Package 'yamlet'

October 11, 2024

Type Package

Title Versatile Curation of Table Metadata

Version 1.1.3

Maintainer Tim Bergsma <bergsmat@gmail.com>

BugReports https://github.com/bergsmat/yamlet/issues

Description A YAML-based
mechanism for working with table metadata. Supports
compact syntax for creating, modifying, viewing, exporting,
importing, displaying, and plotting metadata coded as column
attributes. The 'yamlet' dialect is valid 'YAML' with
defaults and conventions chosen to improve readability.
See ?yamlet, ?decorate, ?modify, ?io_csv, and ?ggplot.decorated.

License GPL-3

Encoding UTF-8

Imports yaml, csv (>= 0.6.2), encode, units, spork (>= 0.3.3),
ggplot2, scales, dplyr (>= 1.1.0), rlang, xtable, tidyr, vctrs,
pillar, knitr

RoxygenNote 7.3.2

VignetteBuilder knitr

Suggests testthat (>= 2.1.0), magrittr, table1, rmarkdown, gridExtra,
haven, tablet (>= 0.6.8), kableExtra

NeedsCompilation no

Author Tim Bergsma [aut, cre]

Repository CRAN

Date/Publication 2024-10-11 04:30:02 UTC

Contents

`as.integer.classified` *Coerce Classified to Integer*

Description

Coerces classified to integer. Result is like `as.integer(as.numeric(x)) + offset` but has a guide attribute: a list of integers whose names are the original levels of x. If you need a simple integer, consider coercing first to numeric.

Usage

```
## S3 method for class 'classified'
as.integer(
  x,
  offset = 0L,
  ...,
  persistence = getOption("yamlet_persistence", TRUE)
)
```

Arguments

x	classified, see classified
offset	an integer value to add to intermediate result
...	passed to desolve
persistence	whether to return 'dvec' (is.integer(): TRUE) or just integer.

Value

integer (possibly of class dvec)

See Also

Other classified: [.classified(), [<-.classified(), [[.classified(), [[<-.classified(), c.classified(), classified(), classified.classified(), classified.data.frame(), classified.default(), classified.dvec(), classified.factor(), desolve.classified(), unclassified(), unclassified.classified(), unclassified.data.frame()

Examples

```
library(magrittr)

# create factor with codelist attribute
classified(c('knife','fork','spoon'))

# give back a simple numeric
classified(c('knife','fork','spoon')) %>% as.numeric

# intentionally preserve levels as 'guide' attribute
classified(c('knife','fork','spoon')) %>% as.integer

# implement offset
classified(c('knife','fork','spoon')) %>% as.integer(-1)

# globally defeat the 'persistence' paradigm
options(yamlet_persistence = FALSE)
c('knife','fork','spoon') %>%
  classified %>%
  as.integer %>%
  class # integer

# remove option to restore default persistence paradigm
options(yamlet_persistence = NULL)
c('knife','fork','spoon') %>%
  classified %>%
  as.integer %>%
  class # dvec

# locally defeat persistence paradigm
c('knife','fork','spoon') %>%
  classified %>%
```

```
  as.integer(persistence = FALSE) %>%
  class # integer
```

`as_dvec.units` *Coerce Units to Decorated Vector*

Description

Coerces units to dvec.

Usage

```
## S3 method for class 'units'
as_dvec(x, ...)
```

Arguments

`x`	units
`...`	passed arguments

Examples

```
library(magrittr)
library(dplyr)
a <- data.frame(id = 1:4, wt = c(70, 80, 70, 80), sex = c(0,1,0,1))
a %<>% decorate('wt: [ body weight, kg ]')
a %<>% decorate('sex: [ sex, [ female: 0, male: 1]]')
a %<>% decorate('id: identifier')
a %<>% resolve
a %<>% mutate(wt = as_units(wt))
a %<>% mutate(wt = as_dvec(wt))
str(a$wt)
```

`as_units.dvec` *Coerce Decorated Vector to Units*

Description

Coerces dvec to units. If x has a units attribute, it is used to create class 'units'. It is an error if x has no units attribute.

Usage

```
## S3 method for class 'dvec'
as_units(x, ..., preserve = getOption("yamlet_as_units_preserve", "label"))
```

Arguments

x	dvec
...	ignored
preserve	attributes to preserve; just label by default (class and units are handled implicitly)

Examples

```
library(magrittr)
a <- data.frame(id = 1:4, wt = c(70, 80, 70, 80), sex = c(0,1,0,1))
a %<>% decorate('wt: [ body weight, kg ]')
a %<>% decorate('sex: [ sex, [ female: 0, male: 1]]')
a %<>% decorate('id: identifier')
a %<>% resolve
a$wt %>% as_units
```

---

canonical.decorated *Sort Decorations*

Description

Enforces canonical attribute order for class 'decorated'. Set of default_keys will be augmented with all observed attribute names and will be expanded or reduced as necessary for each data item.

Usage

```
## S3 method for class 'decorated'
canonical(
  x,
  default_keys = getOption("yamlet_default_keys", list("label", "guide")),
  ...
)
```

Arguments

x	decorated
default_keys	attribute names in preferred order
...	ignored

Value

decorated

See Also

Other canonical: canonical(), canonical.yamlet()

Other interface: classified.data.frame(), decorate.character(), decorate.data.frame(), desolve.decorated(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Examples

```
# make some decorated data
library(magrittr)
x <- data.frame(x = 1, y = 1, z = factor('a'))
x %<>% decorate('
x: [ guide: mm, desc: this, label: foo ]
"y": [ guide: bar, desc: other ]
')

# retrieve decorations: label not first!
decorations(x)

# sort label first by default
decorations(canonical(x))

# equivalent invocation
canonical(decorations(x))
```

---

classified.default *Create Classified by Default*

Description

Creates a factor of subclass 'classified', for which there are attribute-preserving methods. In particular, classified has a codelist attribute indicating the origin of its levels: it is constructed from the codelist attribute of x if available, or from 'levels' and 'labels' by default. Unlike the case for factor, length of labels cannot be one (i.e., different from length of levels).

