potentially nested lists of OPMS, OPM or OPMA objects, or really nested lists whose contained lists can be converted to an OPM or OPMA object.
precomputed	Logical scalar. If TRUE, contained lists have already been converted to one of the three classes. Otherwise, suitable contained lists will be converted.
skip	Logical scalar. If precomputed is TRUE, silently skip non-list elements of nested lists? If precomputed is FALSE, silently skip objects that do not belong to the three target classes? Otherwise, an error is generated if such a list element is encountered.

group — Logical or character scalar. If TRUE, split the list of collected OPM objects according to the plate type and convert the contained lists separately if they contain more than one plate; otherwise just keep the OPM object. FALSE is the default: all plates are tried to be forced into a single OPMS object. If a character scalar, the name of the plate type to be extracted.

Details

While otherwise rather flexible, this function will fail to return an OPMS object if group is set to FALSE and the plate types do not match (simply because such OPMS objects are disallowed). But if group is set to TRUE, a list (MOPMX object), not a single OPMS object will be returned; and if group is of mode 'character', this extracts the plate type(s) of interest.

Note that read_opm already has plate-type selection options.

Value

OPMS object, or list (MOPMX object) of such objects (and/or OPM objects), or OPM object, or NULL.

See Also

Other combination-functions: $<-, [<-, [[<-, c, plus

Examples

```
## Testing distinct OPM/OPMS combinations -- all should work.
## Note the number of contained plates in the generated objects.

(x <- opms()) # 0 objects
stopifnot(is.null(x))
(x <- opms(group = TRUE)) # 0 also objects
stopifnot(is(x, "MOPMX"), length(x) == 0)

dim(x <- opms(vaas_1)) # 1 object
stopifnot(identical(x, vaas_1))
dim(x <- opms(vaas_4, group = plate_type(vaas_4)))
stopifnot(identical(x, vaas_4))
dim(x <- opms(vaas_4, group = "PM01"))
stopifnot(is.null(x)) # no such plate type => empty object!

dim(x <- opms(vaas_1, vaas_1)) # 2 objects
stopifnot(is(x, "OPMS"), length(x) == 2L)
dim(x <- opms(vaas_4, vaas_1))
stopifnot(is(x, "OPMS"), length(x) == 5L)
dim(x <- opms(vaas_1, vaas_4))
stopifnot(is(x, "OPMS"), length(x) == 5L)
dim(x <- opms(vaas_4, vaas_4))
stopifnot(is(x, "OPMS"), length(x) == 8L)
```

opmx *Convert user-defined objects to* OPMX

Description

Convert data frames with user-defined plate types to OPMX or MOPMX objects.

Usage

```
  ## S4 method for signature 'data.frame'
opmx(object,
    format = c("horizontal", "rectangular", "vertical"), plate.type = NULL,
    position = NULL, well = NULL, prefix = "T_", sep = "<>", full.name = NULL,
    setup.time = date(), filename = "", interval = NULL)
```

Arguments

`object` Data frame containing numeric data with growth or respiration measurements and optional or mandatory additional columns, depending on the `format` argument.

`format` Character scalar indicating the data layout within `object`. See below for examples. In brief, the formats are:

horizontal One row per well, with additional columns providing the substrate names, metadata that identify the plate, and optionally other columns to be used as `metadata` or `csv_data` entries. The time points must be given in columns that can be identified with a certain prefix. The part after the prefix must be convertible to numeric data (the time points, ideally given in hours). Several plates and even several plate types can be contained within `object`.

rectangular Several rows and columns per time point, yielding a set of rectangles per plate. Only a single plate, and only measurements are contained within `object`, thus some of the other arguments cannot be empty. See the `interval` argument for setting time points.

vertical One column per well. Only a single plate, and only measurements and time points are contained within `object`, thus some of the other arguments cannot be empty. The time points are either contained as column or can be read from the row names. Ideally, they are given in hours.If `object` has no column names, the first column or the row names yield the time points.

`position` Character vector. In 'horizontal' format, the name of one to several columns to be joined yielding 'position' indicators. These will be used to uniquely identify each plate. The columns to be joined will be kept, too; usually they will end up in the `metadata`. In this case, the resulting 'position' indicators are newly generated rather than literally taken from the input, but yield the same grouping. An empty `position` argument is possible, but then an accordingly named column must already be present, whose content is used literally.

The position argument is mandatory for the 'rectangular' and 'vertical' formats. It should be chosen so as to identify the resulting OPM object again once it is combined with others into an OPMS object. (By default the setup time is additionally considered but the default for the setup.time argument is just the time of the call to opmx.)

For plate position values to be used literally, integers between 0 and 99 (inclusively) followed by a single letters are recommended, because this allows **opm** to normalise this entry. See the eponymous argument of csv_data.

plate.type — Character scalar. In 'horizontal' mode, the name of the column containing the plate-type indicators. After normalisation, these will be used for storing the mapping of well coordinates. The argument is ignored if empty. But then an accordingly named column must already be present **OR** full.name must contain a single named element, whose name is then inserted as plate name.

The plate.type argument is mandatory for the 'rectangular' and vertical formats. An according plate type must already have been stored using register_plate and contain the well coordinates found in object. Normalisation of the plate-type name is done, however.

In 'horizontal' mode, the plate type can be registered beforehand, too, which is useful to enforce a certain ordering of wells. But then the registered well-coordinate map must contains all well coordinates found in object.

well — Character scalar. In 'horizontal' format, the name of the column containing the well indicators. These should be substrate names; an according mapping from (newly assigned) well coordinates to these substrate names will then be stored using register_plate if it is not yet present. Ignored if empty (but then an accordingly named column must be present).

prefix — Character scalar. In 'horizontal' format, used for identifying the measurements columns.

sep — Character scalar. In 'rectangular' format, used for identifying the rows and column with time-points and well coordinates.

full.name — Named character vector indicating the full plate names. Ignored if empty. Names should be names of the plate types found within object, if any, but normalisation will be done. Values should be the respective full names. Missing ones are silently ignored.

If the plate type is not found within object, then it is taken from the name of full.name, assuming a uniform plate type throughout object. In that case, full.name must contain only a single element (and a single name).

setup.time — Character scalar to be inserted if missing in the data. Like the next argument, the value goes into the csv_data.

filename — Character scalar to be inserted if missing in the data.

interval — Numeric vector. If of length one, indicating the time interval between measurements in the 'rectangular' format. If the length corresponds to the number of measurements per well in object, it is interpreted directly as the time points. This is useful if the intervals are non-unique. Ignored if empty, causing `0, 1, 2, ...` to be used as time points. (This is often acceptable as it only causes a different scaling; it is not acceptable if the time points were not in regular intervals.)

In the case of the `vertical` format, a non-empty `interval` value causes the time points to not be extracted from `object` but constructed from `interval`. Ideally, `interval` is given in hours (because this corresponds to the default axis labelling of some plotting functions).

Details

The main purpose of this function is to convert objects that hold non-PM data to `OPMX` objects that can be analysed with **opm**. The mechanism for dealing with user-defined plate types is implemented in `register_plate`, whereas `opmx` also takes care of the necessary changes in format and naming for converting a data frame to a `MOPMX` object.

Value

`OPMX` or `MOPMX` object or `NULL`, depending on how many distinct plate types are encountered within `object`.

See Also

Other conversion-functions: `as.data.frame`, `extract`, `extract_columns`, `flatten`, `merge`, `oapply`, `plates`, `rep`, `rev`, `sort`, `split`, `to_yaml`, `unique`

Examples

```
## 'horizontal' input format

# fake data frame containing growth or respiration measurements
x <- data.frame(
  Treatment = c(rep("Control", 4), rep("Heat stress", 4)),
  Strain = paste0("X", c(1, 2, 2, 1, 2, 1, 1, 2)),
  Substrate = c(rep("Glucose", 2), rep("Galactose", 4), rep("Glucose", 2)),
  T_0 = c(12, 5, 8, 6, 8, 9, 7, 10),
  T_5 = c(23, 7, 7, 18, 30, 10, 8, 9),
  T_10 = c(79, 9, 10, 64, 67, 8, 6, 11),
  T_15 = c(103, 8, 46, 99, 101, 17, 9, 8),
  T_20 = c(105, 9, 77, 112, 103, 44, 8, 12)
)

# The plate type is not contained and thus taken from 'full.name', but
# the wells are obviously within 'Substrate', and each combination of
# 'Treatment' and 'Strain' is apparently one group of measurements
# (interpreted as 'plate').
y <- opmx(x, well = "Substrate", position = c("Treatment", "Strain"),
  full.name = c(sugars = "Fake sugar test plate"))

# This yields a single OPMX object as there is only one plate type.
stopifnot(is(y, "OPMX"), dim(y) == c(4, 5, 2))
print(xy_plot(y, include = list("Strain", "Treatment"),
  theor.max = FALSE, main = list(in.parens = FALSE), ylab = "Hours"))

## 'rectangular' input format
```

```
# Get the input file. The rectangular format is hardly suitable for R
# but produced by plate readers such as those distributed by the TECAN
# company.
growth.data.file <- grep("tecan", opm_files("growth"), ignore.case = TRUE,
  value = TRUE)

if (length(growth.data.file)) { # if the file was found

  x <- read.table(growth.data.file)
  head(x)

  # Creating a fake well map. For really making sense of these data, one
  # would need to know the real substrate names.
  well.map <- rbind(1:6, 1:6, 1:6, 1:6)
  well.map[] <- paste0("Substrate ", LETTERS[1:4], well.map)

  # Registering the plate type beforehand is mandatory here because the
  # file does not contain the real substrate names.
  register_plate(XYZ = well.map)
  (y <- opmx(x, "rectangular", plate.type = "XYZ", position = 1,
    interval = 0.25))
  plate_type(y) # => a custom (user-defined) plate
  stopifnot(setequal(wells(y, full = TRUE, in.parens = FALSE), well.map))

  register_plate(XYZ = NULL) # tidying up

} else {
  warning("file with growth data not found")
}

## 'vertical' input format

# Fake data frame. It is safer to set all column names explicitly.
# If none are there, the first column yields the time points unless
# there are explicitly set row names.
x <- data.frame(
  c(0, 5, 10, 15, 20),
  c(12, 23, 79, 103, 105),
  c(5, 7, 9, 8, 9),
  c(8, 7, 10, 46, 77),
  c(6, 18, 64, 99, 112),
  c(8, 30, 67, 101, 103),
  c(9, 10, 8, 17, 44),
  c(7, 8, 6, 9, 8),
  c(10, 9, 11, 8, 12)
)
colnames(x) <- NULL # necessary for this example

# Creating a fake well map for the fake data frame.
well.map <- paste("Substrate", 1:8)
names(well.map) <- paste0("A", 1:8)

# Registering the plate type beforehand is mandatory here because the
```

```
# input data frame does not contain the real substrate names.
register_plate(XYZ = well.map)
wells(plate = "CUSTOM:XYZ")[1:10]

(y <- opmx(x, "vertical", plate.type = "XYZ", position = 1))

plate_type(y) # => a custom (user-defined) plate
stopifnot(setequal(wells(y, full = TRUE, in.parens = FALSE), well.map))

register_plate(XYZ = NULL) # tidying up
```

OPM_DB *Classes for* **opm** *database I/O*

Description

These child classes of DBTABLES from the **pkgutils** package hold intermediary objects that can be used for database input and output of OPMX objects. These classes are not normally directly dealt with by an **opm** user but are documented here for completeness. See opm_dbput for methods that internally use these classes for database I/O.

Details

See their documentation for details on OPMX objects themselves. We here define the following additional classes:

OPM_DB Holds all data that occur in an OPM object, or in several such objects as contained in an OPMS object.

OPMA_DB Holds all data that occur in an OPMA object, or in several such objects as contained in an OPMS object.

OPMD_DB Holds all data that occur in an OPMD object, or in several such objects as contained in an OPMS object.

The inheritance relationships thus mirror those of the OPMX objects (with the exception of OPMS). Conversion with as is implemented from all OPMX classes to all classes defined here. Lists can also be converted provided they only contain OPMX objects (or lists of such objects).

Conversion in the other direction, yielding one of the OPMX classes, is also implemented. Attempting to convert several plates to an OPMX class other than OPMS will yield an error, however, as well as trying to convert a single plate to OPMS, or several plates with distinct plate types. In contrast, conversion to a list will work in all instances, and such a list could further be processed with the opms function, irrespective of the number of plates contained.

In contrast to the OPMX classes, the three ones defined here can be created using new, yielding empty objects. These can neither be converted to OPMX objects nor combined with them using c. Instead, they are useful in conjunction with by from the **pkgutils** package with do_inline set to TRUE. They contain all fkeys information and can be filled using a suitable FUN argument.

See Also

methods::Methods methods::new opm::opms

Other classes: FOE, MOPMX, OPM, OPMA, OPMD, OPMS, OPMX, OPM_MCP_OUT, WMD, WMDS, WMDX, YAML_VIA_LIST

Examples

```
library(pkgutils)

## overview on the classes
showClass("OPM_DB")
showClass("OPMA_DB")
showClass("OPMD_DB")

## show all conversions with as()
showMethods("coerce", classes = c("OPM_DB", "OPMA_DB", "OPMD_DB"))

## conversions back and forth, OPMD as starting point
(x <- as(vaas_1, "OPMD_DB"))
(y <- as(x, "OPMD"))
stopifnot(
  dim(y) == dim(vaas_1),
  # numeric data remain except for rounding errors:
  all.equal(measurements(y), measurements(vaas_1)),
  all.equal(aggregated(y), aggregated(vaas_1)),
  all.equal(discretized(y), discretized(vaas_1)),
  # file names get normalized, hence CSV dat may get unequal:
  !isTRUE(all.equal(csv_data(y), csv_data(vaas_1)))
)
(y <- try(as(x, "OPMS"), silent = TRUE))
stopifnot(inherits(y, "try-error")) # does not work because only 1 plate

## conversions back and forth, OPMS as starting point
(small <- vaas_4[, , 11:15])
(x <- as(small, "OPMD_DB"))
(y <- as(x, "OPMS"))
stopifnot(sapply(1:length(y), # same values
  function(i) dim(y[i]) == dim(small[i])))
(y <- try(as(x, "OPMD"), silent = TRUE)) # does not work because > 1 plate
stopifnot(inherits(y, "try-error"))
(y <- as(x, "list")) # one can always go through a list
stopifnot(sapply(y, is, "OPMD")) # opms() could now be called

## one can create new objects without data
(y <- new("OPMD_DB"))
stopifnot(fkeys_valid(y), fkeys(y) == fkeys(x), !length(y))
# such objects cannot be converted to OPMX but can be filled using by()
```

---

opm_dbput *Database I/O for* **opm**

Description

Methods for inserting, querying and deleting OPMX objects into or from (SQL-based) relational databases. A common database scheme is assumed as defined in the auxiliary SQL files of this package (run opm_files in "sql" mode), but tables could be named differently, and columns could be added containing user-defined combinations of metadata.

Usage

```
  ## S4 method for signature 'ANY'
opm_dbcheck(conn, metadata = NULL,
    time.points = TRUE, wells = TRUE)

  ## S4 method for signature 'MOPMX'
opm_dbclass(object)
  ## S4 method for signature 'OPM'
opm_dbclass(object)
  ## S4 method for signature 'OPMS'
opm_dbclass(object)
  ## S4 method for signature 'numeric'
opm_dbclass(object)

  ## S4 method for signature 'character,ANY'
opm_dbclear(object, conn,
    map.tables = NULL)
  ## S4 method for signature 'integer,DBIConnection'
opm_dbclear(object, conn,
    map.tables = NULL)
  ## S4 method for signature 'integer,RODBC'
opm_dbclear(object, conn,
    map.tables = NULL)

  ## S4 method for signature 'character,DBIConnection'
opm_dbfind(object, conn,
    map.tables = NULL)
  ## S4 method for signature 'character,RODBC'
opm_dbfind(object, conn,
    map.tables = NULL)

  ## S4 method for signature 'character,ANY'
opm_dbget(object, conn,
    map.tables = NULL, include = 2L)
  ## S4 method for signature 'integer,DBIConnection'
opm_dbget(object, conn,
    map.tables = NULL, include = 2L)
  ## S4 method for signature 'integer,RODBC'
opm_dbget(object, conn,
    map.tables = NULL, include = 2L)
```

```
  ## S4 method for signature 'ANY,ANY'
opm_dbnext(object, conn,
    map.tables = NULL)
  ## S4 method for signature 'OPM_DB,DBIConnection'
opm_dbnext(object, conn,
    map.tables = NULL)
  ## S4 method for signature 'OPM_DB,RODBC'
opm_dbnext(object, conn,
    map.tables = NULL)

  ## S4 method for signature 'ANY,ANY'
opm_dbput(object, conn, ...)
  ## S4 method for signature 'OPM_DB,DBIConnection'
opm_dbput(object, conn,
    map.tables = NULL, start = opm_dbnext(object, conn, map.tables))
  ## S4 method for signature 'OPM_DB,RODBC'
opm_dbput(object, conn,
    map.tables = NULL, start = opm_dbnext(object, conn, map.tables))
```

Arguments

object	OPMX, MOPMX or OPM_DB object, integer vector containing real or potential primary keys of a database table, or character scalar containing a partial SQL query (the part after the 'WHERE' keyword).
conn	Database connection object. Currently DBIConnection objects from one of the reverse dependencies of **DBI** (recommended) and RODBC objects as created by the **RODBC** package are supported.
map.tables	Passed as do_map argument to by from the **pkgutils** package. Necessary if table names that deviate from the defaults are to be used.
include	Integer scalar indicating whether aggregated data (1) or aggregated and discretised data (2) or neither (0) should be added to the result. The numeric method of opm_dbnext needs the same kind of object argument.
start	Integer vector determining the minimum primary keys to which those in object should be coerced. Necessary for appending to a database table without overwriting previously inserted data.
metadata	Empty or data frame with metadata to be added to the check object vaas_4. If a data frame, it must contain exactly 2 rows. Adding metadata makes only sense if according columns have been added to the database table for the plates; see the examples below. The original metadata from vaas_4 are always removed.
time.points	Index of one to several time points. Selection speeds up database I/O during checking.
wells	Index of one to several wells. Selection speeds up database I/O during checking.
...	Optional arguments passed between the methods.

Details

The DBIConnection methods send table and column names are through make.db.names from the **DBI** package or its dependencies before including them into SQL queries, if any. As dictated by

by from the **pkgutils** packages, this is done after applying map.tables. The RODBC methods use a simple quoting scheme.

opm_dbcheck attempts to insert, query and delete the first two plates from the object vaas_4 into the database. If everything is correctly set up, this should work without error **unless** these two plates from vaas_4 have already been inserted. If errors occur, it is up to the user to clean up the data base (as far as necessary).

Note that the deletion mechanism is based on ON DELETE CASCADE. To enable this in SQLite, PRAGMA foreign_keys = ON; has to be called each time a database is opened. See the according demo entry.

Value

The main functions are those for create, search, read and delete operations:

- opm_dbput returns an integer vector containing the primary keys of the inserted plates.
- opm_dbfind returns an integer vector containing the primary keys of the found plates.
- opm_dbget returns a MOPMX object with one element per plate type.
- opm_dbclear invisibly returns the result of dbGetQuery (which is usually NULL).

Regarding the helper functions, opm_dbnext returns an integer scalar that is suitable as start argument of opm_dbput, whereas opm_dbclass returns a character scalar with the name of the intermediary class (derived from OPM_DB) to be created for database I/O. These need not normally be called by an **opm** user.

For checking whether a database (connection) is correctly set up, opm_dbcheck is available, which returns a character vector whose elements are either ok or a description of the error that has occurred at that step of the checking process.

See Also

DBI::make.db.names pkgutils::by

Examples

```
# The SQL files for generating the expected database tables. Tables can
# be renamed, but then an according 'map.tables' argument must be used.
opm_files("sql")

# Usage examples are given in these demos. An according database must be
# made accessible beforehand.
if (interactive())
  demo(package = "opm")
```

opm_files *Files, parameter names and colour sets used by the package*

Description

Get list of files from the **opm** package of interest for the user, or get list of predefined parameter names of interest for the user, or select from some predefined colour sets for plotting.

Usage

```
opm_files(what = c("scripts", "testdata", "auxiliary", "demo", "doc", "css", "sql",
  "omnilog", "single", "multiple", "growth"))

param_names(what = c("param.names", "disc.name", "reserved.md.names", "split.at"))

select_colors(set = c("w3c", "w3c.i", "nora", "nora.i", "brewer", "brewer.i", "roseo",
  "roseo.i"))
```

Arguments

what Character scalar indicating the subdirectory to search in or the kind of names to obtain. Currently the following subdirectories are included:

auxiliary Miscellaneous files which are neither executable scripts nor test data.

css Predefined CSS files for HTML files generated by, e.g., phylo_data.

doc The vignette (documentation) in several formats, including the extracted R code.

demo Example R code using the **opm** package that neither fitted into these help pages nor into the vignette. Can directly be loaded via demo; see demo(package = "opm").

growth Growth-measurement example files.

multiple Not directly readable (i.e., multiple-plate) test files.

omnilog Directly readable (i.e., single-plate) test files from OmniLog® runs.

scripts R script files for non-interactive uses of the **opm** package, particularly for the batch processing of many files. When called without input arguments or with the '-h' switch, the scripts output usage information.

single Directly readable (i.e., single-plate) test files.

sql SQL files for working with relational databases.

testdata Files as output by the devices such as the OmniLog® instrument. Included here as examples for data input (and metadata management).

and the following kinds of parameter names:

param.names Names of the estimated curve parameters used internally and in the output.

disc.name Alternative name used to select discretised values instead.

reserved.md.names Names that should not be used in metadata entries because they are used as predefined column names by functions such as `flatten`.

split.at The name of the column in data frames generated by `extract` that separates data from metadata.

`set` Character scalar. Name of the colour vector to use. Colour vectors have been optimised for maximum contrast between adjacent colours, either manually or using `max_rgb_contrast` from the **pkgutils** package. Names ending in '.i' indicate vectors in inverse order (compared to the vector with the same name except '.i').

