Package 'yamlet'

November 25, 2020

Type Package

Title Versatile Curation of Table Metadata

Version 0.5.2

Author Tim Bergsma

Maintainer Tim Bergsma <bergsmat@gmail.com>

Description A file-based
mechanism for documenting datasets. It reads and writes
YAML-formatted metadata and applies it as data
item attributes. Data and metadata are stored independently
but can be coordinated by using similar file paths with
different extensions. The 'yamlet' dialect is valid 'YAML',
but some conventions are chosen to improve readability.
Defaults and conventions can be over-ridden by the user.
See ?yamlet and ?decorate.data.frame.
See ?read_yamlet ?write_yamlet, and ?io_csv.

License GPL-3

Encoding UTF-8

LazyData true

Imports yaml, csv (>= 0.5.4), encode, units, spork, ggplot2, dplyr (>= 0.8.1), rlang, xtable

RoxygenNote 7.1.1

VignetteBuilder knitr

Suggests testthat (>= 2.1.0), magrittr, table1, knitr, rmarkdown

NeedsCompilation no

Repository CRAN

Date/Publication 2020-11-25 07:30:02 UTC

R topics documented:

`as_decorated.default` *Coerce to Decorated by Default*

Description

Coerces to class 'decorated' by decorating (by default) with an empty list.

Usage

```
as_decorated.default(x, meta = "-", ...)
```

Arguments

`x`	object
`meta`	see `decorate.list`
`...`	passed arguments

Value

decorated

See Also

Other decorate: `as_decorated()`, `decorate.character()`, `decorate.data.frame()`, `decorate.list()`, `decorate()`, `decorations.data.frame()`, `decorations()`, `print.decorations()`, `redecorate()`

Examples

```
class(Puromycin)
class(as_decorated(Puromycin))
```

`as_yamlet` *Coerce to Yamlet*

Description

Coerces something to yamlet format. If the object or user specifies default keys, these are applied. See `as_yamlet.character`.

Usage

```
as_yamlet(x, ...)
```

Arguments

`x`	object
`...`	passed arguments

Value

a named list

See Also

Other yamlet: `as_yamlet.character()`, `as_yamlet.yam()`, `print.yamlet()`

Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
file
identical(as_yamlet(as_yam(file)), as_yamlet(file))

# Read yamlet from storage and apply default keys.
as_yamlet(file)
```

`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.default()`, `as_decorated()`, `decorate.data.frame()`, `decorate.list()`, `decorate()`, `decorations.data.frame()`, `decorations()`, `print.decorations()`, `redecorate()`

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

Examples

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

```
)
identical(
  decorate(file, meta = as_yamlet(meta)),
  decorate(file, meta = meta)
)
a <- decorate(file)
b <- resolve(decorate(file))
c <- resolve(decorate(
  file,
  read = read.table,
  quote = "",
  as.is = FALSE,
  sep = ',',
  header = TRUE,
  na.strings = c('', '\\s', '.','NA'),
  strip.white = TRUE,
  check.names = FALSE
))
d <- decorate(
  file,
  read = read.table,
  quote = "",
  as.is = FALSE,
  sep = ',',
  header = TRUE,
  na.strings = c('', '\\s', '.','NA'),
  strip.white = TRUE,
  check.names = FALSE
)

# Importantly, b and c are identical with respect to factors
cbind(
  `as.is/!resolve`   = sapply(a, class), # no factors
  `as.is/resolve`    = sapply(b, class), # factors made during decoration
  `!as.is/resolve`   = sapply(c, class), # factors made twice!
  `!as.is/!resolve`  = sapply(d, class)  # factors made during read
)
str(a$Smoke)
str(b$Smoke)
str(c$Smoke)
str(d$Smoke)
levels(c$Creatinine)
levels(d$Creatinine) # level detail retained as 'guide'
```

`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, ...)
```

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

Value

class 'decorated' 'data.frame'

See Also

decorate.list

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

Other decorate: as_decorated.default(), as_decorated(), decorate.character(), decorate.list(), decorate(), decorations.data.frame(), decorations(), print.decorations(), redecorate()

Examples

```
library(csv)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
meta <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
a <- decorate(as.csv(file))
b <- decorate(as.csv(file), meta = as_yamlet(meta))
c <- decorate(as.csv(file), meta = meta)
d <- decorate(as.csv(file), meta = file)
e <- resolve(decorate(as.csv(file)))

# Most import methods are equivalent.
identical(a, b)
identical(a, c)
identical(a, d)
identical(a, e)
```

`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` attributes are excluded from the result, as this is an attribute you likely don't want to manipulate independently. Consider carefully whether the default handling of factor levels (see `coerce` argument) is appropriate for your application.