Usage

```
## Default S3 method:
classified(
  x = character(),
  levels,
  labels,
  exclude = NA,
  ordered = is.ordered(x),
```

```
  nmax = NA,
  token = character(0),
  ...
)
```

Arguments

x	see factor
levels	see factor
labels	see factor, must have same length as levels
exclude	see factor
ordered	see factor
nmax	see factor
token	informative label for messages
...	ignored

Value

'classified' 'factor'

See Also

Other classified: [.classified(), [<-.classified(), [[.classified(), [[<-.classified(), as.integer.classified(), c.classified(), classified(), classified.classified(), classified.data.frame(), classified.dvec(), classified.factor(), desolve.classified(), unclassified(), unclassified.classified(), unclassified.data.frame()

Examples

```
# classified creates a factor with a corresponding codelist attribute
classified(c('a','b','c'))

# codelist 'remembers' the origins of levels
classified(c('a','b','c'), labels = c('A','B','C'))

# classified is 'reversible'
library(magrittr)
c('a','b','c') %>%
  classified(labels = c('A','B','C')) %>%
  unclassified
```

`classified.factor` *Create Classified from Factor*

Description

Creates classified from factor. Uses `classified.default`, but supplies existing levels by default.

Usage

```
## S3 method for class 'factor'
classified(
  x = character(),
  levels,
  labels,
  exclude = NA,
  ordered = is.ordered(x),
  nmax = NA,
  token = character(0),
  ...
)
```

Arguments

`x`	see `factor`
`levels`	passed to `classified.default`; defaults to `levels(x)`
`labels`	passed to `classified.default`; must be same length as levels(after removing values in `exclude`) and must not contain duplicates
`exclude`	see `factor`
`ordered`	see `factor`
`nmax`	see `factor`
`token`	informative label for messages
`...`	ignored

Value

'classified' 'factor'

See Also

Other classified: `[.classified()`, `[<-.classified()`, `[[.classified()`, `[[<-.classified()`, `as.integer.classified()`, `c.classified()`, `classified()`, `classified.classified()`, `classified.data.frame()`, `classified.default()`, `classified.dvec()`, `desolve.classified()`, `unclassified()`, `unclassified.classified()`, `unclassified.data.frame()`

Examples

```
a <- factor(c('c','b','a'))
levels(classified(a))
attr(classified(a), 'codelist')
```

`decorate.character` *Decorate Character*

Description

Treats x as a file path. By default, metadata is sought from a file with the same base but the 'yaml' extension.

Usage

```
## S3 method for class 'character'
decorate(
  x,
  meta = NULL,
  ...,
  read = getOption("yamlet_import", as.csv),
  ext = getOption("yamlet_extension", ".yaml")
)
```

Arguments

x	file path for table data
meta	file path for corresponding yamlet metadata, or a yamlet object
...	passed to read (if accepted) and to as_yamlet.character
read	function or function name for reading x
ext	file extension for metadata file, if relevant

Value

class 'decorated' 'data.frame'

See Also

Other decorate: as_decorated(), as_decorated.default(), decorate(), decorate.data.frame(), decorate.list(), decorate_groups(), decorate_groups.data.frame(), decorations(), decorations.data.frame(decorations_groups(), decorations_groups.data.frame(), group_by_decorations(), group_by_decorations.dat redecorate()

Other interface: canonical.decorated(), classified.data.frame(), decorate.data.frame(), desolve.decorated(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Examples

```
# find data file
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
file

# find metadata file
meta <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
meta

# decorate with explicit metadata reference
a <- decorate(file, meta)

# rely on default metadata path
b <- decorate(file)

# in this case: same
stopifnot(identical(a, b))
```

`decorate.data.frame` *Decorate Data Frame*

Description

Decorates a data.frame. Expects metadata in yamlet format, and loads it onto columns as attributes.

Usage

```
## S3 method for class 'data.frame'
decorate(
  x,
  meta = NULL,
  ...,
  persistence = getOption("yamlet_persistence", TRUE)
)
```

Arguments

`x`	data.frame
`meta`	file path for corresponding yaml metadata, or a yamlet; an attempt will be made to guess the file path if x has a 'source' attribute
`...`	passed to `decorate.list`
`persistence`	whether to coerce decorated columns to 'dvec' where suitable method exists

Details

As of v0.8.8, the data.frame method for decorate() coerces affected columns using `as_dvec` if `persistence` is true and a suitable method exists. 'vctrs' methods are implemented for class dvec to help attributes persist during tidyverse operations. Details are described in `c.dvec`. Disable this functionality with `options(yamlet_persistence = FALSE)`.

