# Package 'yamlet'

June 18, 2020

**Type** Package

**Title** Versatile Curation of Table Metadata

**Version** 0.4.8

**Author** Tim Bergsma

**Maintainer** Tim Bergsma <bergsmat@gmail.com>

**Description** The 'yamlet' package implements 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** 7.1.0

**VignetteBuilder** knitr

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

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-06-18 18:00:03 UTC

## R topics documented:

---

`alias.data.frame` *Alias a Data Frame*

---

## Description

Aliases a data.frame. Replaces column names with labels, where present. Stores column name as 'alias' attribute.

## Usage

```
## S3 method for class 'data.frame'
alias(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | data.frame |
| `...` | ignored arguments |

## Value

aliased data.frame

## See Also

Other labels: append_units.data.frame(), append_units.default(), append_units(), sub_units()

## Examples

```
x <- data.frame(x = 1:10, y = 1:10, z = 1:10)
attr(x$x, 'label') <- 'Independent Value'
attr(x$y, 'label') <- 'Dependent Value'
x
alias(x)
```

---

append_units.data.frame

*Append Units for Data Frame*

---

## Description

Appends units for data.frame. For finer control, consider applying append_units.default to individual columns.

## Usage

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

## Arguments

| | |
|---|---|
| x | data.frame |
| ... | passed to default method |

## Value

data.frame

## See Also

Other labels: alias.data.frame(), append_units.default(), append_units(), sub_units()

## Examples

```
library(magrittr)
library(ggplot2)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
file %>% decorate %>% explicit_guide %>% append_units %>% as_yamlet
```

## as_classified.factor *Coerce Factor to Classified*

### Description

Coerce factor to classified. Creates a factor that retains attributes during subsetting.

### Usage

```
## S3 method for class 'factor'
as_classified(x, ...)
```

### Arguments

| | |
|---|---|
| x | factor |
| ... | ignored arguments |

### Value

class 'classified' 'factor'

### See Also

Other classified: [.classified(), as_classified()

Other interface: conditionalize.data.frame(), decorate.character(), decorate.data.frame(), ggplot.decorated(), ggready.data.frame(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame(), write_yamlet()

### Examples

```
class(as_classified(factor(letters)))
```

## 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)
```

---

## conditionalize.data.frame

*Conditionalize Attributes of Data Frame*

---

### Description

Conditionalizes attributes of data.frame. Creates a conditional `attribute` definition for `column` by mapping `value` to `test`. Only considers records where both `test` and `value` are defined, and gives an error if there is not one-to-one mapping. Can be used with write methods as an alternative to hand-coding conditional metadata.

### Usage

```
## S3 method for class 'data.frame'
conditionalize(x, column, attribute, test, value, ...)
```

### Arguments

| | |
|---|---|
| x | data.frame |
| column | unquoted name of column to conditionalize |
| attribute | unquoted name of attribute to create for column |
| test | unquoted name of column to test |
| value | unquoted name of column supplying attribute value |
| ... | ignored arguments |

## Details

If the test column is character, individual elements should not contain both single and double quotes. For the conditional expressions, these values will be single-quoted by default, or double-quoted if they contain single quotes.

## Value

class 'decorated' 'data.frame'

## See Also

Other interface: `as_classified.factor()`, `decorate.character()`, `decorate.data.frame()`, `ggplot.decorated()`, `ggready.data.frame()`, `io_csv.character()`, `io_csv.data.frame()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `read_yamlet()`, `resolve.data.frame()`, `write_yamlet()`

Other conditionalize: `conditionalize()`

## Examples

```
library(magrittr)
library(dplyr)
library(csv)
file <- system.file(package = 'yamlet', 'extdata','phenobarb.csv')
x <- as.csv(file)
head(x,3)

# suppose we have an event label stored as a column:

x %<>% mutate(evid = ifelse(
  event == 'dose',
  'dose of drug administered',
  'serum phenobarbital concentration'
 )
)

# We can define a conditional label for 'value'
# by mapping evid to event:

x %<>% conditionalize(value, label, event, evid)

x %>% as_yamlet
x %>% write_yamlet
```