Usage

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

Arguments

`x`	data.frame
`...`	optional unquoted column names to limit output (passed to `select`)
`coerce`	logical: whether to coerce factor levels to guide; alternatively, a key for the levels
`exclude_attr`	attributes to remove from the result

Value

named list of class 'decorations'

See Also

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

Examples

```
library(csv)
library(magrittr)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(as.csv(file))[,c('conc','Race')]
y <- decorate(as.csv(file))[,c('conc','Race')] %>% resolve
decorations(x)
decorations(y)
```

```
decorations(y, conc)
decorations(y, coerce = TRUE)
decorations(y, coerce = 'codelist')
decorations(y, exclude_attr = NULL)
```

`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`. Currently, there is only one method for resolve() (`resolve.decorated`) with the result that all 'resolved' objects inherit 'decorated' and thus can use `ggplot.decorated`.

For finer control, you can switch between 'data.frame' and to '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: `decorate.character()`, `decorate.data.frame()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `modify.default()`, `promote.default()`, `read_yamlet()`, `resolve.decorated()`, `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 unordered 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.
x %>% ggplot(map) + geom_point()

# We can resolve guide for a chance to enrich the output with units.
# Notice CHF levels are now ordered.
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))
```

`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.character(), io_res(), io_table.character(), io_table.data.frame(), io_table(), io_yamlet.character(), io_yamlet.data.frame(), io_yamlet.yamlet(), io_yamlet()

Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)
out <- file.path(tempdir(), 'out.csv')
foo <- io_csv(x, out)
identical(out, foo)
y <- io_csv(foo)
attr(x, 'source') <- NULL
attr(y, 'source') <- NULL
identical(x, y) # lossless 'round-trip'
```

---

io_res *Import Resolved Tables*

Description

Imports 'resolved' tables as comma-separated variable. Generic, with character method that extends io_csv.

Usage

```
io_res(x, ...)
```

Arguments

x	object
...	passed arguments

Value

See methods.

See Also

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

Examples

```
example(io_res.character)
```

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.character()`, `io_csv.data.frame()`, `io_csv()`, `io_res.character()`, `io_res()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `io_yamlet.yamlet()`, `io_yamlet()`

Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)
out <- file.path(tempdir(), 'out.tab')
foo <- io_table(x, out)
identical(out, foo)
y <- io_table(foo, as.is = TRUE)
attr(x, 'source') <- NULL
rownames(x) <- NULL
rownames(y) <- NULL
identical(x, y) # lossless 'round-trip'
```

io_yamlet *Import and Export Yamlet*

Description

Imports and exports yamlet. Generic, with a read method `read_yaml` for character and a write method `write_yaml` for data.frame.

Usage

```
io_yamlet(x, ...)
```

Arguments

x	object
...	passed arguments

Value

see methods

See Also

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

Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.yaml')
x <- io_yamlet(file)
tmp <- tempdir()
out <- file.path(tmp, 'tmp.yaml')

# we can losslessly 'round-trip' x using to generic calls
identical(x, io_yamlet(io_yamlet(x, out)))
```

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.

Value

same class as x

See Also

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

Other interface: `decorate.character()`, `decorate.data.frame()`, `ggplot.decorated()`, `io_csv.character()`, `io_csv.data.frame()`, `io_res.character()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `promote.default()`, `read_yamlet()`, `resolve.decorated()`, `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)
```

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

Description

At time of printing, default labels will be used as column names to search `data` for more meaningful labels, taking first available from attributes with names in `search`.

Usage

```
## S3 method for class 'decorated_ggplot'
print(
  x,
  ...,
 search = getOption("decorated_ggplot_search", c("expression", "title", "label"))
)
```

Arguments

`x`	class 'decorated_ggplot' from `ggplot.decorated`
`...`	passed arguments
`search`	attribute names from which to seek label substitutes

Value

see `print.ggplot`

See Also

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

Examples

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

`print.decorations` *Print Decorations*

Description

Prints decorations. Coerces to yamlet and prints result.

Usage

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

Arguments

`x`	decorations, i.e. a named list of class 'decorations'
`...`	ignored

Value

invisible x (yamlet)

See Also

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

Examples