Value

class 'decorated' 'data.frame'

See Also

decorate.list

Other interface: canonical.decorated(), classified.data.frame(), decorate.character(), desolve.decorated(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Other decorate: as_decorated(), as_decorated.default(), decorate(), decorate.character(), decorate.list(), decorate_groups(), decorate_groups.data.frame(), decorations(), decorations.data.frame(decorations_groups(), decorations_groups.data.frame(), group_by_decorations(), group_by_decorations.dat redecorate()

Examples

```
# find data path
library(csv)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
file
dat <- as.csv(file) # dat now has 'source' attribute

# use source attribute to find metadata
a <- decorate(as.csv(file))

# supply metadata path (or something close) explicitly
b <- decorate(dat, meta = file)

# these are equivalent
stopifnot(identical(a, b))
```

---

decorate_groups.data.frame

Capture Groups as Decorations for Data Frame

Description

Captures groups as decorations for class 'data.frame'. Creates a sequentially-valued integer attribute with name 'groups' for each corresponding column (after clearing all such existing designations). It is an error if not all such columns are present. Defaults to `groups(x)`. If no columns are specified and x has no groups, x is returned with any existing column-level 'groups' attributes removed.

Usage

```
## S3 method for class 'data.frame'
decorate_groups(x, ...)
```

Arguments

x	data.frame
...	unquoted names of columns to assign as groups; defaults to groups(x)

Value

same class as x

See Also

Other decorate: as_decorated(), as_decorated.default(), decorate(), decorate.character(), decorate.data.frame(), decorate.list(), decorate_groups(), decorations(), decorations.data.frame(), decorations_groups(), decorations_groups.data.frame(), group_by_decorations(), group_by_decorations.dat redecorate()

Examples

```
library(magrittr)
library(dplyr)
Theoph %>% decorate_groups(Subject, Time) %>% groups # nothing!
Theoph %>% decorate_groups(Subject, Time) %>% decorations # note well
Theoph %>% group_by(Subject, Time) %>% decorate_groups %>% decorations # same
```

---

decorations.data.frame

Retrieve Decorations for Data Frame

Description

Retrieve the decorations of a data.frame; i.e., the metadata used to decorate it. Returns a list with same names as the data.frame. By default, 'class' and 'level' attributes are excluded from the result, as you likely don't want to manipulate these independently.

Usage

```
## S3 method for class 'data.frame'
decorations(
  x,
  ...,
  exclude_attr = getOption("yamlet_exclude_attr", c("class", "levels"))
)
```

Arguments

x	data.frame
...	optional unquoted column names to limit output (passed to select)
exclude_attr	attributes to remove from the result

Value

named list of class 'yamlet'

See Also

Other decorate: as_decorated(), as_decorated.default(), decorate(), decorate.character(), decorate.data.frame(), decorate.list(), decorate_groups(), decorate_groups.data.frame(), decorations(), decorations_groups(), decorations_groups.data.frame(), group_by_decorations(), group_by_decorations.data.frame(), redecorate()

Examples

```
# prepare a decorated data.frame
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)

# retrieve the decorations
decorations(x, Subject, time, conc)
```

---

decorations_groups.data.frame

Recover Groups Decorations for Data Frame

Description

Recovers groups decorations for class 'data.frame'. Seeks a sequentially-valued integer attribute with name 'groups' for each column, sorts these, and returns a character vector like group_vars(x).

Usage

```
## S3 method for class 'data.frame'
decorations_groups(x, ...)
```

Arguments

x	data.frame
...	ignored

Value

character: names of groups columns

See Also

Other decorate: as_decorated(), as_decorated.default(), decorate(), decorate.character(), decorate.data.frame(), decorate.list(), decorate_groups(), decorate_groups.data.frame(), decorations(), decorations.data.frame(), decorations_groups(), group_by_decorations(), group_by_decorations.data.frame(), redecorate()

Examples

```
library(magrittr)
library(dplyr)
Theoph %<>% group_by(Subject, Time)
Theoph %>% group_vars
Theoph %>% decorations_groups # nothing!
Theoph %<>% decorate_groups
Theoph %>% decorations_groups # something!
Theoph %<>% ungroup
Theoph %>% group_vars # gone!
Theoph %<>% group_by(across(all_of(decorations_groups(.))))
Theoph %>% group_vars # recovered!
Theoph %<>% group_by_decorations
Theoph %>% group_vars # same
rm(Theoph)
```

---

desolve.classified *Desolve Guide for Classified*

Description

Un-resolves explicit usage of default key 'guide' to implicit usage for class 'classified'. Calls drop_title (a non-action by default), unclassified, followed by implicit_guide.

Usage

```
## S3 method for class 'classified'
desolve(x, ...)
```

Arguments

x	classified
...	passed to drop_title, unclassified, and unclassified

Value

dvec

See Also

Other resolve: desolve(), desolve.data.frame(), desolve.decorated(), desolve.dvec(), resolve(), resolve.classified(), resolve.data.frame(), resolve.decorated(), resolve.dvec()

Other classified: [.classified(), [<-.classified(), [[.classified(), [[<-.classified(), as.integer.classified(), c.classified(), classified(), classified.classified(), classified.data.frame(), classified.default(), classified.dvec(), classified.factor(), unclassified(), unclassified.classified(), unclassified.data.frame()

Examples

```
library(magrittr)
x <- as_dvec(
  4:6,
  guide = list(a = 4L, b = 5L, c = 6L)
)

# untouched
x %>% str

# resolved
x %>% resolve %>% str

# resolved and desolved
x %>% resolve %>% desolve %>% str
```

---

desolve.decorated *Desolve Guide for Decorated*

Description

Un-resolves explicit usage of default key 'guide' to implicit usage for 'decorated' class. Simply calls drop_title, unclassified, and implicit_guide.