---

`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 |
| read | function or function name for reading x |
| ext | file extension for metadata file, if relevant |
| ... | passed to read (if accepted) and to as_yamlet.character |

## Value

class 'decorated' 'data.frame'

## See Also

Other decorate: decorate.data.frame(), decorate.list(), decorate(), decorations.data.frame(), decorations(), redecorate()

Other interface: as_classified.factor(), conditionalize.data.frame(), decorate.data.frame(), ggplot.decorated(), ggready.data.frame(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame(), 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 = 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: as_classified.factor(), conditionalize.data.frame(), decorate.character(), ggplot.decorated(), ggready.data.frame(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame(), write_yamlet()

Other decorate: decorate.character(), decorate.list(), decorate(), decorations.data.frame(), 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 |
| coerce | logical: whether to coerce factor levels to guide; alternatively, a key for the levels |
| exclude_attr | attributes to remove from the result |
| ... | ignored |

## Value

named list

## See Also

Other decorate: `decorate.character()`, `decorate.data.frame()`, `decorate.list()`, `decorate()`, `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, coerce = TRUE)
decorations(y, coerce = 'codelist')
decorations(y, exclude_attr = NULL)
```

---

# encode.yamlet *Encode Yamlet*

---

## Description

Encodes yamlet. Each 'guide' element with length > 1 is converted to an encoding, if possible. If `data` is supplied, conditional guides will be ignored.

### Usage

```
## S3 method for class 'yamlet'
encode(x, target = "guide", data = NULL, ...)
```

### Arguments

| | |
|---|---|
| x | yamlet |
| target | attribute to encode |
| data | optional data.frame for guide context |
| ... | ignored |

### Value

yamlet, with guide elements possibly transformed to encodings

### See Also

Other encode: `list2encoding()`

### Examples

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

---

## explicit_guide *Coerce Guide to Something More Explicit*

---

### Description

Coerces 'guide' to something more explicit. Generic, with methods for data.frame and yamlet. The key 'guide' generally suggests a guide to interpretation of a data item, such as units, formats, codelists, and encodings. The idea here is to replace 'guide' with something explicit in case required downstream.

### Usage

```
explicit_guide(x, ...)
```

### Arguments

| | |
|---|---|
| x | object of dispatch |
| ... | passed arguments |

### Value

see methods

## See Also

Other explicit_guide: explicit_guide.data.frame(), explicit_guide.yamlet()

---

`factorize_codelist` *Coerce Codelist to Factor*

---

## Description

Coerces codelist to factor. Generic, with default and data.frame methods. Returns 'classified' 'factor' which as an attribute-preserving subset method.

## Usage

```
factorize_codelist(x, ...)
```

## Arguments

| | |
|---|---|
| x | object |
| ... | passed arguments |

## Value

class 'classified' 'factor'

## See Also

Other factorize_codelist: factorize_codelist.character(), factorize_codelist.data.frame(), factorize_codelist.default(), factorize_codelist.factor()

---

`footnote.decorated` *Footnote Decorated*

---

## Description

Footnotes a decorated data.frame. Generates a text string that defines column names using label and unit attributes.

## Usage

```
## S3 method for class 'decorated'
footnote(x, equal = ":", collapse = "; ", ...)
```

### Arguments

| | |
|---|---|
| x | decorated |
| equal | character: a symbol suggesting equality between a name and its note |
| collapse | used to paste column-wise footnotes |
| ... | passed to append_units |

### Value

character

### See Also

Other footnote: footnote()

### Examples

```
library(magrittr)
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]]')
footnote(x)
```

---

## 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 tries to implement automatic labels and units in axes and legends in association with print.dg. Use ggplot(as.data.frame(x)) to get default ggplot() behavior. Use ggplot(as_decorated(x)) to enforce custom behavior.

### Usage

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

### Arguments

| | |
|---|---|
| data | data.frame or similar |
| ... | passed to ggplot |

## Value

return value like ggplot

## See Also

Other dg: print.dg()

Other interface: as_classified.factor(), conditionalize.data.frame(), decorate.character(), decorate.data.frame(), ggready.data.frame(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame(), write_yamlet()

## Examples