Details

In addition to the results of 'reserved.md.names', it should be avoided to use metadata keys that start with a dot, as such keys might also be created intermediary by methods that have to compile metadata together with other information.

Note that `pkg_files` might fail with very unusual installations of the **opm** package.

See `xy_plot` for a usage example of `select_colors`. This function is not normally directly called by an **opm** user but could be used for testing before doing some serious plotting.

Value

Character vector of file names or reserved parameter names or names of colours.

References

http://www.colorbrewer.org

See Also

pkgutils::pkg_files utils::demo grDevices::colors grDevices::rainbow grDevices::grey

Other naming-functions: `find_positions`, `find_substrate`, `gen_iii`, `listing`, `plate_type`, `register_plate`, `substrate_info`, `wells`

Examples

```
## example input files
isRfile <- function(x) grepl("\\.R$", x, ignore.case = TRUE)
(x <- opm_files("auxiliary"))
stopifnot(!isRfile(x))
(x <- opm_files("demo"))
stopifnot(isRfile(x))
(x <- opm_files("scripts"))
stopifnot(isRfile(x))
(x <- opm_files("testdata"))
stopifnot(!isRfile(x))
for (name in c("growth", "single", "multiple", "omnilog")) {
  print(y <- opm_files(name))
  stopifnot(y %in% x) # i.e., a subset of the set of all input example files
}
```

```
# On UNIX systems you should be able to do this if Rscript and the optparse
# package are properly installed:
# invisible(sapply(paste("Rscript", opm_files()), system))
# ...and get the usage messages of all scripts.

## reserved parameter names
(x <- param_names())
stopifnot(is.character(x), length(x) > 1, identical(unique(x), x))
(x <- param_names("reserved"))
stopifnot(is.character(x), length(x) > 1, identical(unique(x), x))
stopifnot(param_names("split.at") %in% x)

## colours
(x <- select_colors("nora"))
(y <- select_colors("nora.i")) # same in reverse order
stopifnot(is.character(x), length(x) > 0L, identical(x, rev(y)))
```

---

opm_mcp *Multiple comparison of group means*

Description

This function provides linear-hypothesis testing and multiple comparisons for group means of curve parameters. Note that for most `output` modes (see below) it would be an error if this function was called and the package **multcomp** was unavailable (even though that package is not formally a dependency of **opm**).

Usage

```
  ## S4 method for signature 'MOPMX'
opm_mcp(object, model, linfct = 1L,
    m.type = "glm", rhs = 0, alternative = "two.sided", glht.args = list(),
    ops = "+", output = "mcp", sep = opm_opt("comb.value.join"), ...)
  ## S4 method for signature 'OPMS'
opm_mcp(object, model, linfct = 1L,
    m.type = "glm", rhs = 0, alternative = "two.sided", glht.args = list(),
    ops = "+", output = "mcp", sep = opm_opt("comb.value.join"), ...)
  ## S4 method for signature 'data.frame'
opm_mcp(object, model, linfct = 1L,
    m.type = c("glm", "lm", "aov"), rhs = 0, alternative = "two.sided",
    glht.args = list(), ops = "+",
    output = c("mcp", "data", "model", "linfct", "contrast"),
    sep = opm_opt("comb.value.join"), split.at = param_names("split.at"))
```

Arguments

`object`	Either an `OPMS` object, or a data frame derived via `extract` containing factor variables that determine experimental groups for multiple comparison of means and can be selected using `model`.

`model` A model formula, or a character vector or a list containing the names of factors to be included in the model for fitting. In order to join two or more metadata-variables into one factor use pseudo-function `J` (described in `extract`). This is necessary especially when `linfct = Pairs`, see the examples below. For model specifications using formulae in general, see `formula` (in the **stats** package). For the way models are used by **opm** for selecting metadata entries, see `metadata`.

If `object` is of class `OPMS`, `model` is passed to `extract` after removal of the relevant reserved names (see `param_names`, which can nevertheless be included in the model formula as they are always contained in the resulting data frame. If `model` is a list or vector, it is used for selecting metadata, too, and for running the tests automatically converted to a formula, using one to several operators as specified with `ops`. Non-syntactical names within `formula` are converted to syntactical ones for use with `glht`. This is done in the same way in the data frame passed to that function.

If the `output` argument is set to 'model', the function returns the converted `model` argument, which is useful in exploring the conducted conversions before running the proper tests.

`linfct` A specification of the linear hypotheses to be tested analogously to `linfct` in `glht`. A variety of objects can be used for this argument:

- One of the classes of objects accepted by `glht` from the **multcomp** package as `linfct` argument. Such objects will not be modified. Linear functions can be specified by either the matrix of coefficients or by symbolic descriptions of one or more linear hypotheses. The set of existing types of contrast is extended with the contrast type 'Pairs'. Here all pair-wise comparison concerning the first entry in `model` are computed. Alternatively, the factor whose levels should determine the pairs can be addressed directly with, for example, `linfct = c(Pairs.Well = 1)`.
  The Dunnett-type contrast has the special feature, that users are free to directly define the group which should serve as the control or base in the set of contrasts. In analogy to `Pairs`, the name of the level, separated from the string 'Dunnett' by any sign, can be stated. See examples below and in the vignettes and further `contrMat` from the **multcomp** package.
  In situations where metadata have non-syntactic names, special signs are exchanged against dots. When applying `linfct = c(Pairs = 1)` or `linfct = c(Dunnett = 1)` with the above mentioned extension, the sign between the `linfct` name and the metadata name must not be a dot.
- An object inheriting from the `AsIs` as created by `I` from the **base** package. Such objects, irrespective of their class, will be converted to an argument list for and then passed to `mcp` from the **multcomp** package.
- Other objects will be treated as a selection of factors from the data just like `model`, i.e. they will be converted like any `metadata` key (but note that character vectors would be passed to `glht`). If this yielded a numeric or logical vector, it would be treated as specifying the positions of factors within `model`. If names were present, they would be used as the values of the arguments passed to `mcp`. Otherwise `opm_opt("contrast.type")` would be used. (See the `type` argument of `contrMat`.) The modified object would then be used as the argument list in a call to `mcp`.

After the conversions, if any, this argument is passed to `glht` as `linfct` argument.

If the `output` argument is set to `linfct`, the function returns the converted `linfct` argument, which is useful in exploring the conducted conversions before running the proper tests.

`m.type` Character scalar indicating which of the following model types to use in model fitting: `glm`, `aov` or `lm`. See the eponymous functions in the **stats** package for details.

`rhs` Numeric vector passed to `glht` in the **multcomp** package. Also considered when creating contrasts of the 'Pairs' type.

`alternative` Character scalar also passed to that function (but only if `linfct` is or yields a matrix), and also relevant for 'Pairs'-type contrasts.

`glht.args` List of additional arguments for the multiple comparison procedure passed to `glht`. See `glht` in the **multcomp** package for details.

`ops` Character vector. `ops` is ignored if `model` is directly provided as a formula. Otherwise the provided list or character vector is converted to formula, and `ops` then specifies the operator(s) to insert between the variables in the right part of the `model` formula. Thus, `ops` should contain '+', '*', or ':' as elements. '+' is the default, and the elements are recycled as usual if necessary. See the description of `formula` for further details (in the **stats** package).

`output` Character scalar determining the main output mode. See below under 'Value'.

`split.at` Character vector. See `extract`. Cannot be set in the case of the `OPMS` method.

`sep` Character scalar (comprising a single character) passed to `extract`.

`...` Optional arguments passed to `extract`. Most of them would be passed to `wells` for creating substrate names. Some restrictions are necessary here if the resulting object shall latter on be analysed with `annotated`; see there for details. This holds particularly if `object` is of class `MOPMX`. In that case, setting `full` to `FALSE` is likely to cause most combinations of wells and plates to be omitted because the well names get non-unique.

Details

This function internally reshapes the data in `object` into a 'flat' data frame the structure of which is described under 'value'. In the default `output` mode, `glht` from the **multcomp** package is applied to this data frame. This causes (general linear) models and, by indicating a contrast type, user-defined simultaneous multiple testing procedures to be inferred.

Since either the user or this function itself makes use of `mcp`, we refer to the 'Details' section of the `glht` function. The `mcp` function must be used with care when defining parameters of interest in two-way ANOVA or ANCOVA models. The definition of treatment differences might be problem-specific. An automated determination of the parameters of interest would be impossible and thus only comparisons for the main effects (ignoring covariates and interactions) would be generated and a warning issued.

Value

The kind of object returned by this function are determined by the `output` argument:

mcp The default. An object of class `glht` as the result of the multiple comparison of means. Methods for `print`, `summary`, `confint`, `coef` and `vcov` are available for this class. See `glht` in the **multcomp** package for details.

data Reshaped ('flattened') data frame of the class `OPM_MCP_OUT`. It contains one column for the measured values, one factorial variable determining the well, one factorial variable for the curve parameter (see `param_names`) and additional factorial variables selected by `model` as factors. The column names are converted to syntactical names. Such a data frame might be of use for model-building approaches not covered by this function.

model The `model` argument *after* the conversions conducted by `opm_mcp`, if any.

linfct The `linfct` argument *after* the conversions conducted by `opm_mcp`, if any.

contrast A list of contrast matrices as specified by `model` and `linfct`. As these matrices would be guaranteed to fit to `object`, they could serve as template matrices to be further specified by the user. Note that this only works if `linfct` either is an object of class `mcp` or convertible to such an object, and if its values are character scalars that can be passed as `type` argument to `contrMat` from the **multcomp** package.

Author(s)

Lea A.I. Vaas, Markus Goeker

See Also

multcomp::glht multcomp::contrMat stats::lm stats::formula

Other multcomp-functions: `annotated`

Examples

```
# helper function for plotting with better suitable margins
plot_with_margin <- function(x, mar, ...) {
  old.mar <- par(mar = mar)
  on.exit(par(old.mar)) # tidy up
  plot(x, ...)
}
do.plot <- FALSE # change this to see the plots

## OPMS method

# return the intermediary data frame, do not conduct statistical tests
head(x <- opm_mcp(vaas_4, model = list("Species", "Strain"),
  output = "data"))
stopifnot(is.data.frame(x), dim(x) == c(384, 5))

# watch the converted 'model' argument
(x <- opm_mcp(vaas_4, model = list("Species", "Strain"),
  output = "model"))
stopifnot(inherits(x, "formula")) # left side is set automatically

if (require("multcomp")) {
```

```
# watch the converted 'linfct' argument
(x <- opm_mcp(vaas_4, model = list("Species", "Strain"),
  linfct = c(Dunnett = 1), output = "linfct"))
stopifnot(inherits(x, "mcp"))

# create a template contrast matrix
(x <- opm_mcp(vaas_4, model = list("Species", "Strain"),
  linfct = c(Dunnett = 1), output = "contrast"))
stopifnot(is.list(x), sapply(x, inherits, "contrMat"),
  names(x) == "Species") # the selection is as specified by 'linfct'

# comparison using specified model comparing 'Species' pooled over
# complete plates
(x <- opm_mcp(vaas_4, model = list("Species"), m.type = "lm",
  linfct = c(Dunnett = 1))) # refers to 'Species'
stopifnot(inherits(x, "glht"), length(coef(x)) == 1)
if (do.plot)
  plot_with_margin(x, c(3, 20, 3, 2), main = "Species")

# comparison of only A01 - A04 against each other, Tukey style
# note that the left side of the model is set automatically
(x <- opm_mcp(vaas_4[, , 1:4],
  model = ~ Well + Species, m.type = "lm",
  linfct = c(Tukey = 1))) # the number refers to 'Well'
stopifnot(inherits(x, "glht"), length(coef(x)) == 6)
if (do.plot)
  plot_with_margin(x, c(3, 18, 3, 2), main = "Tukey, A01 - A04")

# Dunnett-type comparison of selected wells
(x <- opm_mcp(vaas_4[, , 1:4], model = ~ Well,
  m.type = "lm", linfct = c(Dunnett = 1)))
stopifnot(inherits(x, "glht"), length(coef(x)) == 3)
if (do.plot)
  plot_with_margin(x, c(3, 20, 3, 2), main = "Dunnett, A01 vs. A02 - A04")
# by default 'Dunnett' uses first level as reference

# Dunnett-type comparison with selected control-group
(x <- opm_mcp(vaas_4[, , 1:5], output = "mcp", model = ~ Well,
  linfct = c(`Dunnett.A05 (D-Cellobiose)` = 1)))
if (do.plot)
  plot_with_margin(x, c(3, 20, 3, 2), main = "Dunnett, vs. A05")

# manually defined contrast matrix
(contr <- opm_mcp(vaas_4[, , 1:4], linfct = c(Tukey = 1),
  model = ~ Well, output = "contrast")) # create template, Tukey style
contr <- contr$Well[c(1:3, 6), ] # select comparisons of interest
(x <- opm_mcp(vaas_4[, , 1:4],
  model = ~ Well, m.type = "lm", linfct = contr)) # run tests
if (do.plot)
  plot_with_margin(x, c(3, 20, 3, 2), main = "My own contrasts")

# joining of selected metadata using pseudofunction J
(x <- opm_mcp(vaas_4[, , 1:4], model = ~ J(Well + Species),
```

```
  linfct = c(Dunnett = 1), full = FALSE)) # use short well names
if (do.plot)
  plot_with_margin(x, c(3, 22, 3, 2), main = "Dunnett, Well/Species joined")

# comparing wells pairwise regarding the tested species
(x <- opm_mcp(vaas_4[, , 1:4], model = ~ J(Well + Species),
  linfct = c(Pairs.Well = 1), full = FALSE)) # use short well names
if (do.plot)
  plot_with_margin(x, c(3, 22, 3, 2),
    main = "Wells compared between species")
# i.e. 'Pairs.Well' means 'Pairs' type of comparison for each 'Well'
# separately within a joined factor (the first one in 'model', hence
# 'c(Pairs.Well = 1)', with '1' referring to the elements of 'model').

# pairwise comparison of Species regarding the tested strains
xx <- c(vaas_4, vaas_4) # temporary test data
(x <- opm_mcp(xx[, , 1:4], model = ~ J(Strain + Species),
  linfct = c(Pairs.Species = 1), full = FALSE)) # use short well names
if (do.plot)
  plot_with_margin(x, c(3, 22, 3, 2),
    main = "Strains compared within species")
# i.e. 'Pairs.Species' means 'Pairs' type of comparison for each 'Species'
# separately within a joined factor (the first one in 'model', hence
# 'c(Pairs.Species = 1)', with '1' referring to the elements of 'model').

## one could check the number of calculated tests as follows:
#if (nrow(confint(result)$confint) > 20L)
#  warning("number of performed comparisons exceeds 20")

## data-frame method (usually unnecessary to directly apply it)
x <- extract(vaas_4, as.labels = list("Species", "Strain"), subset = "A",
  dataframe = TRUE)

# without the tests, returning the converted data frame
head(y <- opm_mcp(x, output = "data", model = list("Species", "Strain")))
stopifnot(is.data.frame(y), dim(y) == c(384, 5)) # same result as above

# now with conducting the tests
(y <- opm_mcp(x, model = "Species", m.type = "lm",
  linfct = c(Dunnett = 1)))
stopifnot(inherits(y, "glht"), length(coef(y)) == 1)
if (do.plot)
  plot_with_margin(y, c(3, 20, 3, 2), main = "Species (from data frame)")

# testing for subsets of object
(y <- opm_mcp(subset(x, x$Species == "Escherichia coli"),
  linfct = c(Dunnett = 1), model = "Strain", m.type = "lm"))
stopifnot(inherits(y, "glht"), length(coef(y)) == 1)
if (do.plot)
  plot_with_margin(y, c(3, 15, 3, 2), main = "Dunnett (from data frame)")

}
```

opm_opt *Global* **opm** *options*

Description

Get and set global **opm** options.

Usage

```
  ## S4 method for signature 'character'
opm_opt(x)
  ## S4 method for signature 'list'
opm_opt(x)
  ## S4 method for signature 'missing'
opm_opt(x, ...)
```

Arguments

x	Character scalar or list or missing. If not given, all current settings are returned (as a named list). If x is a character scalar, it is used for querying for a single value. If x is a list, it is expected to contain key-value pairs that can be set. In that case, it is an error if a key is unknown, i.e. it is an error to use a name that is not already contained (**opm** would never query for it anyway). It is also illegal to attempt set novel values whose classes are not identical to, or derived from, the classes of the old value. Further, it is illegal to set a zero-length value.
...	Optional arguments. If x is missing, these arguments are concatenated into a list and used as if x was given as a list (see above). That is, the argument names are used as the keys for setting values. This is usually easier than working with a list.

Details

The following keys can be used with the following kinds of values:

colors Default colour set used by the OPMS method of xy_plot and other plotting functions.

color.borders Character vector with default colour borders between which level_plot interpolates to obtain a colour palette.

comb.key.join Used by functions that support combination of metadata entries converted to dataframe columns immediately after their selection. Sets the character string that is used when joining old names to new name. Should normally only be a single character.

comb.value.join Used by functions that support combination of metadata entries converted to dataframe columns immediately after their selection. Sets the character string that is used when joining old values to new values. Should normally only be a single character; must be a single character when used by opm_mcp.

contrast.type Character scalar indicating the default type of contrast used by opm_mcp.

css.file Character scalar. Default CSS file linked by `phylo_data` when producing HTML output. Ignored if empty.

csv.keys Character vector with names of entries of `csv_data` be used by `include_metadata`. Should be kept a subset of `opm_opt("csv.selection")`.

csv.selection Character vector with names of entries of `csv_data` (must be a valid 'keys' argument) to be extracted by `collect_template`.

curve.param Character scalar. Default 'subset' argument of `extract` and the plotting functions.

disc.param Character scalar. Default 'subset' argument of `do_disc`. It is usually not advisable to change it.

digits Integer scalar. Number of digits used by some functions generating output text.

file.encoding Character scalar. Character encoding in input files as assumed by `read_opm`.

file.split.tmpl Character scalar. Template used as 'format' argument by `split_files`.

gen.iii Character scalar indicating whether `read_opm` and other IO functions based on it automatically convert to this plate type. If empty, nothing is changed.

group.name Character scalar used as column name for trivial groups (either all items in the same group or each item in a group of its own) created by `extract`.

heatmap.colors Colour palette used by `heat_map`.

html.attr Used by `phylo_data` for automatically creating HTML 'title' and 'class' attributes.

input.try.order Integer vector indicating the preferred order when trying to read CSV files with `read_single_opm`. See there for details.

key.join Used by `metadata` and some other functions that must be in sync with it for joining metadata keys used in nested queries (because the resulting object is 'flat').

machine.id Integer scalar that can be used for identifying an OmniLog® instrument. Useful for `collect_template` if several such machines are in use.

max.chars Integer scalar used when abbreviating full substrate names. See `wells` for an example.

min.mode Used when making discretisation results uniform within a group. The minimum proportion the most frequent value much reach to be used for representing all values (if less, frequent, `NA` is used). Must be a numeric scalar between 0 and 1.

phylo.fmt Character scalar indicating the default output format used by `phylo_data`.

split Character scalar indicating the default splitting characters used by `separate`.

strict.OPMD Logical scalar indicating whether `OPMD` objects can only be created if the discretised data are consistent with the parameter from which they have been estimated.

threshold Numeric scalar indicating the default threshold used by `annotated`.

time.zone Character scalar indicating the time zone to be used when parsing `setup_time` entries. This is relevant for `merge`, which by default attempts to sort by parsed setup times

time.fmt Character vector indicating the time formats used for parsing the `setup_time` entries (in the given order). Also relevant for `merge` by default. It is advisable to put the more specific formats to the front because otherwise information such as an 'AM' or 'PM' indication might be lost. A wrong format might well match a given entry, causing **opm** to misinterpret the time or even the date.

For parameter names used by **opm** that cannot be modified by the user see `param_names`.

Value

List or atomic vector. If one to several values are set, the previous entries are returned invisibly.

See Also

base::options base::getOption

Other auxiliary-functions: separate

Examples

```
# fetching a value
(digits <- opm_opt("digits"))
stopifnot(digits == 4)

# setting a value; previous value is returned as list
(old.opts <- opm_opt(digits = 5L))
stopifnot(is.list(old.opts), length(old.opts) == 1L)
stopifnot(old.opts$digits == 4)

# fetching the value again: should now be changed
(digits <- opm_opt("digits"))
stopifnot(digits == 5)

# resetting the value
(old.opts <- opm_opt(old.opts))
stopifnot(is.list(old.opts), length(old.opts) == 1L)
stopifnot(old.opts$digits == 5)
(digits <- opm_opt("digits"))
stopifnot(digits == 4)
```

---

parallelplot *Parallel plot*

Description

Customised plotting of estimated curve parameter values from single or multiple PM plates using `parallelplot` from the **lattice** package with some adaptations likely to be useful for OmniLog® data.