```
example(decorations.data.frame)
```

`promote.default` *Promote by Default.*

Description

Promotes attributes of list-like objects. For the plural attributes of each element, any singularity is promoted to the sole attribute. Reserved attributes are untouched. Methods `filter.decorated` and `[.decorated` automatically attempt to promote attributes for all elements.

Usage

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

Arguments

x	object
...	indicated elements
.reserved	attributes to leave untouched

Value

same class as x

See Also

filter.decorated [.decorated

Other promote: `[.decorated()`, `filter.decorated()`, `promote()`, `singularity()`

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

Examples

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

# Note that there are two elements each for value label and value guide.
x %>% decorations(event, value)

# After filtering, only one set is relevant.
# promote() identifies and retains such.
x %>% dplyr:::filter.data.frame(event == 'dose') %>% decorations(value)
x %>% dplyr:::filter.data.frame(event == 'dose') %>% promote %>% decorations(value)

# If for some reason we do a partial promote, value attributes are unaffected.
# Nonsense example:
x %>% dplyr:::filter.data.frame(event == 'dose') %>% promote(event) %>% decorations(value)

# However, the 'decorated' method for filter() calls promote() internally.
x %>% filter(event == 'dose') %>% decorations(value)
```

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: decorate.character(), decorate.data.frame(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), modify.default(), promote.default(), resolve.decorated(), 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)
identical(x, decorate(file))
```

`redecorate` *Redecorate a List-like Object*

Description

Redecorates a list-like object. Equivalent to `decorate( ...,overwrite = TRUE)`.

Usage

```
redecorate(x, ..., overwrite = TRUE)
```

Arguments

`x`	object
`...`	passed arguments
`overwrite`	passed to `decorate`

Value

a list-like object, typically data.frame

See Also

Other decorate: `as_decorated.default()`, `as_decorated()`, `decorate.character()`, `decorate.data.frame()`, `decorate.list()`, `decorate()`, `decorations.data.frame()`, `decorations()`, `print.decorations()`

Examples

```
library(dplyr)
library(magrittr)
library(csv)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(as.csv(file))
x %>% select(Subject) %>% as_yamlet
x %<>% redecorate('Subject: Patient Identifier')
x %>% select(Subject) %>% as_yamlet
```

`resolve.decorated` *Resolve Guide for Decorated*

Description

Resolves implicit usage of default key 'guide' to explicit usage for decorated class. Simply calls `explicit_guide` followed by `factorize_codelist`.

Usage

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

Arguments

`x`	object
`...`	passed to `explicit_guide` and `factorize_codelist`

Value

class 'resolved' (and inherited classes)

See Also

Other resolve: `resolve()`

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

Examples

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

to_yamlet *Coerce to Yamlet Storage Format*

Description

Coerces to yamlet storage format. Generic, with methods for default, null, character and list which together implement the yamlet storage syntax. Always returns length-one character, possibly the empty string.

Usage

```
to_yamlet(x, ...)
```

Arguments

x	object
...	ignored

Value

length-one character

See Also

Other to_yamlet: to_yamlet.NULL(), to_yamlet.character(), to_yamlet.default(), to_yamlet.list()

write_yamlet *Write Yamlet*

Description

Writes yamlet to file. Similar to io_yamlet.data.frame 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"),
  ...
)
```

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
...	passed to as_yamlet

Value

invisible character representation of yamlet (storage syntax)

See Also

decorate.list

Other interface: decorate.character(), decorate.data.frame(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_res.character(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), modify.default(), promote.default(), read_yamlet(), resolve.decorated(), 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)
identical(read_yamlet(meta), read_yamlet(tmp))
```

---

xtable.decorated *Create Export Table for Decorated*

Description

Creates an export table for decorated data.frame by adding a footnote attribute.

Usage

```
## S3 method for class 'decorated'
xtable(x, ..., label = NULL, style = "latex")
```

Arguments

x	decorated
...	passed to footnote and (if named) xtable
label	passed to xtable
style	passed to footnote

Value

class 'decorated_xtable','xtable', 'data.frame'

Examples

```
library(magrittr)
library(xtable)
set.seed(0)
x <- data.frame(
 auc = rnorm(100, mean = 2400, sd = 200),
 bmi = rnorm(100, mean = 20, sd = 5),
 gen = 0:1
)
x %<>% decorate('auc: [AUC_0-24, ng*h/mL]')
x %<>% decorate('bmi: [Body Mass Index, kg/m^2]')
x %<>% decorate('gen: [Gender, [Male: 1, Female: 0]]')
y <- xtable(x)
attr(y, 'footnote')
y <- xtable(x, auc:bmi)
attr(y, 'footnote')
```

---

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, and therefore has utility over a longer life cycle of data development.

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: [unit: miles/gallon]`

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

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

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 ]]`

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.dg` 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.

Index