```
meta <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(meta)
library(ggplot2)
class(ggplot(data = x) + geom_path(aes(x = time, y = conc)))
class(ggplot(data = x, aes(x = time, y = conc)) + geom_path())
example(print.dg)
```

---

# ggready.data.frame *Prepare Data Frame for GGplot*

---

## Description

Prepares data.frame for ggplot. Calls resolve and appends units to label using append_units (passing style = 'plotmath' if parse is true, else style = 'plain'). Enforces class 'decorated'.

## Usage

```
## S3 method for class 'data.frame'
ggready(x, parse = getOption("ggready_parse", TRUE), ...)
```

## Arguments

| | |
|---|---|
| x | object |
| parse | passed to append_units |
| ... | passed to append_units |

## Value

decorated

## See Also

Other resolve: ggready(), resolve.data.frame(), resolve()

Other interface: as_classified.factor(), conditionalize.data.frame(), decorate.character(), decorate.data.frame(), ggplot.decorated(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame(), write_yamlet()

## Examples

```
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)
x <- ggready(x)
str(x$conc)
library(magrittr)
library(ggplot2)
file %>%
 decorate %>%
 filter(!is.na(conc)) %>%
 ggready %>%
 ggplot(aes(x = time, y = conc, color = Heart)) +
 geom_point()
```

---

# 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_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_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_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_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)))
```

## is_parseable.default *Check Something is Parseable as Units by Default*

### Description

Checks if something is parseable as units. Tests against the udunits library in **units**. See `as_units`. See also `install_symbolic_unit` for finer control.

### Usage

```
## Default S3 method:
is_parseable(x, ...)
```

### Arguments

| | |
|---|---|
| x | character |
| ... | passed arguments |

### Value

logical

### See Also

Other parseable: `is_parseable()`

Other interface: `as_classified.factor()`, `conditionalize.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `ggplot.decorated()`, `ggready.data.frame()`, `io_csv.character()`, `io_csv.data.frame()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `read_yamlet()`, `resolve.data.frame()`, `write_yamlet()`

### Examples

```
is_parseable(c('kg/m2','kg/m^2','kg.m/s2','Âµg/L'))
is_parseable('foo')
library(units)
install_symbolic_unit('foo')
is_parseable('foo')
```

---

## print.dg *Print Automatic Labels and Units for ggplot*

---

### Description

Prints automatic labels and units for ggplot. Substitutes column label, if present, for default.

### Usage

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

### Arguments

| | |
|---|---|
| x | class 'dg' from `ggplot.decorated` |
| ... | passed arguments |

## Value

see print.ggplot

## See Also

Other dg: ggplot.decorated()

## Examples

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

# resolve() promotes factors to a class
# that retains attributes when subsetting,
# so legend has access to the label from Heart,
# even after a filter operation.

file %>% decorate %>% resolve %>% filter(!is.na(conc)) %>%
ggplot(aes(x = time, y = conc, color = Heart)) + geom_point()

# No factors created here, but print.dg promotes guide to factor if it can:

file %>% decorate %>% filter(!is.na(conc)) %>%
ggplot(aes(x = time, y = conc, color = Heart)) + geom_point()

# facet_wrap() should use decodes where available.
# resolve() makes them available by promoting to
# (a subclass of) factor.

file %>% decorate %>% filter(!is.na(conc)) %>% resolve %>%
ggplot(aes(x = time, y = conc)) + geom_point() + facet_wrap(~Creatinine)

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

file <- system.file(package = 'yamlet', 'extdata','phenobarb.csv')

# Note that there are two elements each for value label and value guide.
#'
file %>% decorate %>% as_yamlet
# Guide might have been mistaken for an attempt to provide codes/decodes
# for a factor.  However, the keys evaluate to logical on the data.frame.
# Seeing that, we test for one of them being all true, and if so we select it.

file %>% decorate %>% ggplot(aes(x = time, y = value, color = event)) + geom_point()

# In the above example, we are plotting doses and concentrations, which have
# different labels and units, so we can't improve on the y axis label.
# But if we subset to just one of these, then only one of the named conditions
```

```
# will be always true (and will therefore be promoted).