Usage

```
  ## S4 method for signature 'NULL,OPMX'
parallelplot(x, data, ...)
  ## S4 method for signature 'OPMX,ANY'
parallelplot(x, data, groups = 1L,
  panel.var = NULL, pnames = param_names(), col = opm_opt("colors"),
  strip.fmt = list(), striptext.fmt = list(), legend.fmt = list(),
  legend.sep = " ", draw.legend = TRUE, space = "top", ...)
```

```
  ## S4 method for signature 'OPMX,missing'
parallelplot(x, data, ...)
  ## S4 method for signature 'formula,OPMX'
parallelplot(x, data, ...)
  ## S4 method for signature 'missing,OPMX'
parallelplot(x, data, ...)
  ## S4 method for signature 'vector,OPMX'
parallelplot(x, data, ...)
```

Arguments

x	An OPMA or OPMS object with aggregated data. This and the following argument can swap their places.
data	Any kind of object that can be used for selecting metadata. Either NULL, a character vector, a list of character vectors or a formula indicating which metadata should be included and used to determine the shape of the plot. The next argument by default accesses the first metadata entry. If numeric, panel.var also accesses the results from applying data. Most flexibility is available if data is a formula. For instance, as usual the J pseudo-function can be used to join metadata entries. Further, if a left part is present, this can indicate the parameters that should be plotted on the Y-axes (in place of the pnames argument; see below for further details). As usual, the right part of the formula states the meta-information to be included.
groups	Character or numerical scalar determining which metadata entry or other information, such as the well indexes, (see the examples) is used for assigning colours to the curves. If a numeric scalar, it refers to the position of the (potentially merged) metadata entry within data. If that argument were empty, a numeric groups argument would be ignored. Empty groups arguments are always ignored; the (constant) plate type is then used for creating a header.
panel.var	Character or numeric vector indicating which metadata entry or other information, such as the well indexes, (see the examples) is used for creating sub-panels. If a numeric vector, it refers to the position of the (potentially merged) metadata entry within data. If that argument were empty, a numeric panel.var argument would be ignored.
pnames	Character vector or formula to select the curve parameters for plotting. It has to comprise at least two of the names given by param_names(). If explicitly provided, this argument overrules the left side, if any, of a formula given as data argument. (But the left side, if any, of such a formula would overrule the default.)
col	Character or numerical scalar or vector. This and the following arguments work like the eponymous arguments of xy_plot. See there for details.
strip.fmt	List.
striptext.fmt	List.
legend.fmt	List.
legend.sep	Character scalar.
draw.legend	Logical scalar.

space	Character scalar.
...	Optional arguments passed to parallelplot from the **lattice** package.

Details

The main application of this function is to include all four estimated curve parameters into a single comprehensive overview. This assists in addressing questions such as

- Are there any consistent patterns of individual curves that may be explained by specific class membership? For instance, which curve parameter best reflects the origin of the tested strains?
- Are there any patterns of individual curves with unexpected deviations? For instance, do differences between experimental repetitions occur?

Value

An object of class 'trellis'. See xyplot from the **lattice** package for details.

Author(s)

Lea A.I. Vaas

References

Sarkar, D. 2008 *Lattice: Multivariate Data Visualization with R*. New York: Springer, 265 p.

See Also

lattice::xyplot lattice::parallelplot

Other plotting-functions: ci_plot, heat_map, level_plot, radial_plot, summary, xy_plot

Examples

```
## OPM objects

parallelplot(vaas_1)
parallelplot(vaas_1, data = list("Species", "Strain"))
# ... no effect on selection but on header

# value of 'groups' not found in the data: per default no metadata are used
x <- try(parallelplot(vaas_1, groups = "Species"), silent = TRUE)
stopifnot(inherits(x, "try-error"))
# same problem: metadata selected but 'groups' is not contained
x <- try(parallelplot(vaas_1, data = list("Species", "Strain"),
  groups = "missing"), silent = TRUE)
stopifnot(inherits(x, "try-error"))
# ... thus it is safer to use a positional 'groups' argument

## OPMS objects

# per default metadata are ignored
parallelplot(vaas_4[, , 1:10])
```

```
# otherwise selecting metadata is as usual
parallelplot(vaas_4[, , 1:10], data = ~ J(Species, Strain))
parallelplot(vaas_4[, , 1:10], data = list("Species", "Strain"))

# value of 'groups' not found in the data: per default no metadata are used
x <- try(parallelplot(vaas_4[, , 1:10], groups = "Species"), silent = TRUE)
stopifnot(inherits(x, "try-error"))
# now 'groups' is all present but not a character scalar
x <- try(parallelplot(vaas_4[, , 1:10], data = list("Species", "Strain"),
  groups = c("Strain", "Species")), silent = TRUE)
stopifnot(inherits(x, "try-error"))
# here 'groups' is positional but beyond the last element
x <- try(parallelplot(vaas_4[, , 1:10], data = list("Species", "Strain"),
  groups = 3), silent = TRUE)
stopifnot(inherits(x, "try-error"))

# 'groups' and 'panel.var' arguments that work
parallelplot(vaas_4[, , 1:10], data = ~ J(Species, Strain),
  panel.var = "Species", groups = "Strain")
parallelplot(vaas_4[, , 1:10], data = "Species", panel.var = "Species",
  groups = NULL)
parallelplot(vaas_4[, , 1:10], data = list("Species", "Strain"),
  panel.var = "Species")

# use of non-metadata information: here the names of the wells
parallelplot(vaas_4[, , 1:10], data = "Species", panel.var = "Well",
  groups = "Species")

# selection of parameters via 'pnames'
parallelplot(vaas_4[, , 1:10], pnames = ~ A + AUC + mu,
  data = ~ Species + Strain, panel.var = "Species",
  col = c("black", "red"), groups = "Species")
x <- try(parallelplot(vaas_4[, , 1:10], pnames = "A",
  data = ~ Species + Strain, panel.var = "Species",
  col = c("black", "red"), groups = "Species"), silent = TRUE)
stopifnot(inherits(x, "try-error")) # => at least two 'pnames' needed

# selecting the parameters via the left side of a 'data' formula
parallelplot(vaas_4[, , 1:10], data = A + AUC ~ J(Species, Strain))
parallelplot(vaas_4[, , 1:10], data = A + AUC ~ J(Species, Strain),
  groups = "Species")

# 'pnames' explicitly given => left side of formula ignored
parallelplot(vaas_4[, , 1:10], data = A + AUC ~ J(Species, Strain),
  pnames = c("A", "mu", "AUC"), groups = "Species")

# again: at least two 'pnames' needed
x <- try(parallelplot(vaas_4[, , 1:10], data = AUC ~ J(Species, Strain),
  groups = "Species"), silent = TRUE)
stopifnot(inherits(x, "try-error"))
```

phylo_data *Export phylogenetic data*

Description

Create entire character matrix (include header and footer) in a file format suitable for exporting phylogenetic data. Return it or write it to a file. This function can also produce HTML tables and text paragraphs suitable for displaying PM data in taxonomic journals such as IJSEM.

Usage

```
  ## S4 method for signature 'OPMD_Listing'
phylo_data(object,
    html.args = html_args(), run.tidy = FALSE)
  ## S4 method for signature 'OPMS_Listing'
phylo_data(object,
    html.args = html_args(), run.tidy = FALSE)
  ## S4 method for signature 'XOPMX'
phylo_data(object, as.labels,
    subset = param_names("disc.name"), sep = " ", extract.args = list(),
    join = TRUE, discrete.args = list(range = TRUE, gap = TRUE), ...)
  ## S4 method for signature 'data.frame'
phylo_data(object, as.labels = NULL,
    subset = "numeric", sep = " ", ...)
  ## S4 method for signature 'matrix'
phylo_data(object,
    format = opm_opt("phylo.fmt"), outfile = "", enclose = TRUE, indent = 3L,
    paup.block = FALSE, delete = c("none", "uninf", "constant", "ambig"),
    join = FALSE, cutoff = 0, digits = opm_opt("digits"),
    comments = comment(object), html.args = html_args(),
    prefer.char = format == "html", run.tidy = FALSE, ...)
```

Arguments

object	Data frame, numeric matrix or OPMS or MOPMX object (with aggregated values). Currently only 'integer', 'logical', 'double' and 'character' matrix content is supported. The data-frame and OPMS methods first call extract and then the matrix method. The methods for OPMD_Listing and OPMS_Listing objects can be applied to the results of listing.
format	Character scalar determining the output format, either epf (Extended PHYLIP Format), nexus, phylip, hennig or html. If NEXUS or 'Hennig' format is chosen, a non-empty comment attribute will be output together with the data (and appropriately escaped). In case of HTML format, a non-empty comment yields the title of the HTML document. EPF or 'extended PHYLIP' is sometimes called 'relaxed PHYLIP'. The main difference between EPF and PHYLIP is that the former can use labels with more

than ten characters, but its labels must not contain whitespace. (These adaptations are done automatically with safe_labels.)

outfile — Character scalar. If a non-empty character scalar, resulting lines are directly written to this file. Otherwise, they are returned.

enclose — Logical scalar. Shall labels be enclosed in single quotes? Ignored unless format is 'nexus'.

indent — Integer scalar. Indentation of commands in NEXUS format. Ignored unless format is 'nexus' (and a matter of taste anyway).

paup.block — Logical scalar. Append a PAUP* block with selected (recommended) default values? Has no effect unless 'nexus' is selected as 'format'.

delete — Character scalar with one of the following values:

uninf Columns are removed which are either constant (in the strict sense) or are columns in which some fields contain polymorphisms, and no pairs of fields share no character states.

ambig Columns with ambiguities (multiple states in at least one single field) are removed.

constant Columns which are constant in the strict sense are removed.

delete is currently ignored for formats other than HTML, and note that columns become rows in the final HTML output.

join — Logical scalar, vector or factor. Unless FALSE, rows of object are joined together, either according to the row names (if join is TRUE), or directly according to join. This can be used to deal with measurements repetitions for the same organism or treatment.

cutoff — Numeric scalar. If joining results in multiple-state characters, they can be filtered by removing all entries with a relative frequency less than 'cutoff'. Makes not much sense for non-integer numeric data.

digits — Numeric scalar. Used for rounding, and thus ignored unless object is of mode 'numeric'.

comments — Character vector. Comments to be added to the output (as title if HTML is chosen). Ignored if the output format does not allow for comments. If empty, a default comment is chosen.

html.args — List of arguments used to modify the generated HTML. See html_args for the supported list elements and their meaning.

prefer.char — Logical scalar indicating whether or not to use NA as intermediary character state. Has only an effect for 'logical' and 'integer' characters. A warning is issued if integers are not within the necessary range, i.e. either 0 or 1.

run.tidy — Logical scalar. Filter the resulting HTML through the Tidy program? Ignored unless format is html. Otherwise, if TRUE, it is an error if the Tidy executable is not found.

as.labels — Vector of data-frame indexes or OPMS metadata entries. See extract.

sep — Character scalar. See extract.

subset — Character scalar. For the OPMS method, passed to the OPMS method of extract. For the data-frame method, a selection of column classes to extract.

extract.args	Optional list of arguments passed to that method.
discrete.args	Optional list of arguments passed from the OPMS method to discrete. If set to NULL, discretisation is turned off. Ignored if precomputed discretised values are chosen by setting subset to param_names("disc.name").
...	Optional arguments passed between the methods (i.e., from the other methods to the matrix method) or to hwrite from the **hwriter** package. Note that 'table.summary' is set via html.args and that 'page', 'x' and 'div' cannot be used.

Details

Exporting PM data in such formats allows one to either infer trees from the data under the maximum-likelihood and/or the maximum-parsimony criterion, or to reconstruct the evolution of PM characters on given phylogenetic trees, or to nicely display the data in HTML format.

For exporting NEXUS format, the matrix should normally be converted beforehand by applying discrete. Exporting HTML is optimised for data discretised with gap set to TRUE. For other data, the character.states argument should be modified, see html_args. The hennig (Hennig86) format is the one used by TNT; it allows continuous characters to be analysed as such. Regarding the meaning of 'character' as used here, see the 'Details' section of discrete.

The generated HTML is guaranteed to produce neither errors nor warnings if checked using the Tidy program. It deliberately contains no formatting instructions but a rich annotation with 'class' attributes which allows for CSS-based formatting. This annotation includes the naming of all sections and all kinds of textual content. Whether the characters show differences between at least one organism and the others is also indicated. For the CSS files that come with the package, see the examples below and opm_files.

Value

Character vector, each element representing a line in a potential output file, returned invisibly if outfile is given.

References

Berger, S. A., Stamatakis, A. 2010 Accuracy of morphology-based phylogenetic fossil placement under maximum likelihood. *8th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-10).* Hammamet, Tunisia [analysis of phenotypic data with RAxML].

Felsenstein, J. 2005 PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author. Seattle: University of Washington, Department of Genome Sciences [the PHYLIP program].

Goloboff, P.A., Farris, J.S., Nixon, K.C. 2008 TNT, a free program for phylogenetic analysis. *Cladistics* **24**, 774–786 [the TNT program].

Goloboff, P.A., Mattoni, C., Quinteros, S. 2005 Continuous characters analysed as such. *Cladistics* **22**, 589–601.

Maddison, D. R., Swofford, D. L., Maddison, W. P. 1997 Nexus: An extensible file format for systematic information. *Syst Biol* **46**, 590–621 [the NEXUS format].

Stamatakis, A. 2006 RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models *Bioinformatics* **22**, 2688–2690. [the RAxML program].

Swofford, D. L. 2002 *PAUP*: Phylogenetic Analysis Using Parsimony (*and Other Methods), Version 4.0 b10*. Sunderland, Mass.: Sinauer Associates [the PAUP* program].

http://ijs.sgmjournals.org/ [IJSEM journal]

http://tidy.sourceforge.net/ [HTML Tidy]

See Also

base::comment base::write hwriter::hwrite

Other phylogeny-functions: html_args, safe_labels

Examples

```
# simple helper functions
echo <- function(x) write(substr(x, 1, 250), file = "")
is_html <- function(x) is.character(x) &&
  c("<html>", "<head>", "<body>", "</html>", "</head>", "</body>") %in% x
longer <- function(x, y) any(nchar(x) > nchar(y)) &&
  !any(nchar(x) < nchar(y))

## examples with a dummy data set
x <- matrix(c(0:9, letters[1:22]), nrow = 2)
colnames(x) <- LETTERS[1:16]
rownames(x) <- c("Ahoernchen", "Behoernchen") # Chip and Dale in German

# EPF is a comparatively restricted format
echo(y.epf <- phylo_data(x, format = "epf"))
stopifnot(is.character(y.epf), length(y.epf) == 3)
stopifnot(identical(y.epf, phylo_data(as.data.frame(x), subset = "factor",
  format = "epf")))

# PHYLIP is even more restricted (shorter labels!)
echo(y.phylip <- phylo_data(x, format = "phylip"))
stopifnot((y.epf == y.phylip) == c(TRUE, FALSE, FALSE))

# NEXUS allows for more content; note the comment and the character labels
echo(y.nexus <- phylo_data(x, format = "nexus"))
nexus.len.1 <- length(y.nexus)
stopifnot(is.character(y.nexus), nexus.len.1 > 10)

# adding a PAUP* block with (hopefully useful) default settings
echo(y.nexus <- phylo_data(x, format = "nexus", paup.block = TRUE))
stopifnot(is.character(y.nexus), length(y.nexus) > nexus.len.1)

# adding our own comment
comment(x) <- c("This is", "a test") # yields two lines
echo(y.nexus <- phylo_data(x, format = "nexus"))
stopifnot(identical(length(y.nexus), nexus.len.1 + 1L))

# Hennig86/TNT also includes the comment
echo(y.hennig <- phylo_data(x, format = "hennig"))
hennig.len.1 <- length(y.hennig)
```

```
stopifnot(is.character(y.hennig), hennig.len.1 > 10)

# without an explicit comment, the default one will be used
comment(x) <- NULL
echo(y.hennig <- phylo_data(x, format = "hennig"))
stopifnot(identical(length(y.hennig), hennig.len.1 - 1L))

## examples with real data and HTML

# setting the CSS file that comes with opm as default
opm_opt(css.file = opm_files("css")[[1]])

# see discrete() for the conversion and note the OPMS example below: one
# could also get the results directly from OPMS objects
x <- extract(vaas_4[, , 1:10], as.labels = list("Species", "Strain"),
  in.parens = FALSE)
x <- discrete(x, range = TRUE, gap = TRUE)
echo(y <- phylo_data(x, format = "html",
  html.args = html_args(organisms.start = "Strains: ")))
# this yields HTML with the usual tags, a table legend, and the table itself
# in a single line; the default 'organisms.start' could also be used
stopifnot(is_html(y))

# now with joining of the results per species (and changing the organism
# description accordingly)
x <- extract(vaas_4[, , 1:10], as.labels = list("Species"),
  in.parens = FALSE)
x <- discrete(x, range = TRUE, gap = TRUE)
echo(y <- phylo_data(x, format = "html", join = TRUE,
  html.args = html_args(organisms.start = "Species: ")))
stopifnot(is_html(y))
# Here and in the following examples note the highlighting of the variable
# (uninformative or informative) characters. The uninformative ones are those
# that are not constant but show overlap regarding the sets of character
# states between all organisms. The informative ones are those that are fully
# distinct between all organisms.

# 'OPMS' method, yielding the same results than above but directly
echo(yy <- phylo_data(vaas_4[, , 1:10], as.labels = "Species",
  format = "html", join = TRUE, extract.args = list(in.parens = FALSE),
  html.args = html_args(organisms.start = "Species: ")))
# the timestamps might differ, but otherwise the result is as above
stopifnot(length(y) == length(yy) && length(which(y != yy)) < 2)

# appending user-defined sections
echo(yy <- phylo_data(vaas_4[, , 1:10], as.labels = "Species",
  format = "html", join = TRUE, extract.args = list(in.parens = FALSE),
  html.args = html_args(organisms.start = "Species: ",
  append = list(section.1 = "additional text", section.2 = "more text"))))
stopifnot(length(y) < length(yy), length(which(!y %in% yy)) < 2)
# note the position -- there are also 'prepend' and 'insert' arguments

# effect of deletion
```

```
echo(y <- phylo_data(x, "html", delete = "none", join = FALSE))
echo(y.noambig <- phylo_data(x, "html", delete = "ambig", join = FALSE))
stopifnot(length(which(y != y.noambig)) < 2) # timestamps might differ
# ambiguities are created only by joining
echo(y <- phylo_data(x, "html", delete = "none", join = TRUE))
echo(y.noambig <- phylo_data(x, "html", delete = "ambig", join = TRUE))
stopifnot(longer(y, y.noambig))
echo(y.nouninf <- phylo_data(x, "html", delete = "uninf", join = TRUE))
stopifnot(longer(y, y.nouninf))
echo(y.noconst <- phylo_data(x, "html", delete = "const", join = TRUE))
stopifnot(longer(y.noconst, y.nouninf))

# getting real numbers, not discretised ones
echo(yy <- phylo_data(vaas_4[, , 1:10], as.labels = "Species",
  format = "html", join = TRUE, extract.args = list(in.parens = FALSE),
  subset = "A", discrete.args = NULL,
  html.args = html_args(organisms.start = "Species: ")))
stopifnot(is_html(yy), length(yy) == length(y) - 1) # no symbols list
# the highlighting is also used here, based on the following heuristic:
# if mean+/-2*sd does not overlap, the character is informative; else
# if mean+/-sd does not overlap, the character is uninformative; otherwise
# it is constant

# this can also be used for formats other than HTML (but not all make sense)
echo(yy <- phylo_data(vaas_4[, , 1:10], as.labels = "Species",
  format = "hennig", join = TRUE, extract.args = list(in.parens = FALSE),
  subset = "A", discrete.args = NULL))
stopifnot(is.character(yy), length(yy) > 10)

## 'OPMD_Listing' method
echo(x <- phylo_data(listing(vaas_1, NULL)))
stopifnot(is.character(x), length(x) == 1)
echo(x <- phylo_data(listing(vaas_1, NULL, html = TRUE)))
stopifnot(is.character(x), length(x) > 1)

## 'OPMS_Listing' method
echo(x <- phylo_data(listing(vaas_4, as.groups = "Species")))
stopifnot(is.character(x), length(x) == 2, !is.null(names(x)))
echo(x <- phylo_data(listing(vaas_4, as.groups = "Species", html = TRUE)))
stopifnot(is.character(x), length(x) > 2, is.null(names(x)))
```

`plates` *Get available plates or apply function to them*

Description

Get all `plates` contained in an `OPMS` object or a list, or create a list containing a single `OPM` object as element, or apply a function to a collection of `OPM` objects.

Usage

```
  ## S4 method for signature 'MOPMX'
oapply(object, fun, ...,
    simplify = TRUE)
  ## S4 method for signature 'OPM'
oapply(object, fun, ...,
    simplify = TRUE)
  ## S4 method for signature 'OPMS'
oapply(object, fun, ...,
    simplify = TRUE)

  ## S4 method for signature 'MOPMX'
plates(object)
  ## S4 method for signature 'WMD'
plates(object)
  ## S4 method for signature 'WMDS'
plates(object)
  ## S4 method for signature 'list'
plates(object)
```

Arguments

object	List, OPM, OPMS or MOPMX object.
fun	A function. Should accept an OPM object as first argument.
...	Optional other arguments passed to fun.
simplify	Logical scalar. If FALSE, the result is a list. If TRUE, it is attempted to simplify this to a vector, matrix or OPMS object (if the result is a list of OPM or OPMA objects). If this is impossible, a list is returned. The MOPMX method tries creating a MOPMX object again after removing NULL values, if any.

Details

The list method of plates traverses the input recursively and skips all objects of other classes than OPM. See also opms, which is somewhat similar but more flexible.

oapply applies a function to all OPM objects within an OPMS object. Optionally it simplifies the result to an OPMS object if possible, or other structures simpler than a list. The OPM method of simply applies fun once (to object).

Value

For plates, a list of OPM objects (may be empty instead if object is a list). The result of oapply depends on fun and simplify: a list, vector, matrix or OPMS object are possible outcomes.

See Also

base::list base::as.list base::sapply

Other conversion-functions: as.data.frame, extract, extract_columns, flatten, merge, opmx, rep, rev, sort, split, to_yaml, unique

Examples

```
# plates(), 'OPM' method
summary(x <- plates(vaas_1)) # => list of OPM objects
stopifnot(is.list(x), length(x) == 1L, sapply(x, inherits, what = "OPM"))

# plates(), 'OPMS' method
summary(x <- plates(vaas_4)) # => list of OPM objects
stopifnot(is.list(x), length(x) == 4L, sapply(x, inherits, what = "OPM"))

# plates(), list method
x <- list(vaas_1, letters, vaas_4, 1:10)
summary(x <- plates(x)) # => list of OPM objects
stopifnot(is.list(x), length(x) == 5, sapply(x, inherits, what = "OPM"))

## oapply()
summary(x <- oapply(vaas_4, identity)) # trivial
stopifnot(identical(x, vaas_4))
summary(x <- oapply(vaas_4, identity, simplify = FALSE)) # => yields list
stopifnot(is.list(x), length(x) == 4, sapply(x, class) == "OPMD")
```

---

plate_type *Plate type displayed or modified, registered or deleted*

Description

Get the type of the OmniLog® plate used in the measuring, normalise plate-type names, display known names, or modify the plate type after inputting the plate data. Alternatively, register or remove user-defined plate types.