Usage

```
## S3 method for class 'decorated'
desolve(x, ...)
```

Arguments

x	decorated
...	passed to drop_title, unclassified, and implicit_guide

Value

decorated

See Also

Other resolve: desolve(), desolve.classified(), desolve.data.frame(), desolve.dvec(), resolve(), resolve.classified(), resolve.data.frame(), resolve.decorated(), resolve.dvec()

Other interface: canonical.decorated(), classified.data.frame(), decorate.character(), decorate.data.frame(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Examples

```
library(magrittr)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)

# this is how Age, glyco, Race look when resolved
x %>% resolve %>% decorations(Age, glyco, Race)

# we can resolve two of them and then 'unresolve' all of them
x %>% resolve(glyco, Race) %>% desolve %>% decorations(Age, glyco, Race)
```

---

ggplot.decorated *Create a New ggplot for a Decorated Data Frame*

Description

Creates a new ggplot object for a decorated data.frame. This is the ggplot() method for class 'decorated'. It creates a ggplot object using the default method, but reclassifies it as 'decorated_ggplot' so that a custom print method is invoked; see print.decorated_ggplot.

Usage

```
## S3 method for class 'decorated'
ggplot(data, ...)
```

Arguments

data	decorated, see decorate
...	passed to ggplot

Details

This approach is similar to but more flexible than the method for ggready. For fine control, you can switch between 'data.frame' and 'decorated' using as_decorated (supplies null decorations) and as.data.frame (preserves decorations).

Value

return value like ggplot but inheriting 'decorated_ggplot'

See Also

decorate resolve ggready

Other decorated_ggplot: ggplot_build.decorated_ggplot(), print.decorated_ggplot()

Other interface: canonical.decorated(), classified.data.frame(), decorate.character(), decorate.data.frame(), desolve.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
library(ggplot2)
library(dplyr)
library(magrittr)
# par(ask = FALSE)

x <- decorate(file)
x %<>% filter(!is.na(conc))

# Manipulate class to switch among ggplot methods.
class(x)
class(data.frame(x))
class(as_decorated(data.frame(x)))

# The bare data.frame gives boring labels and un-ordered groups.
map <- aes(x = time, y = conc, color = Heart)
data.frame(x) %>% ggplot(map) + geom_point()

# Decorated data.frame uses supplied labels.
# Notice CHF levels are still not ordered. (Moderate first.)
x %>% ggplot(map) + geom_point()

# If we resolve Heart, CHF levels are ordered.
x %<>% resolve(Heart)
x %>% ggplot(map) + geom_point()

# We can map aesthetics as decorations.
x %<>% decorate('Heart: [ color: [gold, purple, green]]')
x %>% ggplot(map) + geom_point()

# Colors are matched to particular levels. Purple drops out here:
x %>% filter(Heart != 'Moderate') %>% ggplot(map) + geom_point()

# We can resolve other columns for a chance to enrich the output with units.
x %<>% resolve
```

```
suppressWarnings( # because this complains for columns with no units
  x <- modify(x, title = paste0(label, '\n(', units, ')'))
)
x %>% ggplot(map) + geom_point()

# Or something fancier.
x %<>% modify(conc, title = 'conc_serum. (mg*L^-1.)')
x %>% ggplot(map) + geom_point()

# The y-axis title is deliberately given in spork syntax for elegant coercion:
library(spork)
x %<>% modify(conc, expression = as.expression(as_plotmath(as_spork(title))))
x %>% ggplot(map) + geom_point()
# Add a fancier label for Heart, and facet by a factor:
x %<>% modify(Heart, expression = as.expression(as_plotmath(as_spork('CHF^\\*'))))
x %>% ggplot(map) + geom_point() + facet_wrap(~Creatinine)

# ggready handles the units and plotmath implicitly for a 'standard' display:
x %>% ggready %>% ggplot(map) + geom_point() + facet_wrap(~Creatinine)

# Notice that instead of over-writing the label
# attribute, we are creating a stack of label
# substitutes (title, expression) so that
# label is still available as an argument
# if we want to try something else.  The
# print method by default looks for all of these.
# Precedence is expression, title, label, column name.
# Precedence can be controlled using
# options(decorated_ggplot_search = c(a, b, ...) ).

# Here we try a dataset with conditional labels and units.

file <- system.file(package = 'yamlet', 'extdata','phenobarb.csv')
x <- file %>% decorate %>% resolve
# Note that value has two elements for label and guide.
x %>% decorations(value)

# The print method defaults to the first, with warning.
map <- aes(x = time, y = value, color = event)

x %>% ggplot(map) + geom_point()


# If we subset appropriately, the relevant value is substituted.
x %>% filter(event == 'conc') %>% ggplot(map) + geom_point()

x %>% filter(event == 'conc') %>%
ggplot(aes(x = time, y = value, color = ApgarInd)) + geom_point()

x %>% filter(event == 'dose') %>%
ggplot(aes(x = time, y = value, color = Wt)) +
geom_point() +
scale_y_log10() +
```

```
scale_color_gradientn(colours = rainbow(4))

# print.decorated_ggplot will attempt to honor coordinated aesthetics.
x <- data.frame(x = c(1:6, 3:8), y = c(1:6,1:6), z = letters[c(1:6,1:6)])
x %<>% decorate('z: [color: ["red", "blue", "green", "gold", "black", "magenta"]]')
x %<>% decorate('z: [fill: ["red", "blue", "green", "gold", "black", "magenta"]]')
x %<>% decorate('z: [shape: [20, 21, 22, 23, 24, 25]]')
x %<>% decorate('z: [linetype: [6, 5, 4, 3, 2, 1]]')
x %<>% decorate('z: [alpha: [ .9, .8, .7, .6, .5, .4]]')
x %<>% decorate('z: [size: [1, 1.5, 2, 2.5, 3, 3.5]]')
x %>% ggplot(aes(
 x, y,
  color = z,
  fill = z,
  shape = z,
  linetype = z,
  alpha = z,
  size = z,
)) +
  geom_point() +
  geom_line(size = 1)
```

`group_by_decorations.data.frame` *Groups by Decorations for Data Frame*

Description

Invokes `group_by` using whatever groups are recovered by `decorations_groups`.

Usage