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

file %>% decorate %>%
filter(event == 'dose') %>%
ggplot(aes(x = time, y = value, color = Wt)) +
geom_point() +
scale_y_log10() +
scale_color_gradientn(colours = rainbow(4))
```

---

`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 |
| `default_keys` | character: default keys for the first n anonymous members of each element |
| `...` | passed to `as_yamlet` |

## Value

yamlet: a named list with default keys applied

## See Also

`decorate.data.frame`

Other interface: `as_classified.factor()`, `conditionalize.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `ggplot.decorated()`, `ggready.data.frame()`, `io_csv.character()`, `io_csv.data.frame()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `resolve.data.frame()`, `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: `decorate.character()`, `decorate.data.frame()`, `decorate.list()`, `decorate()`, `decorations.data.frame()`, `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.data.frame` *Resolve Guide for Data Frame*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `x` | object |
| `...` | passed arguments |

## Value

data.frame

## See Also

Other resolve: `ggready.data.frame()`, `ggready()`, `resolve()`

Other interface: `as_classified.factor()`, `conditionalize.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `ggplot.decorated()`, `ggready.data.frame()`, `io_csv.character()`, `io_csv.data.frame()`, `io_table.character()`, `io_table.data.frame()`, `io_yamlet.character()`, `io_yamlet.data.frame()`, `is_parseable.default()`, `read_yamlet()`, `write_yamlet()`

## Examples

```
library(magrittr)
file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
x <- decorate(file)
x %>% resolve(default = 'unit') %>% as_yamlet
```

sub_units *Place below Under Label*

## Description

Places units attribute below label attribute. Makes the most sense for figures (`style = 'plotmath'`) and useful for tables (`style = 'latex'`) in combination with `alias`. See also `append_units`.

## Usage

```
sub_units(
  x,
  open = if (style == "plain") "\n(" else "\\n(",
  close = ")",
  style = "latex",
  math_open = "",
  math_close = "",
  label_open = "$\\begin{gathered}",
  label_close = "\\end{gathered}$",
  newline = "\\\\",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | object |
| `open` | character to precede units |
| `close` | character to follow units |
| `style` | one of 'plain', 'latex', or 'plotmath' |
| `math_open, math_close, label_open, label_close, newline` | passed to `as_latex.spar` if style = 'latex' |
| `...` | passed arguments |

## Value

see methods for append_units

## See Also

Other labels: `alias.data.frame()`, `append_units.data.frame()`, `append_units.default()`, `append_units()`

## Examples

```
library(units)
library(magrittr)
library(dplyr)
library(ggplot2)
x <- 1:10
attr(x, 'label') <- 'acceleration'
units(x) <- 'm/s^2'
y <- as_units('kg')
x %>% attr('label')
x %>% sub_units %>% attr('label')
x %>% sub_units(style = 'plotmath') %>% attr('label')
x %>% sub_units(style = 'plain') %>% attr('label') %>% writeLines
y %>% attr('label')
y %>% sub_units(style = 'plain') %>% attr('label')
x %>% sub_units(style = 'plotmath')
x %>% sub_units(style = 'latex')

file <- system.file(package = 'yamlet', 'extdata','quinidine.csv')
file %>% decorate %>% resolve %>%
sub_units(style = 'plotmath') %>%
ggplot(data = ., aes(x = conc, y = time, color = Heart)) %>%
add(geom_point())
```

---

# 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: as_classified.factor(), conditionalize.data.frame(), decorate.character(), decorate.data.frame(), ggplot.decorated(), ggready.data.frame(), io_csv.character(), io_csv.data.frame(), io_table.character(), io_table.data.frame(), io_yamlet.character(), io_yamlet.data.frame(), is_parseable.default(), read_yamlet(), resolve.data.frame()

## 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, style = "latex", ...)
```

## Arguments

| | |
|---|---|
| x | decorated |
| style | passed to footnote |
| ... | passed to footnote and xtable |

## Value

class 'decorated', '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')
```

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