Usage

```
  ## S4 method for signature 'MOPMX'
gen_iii(object, ...)
  ## S4 method for signature 'OPM'
gen_iii(object, to = "gen.iii")
  ## S4 method for signature 'OPMS'
gen_iii(object, ...)

  ## S4 method for signature 'MOPMX'
plate_type(object, ..., normalize = FALSE,
    subtype = FALSE)
  ## S4 method for signature 'OPM'
plate_type(object, ..., normalize = FALSE,
    subtype = FALSE)
  ## S4 method for signature 'OPMS'
plate_type(object, ...)
  ## S4 method for signature 'OPM_DB'
plate_type(object, ..., normalize = FALSE,
```

```
    subtype = FALSE)
  ## S4 method for signature 'character'
plate_type(object, full = FALSE,
    in.parens = TRUE, max = opm_opt("max.chars"), clean = TRUE,
    brackets = FALSE, word.wise = FALSE, paren.sep = " ", downcase = FALSE,
    normalize = TRUE, subtype = FALSE)
  ## S4 method for signature 'factor'
plate_type(object, ...)
  ## S4 method for signature 'logical'
plate_type(object, ...)
  ## S4 method for signature 'missing'
plate_type(object, ...)

  ## S4 method for signature 'character'
register_plate(object, ...)
  ## S4 method for signature 'list'
register_plate(object, ...)
  ## S4 method for signature 'missing'
register_plate(object, ...)
```

Arguments

object	OPM, OPMS or MOPMX object, or character vector of original plate name(s), or factor, or logical scalar. If missing, the function displays the plate types **opm** knows about. If a logical scalar, the behaviour is the same, but optionally restricted to the user-defined or the **opm**-defined plate types. For register_plate, object is either missing (the easiest way to apply the function), causing the arguments within ... (which should be named) to be used, if any, a named list of NULL values or character vectors, or a character vector whose elements are interpretable as file names. See below for details on the input format.
full	Logical scalar. If TRUE, add (or replace by) the full name of the plate type (if available); otherwise, return it as-is.
in.parens	Logical scalar. This and the five next arguments work like the eponymous ones of wells, but here are applied to the plate name. See there for details.
max	Numeric scalar.
clean	Logical scalar.
brackets	Logical scalar.
word.wise	Logical scalar.
paren.sep	Character scalar.
downcase	Logical scalar.
normalize	Logical scalar. Attempt to normalise the plate-type string before interpreting it?
subtype	Logical scalar. Keep the plate subtype indicator, if any? Only relevant for the character or factor method.

`to` Character scalar indicating the plate type. User-defined plate types must be given literally. For generation-III plates, use `gen.iii`; for the EcoPlate™, use `eco`; the remaining allowed values are only `sf.n2`, `sf.p2`, `an2`, `ff` and `yt`, but matching is case-insensitive.

A character vector of length greater than one can be passed to the `MOPMX` method, which recycles its `to` argument.

`...` Optional arguments passed between the methods. For `register_plate`, named arguments to be used if `object` is missing.

Details

The `OPM` and `OPMS` methods of `plate_type` are convenience methods for one of the more important entries of `csv_data` with additional options useful for creating plot titles.

The character method normalises the names of OmniLog® PM plates to the internally used naming scheme. Unrecognised names are returned unchanged. This needs not normally be called by the **opm** user but might be of interest.

Factors are treated by passing their levels through the character method.

If a logical scalar is given as `object` argument, `TRUE` restricts the output to user-defined plate types (an empty set by default), `FALSE` to the plate types that ship with **opm**, and `NA` shows all plates.

`gen_iii` change the plate type of an `OPM` object to 'Generation III' or another plate type. This is currently the only function to change plate names. It is intended for Generation-III or other plates that were not devised for the OmniLog instrument but can be run just like PM plates. Usually they will be annotated as some PM plate by the OmniLog® system. In contrast, input ID-mode plates are automatically detected (see `read_single_opm`). For this reason, `gen_iii` does not enable changes to PM plate types.

User-defined plate types are allowed but need the according prefix (which must currently case-insensitively match 'Custom:'), even though additional normalisation is done. It is an error to set a user-defined plate type that has not beforehand been registered with `register_plate`.

The actual spelling of the plate type used might (in theory) differ between distinct versions of **opm** but is internally consistent. It is an error to set one of the PM plate types or to assign an unknown plate type.

Two kinds of information can be registered for user-defined plates: the full name of the plate and/or the full names of the substrates for all well coordinates. Both kinds of information can be deleted again. In any case, the name of the argument within `...` or within `object` given as list must indicate the plate type. Normalisation is done, as well as adding the usual prefix for user-defined plates.

For registering a full plate name, an unnamed character scalar must be provided. For registering the mapping from well coordinates to substrate names, a named character vector must be provided with the names indicating the well coordinates and the elements indicating the according substrate names. Alternatively, a matrix or data frame can be provided that imitates that physical structure of the plate. That is, the rows are the plate rows (A, B, C, ...) and the columns are the plate columns (1, 2, 3, ...). If row or column names are used, they are honoured.

For deleting a user-defined plate, an empty value (such as `NULL` or an empty vector) must be provided. Deletion is done for both full plate names and mappings from well coordinates to substrate names. It is ignored whether or not this information had been registered beforehand.

Value

Character scalar in the case of the OPM and OPMS methods of plate_type, otherwise a character vector with the same length than object, or a corresponding factor. If object is not given, a character vector of normalised plate-type names.

gen_iii returns a novel OPMX object.

register_plate returns a logical vector whose values indicate whether information was registered or deleted and whose names are the normalised plate-type names.

See Also

base::strtrim base::abbreviate base::gsub

Other naming-functions: find_positions, find_substrate, listing, opm_files, param_names, select_colors, substrate_info, wells

Examples

```
## 'OPM' method
(x <- plate_type(vaas_1, full = FALSE))
(y <- plate_type(vaas_1, full = TRUE))
(z <- plate_type(vaas_1, full = TRUE, in.parens = FALSE))
# strings lengths differ depending on the selection
stopifnot(nchar(x) < nchar(y), nchar(z) < nchar(y))

## Not run:

# Splitting a list of 'OPM' objects according to the plate type is easy:
x <- split(x), sapply(x, plate_type))
# but see also opms() and read_opm(), which can do this internally

## End(Not run)

## 'OPMS' method
(xx <- plate_type(vaas_4, full = FALSE))
# plate type is guaranteed to be uniform within an OPMS object
stopifnot(identical(x, xx))

## character and factor methods

# Entirely unrecognized strings are returned as-is
(x <- plate_type(letters))
stopifnot(identical(x, letters))

# Something more realistic
(x <- plate_type(y <- c("PM1", "PM-11C", "PMM04-a"), subtype = TRUE))
stopifnot(x != y)

# Factors
(z <- plate_type(as.factor(y), subtype = TRUE))
stopifnot(is.factor(z), z == x) # same result after conversion
```

```
## 'missing' method
(x <- plate_type())
stopifnot(is.character(x), plate_type(vaas_1) %in% x)

## changing the plate type

# 'OPM' method
plate_type(copy <- gen_iii(vaas_1))
stopifnot(identical(vaas_1, copy)) # the data set already had that plate type
plate_type(copy <- gen_iii(vaas_1, "eco")) # which is wrong, actually
stopifnot(!identical(vaas_1, copy))

# 'OPMS' method
plate_type(copy <- gen_iii(vaas_4))
stopifnot(identical(vaas_4, copy)) # as above
plate_type(copy <- gen_iii(vaas_4, "eco"))
stopifnot(!identical(vaas_4, copy)) # as above

## registering plate types

# well map and full name of a plate can be simultaneously registered
register_plate(myplate = c(A01 = "Glucose", A02 = "Fructose"),
  myplate = "Simple fake test plate")
# note standardization of name
stopifnot("CUSTOM:MYPLATE" %in% plate_type(TRUE))
# queries can be done ignoring case differences
listing(wells(plate = "custom:myplate"))

# input/output of plate types
plate.file <- tempfile()
write(to_yaml(listing(wells(plate = "custom:myplate"))), plate.file)
register_plate(plate.file)
unlink(plate.file) # tidying up

# erasing this plate type again; this will delete well map and full name
register_plate(myplate = NULL)
stopifnot(!"CUSTOM:MYPLATE" %in% plate_type(TRUE))
```

`potato` *Potato cell-line growth data set*

Description

Example data set for analysing growth curves with **opm** containing manually entered fresh-mass and dry-mass measurements over time from three distinct potato (*Solanum tuberosum*) cell lines under several stress treatments.

Format

Data frame with 540 rows and six columns. The columns are:

Genotype Factor containing the names of the cell lines. See Vaas *et al.* (2013) for further details.

Treatment Factor describing the three applied stress conditions. There is also one level for the control.

Replicate Integer vector with the number of the replicate. There are five replicates per combination of 'Genotype' and 'Treatment'.

Time Integer vector containing the measurement times in days.

FM Integer vector containing the fresh-mass measurements in milligramme.

DM Integer vector containing the dry-mass measurements in milligramme.

Details

How to convert and analyse this data set is explained in the vignette on working with growth curves in **opm**.

References

Vaas, L. A. I., Marheine, M., Sikorski, J., Goeker, M., Schumacher, H.-M. 2013 Impacts of pr-10a overexpression at the molecular and the phenotypic level. *International Journal of Molecular Sciences* **14**: 15141–15166.

El-Banna, A., Hajirezaei, M. R., Wissing, J., Ali, Z., Vaas, L., Heine-Dobbernack, E., Jacobsen, H.-J., Schumacher, H.-M., Kiesecker, H. 2010 Over-expression of PR-10a leads to increased salt and osmotic tolerance in potato cell cultures. *Journal of Biotechnology* **150**: 277–287.

Sandford, S. A. 1995. Apples and Oranges – A Comparison. *Annals of Improbable Research* **1** (3).

Examples

```
## Not run:

# Calling this yielded a variable 'potato' containing the data. The opm
# package must be loaded beforehand using library().
data(potato)

## End(Not run)
```

---

radial_plot *Radial plot*

Description

A wrapper for `radial.plot` from the **plotrix** package with some adaptations likely to be useful for OmniLog® data.

Usage

```
  ## S4 method for signature 'OPMS'
radial_plot(object, as.labels,
    subset = opm_opt("curve.param"), sep = " ", extract.args = list(), ...)
  ## S4 method for signature 'data.frame'
radial_plot(object, as.labels,
    subset = "numeric", sep = " ", extract.args = list(), ...)
  ## S4 method for signature 'matrix'
radial_plot(object, as.labels = NULL,
    subset = TRUE, sep = " ", extract.args = list(), rp.type = "p",
    radlab = FALSE, show.centroid = TRUE, show.grid.labels = 1, lwd = 3,
    mar = c(2, 2, 2, 2), line.col = opm_opt("colors"), draw.legend = TRUE,
    x = "bottom", y = NULL, xpd = TRUE, pch = 15, legend.args = list(),
    group.col = FALSE, point.symbols = 15, point.col = opm_opt("colors"),
    poly.col = NA, main = paste0(as.labels, sep = sep), ...)
```

Arguments

object	OPMS object (with aggregated values) to be plotted. Alternatively a data frame or a numeric matrix, but the methods for these objects are rarely needed.
as.labels	For the OPMS method, a metadata selection defining the data to be joined and used as row names, ultimately yielding the legend. If NULL or empty, ignored. See extract for details. For the data-frame method, a selection of columns used in the same way.
subset	For the OPMS method, a character scalar passed to the OPMS method of extract. The data-frame method uses this to select the columns; currently only the default makes sense. For the matrix method, a selection of columns to be plotted.
sep	Character scalar determining how to join row names. See extract for details.
extract.args	Optional list of arguments passed to the OPMS method of extract. Currently ignored by the other methods.
rp.type	Character scalar. Among the possible values 'p' for polygons, 's' for symbols and 'r' for radial lines, the latter is not recommended. These and the following arguments are passed to plotrix::radial.plot. See there for details.
radlab	Logical scalar. Rotation of the outer labels to a radial orientation might safe some space in the graphic.
show.centroid	Logical scalar.
show.grid.labels	Logical scalar. Indicates whether and where display labels for the grid are shown.
lwd	Numeric scalar for the line width.
mar	Numeric vector of length 4 determining the margins of the plot.
line.col	Character or numeric vector for determining the line colour.
point.symbols	Like the following arguments, passed to radial.plot from the **plotrix** package. See there for details. Explicitly provided here to silence some radial.plot warnings occurring as of R 3.0.0.

point.col	Indicates the colour(s) of the symbols.
poly.col	Indicates the colour for filling the drawn polygons, if any. Use NA for no fill (recommended).
group.col	Logical scalar indicating whether or not wells from plates that belong to the same group shall have the same colour.
main	The main title of the plot.
...	Optional other arguments passed to radial.plot from the **plotrix** package.
draw.legend	Logical scalar. Whether to draw a legend. Ignored unless object has row names (because these are used to generate the description).
x	Legend position, passed to legend from the **graphics** package. Ignored unless draw.legend is TRUE.
y	Optional Second legend coordinate. Also passed to that function.
xpd	Logical scalar. Also passed to that function.
pch	Integer scalar. Also passed to that function.
legend.args	List of optional other arguments passed to that function.

Details

The default positioning of the legend is not necessarily very useful, but suitable combinations of margin, x and y can be found for given data sizes. Plotting entire plates usually makes not much sense (see the examples).

The data frame and OPMS methods extract a numeric matrix from a given data frame or OPMS object and pass the result to the matrix method.

Value

A vector with the row names of object as names and the corresponding colours as values, equivalent to the legend; NULL if no row names are present.

Author(s)

Lea A.I. Vaas and Markus Goeker

See Also

plotrix::radial.plot graphics::legend

Other plotting-functions: ci_plot, heat_map, level_plot, parallelplot, summary, xy_plot

Examples

```
data("vaas_4")

## 'OPMS' method
# Note that this visualization is not useful when applied to too many wells.
# Thus, we here use only a subset of the wells for plotting.
(y <- radial_plot(vaas_4[, , 1:5], as.labels = list("Species", "Strain"),
```

```
  main = "Test", x = 200, y = 200))

# with some fine tuning; note the centroids
(y <-radial_plot(vaas_4[, , 1:5], as.labels = list("Species", "Strain"),
  main = "Test", x = 200, y = 200, rp.type = "s", show.centroid = TRUE))

# with the same colour for members of the same group
(xy <-radial_plot(vaas_4[, , 1:5], as.labels = list("Species"),
  group.col = TRUE, main = "Test", x = 200, y = 200, rp.type = "s",
  show.centroid = TRUE))

## Data-frame method (rarely needed)
x <- extract(vaas_4, as.labels = list("Species", "Strain"), dataframe = TRUE)
(yy <- radial_plot(x[, 1:8], as.labels = c("Species", "Strain"),
  main = "Test"))
stopifnot(identical(y, yy)) # should also yield the same picture than above
stopifnot(is.character(yy), names(yy) == paste(x$Species, x$Strain))

## Matrix method (rarely needed)
x <- extract(vaas_4, as.labels = list("Species", "Strain"))
(yy <- radial_plot(x[, 1:5], main = "Test"))
stopifnot(identical(y, yy)) # should also yield the same picture than above
stopifnot(is.character(yy), names(yy) == rownames(x))
```

`read_opm` *Read multiple* PM *files at once or read single* PM *file*

Description

Read OmniLog® or **opm** data file(s) in one of three possible formats: either new- or old-style OmniLog® CSV or **opm** YAML (including JSON) format. MicroStation™ CSV are also understood, as well as files compressed using `gzip`, `bzip2`, `lzma` or `xz`. (Files can be specifically excluded using `include` and/or `exclude`).

Usage

```
read_opm(names,
  convert = c("try", "no", "yes", "sep", "grp"),
  gen.iii = opm_opt("gen.iii"), include = list(), ...,
  demo = FALSE)

read_single_opm(filename)
```

Arguments

`names`	Character vector with names of files in one of the formats accepted by `read_single_opm`, or names of directories containing such files, or both; or convertible to such a vector. See the `include` argument of `read_opm` and `explode_dir` for how to select subsets from the input files or directories.

`filename` Character scalar, or convertible to such, with the obvious meaning.

`convert` Character scalar with one of the following values:

no Always return a list of `OPM` objects. (This list is rather a `MOPMX` object than a plain list.)

yes Convert to `NULL`, `OPM` object, or `OPMS` object, depending on the number of files read (0, 1, or more).

try Behave like 'yes' but do not result in an error message if conversion to `OPMS` is impossible; return a list (`MOPMX` object) in that case.

sep Return a nested list, each partial list (`MOPMX` object) containing `OPM` objects of the same plate type.

grp Also split into such contained lists but convert them to `OPMS` objects if more than one plate is encountered. An error is raised if this is impossible (in contrast to 'try'). Return a list (`MOPMX` object).

`gen.iii` Logical or character scalar. If `TRUE`, invoke `gen_iii` on each plate. This is automatically done with CSV input if the plate type is given as OTH (which is usually the case for plates run in ID mode). If a character scalar, it is used as the `to` argument of `gen_iii` to set other plate types unless it is empty.

`include` Pattern for selecting from the input files. The default value results in the output of `file_pattern`, which should be sufficient in most cases. See `explode_dir` for details on other possibilities.

`...` Optional further arguments passed to `explode_dir`.

`demo` Logical scalar. Do not read files, but print a vector with the names of the files that would be (attempted to) read, and return them invisibly?

Details

The expected CSV format is what is output by an OmniLog® instrument, one plate per file, or a MicroStation™ instrument, with one to many plates per file. Other formats, or OmniLog® files re-saved with distinct CSV settings, are not understood. For this reason, if any editing of the files was necessary at all, it is advisable to do this in an editor for plain text, not in a spreadsheet program.

Plates run in ID mode are automatically detected as such (their plate type is changed from OTH to the internally used spelling of 'Generation III'). A generation-III or other plate type can also be forced later on by using `gen_iii`.

It is **impossible** to read CSV files that contain more than one plate. For splitting old-style and new-style CSV files into one file per plate, see the example under `split_files`. In contrast, input YAML files can contain data from more than one plate. The format (which includes JSON) is described in detail under `batch_opm`.

For splitting lists of `OPM` objects according to the plate type, see `plate_type`, and consider the plate-type selection options of `opms`.

The order in which it is tried to read distinct formats of CSV files can be modified using the 'input.try.order' key of `opm_opt`. The value is an integer vector whose elements have the following meaning:

1. New-style OmniLog® CSV.
2. Old-style OmniLog® CSV.

3. MicroStation™ CSV.

For instance, `opm_opt(input.try.order = 2:1` would change the order in which OmniLog® formats are tried and deselect MicroStation™ files entirely. Negative indexes can be used, but non-negative values not within the range listed above will result in an error. If it known in advance which formats are (not) to be expected, subset creation or just changing the order can be used to accelerate data input.

Value

`read_opm` returns an `OPM` object (maybe `OPMA` in case of YAML input), or list (`MOPMX` object) of such objects, or `OPMS` object. If `demo` is `TRUE`, a character vector instead.

`read_single_opm` also returns an`OPM` object. In the case of YAML input, this might also be an `OPMA` object or a list of such objects, but **not** an `OPMS` object.

References

http://www.yaml.org/

http://www.json.org/

http://www.biolog.com/

See Also

utils::read.csv yaml::yaml.load_file

Other io-functions: `batch_collect`, `batch_opm`, `batch_process`, `clean_filenames`, `collect_template`, `explode_dir`, `file_pattern`, `glob_to_regex`, `split_files`, `to_metadata`

Examples

```
test.files <- opm_files("omnilog")
if (length(test.files) > 0) { # if the folder is found
  x <- read_opm(test.files, demo = TRUE) # check first what you would get
  stopifnot(identical(test.files, x))
  x <- read_opm(test.files[1:2]) # these two have the same plate type
  class(x)
  dim(x)
  summary(x)
  stopifnot(is(x, "OPMS"), identical(dim(x), c(2L, 384L, 96L)))
} else {
  warning("test files not found")
}
# This can be repeated for the other input test files. Instead of a several
# file names one can also provide a single one, one to several directory
# names, or mixture of file and directory names.

## Not run:

# Reading all files from the current working directory is also easy:
x <- read_opm(getwd())
# or
```

```
x <- read_opm(".")

## End(Not run)

# read_single_opm()
test.files <- opm_files("omnilog")
if (length(test.files) > 0) { # if the folder is found
  x <- read_single_opm(test.files[1]) # => 'OPM' object
  class(x)
  dim(x)
  summary(x)
  stopifnot(is(x, "OPM"), identical(dim(x), c(384L, 96L)))
} else {
  warning("test-file folder not found")
}
# this can be repeated for the other input test files
```

`run_kmeans` *Conduct k-means partitioning*

Description

Run a k-means partitioning analysis. This function is used by `discrete` in 'gap' mode to automatically determine the range of ambiguous data. If applied to such one-dimensional data, it uses an exact algorithm from the **Ckmeans.1d.dp** package.

Usage

```
  ## S4 method for signature 'matrix,numeric'
run_kmeans(object, k,
    cores = 1L, nstart = 10L, ...)
  ## S4 method for signature 'numeric,numeric'
run_kmeans(object, k,
    cores = 1L)
```

Arguments

`object`	Numeric vector or matrix.
`k`	Numeric vector. Number of clusters requested.
`nstart`	Numeric scalar. Ignored if 'Ckmeans.1d.dp' is called. Otherwise passed to `kmeans` from the **stats** package.
`cores`	Numeric scalar indicating the number of cores to use.
`...`	List of optional arguments passed to `kmeans` from the **stats** package.