```
## S3 method for class 'data.frame'
group_by_decorations(x, ...)
```

Arguments

x	grouped_df
...	ignored

Value

list of symbols

See Also

Other decorate: `as_decorated()`, `as_decorated.default()`, `decorate()`, `decorate.character()`, `decorate.data.frame()`, `decorate.list()`, `decorate_groups()`, `decorate_groups.data.frame()`, `decorations()`, `decorations.data.frame()`, `decorations_groups()`, `decorations_groups.data.frame()`, `group_by_decorations()`, `redecorate()`

Examples

```
library(magrittr)
library(dplyr)
Theoph %>% group_vars # nothing!
Theoph %<>% decorate_groups(Subject, Time)
Theoph %<>% group_by_decorations
Theoph %>% group_vars # something
rm(Theoph)
```

---

io_csv *Import and Export Documented Tables as CSV*

Description

Imports or exports documented tables as comma-separated variable. Generic, with methods that extend as.csv.

Usage

```
io_csv(x, ...)
```

Arguments

x	object
...	passed arguments

Value

See methods.

See Also

Other io: io_csv.character(), io_csv.data.frame(), io_res(), io_res.character(), io_res.decorated(), io_table(), io_table.character(), io_table.data.frame(), io_yamlet(), io_yamlet.character(), io_yamlet.data.frame(), io_yamlet.yamlet()

Examples

```
# generate some decorated data
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)

# get a temporary filepath
out <- file.path(tempdir(), 'out.csv')

# save file using io_csv (returns filepath)
foo <- io_csv(x, out)
```

```
stopifnot(identical(out, foo))

# read using this filepath
y <- io_csv(foo)

# lossless round-trip (ignoring source attribute)
attr(x, 'source') <- NULL
attr(y, 'source') <- NULL
stopifnot(identical(x, y))
```

`io_table` *Import and Export Documented Tables*

Description

Imports or exports documented tables. Generic, with methods that extend `read.table` and `write.table`.

Usage

```
io_table(x, ...)
```

Arguments

`x`	object
`...`	passed arguments

Value

See methods.

See Also

Other io: `io_csv()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res()`, `io_res.character()`, `io_res.decorated()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `io_yamlet.yamlet()`

Examples

```
# generate some decorated data
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)

# get a temporary filepath
out <- file.path(tempdir(), 'out.tab')

# save file using io_table (returns filepath)
foo <- io_table(x, out)
stopifnot(identical(out, foo))
```

```
# read using this filepath
y <- io_table(foo, as.is = TRUE)

# lossless round-trip
attr(x, 'source') <- NULL
rownames(x) <- NULL
rownames(y) <- NULL
stopifnot(identical(x, y))
```

`is_dvec` *Test if Class is dvec*

Description

Tests whether x inherits 'dvec'.

Usage

```
is_dvec(x)
```

Arguments

x	object

Value

logical

Examples

```
is_dvec(1L)
is_dvec(as_dvec(1L))
```

`mimic.default` *Try To Look Like Another Equal-length Variable*

Description

Tries to mimic another vector or factor. If meaningful and possible, x acquires a guide attribute with labels from corresponding values in y. Any codelist attribute is removed. No guide is created for zero-length x. If x is a factor, unused levels are removed.

Usage

```
## Default S3 method:
mimic(x, y = x, ...)
```

Arguments

x	vector-like
y	vector-like, same length as x
...	passed to link{factor}

Value

same class as x

See Also

Other mimic: mimic(), mimic.classified()

Other interface: canonical.decorated(), classified.data.frame(), decorate.character(), decorate.data.frame(), desolve.decorated(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default(), write_yamlet()

Examples

```
library(magrittr)
library(dplyr)
let <- letters[1:5]
LET <- LETTERS[1:5]
int <- 0L:4L
num <- as.numeric(int)
fac <- factor(let)
css <- classified(let)

# any of these can mimic any other
str(mimic(LET, let))
str(mimic(num, let))
str(mimic(let, num))

# factors get a guide and classifieds get a named codelist
str(mimic(fac, int))
str(mimic(css, int))

# int can 'pick up' the factor levels as guide names
str(mimic(int, css))

# if two variables mean essentially the same thing,
# mimic lets you save space
x <- data.frame(id = 1:2, ID = c('A','B'))
x
x %<>% mutate(id = mimic(id, ID)) %>% select(-ID)
x
# ID still available, in principle:
x %>% as_decorated %>% resolve
```

`modify.default` *Modify Attributes of Indicated Components by Default*

Description

Modifies the attributes of each indicated element (all elements by default). Tries to assign the value of an expression to the supplied label, with existing attributes and the object itself (.) available as arguments. Gives a warning if the supplied label is considered reserved. Intends to support anything with one or more non-empty names.

Usage

```
## Default S3 method:
modify(
  x,
  ...,
  .reserved = getOption("yamlet_modify_reserved", c("class", "levels", "labels",
    "names"))
)
```

Arguments

`x`	object
`...`	indicated columns, or name-value pairs
`.reserved`	reserved labels that warn on assignment

Details

The name of the component itself is available during assignments as attribute 'name' (any pre-existing attribute 'name' is temporarily masked). After all assignments are complete, the value of 'name' is enforced at the object level. Thus, `modify` expressions can modify component names.

As currently implemented, the expression is evaluated by `eval_tidy`, with attributes supplied as the `data` argument. Thus, names in the expression may be disambiguated, e.g. with `.data`. See examples.

Value

same class as x

See Also

Other modify: `modify()`, `named()`, `selected()`, `selected.default()`

Other interface: `canonical.decorated()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `ggplot.decorated()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_res.decorated()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `mimic.default()`, `promote.list()`, `read_yamlet()`, `resolve.decorated()`, `scripted.default()`, `selected.default()`, `write_yamlet()`

Examples