Value

S3 object of class `kmeanss`, basically a named list of `kmeans` objects.

References

Wang, H., Song, M. 2011 Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming. *The R Journal* **3**, p. 29–33.

See Also

stats::kmeans Ckmeans.1d.dp::Ckmeans.1d.dp

Other kmeans-functions: `borders`, `calinski`, `hist.Ckmeans.1d.dp`, `hist.kmeans`, `hist.kmeanss`, `plot.kmeanss`, `to_kmeans`,

Examples

```
x <- as.vector(extract(vaas_4, as.labels = NULL, subset = "A"))
summary(x.km <- run_kmeans(x, k = 1:10)) # => 'kmeanss' object
stopifnot(inherits(x.km, "kmeanss"), length(x.km) == 10)
stopifnot(sapply(x.km, class) == "kmeans", names(x.km) == 1:10)
```

---

separate *Regularly split character vectors if possible*

Description

From a given set of splitting characters select the ones that split a character vector in a regular way, yielding the same number of parts for all vector elements. Then apply these splitting characters to create a matrix. The data frame method applies this to all character vectors (and optionally also all factors) within a data frame.

Usage

```
  ## S4 method for signature 'character'
separate(object, split = opm_opt("split"),
    simplify = FALSE, keep.const = TRUE, list.wise = FALSE,
    strip.white = list.wise)
  ## S4 method for signature 'data.frame'
separate(object, split = opm_opt("split"),
    simplify = FALSE, keep.const = TRUE, coerce = TRUE, name.sep = ".", ...)
  ## S4 method for signature 'factor'
separate(object, split = opm_opt("split"),
    simplify = FALSE, keep.const = TRUE, ...)
```

Arguments

`object`	Character vector to be split, or data frame in which character vectors (or factors) shall be attempted to be split, or factor.
`split`	Character vector or `TRUE`.

- If a character vector, used as container of the splitting characters and converted to a vector containing only non-duplicated single-character strings. For instance, the default `split` argument `".-_"` yields `c(".", "-", "_")`.
- If a vector of only empty strings or `TRUE`, strings with parts representing fixed-width fields are assumed, and splitting is done at whitespace-only columns. Beforehand, equal-length strings are created by padding with spaces at the right. After splitting in fixed-width mode, whitespace characters are trimmed from both ends of the resulting strings.

`simplify` Logical scalar indicating whether a resulting matrix with one column should be simplified to a vector (or such a data frame to a factor). If so, at least one matrix column is kept, even if `keep.const` is `FALSE`.

`keep.const` Logical scalar indicating whether constant columns should be kept or removed.

`coerce` Logical scalar indicating whether factors should be coerced to 'character' mode and then also be attempted to be split. The resulting columns will be coerced back to factors.

`name.sep` Character scalar to be inserted in the constructed column names. If more than one column results from splitting, the names will contain (i) the original column name, (ii) `name.sep` and (iii) their index, thus creating unique column names (if the original ones were unique).

`list.wise` Logical scalar. Ignored if `split` is `TRUE`. Otherwise, `object` is assumed to contain word lists separated by `split`. The result is a logical matrix in which the columns represent these words and the fields indicate whether or not a word was present in a certain item contained in `object`.

`strip.white` Logical scalar. Remove whitespace from the ends of each resulting character scalar after splitting? Has an effect on the removal of constant columns. Whitespace is always removed if `split` is `TRUE`.

`...` Optional arguments passed between the methods.

Details

This function is useful if information coded in the elements of a character vector is to be converted to a matrix or data frame. For instance, file names created by a batch export conducted by a some software are usually more or less regularly structured and contain content at distinct positions. In such situations, the correct splitting approach can be recognised by yielding the same number of fields from each vector element.

Value

Character matrix, its number of rows being equal to the length of `object`, or data frame with the same number of rows as `object` but potentially more columns. May be character vector of factor with character or factor input and `simplify` set to `TRUE`.

See Also

base::strsplit utils::read.fwf

Other auxiliary-functions: `opm_opt`

Examples

```
# Splitting by characters
x <- c("a-b-cc", "d-ff-g")
(y <- separate(x, ".")) # a split character that does not occur
stopifnot(is.matrix(y), y[, 1L] == x)
(y <- separate(x, "-")) # a split character that does occur
stopifnot(is.matrix(y), dim(y) == c(2, 3))

# Fixed-with splitting
x <- c("  abd  efgh", " ABCD EFGH ", " xyz")
(y <- separate(x, TRUE))
stopifnot(is.matrix(y), dim(y) == c(3, 2))

# Applied to factors
xx <- as.factor(x)
(yy <- separate(xx, TRUE))
stopifnot(identical(yy, as.data.frame(y)))

# List-wise splitting
x <- c("a,b", "c,b", "a,c")
(y <- separate(x, ",", list.wise = TRUE))
stopifnot(is.matrix(y), dim(y) == c(3, 3), is.logical(y))

# Data-frame method
x <- data.frame(a = 1:2, b = c("a-b-cc", "a-ff-g"))
(y <- separate(x, coerce = FALSE))
stopifnot(identical(x, y))
(y <- separate(x)) # only character/factor columns are split
stopifnot(is.data.frame(y), dim(y) == c(2, 4))
stopifnot(sapply(y, class) == c("integer", "factor", "factor", "factor"))
(y <- separate(x, keep.const = FALSE))
stopifnot(is.data.frame(y), dim(y) == c(2, 3))
stopifnot(sapply(y, class) == c("integer", "factor", "factor"))
```

---

set_spline_options *Spline options*

Description

Function to set up spline options which can be passed to do_aggr.

Usage

```
set_spline_options(type = c("tp.spline", "p.spline", "smooth.spline"),
  knots = NULL, gamma = 1,
  est.method = c("REML", "ML", "GCV"), s.par = NULL,
  correlation = NULL, save.models = FALSE,
  filename = NULL, ...)
```

Arguments

type	Character scalar. Specifies the spline type which should be fitted. This can be either thin plate splines (tp.spline), penalised B-splines (i.e, P-splines p.spline) or smoothing splines (smooth.spline).
knots	Integer scalar. Determines the number of knots. Per default, the number of knots is chosen adaptively to the number of unique observations.
gamma	Integer scalar. Specifies a constant multiplier to inflate the degrees of freedom in the "GCV" method to increase penalisation of models that are too close to the data and thus not smooth enough.
est.method	Character scalar. The smoothing parameter estimation method. Currently, only "REML", code"ML" and "GCV" are supported. This argument is ignored for type = "smooth.spline". For details see gam and gamm (see package **mgcv**).
s.par	list. Further arguments to be passed to the smoother s (see package **mgcv**). Note that the **mgcv** options k and bs are specified using type and knots in **opm**.
correlation	An optional "corStruct" object (see the help topic corClasses in the **nlme** package) as used to define correlation structures in package **nlme**. For better coverage of confidence intervals and slightly improved spline fits it is advised to use an AR process of order 1 or 2. However, this correction for auto-correlated error terms results in increased run time.
save.models	Should the models be saved (on the disk) for further inspections and plotting?
filename	File name of the models. Per default a name is auto-generated based on date and time. The file is always generated in the current working directory.
...	Additional arguments to be passed to gam or smooth.spline.

Value

List of options.

Author(s)

Benjamin Hofner

---

sort *Sort, unify, revert or repeat* OPMS *objects*

Description

Sort an OPMS object based on one to several metadata or CSV data entries, or sort elements of a MOPMX object based on plate type, length, or a metadata entry. Alternatively, remove duplicated elements from a OPMS or MOPMX object, or revert the order of plates within an OPMS object, or, repeat OPMS or OPM objects zero times, once, or several times.

Usage

```
  ## S4 method for signature 'OPM'
rep(x, ...)
  ## S4 method for signature 'OPMS'
rep(x, ...)

  ## S4 method for signature 'OPM'
rev(x)
  ## S4 method for signature 'OPMS'
rev(x)

  ## S4 method for signature 'MOPMX,ANY'
sort(x, decreasing,
    by = c("plate.type", "length"), exact = TRUE, strict = TRUE,
    na.last = TRUE, ...)
  ## S4 method for signature 'MOPMX,missing'
sort(x, decreasing, ...)
  ## S4 method for signature 'OPM,ANY'
sort(x, decreasing, ...)
  ## S4 method for signature 'OPM,missing'
sort(x, decreasing, ...)
  ## S4 method for signature 'OPMS,ANY'
sort(x, decreasing, by = "setup_time",
    parse = by == "setup_time", exact = TRUE, strict = TRUE, na.last = TRUE)
  ## S4 method for signature 'OPMS,missing'
sort(x, decreasing, ...)

  ## S4 method for signature 'MOPMX,ANY'
unique(x, incomparables, ...)
  ## S4 method for signature 'MOPMX,missing'
unique(x, incomparables, ...)
  ## S4 method for signature 'OPM,ANY'
unique(x, incomparables, ...)
  ## S4 method for signature 'OPM,missing'
unique(x, incomparables, ...)
  ## S4 method for signature 'OPMS,ANY'
unique(x, incomparables, ...)
  ## S4 method for signature 'OPMS,missing'
unique(x, incomparables, ...)
```

Arguments

x	OPM or OPMS or MOPMX object.
decreasing	Logical scalar. Passed to `order` or `sort.list` from the **base** package.
by	List or character vector. For OPMS objects, if a list, a list of one to several keys passed as `key` argument to `metadata`. If a character vector of length one, by is passed as 'what' argument to `csv_data`. If longer, passed step-by-step to `csv_data` as `keys` argument.

	For MOPMX objects, either 'plate.type', which sorts according to the plate types, 'length', which sorts the elements according to their lengths (i.e., number of plates), or a metadata query that yields, for each element of x, a vector to which max can be applied. Sorting x is then done according to these maxima.
parse	Logical scalar. Convert the setup_time via strptime before ordering? Has only an effect if by is 'setup_time'. It is an error if the time format is not recognised. See opm_opt, arguments time.fmt and time.zone, for modifying the parsing of setup-time entries, and csv_data for this kind of entries.
exact	Logical scalar. Passed to metadata. Affects only metadata querying, not directly the sorting.
strict	Logical scalar. Is it an error if metadata keys are not found? If FALSE, x gets ordered according to only the found keys, and remains in the original order if none of the keys in by are found at all. Note that it is always an error if keys are found in the metadata of some of the plates but not in those of others.
na.last	Logical scalar. Also passed to order or sort.list.
incomparables	Vector passed to duplicated. The default is FALSE.
...	Optional arguments passed between the methods or to duplicated or to rep from the **base** package. See the examples.

Details

The sort OPM method just returns the input data to avoid destructive effects due to the way the default sort interacts with OPM indexing.

rev should be slightly more efficient than calling the default rev method. There is also an OPM method which just returns the input data (to avoid destructive effects due to the way the default rev interacts with OPM indexing).

The OPM method of unique also returns the passed object.

rev yields an OPMS object with another number of plates, or an OPM object, or NULL.

Value

OPMS object with not necessarily the same order of plates than before, or OPM object.

See Also

base::order base::sort base::strptime base::unique base::rev

base::rep

Other conversion-functions: as.data.frame, extract, extract_columns, flatten, merge, oapply, opmx, plates, split, to_yaml

Examples

```
## 'OPMS' methods

# Existing keys
stopifnot(is.unsorted(metadata(vaas_4, "Strain")))
```

```
x <- sort(vaas_4, by = list("Strain"))
stopifnot(is(x, "OPMS"), !is.unsorted(metadata(x, "Strain")))
x <- sort(vaas_4, by = list("Strain"), decreasing = TRUE)
stopifnot(is(x, "OPMS"), is.unsorted(metadata(x, "Strain")))

# Non-existing keys
x <- try(sort(vaas_4, by = list("Not there", "Missing"), strict = TRUE))
stopifnot(inherits(x, "try-error")) # yields error
x <- try(sort(vaas_4, by = list("Not there", "Missing"), strict = FALSE))
stopifnot(identical(x, vaas_4)) # no error, but no new order

# CSV-data based
copy <- sort(vaas_4) # default: by setup time
csv_data(vaas_4, what = "setup_time")
csv_data(copy, what = "setup_time")
stopifnot(!identical(copy, vaas_4))
copy <- sort(vaas_4, by = c("Position", "Setup Time"))
csv_data(vaas_4, what = "position")
csv_data(copy, what = "position")
stopifnot(!is.unsorted(csv_data(copy, what = "position")))
stopifnot(is.unsorted(csv_data(vaas_4, what = "position")))

# making OPMS objects unique
dim(x <- unique(vaas_4))
stopifnot(identical(x, vaas_4))
dim(x <- unique(c(vaas_4, vaas_4)))
stopifnot(identical(x, vaas_4))
dim(x <- unique(vaas_4, what = "Species")) # species are not unique
stopifnot(dim(x)[1L] < dim(vaas_4)[1L])
dim(x <- unique(vaas_4, what = list("Species", "Strain")))
stopifnot(identical(x, vaas_4)) # organisms are unique

# reverting an OPMS object
dim(x <- rev(vaas_4))
stopifnot(dim(x) == dim(vaas_4), !identical(x, vaas_4))
stopifnot(identical(rev(x), vaas_4))

# repeating an OPMS object
dim(x <- rep(vaas_4))
stopifnot(identical(x, vaas_4))
dim(x <- rep(vaas_4, times = 2))
stopifnot(length(x) == length(vaas_4) * 2)
dim(y <- rep(vaas_4, each = 2))
stopifnot(length(y) == length(vaas_4) * 2, !identical(x, y))
stopifnot(is.null(rep(vaas_4, 0)))

## 'OPM' methods
summary(x <- sort(vaas_1))
stopifnot(identical(x, vaas_1))
dim(x <- unique(vaas_1)) # trivial
stopifnot(identical(x, vaas_1))
dim(x <- unique(vaas_1, what = list("Species", "Strain")))
stopifnot(identical(x, vaas_1))
```

```
dim(x <- rev(vaas_1)) # trivial
stopifnot(identical(x, vaas_1))
dim(x <- rep(vaas_1, 1))
stopifnot(identical(x, vaas_1))
dim(x <- rep(vaas_1, 2)) # conversion to OPMS if > 1 element
stopifnot(length(x) == 2, is(x, "OPMS"))
stopifnot(is.null(rep(vaas_4, 0)))
```

`split_files` *Manipulate files*

Description

Split files or clean file names.

Usage

```
split_files(files, pattern, outdir = "", demo = FALSE,
  single = TRUE, wildcard = FALSE, invert = FALSE,
  include = TRUE, format = opm_opt("file.split.tmpl"),
  compressed = TRUE, ...)

clean_filenames(x, overwrite = FALSE, demo = FALSE,
  empty.tmpl = "__EMPTY__%05i__")
```

Arguments

`files`	Character vector or convertible to such. Names of the files to be split.
`pattern`	Regular expression or shell globbing pattern for matching the separator lines if `invert` is `FALSE` (the default) or matching the non-separator lines if otherwise. Conceptually each of the sections into which a file is split comprises a separator line followed by non-separator lines. That is, separator lines followed by another separator line are ignored. Non-separator lines not preceded by a separator line are treated as a section of their own, however.
`outdir`	Character scalar determining the output directory. If empty, or containing empty strings, each file's input directory is used.
`demo`	Logical scalar. For `split_files`, do not create files, just return the usual list containing all potentially created files. Note that in contrast to the `demo` arguments of other IO functions, this requires the input files to be read. For `clean_filenames`, do not rename files but just return the usual result indicating the renaming actions that would be attempted? (Note that this does not indicate whether the renaming would also by successful.)
`single`	Logical scalar. If there is only one group per file, i.e. only one output file would result from the splitting, create this file anyway? Such cases could be recognised by empty character vectors as values of the returned list (see below).

`wildcard` Logical scalar. Is `pattern` a shell-globbing wildcard that first needs to be converted to a regular expression?

`invert` Logical scalar. Invert pattern matching, i.e. treat all lines that **not** match `pattern` as separators?

`include` Logical scalar. Also include the separator lines in the output files?

`format` Character scalar determining the output file name format. It is passed to `sprintf` and expects three placeholders:

- the base name of the file;
- the index of the section;
- the file extension.

Getting `format` wrong might result in non-unique file names and thus probably in overwritten files; accordingly, it should be used with care.

`compressed` Logical scalar. Passed to `file_pattern`, but here only affects the way file names are split in extensions and base names. Should only be set to `FALSE` if input files are not compressed (and have according file extensions).

`...` Optional arguments passed to `grepl`, which is used for matching the separator lines. See also `invert` listed above.

`x` Character vector or convertible to such. Names of the files to be modified.

`overwrite` Logical scalar. Overwrite already existing files, and do not care for duplicate names created by cleaning the file names?

`empty.tmpl` Character scalar. The template to use for file names that become empty after cleaning. Should include an integer placeholder to enable incrementing an index for creating unique file names. (Empty file names should occur rarely anyway.)

Details

`split_files` subdivides each file into sections which are written individually to newly generated files. Sections are determined with patterns that match the start of a section. This function is useful for splitting OmniLog® multiple-plate CSV files before inputting them with `read_opm`. It is used by `batch_opm` for this purpose. See also the 'Examples'.

`clean_filenames` modifies file names by removing anything else then word characters, dashes, and dots. Also remove trailing and leading dashes and underscores (per part of a file name, with dots separating these parts) and reduce adjacent dashes and underscores to a single one. Note that directory parts within the file names, if any, are not affected. This function might be useful for managing OmniLog® CSV files, which can contain a lot of special characters.

Value

`split_files` creates a list of character vectors, each vector containing the names of the newly generated files. The names of the list are the input file names. The list is returned invisibly.

`clean_filenames` yields a character vector, its names corresponding to the renamed old files, values corresponding to the novel names, returned invisibly.

See Also

base::split base::strsplit base::file.rename

Other io-functions: batch_collect, batch_opm, batch_process, collect_template, explode_dir, file_pattern, glob_to_regex, read_opm, read_single_opm, to_metadata

Examples

```
## split_files()

# Splitting an old-style CSV file containing several plates
(x <- opm_files("multiple"))
if (length(x) > 0) {
  outdir <- tempdir()
  # For old-style CSV, use either "^Data File" as pattern or "Data File*"
  # with 'wildcard' set to TRUE:
  (result <- split_files(x, pattern = "^Data File", outdir = outdir))
  stopifnot(is.list(result), length(result) == length(x))
  stopifnot(sapply(result, length) == 3)
  result <- unlist(result)
  stopifnot(file.exists(result))
  unlink(result) # tidy up
} else {
  warning("opm example files not found")
}
## One could split new-style CSV as follows (if x is a vector of file names):
# split_files(x, pattern = '^"Data File",')
## note the correct setting of the quotes
## A pattern that covers both old and new-style CSV is:
# split_files(x, pattern = '^("Data File",|Data File)')
## This is used by batch_opm() in 'split' mode any by the 'run_opm.R' script

## clean_filenames()

# Check the example files: they should be ok
(x <- clean_filenames(opm_files("testdata"), demo = TRUE))
stopifnot(length(x) == 0)

# Example with temporary files
(x <- tempfile(pattern = "cb& ahi+ si--")) # bad file name
write("test", x)
stopifnot(file.exists(x))
(y <- clean_filenames(x)) # file renamed
stopifnot(!file.exists(x), file.exists(y))
unlink(y) # tidy up
```

`subset` *Select a subset of the plates (or time points)*

Description

Select a subset of the plates in an OPMS object based on the content of the metadata. Alternatively, select a common subset of time points from all plates. thin_out keeps only a regular subset of the time points from OPM measurements. This is a mainly experimental function that might be of use in testing.

Usage

```
  ## S4 method for signature 'MOPMX'
subset(x, query, ...)
  ## S4 method for signature 'OPMX'
subset(x, query, values = TRUE,
    invert = FALSE, exact = FALSE, time = FALSE,
    positive = c("ignore", "any", "all"),
    negative = c("ignore", "any", "all"),
    use = c("i", "I", "k", "K", "n", "N", "p", "P", "q", "Q", "t", "T"))

  ## S4 method for signature 'MOPMX'
thin_out(object, ...)
  ## S4 method for signature 'OPM'
thin_out(object, factor, drop = FALSE)
  ## S4 method for signature 'OPMS'
thin_out(object, ...)
```

Arguments

x	OPMX or MOPMX object.
query	Logical or numeric vector or object accepted as query by the infix operators. If a logical or numeric vector, query is directly used as the first argument of [, and all following arguments, if any, are ignored. If otherwise, it is used for conducting a query based on one of the infix operators as described below.
values	Logical scalar. If TRUE, the values of query are also considered (by using infix.q or infix.largeq). If FALSE only the keys are considered (by using infix.k). That is, choose either the plates for which certain metadata entries contain certain values, or choose the plates for which these metadata have been set at all (to some arbitrary value). See the mentioned functions for details, and note the special behaviour if query is a character vector and values is FALSE.
invert	Logical scalar. If TRUE, return the plates for which the condition is not TRUE.
exact	Logical scalar. If the values of query are considered, should this be done using infix.q (when FALSE) or infix.largeq (when TRUE)? See these functions and contains for details.
time	Logical scalar. If TRUE, all other arguments are ignored and the object is reduced to the common subset of time points (measurement hours and minutes).
positive	Character scalar. If 'ignore', not used. Otherwise all previous arguments except object are ignored, and the function yields an error unless object has been

	discretised throughout, i.e. either it is an OPMD object or all elements of object have discretised values. If object has the necessary discretised values, if 'any', wells are selected that contain positive reactions in at least one plate. If 'all', wells are selected that contain positive reactions in all plates. Using invert means selecting all negative or weak reactions.
negative	Character scalar. Like positive, but returns the negative reactions. Using invert means selecting all positive or weak reactions.
use	Character scalar. An alternative way to specify the settings. If 'i' or 'I', ignored. If 't' or 'T', time is set to TRUE. If 'p' or 'P', positive is set to 'any' or 'all', respectively. If 'n' or 'N', negative is set to 'any' or 'all', respectively. Otherwise, use is taken directly as the one-latter name of the infix operators to use for plate selection, overriding values and exact.
object	OPMX object.
factor	Numeric scalar >= 1 indicating how much the data set shall be thinned out.
drop	Logical scalar. See [.
...	Optional arguments passed between the methods.

Details

The MOPMX method creates subsets of all contained OPMX objects (if any) in turn and then removes those that yielded NULL. Thus subset is not intended for directly creating subsets of MOPMX but of their elements to yield, e.g., elements that have a common set of metadata entries, as required under most circumstances by some other MOPMX methods such as extract.

Thinning the plates out is experimental insofar as it has **not** been tested whether and how this could sensibly be applied before aggregating the data.

Value

NULL or OPM or OPMS object. This depends on how many plates are selected; see [for details. The MOPMX method always returns a MOPMX object.

See Also

base::'[' base::'[[' base::subset

Other getter-functions: aggr_settings, aggregated, anyDuplicated, contains, csv_data, dim, disc_settings, discretized, duplicated, has_aggr, has_disc, hours, max, measurements, minmax, seq, well

Examples

```
# simple object comparison function
mustbe <- function(a, b) stopifnot(identical(a, b))

# all plates have that entry: selection identical to original object
mustbe(vaas_4, vaas_4["Species" %k% vaas_4, ])
```

```
mustbe(vaas_4, subset(vaas_4, list(Species = "Escherichia coli"),
  values = FALSE)) # equivalent
mustbe(vaas_4, subset(vaas_4, ~ Species == "Escherichia coli",
  values = FALSE)) # also equivalent

# two plates also have that value: yielding OPMS object with only two plates
mustbe(vaas_4[1:2], vaas_4[list(Species = "Escherichia coli") %q% vaas_4, ])
mustbe(vaas_4[1:2], subset(vaas_4, list(Species = "Escherichia coli")))
mustbe(vaas_4[1:2], subset(vaas_4, ~ Species == "Escherichia coli"))

# these are also equivalent
mustbe(vaas_4[c(1, 3)],
  vaas_4[list(Strain = c("DSM18039", "DSM1707")) %q% vaas_4])
mustbe(vaas_4[c(1, 3)],
  subset(vaas_4, list(Strain = c("DSM18039", "DSM1707"))))
mustbe(vaas_4[c(1, 3)],
  subset(vaas_4, ~ Strain %in% c("DSM18039", "DSM1707")))
mustbe(vaas_4[c(1, 3)],
  subset(vaas_4, ~ Strain == "DSM18039" || Strain == "DSM1707"))
# note that particularly formulae can be used to set up very complex queries

# select all plates that have aggregated values
dim(x <- subset(vaas_4, has_aggr(vaas_4)))
mustbe(x, vaas_4) # all have such values

# select a common set of time points
dim(x <- subset(vaas_4, time = TRUE))
mustbe(x, vaas_4) # the time points had already been identical
# create unequal time points
dim(copy <- vaas_4[, list(1:10, 1:20, 1:15, 1:10)])
mustbe(hours(copy), c(2.25, 4.75, 3.50, 2.25))
# now restrict to common subset
dim(x <- subset(copy, time = TRUE))
mustbe(hours(x), rep(2.25, 4))
# see also the example with split() given under "["

# select all wells that have positive reactions
dim(x <- subset(vaas_4, use = "p")) # in at least one plate
stopifnot(dim(x)[3] < dim(vaas_4)[3])
dim(y <- subset(vaas_4, use = "P")) # in all plates
stopifnot(dim(y)[3] < dim(x)[3])

# select all wells that have non-negative reactions in at least one plate
dim(y <- subset(vaas_4, use = "N", invert = TRUE))
stopifnot(dim(y)[3] > dim(x)[3])

## thin_out()

# 'OPM' method
(x <- dim(vaas_1))
stopifnot(identical(x, c(384L, 96L)))
copy <- thin_out(vaas_1, 10) # keep every 10th time point and measurement
(x <- dim(copy))
```

```
stopifnot(identical(x, c(38L, 96L)), has_aggr(copy))
copy <- thin_out(vaas_1, 10, drop = TRUE) # also remove the parameters
(x <- dim(copy))
stopifnot(identical(x, c(38L, 96L)), !has_aggr(copy))

# 'OPMS' method
(x <- dim(vaas_4))
stopifnot(identical(x, c(4L, 384L, 96L)))
copy <- thin_out(vaas_4, 10)
(x <- dim(copy))
stopifnot(identical(x, c(4L, 38L, 96L)))
```

`substrate_info` *Provide information on substrates*

Description

Return information on substrates such as their CAS number or other database ID or convert substrate names.

Usage

```
  ## S4 method for signature 'MOPMX'
substrate_info(object, ...)
  ## S4 method for signature 'OPM'
substrate_info(object, ...)
  ## S4 method for signature 'OPMS'
substrate_info(object, ...)
  ## S4 method for signature 'character'
substrate_info(object,
    what = c("cas", "kegg", "drug", "metacyc", "chebi", "mesh", "seed",
      "downcase", "greek", "concentration", "html", "peptide", "peptide2",
      "all"),
    browse = 0L, download = FALSE, ...)
  ## S4 method for signature 'factor'
substrate_info(object, ...)
  ## S4 method for signature 'list'
substrate_info(object, ...)
  ## S4 method for signature 'substrate_match'
substrate_info(object, ...)
```

Arguments

`object`	Query character vector, factor or list, S3 object of class 'substrate_match', `OPM`, `OPMS` or `MOPMX` object.
`what`	Character scalar indicating which kind of information to output.
	all Create object of S3 class 'substrate_data' containing all available information and useful for display.

cas CAS registry number, optionally expanded to an URL.

chebi CHEBI database ID, optionally expanded to an URL.

concentration Attempt to extract concentration information (as used in **opm** substrate names) from `object`. Return `NA` wherever this fails.

downcase Substrate name converted to lower case, protecting one-letter specifiers, acronyms and chemical symbols, and translating relevant characters from the Greek alphabet.

drug KEGG drug database ID, optionally expanded to an URL.

greek Substrate name after translation of relevant characters to Greek letters.

html Like `greek`, but using HTML tags, and also converting other parts of compound names that require special formatting.

kegg KEGG compound database ID, optionally expanded to an URL.

mesh MESH database name (useful for conducting PUBMED searches), optionally expanded to an URL.

metacyc METACYC database ID, optionally expanded to an URL.

peptide List of character vectors representing amino acids in three-letter code, in order, contained in the substrate if it is a peptide. Empty character vectors are returned for non-peptide substrates. Amino acids without 'L' or 'D' annotation are assumed to be in 'L' conformation, i.e. 'L-' is removed from the beginning of the amino acid codes.

peptide2 Like `peptide`, but without removal of 'L-' from the beginning of the amino acid codes.

seed SEED compound database ID, optionally expanded to an URL.

See the references for information on the databases.

`browse` Numeric scalar. If non-zero, an URL is generated from each ID. If positive, this number of URLs (counted from the beginning) is also opened in the default web browser; if negative, the URLs are only returned. It is an error to try this with those values of `what` that do not yield an ID.

`download` Logical scalar indicating whether, using the available IDs, substrate information should be queried from the according web services and returned in customised objects. Note that this is unavailable for most values of `what`. At the moment only `kegg` and `drug` can be queried for if the **KEGGREST** package is available. This would yield S3 objects of the class `kegg_compounds`.

`...` Optional other arguments passed between the methods.

Details

The query names must be written exactly as used in the stored plate annotations. To determine their spelling, use `find_substrate`. Each spelling might include a concentration indicator, but the same underlying substrate name yielded the same ID irrespective of the concentration.

Note that the information is only partially complete, depending on the well and the database. While it is possible to link almost all substrates to, say, CAS numbers, they are not necessarily contained in the other databases. Thanks to the work of the CHEBI staff, which is gratefully acknowledged, CHEBI information is complete as far as possible (large molecules such as proteins or other polymers are not covered by CHEBI).

For some wells, even a main substrate cannot be identified, causing all its IDs to be missing. This holds for all control wells, for all wells that contain a mixture of (usually two) substrates, and for all wells that are only specified by a certain pH.

The generated URLs should provide plenty of information on the respective substrate. In the case of CHEBI, KEGG and METACYC, much information is directly displayed on the page itself, whereas the chosen CAS site contains a number of links providing additional chemical details. The MESH web pages directly link to according PUBMED searches.

Value

The character method returns a character vector with `object` used as names and either a matched entry or `NA` as value. Only if `what` is set to 'peptide' a named list is returned instead. The factor method works like the character method, whereas the list method traverses a list and calls `substrate_info` on suitable elements, leaving others unchanged. The `OPM` and `OPMS` methods work like the character method, using their own substrates.

Depending on the `browse` argument, the returned IDs might have been converted to URLs, and as a side effect tabs in the default web browser might have been opened. For suitable values of `what`, setting `download` to `TRUE` yielded special objects as described above.

The `MOPMX` method yield a list with one element of one of the kinds described above per element of `object`.

References

Bochner, B. R., pers. comm.

http://www.cas.org/content/chemical-substances/faqs

http://www.genome.jp/kegg/

Kanehisa, M., Goto, S., Furumichi, M., Tanabe, M., and Hirakawa, M. 2010 KEGG for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Research* **38**: D355–D360.

http://metacyc.org/

Caspi, R., Altman, T., Dreher, K., Fulcher, C.A., Subhraveti, P., Keseler, I.M., Kothari, A., Krummenacker, M., Latendresse, M., Mueller, L.A., Ong, Q., Paley, S., Pujar, A., Shearer, A.G., Travers, M., Weerasinghe, D., Zhang, P., Karp, P.D. 2012 The MetaCyc database of metabolic pathways and enzymes and the BioCyc collection of pathway/genome databases. *Nucleic Acids Research* **40**: D742–D753.

http://www.ncbi.nlm.nih.gov/mesh

Coletti, M.H., Bleich, H.L 2001 Medical subject headings used to search the biomedical literature. *Journal of the American Medical Informatics Association* **8**: 317–323.

http://www.ebi.ac.uk/chebi/

Hastings, J., de Matos, P., Dekker, A., Ennis, M., Harsha, B., Kale, N., Muthukrishnan, V., Owen, G., Turner, S., Williams, M., Steinbeck, C. 2013 The ChEBI reference database and ontology for biologically relevant chemistry: enhancements for 2013. *Nucleic Acids Research* **41**: D456–D463.

Overbeek, R., Begley, T., Butler, R., Choudhuri, J., Chuang, H., Cohoon, M., de Crecy-Lagard, V., Diaz, N., Disz, T., Edwards, R., Fonstein, M., Frank, E., Gerdes, S., Glass, E., Goesmann, A., Hanson, A., Iwata-Reuyl, D., Jensen, R., Jamshidi, N., Krause, L., Kubal, M., Larsen, N., Linke,

B., McHardy, A., Meyer, F., Neuweger, H., Olsen, G., Olson, R., Osterman, A., Portnoy, V., Pusch, G., Rodionov, D., Rueckert, C., Steiner, J., Stevens, R., Thiele, I., Vassieva, O., Ye, Y., Zagnitko, O., Vonstein, V. 2005 The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes. *Nucleic Acids Research* **33**: 5691–5702.

See Also

utils::browseURL

Other naming-functions: find_positions, find_substrate, gen_iii, listing, opm_files, param_names, plate_type, register_plate, select_colors, wells

Examples

```
# Character method; compare correct and misspelled substrate name
(x <- substrate_info(c("D-Glucose", "D-Gloucose")))
stopifnot(any(is.na(x)), !all(is.na(x)))
stopifnot(identical(x, # Factor method yields same result
  substrate_info(as.factor(c("D-Glucose", "D-Gloucose")))))

# Now with generation of URLs
(y <- substrate_info(c("D-Glucose", "D-Gloucose"), browse = -1))
stopifnot(is.na(y) | nchar(y) > nchar(x))
# NA remains NA (and the function would not try to open it in the browser)

# Character method, safe conversion to lower case
(x <- substrate_info(c("a-D-Glucose", "a-D-Gloucose"), "downcase"))
stopifnot(nchar(x) > nchar(c("a-D-Glucose", "a-D-Gloucose")))
# note the protection of 'D' and the conversion of 'a'
# whether or not substrate names are known does not matter here

# Peptide extraction (note treatment of non-standard amino acids)
(x <- substrate_info(c("Ala-b-Ala-D-Glu", "Glucose", "Trp-Val"), "peptide"))
stopifnot(is.list(x), sapply(x, length) == c(3, 0, 2))

# List method
(x <- substrate_info(find_substrate(c("D-Glucose", "D-Gloucose"))))
stopifnot(length(x[[1]]) > length(x[[2]]))

# OPM and OPMS methods
(x <- substrate_info(vaas_1[, 1:3], "all"))
stopifnot(inherits(x, "substrate_data"))
stopifnot(identical(x, substrate_info(vaas_4[, , 1:3], "all")))
## Not run:

  # this would open up to 96 tabs in your browser...
  substrate_info(vaas_4, "kegg", browse = 100)

## End(Not run)
```

summary *Summarise* OPMX *or* MOPMX *objects*

Description

Generate a summary (which also prints nicely to the screen), or display an OPM, OPMS or MOPMX object on screen.

Usage

```
  ## S4 method for signature 'CMAT'
show(object)
  ## S4 method for signature 'MOPMX'
show(object)
  ## S4 method for signature 'OPMX'
show(object)

  ## S4 method for signature 'MOPMX'
summary(object, ...)
  ## S4 method for signature 'OPM'
summary(object, ...)
  ## S4 method for signature 'OPMS'
summary(object, ...)
```

Arguments

object	OPM, OPMS or MOPMX object.
...	Optional arguments to be included in the output.

Details

Currently the show methods are just wrappers for the summary methods for these objects with an additional call to print. The CMAT method is only for internal use.

Value

For the OPM method, a named list of the class OPM_Summary, returned invisibly. The 'Metadata' entry is the number of non-list elements in metadata. For the OPMS method, a list of such lists (one per plate), also returned invisibly, with the class set to OPMS_Summary and some information on the entire object in the attribute 'overall'.

See Also

base::summary base::formatDL methods::show base::print

Other plotting-functions: ci_plot, heat_map, level_plot, parallelplot, radial_plot, xy_plot

Examples

```
# OPM method
(x <- summary(vaas_1))
stopifnot(is.list(x), is.object(x))
vaas_1 # calls show()

# OPMS method
(x <- summary(vaas_4))
stopifnot(is.list(x), length(x) == 4L, all(sapply(x, is.list)),
  is.object(x))
vaas_4 # calls show()
```

---

to_kmeans *Work with k-means results*

Description

Calculate or plot the Calinski-Harabasz statistics from `kmeans` results. The result of `plot` is a simple scatter plot which can be modified with arguments passed to `plot` from the **graphics** package. Alternatively, determine the borders between clusters of one-dimensional data, create a histogram in which these borders are plotted, or convert an object to one of class `kmeans`.

Usage

```
 to_kmeans(x, ...)

 ## S3 method for class 'kmeans'
to_kmeans(x, ...)

 ## S3 method for class 'kmeanss'
to_kmeans(x, y, ...)

 ## S3 method for class 'Ckmeans.1d.dp'
to_kmeans(x, y, ...)

 calinski(x, ...)

 ## S3 method for class 'kmeans'
calinski(x, ...)

 ## S3 method for class 'Ckmeans.1d.dp'
calinski(x, y, ...)

 ## S3 method for class 'kmeanss'
calinski(x, ...)

 ## S3 method for class 'kmeanss'
```

```
plot(x, xlab = "Number of clusters",
   ylab = "Calinski-Harabasz statistics", ...)

 borders(x, ...)

 ## S3 method for class 'kmeans'
borders(x, y, ...)

 ## S3 method for class 'Ckmeans.1d.dp'
borders(x, y, ...)

 ## S3 method for class 'kmeanss'
borders(x, ...)

 ## S3 method for class 'kmeans'
hist(x, y, col = "black", lwd = 1L,
   lty = 1L, main = NULL, xlab = "Clustered values", ...)

 ## S3 method for class 'Ckmeans.1d.dp'
hist(x, y, ...)

 ## S3 method for class 'kmeanss'
hist(x, k = NULL, col = "black",
   lwd = 1L, lty = 1L, main = NULL,
   xlab = "Clustered values", ...)
```

Arguments

x	Object of class kmeans, 'Ckmeans.1d.dp' or kmeanss. For plot, only the latter.
y	Vector of original data subjected to clustering. Automatically determined for the kmeanss methods. For to_kmeans, original numeric vector that was used to create a 'Ckmeans.1d.dp' object, or index of an element of a kmeanss object.
k	Numeric vector or NULL. If non-empty, it indicates the number of groups (previously used as input for kmeans) for which vertical lines should be drawn in the plot that represent the cluster borders. If empty, the smallest non-trivial number of clusters is chosen.
col	Graphical parameter passed to abline. If several values of k are given, col is recycled as necessary.
lwd	Like col.
lty	Like col.
main	Passed to hist.default.
xlab	Character scalar passed to hist.default or to plot from the **graphics** package.
ylab	Character scalar passed to plot from the **graphics** package.
...	Optional arguments passed to and from other methods. For the hist method, optional arguments passed to hist.default.

Details

The borders are calculated as the mean of the maximum of the cluster with the lower values and the minimum of the neighbouring cluster with the higher values. The `hist` method plots a histogram of one-dimensional data subjected to k-means partitioning in which these borders can be drawn.

`y` must also be in the order it has been when subjected to clustering, but this is not checked. Using `kmeanss` objects thus might preferable in most cases because they contain a copy of the input data.

Value

`to_kmeans` creates an object of class `kmeans`.

`borders` creates a numeric vector or list of such vectors.

The return value of the `hist` method is like `hist.default`; see there for details.

`calinksi` returns a numeric vector with one element per `kmeans` object. `plot` returns it invisibly. Its 'names' attribute indicates the original numbers of clusters requested.

See Also

graphics::hist graphics::abline Ckmeans.1d.dp::Ckmeans.1d.dp

Other kmeans-functions: `run_kmeans`

Examples

```
x <- as.vector(extract(vaas_4, as.labels = NULL, subset = "A"))
x.km <- run_kmeans(x, k = 1:10)

# plot() method
# the usual arguments of plot() are available
show(y <- plot(x.km, col = "blue", pch = 19))
stopifnot(is.numeric(y), names(y) == 1:10)

# borders() method
(x.b <- borders(x.km)) # => list of numeric vectors
stopifnot(is.list(x.b), length(x.b) == 10, sapply(x, is.numeric))
stopifnot(sapply(x.b, length) == as.numeric(names(x.b)) - 1)

# hist() methods
y <- hist(x.km[[2]], x, col = "blue", lwd = 2)
stopifnot(inherits(y, "histogram"))
y <- hist(x.km, 3:4, col = c("blue", "red"), lwd = 2)
stopifnot(inherits(y, "histogram"))

# to_kmeans() methods
x <- c(1, 2, 4, 5, 7, 8)
summary(y <- kmeans(x, 3))
stopifnot(identical(y, to_kmeans(y)))
# see particularly run_kmeans() which uses this internally if clustering is
# done with Ckmeans.1d.dp::Ckmeans.1d.dp()
```

to_yaml *Convert to* YAML

Description

Convert some R object to YAML or JSON.

Usage

```
  ## S4 method for signature 'MOPMX'
to_yaml(object, ...)
  ## S4 method for signature 'YAML_VIA_LIST'
to_yaml(object, ...)
  ## S4 method for signature 'list'
to_yaml(object, sep = TRUE,
    line.sep = "\n", json = FALSE, listify = nodots, nodots = FALSE, ...)
```

Arguments

object	Object of one of the classes belonging to YAML_VIA_LIST, or MOPMX object.
sep	Logical scalar. Prepend YAML document separator '---'?
line.sep	Character scalar used as output line separator.
json	Logical scalar. Create JSON instead of YAML? If so, sep, line.sep and ... are ignored.
listify	Logical scalar indicating whether after conversion to a list its non-list elements should be converted to lists if they have names. (Names of named vector are **not** conserved by default in output YAML).
nodots	Logical scalar indicating whether dots in list names should be converted to underscores. This is necessary in some situations (we met this problem when storing JSON documents in a document-oriented database). Converted names will additionally be marked by prepending an underscore, which assists in getting the original spelling back but is anything else than fail-safe.
...	Optional other arguments passed to as.yaml from the **yaml** package, or arguments passed between the methods.

Details

YAML is a useful data-serialisation standard that is understood by many programming languages. It is particularly more human readable than XML, and vector-like data structures (such as phenotype microarray measurements) can be much more compactly encoded.

Many PM data sets at once can be batch-converted into YAML format using batch_opm. The output format for the child classes is described in detail there, as well as other aspects relevant in practice.

JSON is a subset of YAML and (in most cases) can also be parsed by a YAML parser. For generating JSON, the toJSON function from the **rjson** package would be used.

Value

Character scalar (YAML string).

References

http://www.yaml.org/

http://www.json.org/

See Also

yaml::as.yaml yaml::yaml.load_file json::toJSON

Other conversion-functions: as.data.frame, extract, extract_columns, flatten, merge, oapply, opmx, plates, rep, rev, sort, split, unique

Examples

```
## Not run:

# Let 'x' be a any convertible object
# Store the data in file 'out.yml' in YAML format.
write(to_yaml(x), "out.yml")

## End(Not run)
```

---

vaas_4 *Example data sets from Vaas et al. (2012)*

Description

Two literature example data sets are provided with the **opm** package, containing one or four plates, respectively.

Format

vaas_4 is an OPMS object with the dimensions 4 x 384 x 96, i.e. 4 plates with 384 time points and 96 wells per plate. vaas_1 is an OPMD object with the dimensions 384 x 96, i.e. a single plate with 384 time points and 96 wells.

Details

The OPMS object vaas_4 contains measurements from four selected plates from the study by Vaas *et al.* (2012). Metadata have been added to fully describe the conducted PM experiments: these plates are the sixth technical replicate from the first biological replicate for the four bacterial strains considered in the study.