```
library(magrittr)
library(dplyr)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)

# modify selected columns
x %<>% modify(title = paste(label, '(', guide, ')'), time)
x %>% select(time, conc) %>% decorations

# modify (almost) all columns
x %<>% modify(title = paste(label, '(', guide, ')'), -Subject)
x %>% select(time, conc) %>% decorations

# use column itself
x %<>% modify(`defined values` = sum(!is.na(.)))
x %>% select(time) %>% decorations

# rename column
x %<>% modify(time, name = label)
names(x)

# warn if assignment fails
## Not run:
\donttest{
x %<>% modify(title = foo, time)
}
## End(Not run)

# support lists
list(a = 1, b = 1:10, c = letters) %>%
modify(length = length(.), b:c)

x %<>% select(Subject) %>% modify(label = NULL, `defined values` = NULL)

# distinguish data and environment
location <- 'environment'
x %>% modify(where = location) %>% decorations
x %>% modify(where = .env$location) %>% decorations
## Not run:
\donttest{
x%>% modify(where = .data$location) %>% decorations
}
## End(Not run)
x %>% modify(location = 'attributes', where = location) %>% decorations
x %>% modify(location = 'attributes', where = .data$location) %>% decorations
```

`print.decorated_ggplot` *Substitute Expressions, Titles, Labels and Aesthetics in ggplots*

Description

Default labels (e.g. mappings for x, y, etc.) will be used to search data for more meaningful labels, taking first available from attributes with names in search. Likewise, if mappings for colour (color), fill, size, etc. (see defaults for discrete) indicate columns that have these defined as attributes, an attempt is made to add a corresponding discrete scale if one does not exist already. Values are recycled if necessary and are specific by ordinal position to the corresponding level of the corresponding variable. Levels are defined in increasing priority by sort(unique(x)), any guide attribute, any factor levels, any codelist attribute, or any plotmath attribute.

Usage

```
## S3 method for class 'decorated_ggplot'
print(
  x,
  ...,
 search = getOption("yamlet_decorated_ggplot_search", c("expression", "title", "label")),
 discrete = getOption("yamlet_decorated_ggplot_discrete", c("colour", "fill", "size",
    "shape", "linetype", "linewidth", "alpha")),
  drop = getOption("yamlet_decorated_ggplot_drop", TRUE)
)
```

Arguments

x	class 'decorated_ggplot' from ggplot.decorated
...	ignored
search	attribute names from which to seek label substitutes
discrete	discrete aesthetics to map from data decorations where available
drop	should unused factor levels be omitted from data-driven discrete scales?

Value

see print.ggplot

See Also

Other decorated_ggplot: ggplot.decorated(), ggplot_build.decorated_ggplot()

Examples

```
example(ggplot.decorated)
```

`read_yamlet` *Read Yamlet*

Description

Reads yamlet from file. Similar to `io_yamlet.character` but also reads text fragments.

Usage

```
read_yamlet(
  x,
  ...,
  default_keys = getOption("yamlet_default_keys", list("label", "guide"))
)
```

Arguments

`x`	file path for yamlet, or vector of yamlet in storage syntax
`...`	passed to `as_yamlet`
`default_keys`	character: default keys for the first n anonymous members of each element

Value

yamlet: a named list with default keys applied

See Also

`decorate.data.frame`

Other interface: `canonical.decorated()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `ggplot.decorated()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_res.decorated()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `mimic.default()`, `modify.default()`, `promote.list()`, `resolve.decorated()`, `scripted.default()`, `selected.default()`, `write_yamlet()`

Examples

```
library(csv)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
meta <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
x <- as.csv(file)
y <- read_yamlet(meta)
x <- decorate(x, meta = y)
stopifnot(identical(x, decorate(file)))
```

`resolve.decorated` *Resolve Guide for Decorated*

Description

Resolves implicit usage of default key 'guide' to explicit usage for decorated class. Calls `explicit_guide`, `classified`, and `make_title`.

Usage

```
## S3 method for class 'decorated'
resolve(x, ...)
```

Arguments

x	decorated
...	passed to `explicit_guide`, `classified`, and `make_title`

Value

decorated

See Also

Other resolve: `desolve()`, `desolve.classified()`, `desolve.data.frame()`, `desolve.decorated()`, `desolve.dvec()`, `resolve()`, `resolve.classified()`, `resolve.data.frame()`, `resolve.dvec()`

Other interface: `canonical.decorated()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `ggplot.decorated()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_res.decorated()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `mimic.default()`, `modify.default()`, `promote.list()`, `read_yamlet()`, `scripted.default()`, `selected.default()`, `write_yamlet()`

Examples

```
# generate some decorated data
library(magrittr)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)
x %>% decorations(Age, glyco)

# resolve everything, and show selected decorations
x %>% resolve %>% decorations(Age, glyco)

# resolve selectively, and show selected decorations
x %>% resolve(glyco) %>% decorations(Age, glyco)
```

scripted.default *Render Scripted Attributes of Indicated Components by Default*

Description

Modifies specific attributes of each indicated element (all elements by default).

Usage