This particular subset served as input for Figure 2 in Vaas *et al.* (2012), which can be reproduced by running xy_plot with vaas_4. Accordingly, Figure 3 in Vaas *et al.* (2012) represents the wells G11 and H11 selected from vaas_4. Figure 4 focuses then further on that subset, namely showing

the data from strain DSM 30083^T (left curve) and strain *Pseudomonas aeruginosa* DSM 1707 (right curve).

The OPMD object vaas_1 contains measurements from a single selected plate from the study by Vaas *et al.* (2012). Metadata have been added to fully describe the conducted PM experiments: this plate is the sixth technical replicate from the first biological replicate for the strain *Escherichia coli* DSM 30083^T (yes, the type strain of *E. coli*). This is a subset of vaas_4.

The complete data set is available as vaas_et_al in the **opmdata** package.

References

Bochner, B.R., Savageau, M.A. 1977. Generalized indicator plate for genetic, metabolic, and taxonomic studies with microorganisms. *Applied and Environmental Microbiology* **33**, 434–444.

Selezska, K., Kazmierczak, M., Muesken, M., Garbe, J., Schobert, M., Haeussler, S., Wiehlmann, L., Rohde, C., Sikorski, J. 2012 *Pseudomonas aeruginosa* population structure revisited under environmental focus: impact of water quality and phage pressure. *Environmental Microbiology* **14**, 1952–1967.

Vaas, L. A. I., Sikorski, J., Michael, V., Goeker, M., Klenk H.-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846.

http://www.dsmz.de/catalogues/details/culture/DSM-1707.html

http://www.dsmz.de/catalogues/details/culture/DSM-18039.html

http://www.dsmz.de/catalogues/details/culture/DSM-30083.html

Examples

```
## Not run:

# Calling this yielded a variable 'vaas_4' containing the data. The opm
# package must be loaded beforehand using library().
data(vaas_4)

# Calling this yielded a variable 'vaas_1' containing the data. The opm
# package must be loaded beforehand using library().
data(vaas_1)

## End(Not run)
```

---

wells *Listing of well names*

Description

Get the names of the wells contained in an OPMX object. Optionally the full substrate names can be added in parentheses or brackets or used instead of the coordinate, and trimmed to a given length. The listing methods create a textual listing of the discretised values. (See do_disc for generating discretised data.) This is useful to describe OmniLog® phenotype microarray results in a scientific manuscript.

Usage

```
  ## S4 method for signature 'OPMD'
listing(x, as.groups,
    cutoff = opm_opt("min.mode"), downcase = TRUE, full = TRUE,
    in.parens = FALSE, html = FALSE, sep = " ", ..., exact = TRUE,
    strict = TRUE)
  ## S4 method for signature 'XOPMX'
listing(x, as.groups, cutoff = opm_opt("min.mode"),
    downcase = TRUE, full = TRUE, in.parens = FALSE, html = FALSE, sep = " ",
    ..., exact = TRUE, strict = TRUE)
  ## S4 method for signature 'well_coords_map'
listing(x)

  ## S4 method for signature 'ANY'
wells(object, full = TRUE, in.parens = FALSE,
    max = opm_opt("max.chars"), brackets = FALSE, clean = TRUE,
    word.wise = FALSE, paren.sep = " ", downcase = FALSE, rm.num = FALSE,
    plate = "PM01", simplify = FALSE)
  ## S4 method for signature 'MOPMX'
wells(object, ...)
  ## S4 method for signature 'OPM'
wells(object, full = FALSE, in.parens = TRUE,
    max = opm_opt("max.chars"), brackets = FALSE, clean = TRUE,
    word.wise = FALSE, paren.sep = " ", downcase = FALSE, rm.num = FALSE,
    plate = plate_type(object), simplify = TRUE)
  ## S4 method for signature 'OPMS'
wells(object, ...)
  ## S4 method for signature 'missing'
wells(object, ...)
```

Arguments

`object`	OPM object, OPMS or MOPMX object or well name or index. If missing, defaults to the selection of all possible wells (for the default plate type, see below).
`full`	Logical scalar. Return the full names of the wells (if available) or just their coordinates on the plate? The following arguments have no effect if `full` is FALSE.
`in.parens`	Logical scalar. If TRUE, add the full name of the substrate in parentheses (or brackets) after the original name. If FALSE, replace by the full substrate name. Note that adding in parentheses (or brackets) is only done if the trimmed substrate names are not empty.
`max`	Numeric scalar. Maximum number of characters allowed in the names. Longer names are truncated and the truncation is indicated by appending a dot.
`brackets`	Logical scalar. Use brackets instead of parentheses?
`clean`	Logical scalar. If TRUE, clean trimmed end of full substrate name from non-word characters; use an empty string if only the dot remained.

word.wise — Logical scalar. If TRUE, abbreviation works by truncating each word separately, and removing vowels first.

paren.sep — Character scalar. What to insert before the opening parenthesis (or bracket). Currently only zero to many whitespace characters are allowed. The ability to insert a line break is the main purpose of this argument. Using the 'at sign' as value is the only alternative and also special, as it causes the plate name itself to be appended to the well coordinate (after an 'at sign', without parentheses or brackets). So mapping is not actually done in that case but the resulting names are understood by certain other **opm** methods which can conduct the mapping at a later stage.

downcase — Logical scalar indicating whether full names should be (carefully) converted to lower case. This uses substrate_info in downcase mode; see there for details.

rm.num — Logical scalar indicating whether numbering (used in the case of replicated substrates per plate) should be stripped from the end of the full well names.

plate — Name of the plate type. Several ones can be given unless object is of class OPM or OPMS. Normalisation as in plate_type is applied before searching for the substrate names but otherwise the match must be exact.

simplify — Logical scalar indicating whether the result should be simplified to a vector. This will never be done if more than a single column is contained, i.e. if data for more than a single plate type are queried for.

... — Optional arguments passed between the methods.

x — OPMD, OPMS or well_coords_map object.

as.groups — Key suitable for querying the metadata, or NULL. If non-empty, passed as eponymous argument to extract. Thus TRUE and FALSE can be used, creating either a single group or one per plate. The extracted metadata define groups for which the discretised data are aggregated.

If x is an OPMD object and as.groups is not empty, it is used to create the row name of the single row of the resulting OPMS_Listing object. Otherwise an OPMD_Listing object is produced.

cutoff — Numeric scalar used if 'as.groups' is non-empty. If the relative frequency of the most frequent entry within the discretised values to be joined is below that cutoff, NA is used. Ignored if x is an OPMD object but added to the result if as.groups is non-empty.

html — Logical scalar. Convert to HTML? This involves Greek letters and paragraph ('div') tags.

sep — Character scalar used for joining the 'as.groups' entries (if any).

exact — Logical scalar passed to metadata.

strict — Logical scalar also passed to metadata.

Details

Do not confuse wells this with well. The purpose of the OPM and OPMS methods for wells should be obvious. The default method is intended for providing a quick overview of the substrates contained in one to several plates if full is TRUE. If full is FALSE, it can be used to study the effect of the well-index translation and well-name normalisation approaches as used by **opm**, particularly by the sub-creation methods (see [).

Value

The wells methods return a named character vector or a named matrix of the S3 class well_coords_map, depending on simplify and plate.

The return value of the listing methods for OPMX objects is a character vector or matrix with additional class attribute OPMD_Listing or OPMS_Listing.

The well_coords_map method creates a nested list of the class well_coords_listing which can be used in conjunction with to_yaml or saveRDS for externally storing well maps. See the examples for details.

See Also

base::strtrim base::abbreviate

Other naming-functions: find_positions, find_substrate, gen_iii, opm_files, param_names, plate_type, register_plate, select_colors, substrate_info

Examples

```
## wells() 'OPM' method
(x <- wells(vaas_1, full = FALSE))[1:10]
(y <- wells(vaas_1, full = TRUE))[1:10]
(z <- wells(vaas_1, full = TRUE, in.parens = FALSE))[1:10]
# string lengths differ depending on selection
stopifnot(nchar(x) < nchar(y), nchar(z) < nchar(y))

## wells() 'OPM' method
(xx <- wells(vaas_4, full = FALSE))[1:10]
# wells are guaranteed to be uniform within OPMS objects
stopifnot(identical(x, xx))

## wells() default method
x <- c("A01", "B10")
(y <- wells(x, plate = "PM1"))
stopifnot(nchar(y) > nchar(x))
(z <- wells(x, plate = "PM1", in.parens = TRUE))
stopifnot(nchar(z) > nchar(y))
# formula yields same result
stopifnot(y == wells(~ c(A01, B10), plate = "PM1"))
# querying for several plate types at once
(y <- wells(~ c(A01, B10), plate = c("PM2", "PM3", "PM10")))
stopifnot(dim(y) == c(2, 3))
(z <- listing(y)) # create a printable nested list
stopifnot(is.list(z), sapply(z, is.list), names(z) == colnames(y))
# using a sequence of well coordinates
stopifnot(nrow(wells(~ C02:C06)) == 5) # well sequence
stopifnot(nrow(wells(plate = "PM1")) == 96) # all wells by default

## listing() 'OPMD' method

# this yields one sentence for each kind of reaction:
(x <- listing(vaas_1, NULL))
```

```
stopifnot(inherits(x, "OPMD_Listing"), is.character(x), length(x) == 3,
  !is.null(names(x)))

# create an 'OPMS_Listing' object
(y <- listing(vaas_1, ~ Species + Strain))
stopifnot(inherits(y, "OPMS_Listing"), is.matrix(y), dim(y) == c(1, 3),
  y == x, colnames(y) == names(x), !is.null(rownames(y)))

# including HTML tags
(y <- listing(vaas_1, NULL, html = TRUE))
stopifnot(inherits(y, "OPMD_Listing"), is.character(x), nchar(y) > nchar(x),
  !is.null(names(x)))

## listing() 'OPMS' method

# no grouping, no names (numbering used instead for row names)
(x <- listing(vaas_4[1:2], as.groups = NULL))
stopifnot(inherits(x, "OPMS_Listing"), is.matrix(x), dim(x) == c(2, 3))
stopifnot(!is.null(rownames(x)), !is.null(colnames(x)))
(y <- listing(vaas_4[1:2], as.groups = FALSE)) # alternative
stopifnot(identical(x, y))

# in effect no grouping, but names
(x <- listing(vaas_4[1:2], as.groups = list("Species", "Strain")))
stopifnot(inherits(x, "OPMS_Listing"), is.matrix(x), dim(x) == c(2, 3))
stopifnot(!is.null(rownames(x)), !is.null(colnames(x)))

# only single group for all plates
(y <- listing(vaas_4[1:2], as.groups = TRUE))
stopifnot(inherits(y, "OPMS_Listing"), is.matrix(y), dim(y) == c(1, 3))
stopifnot(!is.null(rownames(x)), !is.null(colnames(x)))

# two groups
(x <- listing(vaas_4, as.groups = list("Species")))
stopifnot(inherits(x, "OPMS_Listing"), is.matrix(x), dim(x) == c(2, 3))
stopifnot(!is.null(rownames(x)), !is.null(colnames(x)))
```

WMD *Virtual classes of the* **opm** *package*

Description

Classes that are virtual and thus are not directly dealt with by an **opm** user: WMD, WMDS, FOE, OPMX and YAML_VIA_LIST.

Details

WMD is an acronym for 'with metadata'. This is a virtual class facilitating the management of metadata. No objects can be created from it because metadata without data make not much sense. It is used by its child classes such as OPM.

Conceptually, this class treats metadata as arbitrarily nested lists with arbitrary content. Containers of objects that inherit from this class are not forced to contain the same metadata entries. Problems might arise if such data are queried and attempted to be converted to, e.g., data frames because some values might be missing. But metadata can be queried beforehand for the keys as well as the values they contain, and other methods support setting, modifying and deleting metadata.

For OPM and the other **opm** classes that use it, 'metadata' refers to information that, in contrast to, e.g., csv_data must be added by the user **after** reading OmniLog® CSV files. Metadata might already be present in YAML files created by the **opm** package, however.

WMDS is virtual class containing a collection of WMD objects (the name WMDS is just the plural of WMD). It shares many methods with WMD but they often return a collection of the return values of the according WMD method.

WMDX is the class union of WMD and WMDS.

FOE is an acronym for 'formula or expression'. This is a virtual class facilitating the implementation of functionality for both formulae and expressions. Methods defined for objects from the class can be applied to either kind of object. See metadata.set and map_metadata for usage examples.

OPMX stands for 'OPM or OPMS'. It is a virtual class containing helper methods mainly for plotting OPM and OPMS objects. See show and sort for usage examples.

Similarly, XOPMX unifies OPMS and MOPMX.

See to_yaml for a usage example of YAML_VIA_LIST. This is a virtual class facilitating the conversion to YAML format (or its subset, JSON). It can currently be used by any class that can be coerced to a list.

See Also

methods::Methods base::matrix base::array base::expression stats::formula

Other classes: MOPMX, OPM, OPMA, OPMA_DB, OPMD, OPMD_DB, OPMS, OPM_DB, OPM_MCP_OUT

Examples

```
showClass("WMD")
showClass("WMDS")
showClass("OPMX")
showClass("XOPMX")
showClass("FOE")
showClass("YAML_VIA_LIST")
```

---

xy_plot *X-Y plot*

Description

Customised plotting of a single PM plate or multiple plates, using xyplot from the **lattice** package.

Usage

```
  ## S4 method for signature 'OPM'
xy_plot(x, col = "midnightblue", lwd = 1,
    neg.ctrl = "A01", base.col = "grey10", base.lwd = lwd,
    main = list(), xlab = "Time [h]", ylab = "Value [OmniLog units]",
    theor.max = TRUE, draw.grid = TRUE,
    strip.fmt = list(), striptext.fmt = list(), rcr = 0.75,
    ...)
  ## S4 method for signature 'OPMS'
xy_plot(x, col = opm_opt("colors"), lwd = 1,
    neg.ctrl = "A01", base.col = "black", base.lwd = lwd,
    main = list(), xlab = "Time [h]", ylab = "Value [OmniLog units]",
    theor.max = TRUE, draw.grid = TRUE, space = "top",
    strip.fmt = list(), striptext.fmt = list(),
    legend.fmt = list(), legend.sep = " ", draw.legend = TRUE, rcr = 0.75,
    ...)
  ## S4 method for signature 'data.frame'
xy_plot(x, f, groups,
    col = opm_opt("colors"), lwd = 1, neg.ctrl = NULL, base.col = "black",
    base.lwd = lwd, main = groups, xlab = elem(f, 3L:2L), ylab = elem(f, 2L),
    draw.grid = TRUE, space = "top", strip.fmt = list(), striptext.fmt = list(),
    legend.fmt = list(), legend.sep = " ", draw.legend = TRUE, rcr = 0.75,
    ...)
```

Arguments

x	OPM or OPMS object.
col	For the OPM method, just a character scalar (colour name) determining the line colour. For the OPMS method, either a character vector with colour codes or one of the arguments of select_colors (for picking one of the predefined colour sets). It is an error if fewer colours are chosen than the number of plate grouping levels (see the ... argument below). For user-chosen colour sets, keep in mind that the sets are not checked for duplicates, and see max_rgb_contrast from the **pkgutils** package as a method for optimally arranging user-defined colours.
lwd	Numeric scalar determining the line width.
neg.ctrl	Determine the height of a horizontal baseline drawn in each panel. • If NULL or FALSE, no baseline will be drawn. • If TRUE, the baseline's height is the value of minmax. • If a character scalar, neg.ctrl is interpreted as the name of the wells regarded as negative control, and the baseline's height becomes the value of minmax applied to these wells only. • Set neg.ctrl to a numeric value for assigning the height directly (at your own risk).
base.col	Character scalar. Baseline colour (ignored if no baseline is drawn).

`base.lwd` Numeric scalar determining the width of the baseline (ignored if no baseline is drawn).

`main` The settings controlling the construction of the main title. If a list, a named list with the following entries (if missing, they are replaced by the respective defaults):

predef Character scalar or expression. Predefined title. If set, the other entries are ignored.

use Logical scalar. If `FALSE`, returns `NULL`.

... Other arguments are passed to `plate_type`.

If `settings` is not a list but a character scalar or an expression, this is used as the `predef` entry of the above-mentioned list. If not a list but a logical scalar, it is used as the 'use' entry of this list. If not a list but a numeric value, it is used as the 'max' entry of this list.

`xlab` Character scalar. Title of x-axis. Use `NULL` to turn it off.

`ylab` Character scalar. Title of y-axis. Use `NULL` to turn it off.

`theor.max` Logical scalar. Use the theoretical maximum as maximum of the y-axis? If `FALSE`, use the empirical maximum with a small offset.

`draw.grid` Logical scalar. Insert background grid?

`space` Character scalar indicating the position of the legend; either 'top', 'bottom', 'left' or 'right'. Might be overwritten by `legend.fmt`.

`strip.fmt` List controlling the format of the description strip above each panel. For instance, the background colour is set using the `bg` key. For further details, see `strip.custom` from the **lattice** package. Note that the **content** of these descriptions is determined by arguments passed from `xy_plot` to `wells`; see there for details.

`striptext.fmt` List controlling the textual description at the top of each panel. For instance, the relative text size is set using the `cex` key, the colour by 'col', the font by 'font' and the number of lines by 'lines'. The latter might be of interest in conjunction with the `paren.sep` argument of `wells`. See the argument `par.strip.text` of `xyplot` from the **lattice** package for details.

`legend.fmt` List controlling where and how to draw the legend. The content of the legend (mainly a description of the assignment of the colours to the curves) is determined automatically. See argument 'key' of `xyplot` from the **lattice** package for details.

`legend.sep` Character scalar. Relevant only if more than one columns of metadata have been selected; will then be used as separator to join their names in the legend.

`draw.legend` Logical scalar. If `FALSE`, no legend is drawn, and the two aforementioned arguments are ignored.

`rcr` Numeric scalar (row-column-ratio) interpreted as number of rows per number of columns. Determines the arrangement of the subplots.

`...` Arguments that are passed to `flatten`. For the `OPMS` method, `include` is particularly important: the selected metadata are joined into a single factor, and the assignment of plates to this factor's levels determines the curve colour for each plate. That is, each combination of metadata entries as chosen using `include`

	yields one colour. If no metadata are selected (the default), each plate gets a colour of its own. Also note that arguments passed via `flatten` to `wells` can be given here which determine the content of the panel description.
`f`	Formula (for the data-frame method).
`groups`	Character vector (for the data-frame method).

Details

The optimal number of rows and columns is estimated from the number of selected wells. An optimal font size of the panel headers is also chosen automatically, but can also be adapted by the user, much like most aspects of the resulting graphics output.

In the case of the `OPMS` method, if metadata are selected, curve colours are determined according to the combinations of these metadata entries, otherwise each plate gets its own colour.

The data-frame method is not intended for phenotype microarray data. It is currently **undocumented** and potentially subject to frequent changes or even removal. Users interested in the method should contact the authors.

Value

An object of class 'trellis'. See `xyplot` from the **lattice** package for details.

References

Sarkar, D. 2008 *Lattice: Multivariate Data Visualization with R*. New York: Springer, 265 p.

Vaas, L. A. I., Sikorski, J., Michael, V., Goeker, M., Klenk H.-P. 2012 Visualization and curve parameter estimation strategies for efficient exploration of Phenotype Microarray kinetics. *PLoS ONE* **7**, e34846.

See Also

lattice::xyplot

Other plotting-functions: `ci_plot`, `heat_map`, `level_plot`, `parallelplot`, `radial_plot`, `summary`

Examples

```
# OPM method
## Not run:
xy_plot(vaas_1) # note the default main title built from the plate type

## End(Not run)

x <- vaas_1[, 11:22]
# Yields a warning message: we have deleted the default negative control.
xy_plot(x)
# Turn the baseline off => no warning:
xy_plot(x, neg.ctrl = NULL)
# Or guess a baseline:
xy_plot(x, neg.ctrl = 100)
# Some like it ugly:
```

```
xy_plot(x, neg.ctrl = 100, col = "pink", base.col = "yellow", main = "Ugly")

# OPMS method
## Not run:
# Colour by species and strain; note default main title
xy_plot(vaas_4, include = c("Species", "Strain"))
# Use the largest of the negative-control maxima as baseline
xy_plot(vaas_4, include = c("Species", "Strain"),
  neg.ctrl = max(vaas_4, "A01"))

## End(Not run)
```

`[` *Select subset*

Description

Select a subset of the `measurements` (and, if present, of the `aggregated` data and the `discretized` data) or plates. Return this subset (or these subsets) together with the other slots (which are unchanged).

Usage

```
  ## S4 method for signature 'MOPMX,ANY,missing,ANY'
x[i, j, drop]
  ## S4 method for signature 'MOPMX,ANY,missing,missing'
x[i, j, drop]
  ## S4 method for signature 'MOPMX,character,missing,ANY'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,character,missing,missing'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,expression,missing,ANY'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,expression,missing,missing'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,formula,missing,ANY'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,formula,missing,missing'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,list,missing,ANY'
x[i, j, drop]
  ## S4 method for signature 'MOPMX,list,missing,missing'
```

```
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,missing,missing,ANY'
x[i, j,
    drop]
  ## S4 method for signature 'MOPMX,missing,missing,missing'
x[i, j,
    drop]
  ## S4 method for signature 'OPM,ANY,ANY,ANY'
x[i, j, ...,
    drop = FALSE]
  ## S4 method for signature 'OPMA,ANY,ANY,ANY'
x[i, j, ...,
    drop = FALSE]
  ## S4 method for signature 'OPMD,ANY,ANY,ANY'
x[i, j, ...,
    drop = FALSE]
  ## S4 method for signature 'OPMS,ANY,ANY,ANY'
x[i, j, k, ...,
    drop = FALSE]
```

Arguments

x	OPM, OPMA or OPMS object.
i	Vector or missing. For the OPM and OPMA method, the indexes of one to several time points. For the OPMS method, the indexes of one to several plates. A warning is issued if indexing goes beyond the range. If i is neither a numeric nor a logical vector, the OPMS method passes it through infix.q to yield a logical vector for indexing. If i is a formula, its left side can be used to choose another infix operator as infix.q. For the MOPMX method, either missing or a vector or a formula or expression. The latter work like for OPMS objects and yield a list of logical vectors (one per element of x) Such a list is then further used for selecting subset within the elements of x. A list can also be provided directly.
j	Vector or missing. • For the OPM and OPMA method, the indexes or names of one to several wells. Can also be a formula, which allows for sequences of well coordinates, which are translated to their positions within the currently present well names. Be aware that this means that the content of a sequence of well coordinates is dependent on x! • For the OPMS method, the indexes of one to several time points. In that case, if j is a list, its values are passed to the respective OPM object separately, allowing for individual choices of time points. Otherwise j is used as the i argument of the OPM and OPMA method.
k	Vector or missing. The OPMS method passes k as j argument of the OPM and OPMA method. That is, in that case *this* parameter selects the wells. See j for details.

...	This should **not** be set. It is an error to specify additional dimensions.
drop	Logical scalar. Remove the aggregated data (and the discretised data, if any) and turn an OPMA or OPMD object to an OPM object? Has no effect if x already is an OPM object or contains only such objects. For the MOPMX method, TRUE means dropping the class and generating a list.

Details

The OPMA method works like the OPM one, but the function applies the subset creation to the original and the aggregated data in parallel. The OPMD method applies the selection also to the discretised data.

The aggregated and discretised data may also be dropped entirely; this might be appropriate if a subset of the time points is selected, potentially yielding aggregated values that do not fit to the measurements anymore.

In contrast to the usual '[' methods, with respect to the measurements this always return a matrix (as a component of the returned object), even if it could be simplified to a vector. The time column is not counted and always copied. It is an error to delete the entire matrix. In all other respects, the OPM method behaves like the '[' methods from the **base** package.

The OPMS method selects a subset of the plates and/or the measurements of the individual plates. It simplifies the outcome to a OPM or OPMA object if only a single plate remains and to NULL if no plate remains. This is different from creating subsets of a list in R. OPMS subset creation rather behaves like subset creation a three-dimensional array with plates as first dimension, time points as second, and wells as third.

Value

OPM, OPMA or OPMS object, or NULL.

See Also

base::'[' base::'[['

Examples

```
## OPM(A) method

# complete dataset, full 96-well plates
(x <- dim(vaas_1))
stopifnot(x == c(384, 96))

# selecting specific wells
copy <- vaas_1[, 11:22]
(x <- dim(copy))
stopifnot(x == c(384, 12))
# indexing with formulae allows for sequences of well coordinates
copy <- vaas_1[, ~ A11:B10] # "A11" is 11th, "B10" is 22th well name
stopifnot(dim(copy) == c(384, 12)) # same result as above
# can also be combined
copy <- vaas_1[, ~ A11:22]
```

```
stopifnot(dim(copy) == c(384, 12)) # same result as above

# dropping aggregated data
copy <- vaas_1[] # normal selection
stopifnot(has_aggr(copy), identical(copy, vaas_1))
copy <- vaas_1[drop = TRUE] # selection with dropping
stopifnot(!has_aggr(copy), !identical(copy, vaas_1))


## OPMS method

# Create OPMS object with fewer plates (the first two ones)
(x <- vaas_4[1:2])
stopifnot(is(x, "OPMS"), dim(x) == c(2, 384, 96))
# we can select the same objects with a formula (which is passed through
# the infix-q operator)
stopifnot(identical(vaas_4[~ Species == "Escherichia coli"], x))
# we can select another infix operator with the left side of the formula
stopifnot(identical(vaas_4[k ~ Species], vaas_4))

# If only a single plate is selected, this is reduced to OPM(A)
x <- vaas_4[3]
stopifnot(!is(x, "OPMS"), dim(x) == c(384, 96))

# Create OPMS object with fewer time points (the first 100 in that case;
# usually this would correspond to the first 25 hours)
x <- vaas_4[, 1:100]
stopifnot(dim(x) == c(4, 100, 96))

# Create OPMS object with fewer wells
(x <- vaas_4[, , 1:12])
stopifnot(dim(x) == c(4, 384, 12))

# The same with well names
x <- vaas_4[, , ~ A01:A12] # within x, these are well names 1 to 12
stopifnot(dim(x) == c(4, 384, 12))
# to do this with a vector, one would need sprintf("A%02i", 1:12)

# Select all plates that have aggregated values
x <- vaas_4[has_aggr(vaas_4)]
stopifnot(identical(x, vaas_4)) # all have such values!

# Traverse all contained OPM objects
for (i in seq(vaas_4)) { # OR: for (i in 1:length(vaas_4))
  x <- vaas_4[i]
  # now do something with 'x'...
  stopifnot(dim(x) == c(384, 96))
}
# see also oapply() for a more elegant approach

## MOPMX method
(x <- new("MOPMX", list(vaas_1, vaas_4))) # create MOPMX object
stopifnot(is(x, "MOPMX"), length(x) == 2)
```

```
(y <- x[~ Species != "Escherichia coli"])
stopifnot(is(y, "MOPMX"), length(y) == 1)
(y <- x[list(1, 3:4)]) # only 2nd element reduced
stopifnot(is(y, "MOPMX"), length(y) == 2, !identical(x, y))
```

[<- *Assign subset*

Description

Assign subsets of OPMS objects.

Usage

```
  ## S4 replacement method for signature 'MOPMX,ANY'
x$name <- value

  ## S4 replacement method for signature 'MOPMX,ANY,missing,ANY'
x[i, ...] <- value
  ## S4 replacement method for signature 'OPMS,ANY,missing,NULL'
x[i, j] <- value
  ## S4 replacement method for signature 'OPMS,ANY,missing,OPM'
x[i, j] <- value
  ## S4 replacement method for signature 'OPMS,ANY,missing,OPMS'
x[i, j] <- value
  ## S4 replacement method for signature 'OPMS,ANY,missing,list'
x[i, j] <- value

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

Arguments

x	OPMS or MOPMX object.
i	One to several plate indexes. Should be compatible with the length of value. Otherwise any resulting NULL elements will be removed (with a warning), causing the resulting plate indexes to be unequal to i, which might be confusing.
j	Must **not** be set. See the examples.
...	Must neither be used.
name	Unevaluated symbol used for as index of a single element.
value	Value to be assigned. NULL causes the selected plates to be removed. Alternatively, OPM or OPMS objects or lists of OPM objects can be assigned. All assignments are subject to the restrictions explained in the help entries of the OPMS and MOPMX classes.

Value

value.

See Also

Other combination-functions: c, opms, plus

Examples

```
copy <- vaas_4
copy[5] <- NULL # has no effect
stopifnot(identical(vaas_4, copy))
length(copy)
copy[2:3] <- NULL # removes these plates
length(copy)
stopifnot(length(vaas_4) == length(copy) + 2)
copy[1:4] <- vaas_4 # set all plates to the plates from 'vaas_4'
stopifnot(identical(vaas_4, copy))
copy[3] <- copy[3] # no change
stopifnot(identical(vaas_4, copy))
copy[3] <- copy[2] # now assign other plate
stopifnot(!identical(vaas_4, copy))
copy[6] <- copy[1] # gaps will be closed
stopifnot(length(copy) == 5) # not 6
```

%k% *Query metadata keys*

Description

Search for the presence of metadata keys, either using a vector, factor, list, formula, expression or WMD object.

Usage

```
  ## S4 method for signature 'ANY,MOPMX'
x %K% table
  ## S4 method for signature 'MOPMX,ANY'
x %K% table
  ## S4 method for signature 'WMD,ANY'
x %K% table
  ## S4 method for signature 'WMD,WMD'
x %K% table
  ## S4 method for signature 'WMD,WMDS'
x %K% table
  ## S4 method for signature 'WMDS,ANY'
x %K% table
  ## S4 method for signature 'character,WMD'
```

```
x %K% table
  ## S4 method for signature 'character,WMDS'
x %K% table
  ## S4 method for signature 'expression,WMD'
x %K% table
  ## S4 method for signature 'expression,WMDS'
x %K% table
  ## S4 method for signature 'factor,WMD'
x %K% table
  ## S4 method for signature 'factor,WMDS'
x %K% table
  ## S4 method for signature 'formula,WMD'
x %K% table
  ## S4 method for signature 'formula,WMDS'
x %K% table
  ## S4 method for signature 'list,WMD'
x %K% table
  ## S4 method for signature 'list,WMDS'
x %K% table

  ## S4 method for signature 'ANY,MOPMX'
x %k% table
  ## S4 method for signature 'MOPMX,ANY'
x %k% table
  ## S4 method for signature 'WMD,ANY'
x %k% table
  ## S4 method for signature 'WMD,WMD'
x %k% table
  ## S4 method for signature 'WMD,WMDS'
x %k% table
  ## S4 method for signature 'WMDS,ANY'
x %k% table
  ## S4 method for signature 'character,WMD'
x %k% table
  ## S4 method for signature 'character,WMDS'
x %k% table
  ## S4 method for signature 'expression,WMD'
x %k% table
  ## S4 method for signature 'expression,WMDS'
x %k% table
  ## S4 method for signature 'factor,WMD'
x %k% table
  ## S4 method for signature 'factor,WMDS'
x %k% table
  ## S4 method for signature 'formula,WMD'
x %k% table
  ## S4 method for signature 'formula,WMDS'
x %k% table
```

```
  ## S4 method for signature 'list,WMD'
x %k% table
  ## S4 method for signature 'list,WMDS'
x %k% table
```

Arguments

x	Character vector, factor, list, formula, expression or WMD object used as query. See 'Details'. x and table can swap their places.
table	WMD, WMDS or MOPMX object. x and table can swap their places.

Details

The behaviour of these methods depends on the object used as query. infix.largek is usually stricter than infix.k, sometimes equivalent.

- Using a character vector as query, infix.k tests whether all given keys are present in the top-level names of the metadata (these may be nested, but all contained lists are ignored here). An empty query vector results in TRUE. Note that the values of the character vector, not its names, if any, are used for querying the metadata. In contrast, infix.largek tests whether a given key is present in the metadata and fetches an object that is not NULL. If the key has a length > 1, contained lists are queried.
- Using a list as query, both methods tests whether all given keys are present in the names of the metadata. This works like the character method, but because a query list is given, the comparison of keys can be applied recursively (by using, of course, a nested query list). This is based on contains with the values argument set to FALSE.
- When supplying a WMD object as query, its metadata will be used in a list query.
- The factor method first converts x to 'character' mode.
- The formula method attempts to evaluate the right side of the formula in the context of the metadata of table and returns whether or not this fails (yields an error). Symbols that are not found within the metadata are looked up in the enclosing environment infix.k or only in the base environment infix.largek. But note also that missing objects are not the only potential reason of failure.
- The expression method works like the formula method, using the entire expression in place of the right side of the formula.

See subset for usage examples with OPMS objects.

Value

Logical vector of the length of the WMD or WMDS object. For MOPMX objects, a list of such vectors.

Examples

```
# The data set contains the metadata keys 'Species' and 'Experiment' but
# neither 'Trial' nor 'Organism' nor 'Run':
# In the following we use stopifnot(), which fails unless all arguments
# passed are TRUE.
```

```
## Character methods

# Zero-element queries
stopifnot(character() %k% vaas_1) # always results
stopifnot(character() %K% vaas_1)

# Single-element queries
stopifnot("Experiment" %k% vaas_1) # present
stopifnot("Experiment" %K% vaas_1) # present
stopifnot("Species" %k% vaas_1) # present
stopifnot("Species" %K% vaas_1)  # present
stopifnot(!"Run" %k% vaas_1) # missing
stopifnot(!"Run" %K% vaas_1) # missing
stopifnot(!"Organism" %k% vaas_1) # missing
stopifnot(!"Trial" %K% vaas_1) # missing

# Multi-element queries
stopifnot(!c("Species", "Trial") %k% vaas_1) # only one present
stopifnot(!c("Organism", "Experiment") %k% vaas_1) # only one present
stopifnot(c("Species", "Experiment") %k% vaas_1) # all present
# querying with %K% and vectors of length > 1 mean nested queries; compare
# this to the behaviour of %k%!
stopifnot(!c("Species", "Experiment") %K% vaas_1)
# i.e. "Experiment" is not within "Species".

## List methods

stopifnot(list(Experiment = "whatever") %k% vaas_1) # key present
stopifnot(list(Species = "ignored") %k% vaas_1) # key present

stopifnot(vaas_1 %k% vaas_1) # obviously
stopifnot(vaas_1 %K% vaas_1)

# This fails because we query with a named 2nd-order list but the 'Species'
# metadata entry is not even a list.
stopifnot(!list(Species = list(Genus = "X", Epithet = "Y")) %k% vaas_1)

# This is OK because we query with an unnamed 2nd-order list: it has no
# names that one would fail to find.
stopifnot(list(Species = list("X", "Y")) %k% vaas_1)

# More non-nested query examples
stopifnot(!list(Run = 99) %k% vaas_1) # key not present
stopifnot(list(Species = "?", Experiment = NA) %k% vaas_1) # keys present
stopifnot(!list(Species = "?", Trial = NA) %k% vaas_1) # one key missing
stopifnot(!list(Organism = "?", Experiment = NA) %k% vaas_1) # likewise
stopifnot(list() %k% vaas_1) # empty query always results

# Formulae for querying, compare with list examples above
stopifnot((~ Experiment) %k% vaas_1) # key present
stopifnot((~ Experiment) %K% vaas_1)
stopifnot(vaas_1 %k% ~ Experiment) # key present, no parens needed
stopifnot(vaas_1 %K% ~ Experiment)
```

```
stopifnot(vaas_1 %k% ~ Species) # key present, no parens needed
stopifnot(vaas_1 %K% ~ Species)
stopifnot(!vaas_1 %k% ~ Species$Epithet) # nested key not present
stopifnot(!vaas_1 %K% ~ Species$Epithet)
stopifnot(!vaas_1 %k% ~ missing.name) # key not present
stopifnot(!vaas_1 %K% ~ missing.name)
missing.name <- "abc"
stopifnot(vaas_1 %k% ~ missing.name) # key found in enclosing environment
stopifnot(!vaas_1 %K% ~ missing.name) # enclosing environment ignored
rm(missing.name) # tidy up
```

%q% *Query metadata*

Description

Search for the presence of metadata values for given keys, either using a vector, factor, list, formula, expression or WMD object.

Usage

```
  ## S4 method for signature 'ANY,MOPMX'
x %Q% table
  ## S4 method for signature 'MOPMX,ANY'
x %Q% table
  ## S4 method for signature 'WMD,ANY'
x %Q% table
  ## S4 method for signature 'WMD,WMD'
x %Q% table
  ## S4 method for signature 'WMD,WMDS'
x %Q% table
  ## S4 method for signature 'WMDS,ANY'
x %Q% table
  ## S4 method for signature 'character,WMD'
x %Q% table
  ## S4 method for signature 'character,WMDS'
x %Q% table
  ## S4 method for signature 'expression,WMD'
x %Q% table
  ## S4 method for signature 'expression,WMDS'
x %Q% table
  ## S4 method for signature 'factor,WMD'
x %Q% table
  ## S4 method for signature 'factor,WMDS'
x %Q% table
  ## S4 method for signature 'formula,WMD'
x %Q% table
  ## S4 method for signature 'formula,WMDS'
```

```
x %Q% table
  ## S4 method for signature 'list,WMD'
x %Q% table
  ## S4 method for signature 'list,WMDS'
x %Q% table

  ## S4 method for signature 'ANY,MOPMX'
x %q% table
  ## S4 method for signature 'MOPMX,ANY'
x %q% table
  ## S4 method for signature 'WMD,ANY'
x %q% table
  ## S4 method for signature 'WMD,WMD'
x %q% table
  ## S4 method for signature 'WMD,WMDS'
x %q% table
  ## S4 method for signature 'WMDS,ANY'
x %q% table
  ## S4 method for signature 'character,WMD'
x %q% table
  ## S4 method for signature 'character,WMDS'
x %q% table
  ## S4 method for signature 'expression,WMD'
x %q% table
  ## S4 method for signature 'expression,WMDS'
x %q% table
  ## S4 method for signature 'factor,WMD'
x %q% table
  ## S4 method for signature 'factor,WMDS'
x %q% table
  ## S4 method for signature 'formula,WMD'
x %q% table
  ## S4 method for signature 'formula,WMDS'
x %q% table
  ## S4 method for signature 'list,WMD'
x %q% table
  ## S4 method for signature 'list,WMDS'
x %q% table
```

Arguments

x	Character vector, factor, list, formula, expression or WMD object used as query. See 'Details'. x and table can swap their places.
table	WMD, WMDS or MOPMX object. x and table can swap their places.

Details

The behaviour of these methods depends on the object used as query. infix.largeq is usually stricter than infix.q, sometimes equivalent.

- Using a character vector as query, this tests whether all given query keys are present in the top-level names of the metadata and refer to the same query elements. The `names` of the vector are used to select elements from the top level of the metadata. When using `infix.q`, these elements are then converted to 'character' mode before comparison with the values of `x`. A non-empty vector without a `names` attribute is accepted but will always yield `FALSE`. In contrast, an entirely empty vector yields `TRUE`.
- Using a list, a non-exact query with a query list is conducted. The comparison is applied recursively using `contains` with the `values` argument set to `TRUE` and `exact` set to either `FALSE` (`infix.q`) or `TRUE` (`infix.largeq`). The latter might be too strict for most applications. The main advantage of using a list over the character-based search is that it allows for a nested query.
- When supplying a `WMD` object as query, its metadata will be used in a list query.
- The factor method first converts `x` to 'character' mode.
- The formula method attempts to evaluate the right side of the formula in the context of the metadata of `table` and returns the result. For the `WMD` method, it is up to the user to ensure that the result is a logical scalar, but the method would succeed anyway. The `WMDS` method yields an error unless each plate yields a logical scalar. Symbols that are not found within the metadata are looked up in the enclosing environment (`infix.q`) or only in the base environment (`infix.largeq`). The former approach is less strict. Because of missing objects and other reasons the method might nevertheless fail.
- The expression method works like the formula method, using the entire expression in place of the right side of the formula.

See `subset` for usage examples with `OPMS` objects.

Value

Logical vector of the length of the `WMD` or `WMDS` object. For `MOPMX` objects, a list of such vectors.

Examples

```
# The data set vaas_1 contains the metadata keys 'Species' and 'Experiment'
# with the values 'Escherichia coli' and 'First replicate'.

## Character methods

stopifnot(!"Experiment" %q% vaas_1) # wrong query here; compare to %k%
stopifnot(!"First replicate" %q% vaas_1) # again wrong query
stopifnot(c(Experiment = "First replicate") %q% vaas_1) # correct query
stopifnot(c(Experiment = "First replicate") %Q% vaas_1)

stopifnot(!"Species" %q% vaas_1) # wrong query
stopifnot(!"Escherichia coli" %q% vaas_1) # wrong query
stopifnot(c(Species = "Escherichia coli") %q% vaas_1) # correct query

# This does not work because the value has the wrong type
stopifnot(!c(`Plate number` = "6") %Q% vaas_1)
# Compare to %q%
stopifnot(c(`Plate number` = "6") %q% vaas_1)
```

```
stopifnot(c(Species = "Escherichia coli",
  Experiment = "First replicate") %q% vaas_1) # combined query, all TRUE
stopifnot(c(Species = "Escherichia coli",
  Experiment = "First replicate") %Q% vaas_1) # all present

stopifnot(character() %q% vaas_1) # empty query always results
stopifnot(character() %Q% vaas_1)

## List methods

stopifnot(list(Experiment = "First replicate") %q% vaas_1)
stopifnot(list(Experiment = "First replicate") %Q% vaas_1) # present

# Choice among alternatives
stopifnot(list(Experiment = c("First replicate",
  "Second replicate")) %q% vaas_1) # one of them TRUE
stopifnot(!list(Experiment = c("Second replicate",
  "Third replicate")) %q% vaas_1) # none of them TRUE

# Combined query together with choice among alternatives
stopifnot(list(Experiment = c("First replicate", "Second replicate"),
  Species = c("Escherichia coli", "Bacillus subtilis")) %q% vaas_1)

# Choice among alternatives is not done here: this query fails unless this
# two-element vector is contained. Compare to %q%.
stopifnot(!list(Experiment = c("First replicate",
  "Second replicate")) %Q% vaas_1)

stopifnot(list() %q% vaas_1) # empty query always results
stopifnot(list() %Q% vaas_1)

stopifnot(vaas_1 %q% vaas_1) # obviously
stopifnot(vaas_1 %Q% vaas_1)

## Formulae for querying

stopifnot((~ Experiment == "First replicate") %q% vaas_1)
stopifnot((~ Experiment == "First replicate") %Q% vaas_1)
stopifnot(vaas_1 %q% ~ Experiment == "First replicate")
stopifnot(vaas_1 %Q% ~ Experiment == "First replicate")
stopifnot(vaas_1 %q% ~ Species == "Escherichia coli")
stopifnot(vaas_1 %Q% ~ Species == "Escherichia coli")
stopifnot(vaas_1 %q% ~ Species != "Bacillus subtilis")
stopifnot(vaas_1 %Q% ~ Species != "Bacillus subtilis")

x <- try(vaas_1 %q% ~ missing.name == "abc", silent = TRUE) # fails
stopifnot(inherits(x, "try-error"))
x <- try(vaas_1 %Q% ~ missing.name == "abc", silent = TRUE) # also fails
stopifnot(inherits(x, "try-error"))
missing.name <- "abc"  # enclosing environment considered or ignored
stopifnot(vaas_1 %q% ~ missing.name == "abc")
x <- try(vaas_1 %Q% ~ missing.name == "abc", silent = TRUE) # still fails
```

```
    stopifnot(inherits(x, "try-error"))
    rm(missing.name) # tidy up

    # examples for OPMS methods are given under subset()
```

Index