```
## Default S3 method:
scripted(
  x,
  ...,
  open = getOption("yamlet_append_units_open", " ("),
  close = getOption("yamlet_append_units_close", ")"),
 format = getOption("yamlet_format", ifelse(knitr::is_latex_output(), "latex", "html"))
)
```

Arguments

x	object
...	indicated columns, or name-value pairs; passed to resolve and selected
open	character to precede units
close	character to follow units
format	one of 'latex' or 'html'

Details

The goal here is to render labels and units (where present) in a way that supports subscripts and superscripts for both plots and tables in either html or latex contexts.

The current implementation writes an 'expression' attribute to support figure labels and a 'title' attribute to support tables. print.decorated_ggplot will attempt to honor the expression attribute if it exists. tablet.data.frame will attempt to honor the title attribute if it exists (see Details there). An attempt is made to guess the output format (html or latex).

In addition to the 'title' and 'expression' attributes, scripted() writes a 'plotmath' attribute to store plotmath versions of factor levels, where present. print.decorated_ggplot should prefer these over their latex and html counterparts. Furthermore, factor levels (and codelists, where present) are converted to their latex or html equivalents. None of this happens if a 'plotmath' attribute already exists, thus preventing the same variable from being accidentally transformed twice.

To flexibly support latex, html, and plotmath, this function expects column labels and units to be encoded in "spork" syntax. See as_spork for details and examples. Briefly, "_" precedes a subscript, "^" precedes a superscript, and "." is used to force the termination of either a superscript or a subscript where necessary. For best results, units should be written using *, /, and ^; e.g. "kg*m^2/s^2" not "kg m2 s-2" (although both are valid: see is_parseable). A literal backslash

followed by "n" represents a newline. Greek letters are represented by their names, except where names are enclosed in backticks.

`scripted()` always calls `resolve()` for the indicated columns, to make units present where appropriate.

Value

'scripted', a superclass of x

See Also

Other scripted: `scripted()`

Other interface: `canonical.decorated()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `ggplot.decorated()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_res.decorated()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `mimic.default()`, `modify.default()`, `promote.list()`, `read_yamlet()`, `resolve.decorated()`, `selected.default()`, `write_yamlet()`

Examples

```
library(magrittr)
library(ggplot2)
x <- data.frame(time = 1:10, work = (1:10)^1.5)
x %<>% decorate('
  time: [ Time_elapsed, h ]
  work: [ Work_total_observed, kg*m^2/s^2 ]
')

x %>% decorations
x %>% ggplot(aes(time, work)) + geom_point()
x %>% scripted %>% ggplot(aes(time, work)) + geom_point()
x %>% scripted(format = 'html') %$% work %>% attr('title')
testthat::expect_equal(scripted(x), scripted(scripted(x)))
```

---

write_yamlet *Write Yamlet*

Description

Writes yamlet to file. Similar to `io_yamlet.yamlet` but returns invisible storage format instead of invisible storage location.

Usage

```
write_yamlet(
  x,
  con = stdout(),
  eol = "\n",
  useBytes = FALSE,
  default_keys = getOption("yamlet_default_keys", list("label", "guide")),
  fileEncoding = getOption("encoding"),
  block = FALSE,
  ...
)
```

Arguments

x	something that can be coerced to class 'yamlet', like a yamlet object or a decorated data.frame
con	passed to writeLines
eol	end-of-line; passed to writeLines as sep
useBytes	passed to writeLines
default_keys	character: default keys for the first n anonymous members of each element
fileEncoding	if con is character, passed to file as encoding
block	whether to write block scalars
...	passed to as_yamlet and to as.character.yamlet

Value

invisible character representation of yamlet (storage syntax)

See Also

decorate.list

Other interface: canonical.decorated(), classified.data.frame(), decorate.character(), decorate.data.frame(), desolve.decorated(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_res.decorated(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), mimic.default(), modify.default(), promote.list(), read_yamlet(), resolve.decorated(), scripted.default(), selected.default()

Examples

```
library(csv)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
meta <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
x <- as.csv(file)
y <- read_yamlet(meta)
x <- decorate(x, meta = y)
identical(x, decorate(file))
```

```
tmp <- tempfile()
write_yamlet(x, tmp)
stopifnot(identical(read_yamlet(meta), read_yamlet(tmp)))
```

---

yamlet *yamlet: Versatile Curation of Table Metadata*

Description

The **yamlet** package supports storage and retrieval of table metadata in yaml format. The most important function is `decorate.character`: it lets you 'decorate' your data by attaching attributes retrieved from a file in yaml format. Typically your data will be of class 'data.frame', but it could be anything that is essentially a named list.

Storage Format

Storage format for 'yamlet' is a text file containing well-formed yaml. Technically, it is a map of sequences. Though well formed, it need not be complete: attributes or their names may be missing.

In the simplest case, the data specification consists of a list of column (item) names, followed by semicolons. Perhaps you only have one column:

```
mpg:
```

or maybe several:

```
mpg:
cyl:
disp:
```

If you know descriptive labels for your columns, provide them (skip a space after the colon).

```
mpg: fuel economy
cyl: number of cylinders
disp: displacement
```

If you know units, create a sequence with square brackets.

```
mpg: [ fuel economy, miles/gallon ]
cyl: number of cylinders
disp: [ displacement , in^3 ]
```

If you are going to give units, you probably should give a key first, since the first anonymous element is 'label' by default, and the second is 'guide'. (A guide can be units for numeric variables, factor levels/labels for categorical variables, or a format string for dates, times, and datetimes.) You could give just the units but you would have to be specific:

```
mpg: [units: miles/gallon]
```

You can over-ride default keys by providing them in your data:

```
mpg: [units: miles/gallon]
_keys: [label, units]
```

Notice that stored yamlet can be informationally defective while syntactically correct. If you don't know an item key at the time of data authoring, you can omit it:

`race: [race, [white: 0, black: 1, 2, asian: 3 ]]`

Or perhaps you know the key but not the value:

`race: [race, [white: 0, black: 1, asian: 2, ? other ]]`

Notice that `race` is factor-like; the factor sequence is nested within the attribute sequence. Equivalently:

`race: [label: race, guide: [white: 0, black: 1, asian: 2, ? other ]]`

If you have a codelist of length one, you should still enclose it in brackets:

`sex: [Sex, [ M ]]`

To get started using yamlet, see `?as_yamlet.character` and examples there. See also `?decorate` which adds yamlet values to corresponding items in your data. See also `?print.decorated` which uses label attributes, if present, as axis labels.

Note: the quinidine and phenobarb datasets in the examples are borrowed from **nlme** (`?Quinidine`, `?Phenobarb`), with some reorganization.

Author(s)

Maintainer: Tim Bergsma <bergsmat@gmail.com>

See Also

Useful links:

- Report bugs at https://github.com/bergsmat/yamlet/issues

`yamlet_options` *Display Global Yamlet Options*

Description

Displays global yamlet options: those options whose names begin with 'yamlet_'.

- **yamlet_append_units_open**: see `append_units.default`. Controls how labels are constructed for variables with 'units' attributes. In brief, units are wrapped in parentheses, and appended to the label.
- **yamlet_append_units_close**: see `append_units.default`. Controls how labels are constructed for variables with 'units' attributes. In brief, units are wrapped in parentheses, and appended to the label.
- **yamlet_append_units_style**: see `append_units.default`. Determines parsing as 'plotmath' or 'latex', or 'plain' for no parsing.

- **yamlet_append_units_target**: see `append_units.default`. By default, append result is assigned to attribute 'label', but could be something else like 'title'.
- **yamlet_default_keys**: see `as_yamlet.character`. The first two yaml attributes without specified names are assumed to be 'label' and 'guide'.
- **yamlet_persistence**: see `decorate.list` and `as.integer.classified`. By default, persistence of column attributes is implemented by creating 'dvec' objects (decorated vectors) using **vctrs** methodology.
- **yamlet_cell_value**: see `as.data.frame.yamlet`. Controls how cells are calculated when converting yamlet (decorations) to a data.frame.
- **yamlet_import**: see `decorate.character`. Controls how primary data is read from file (default: as.csv()).
- **yamlet_extension**: see `decorate.character`. Controls what file extension is expected for yaml metadata (default: '.yaml')
- **yamlet_overwrite**: see `decorate.list`. Controls whether existing decorations are overwritten.
- **yamlet_exclude_attr**: see `decorations.data.frame` Controls what attributes are excluded from display.
- **yamlet_with_title**: see `make_title.dvec` and `drop_title.dvec`. For objects with (implied) units attributes, titles are by default automatically created on resolve() and destroyed on desolve(). Interacts with yamlet_append_units_*.
- **yamlet_infer_guide**: see `explicit_guide.yamlet`. Identifies the function that will be used to reclassify 'guide' as something more explicit.
- **yamlet_explicit_guide_overwrite**: see `explicit_guide.data.frame` and `explicit_guide.dvec`. In the latter case, controls whether existing attributes are overwritten.
- **yamlet_explicit_guide_simplify**: `explicit_guide.data.frame` and `explicit_guide.dvec`. Ordinarily, the 'guide' attribute is removed if something more useful can be inferred.
- **yamlet_decorated_ggplot_search**: see `print.decorated_ggplot`. The print method for decorated_ggplot populates axis labels by searching first for attributes named 'expression', 'title', and 'label'. Customizable.
- **yamlet_decorated_ggplot_discrete**: see `print.decorated_ggplot`. Discrete aesthetics to map from data decorations where available.
- **yamlet_decorated_ggplot_drop**: see `print.decorated_ggplot`. Should unused factor levels be omitted from data-driven discrete scales?
- **yamlet_ggready_parse**: see `ggready.data.frame`, `ggready.decorated`. Whether to parse axis labels. TRUE by default, but may be problematic if unintended.
- **yamlet_modify_reserved**: see `modify.default`. A list of reserved labels that warn on reassignment.
- **yamlet_promote_reserved**: see `promote.list`. Attributes to leave untouched when promoting singularities.
- **yamlet_promote**: see `filter.decorated`. Whether to promote when filtering 'decorated'.
- **yamlet_as_units_preserve**: `as_units.dvec`. What attributes to preserve when converting dvec to units. Just 'label' by default. Assign `options(yamlet_as_units_preserve = character(0))` to remove all.

- **yamlet_print_simplify**: `print.yamlet`. Whether to collapse interactively-displayed decorations into a single line for lists that have no (nested) names and have the same length when unlisted. True by default. Can be misleading for lists with fine detail, but in most cases fine detail will likely have names.
- **yamlet_format**: `scripted.default`. Choice of 'html' or 'latex', guessed if not supplied.
- **yamlet_warn_conflicted**: `c.classified`. Whether to warn when codelists for combined classified factors have conflicting names (which will be dropped).
- **yamlet_expand_codelist**: `explicit_guide.yamlet`. If TRUE (default) an empty list as a guide attribute is short-hand for `sort(unique(x))`.
- **yamlet_collapse_codelist**: `implicit_guide.data.frame`. An integer (default: 10) giving the maximum number of (un-named) codelist elements to store explicitly. Else, if `sort(unique(x))` has exactly the same values as codelist, `implicit_guide` will substitute an empty list.

Usage

```
yamlet_options()
```

Value

list

Examples

```
yamlet_options()
```

Index