

Package ‘yamlet’

July 19, 2021

Type Package

Title Versatile Curation of Table Metadata

Version 0.7.7

Author Tim Bergsma

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.data.frame and ?modify.default. See ?read_yamlet ?write_yamlet, ?io_csv, and ?ggplot.decorated.

License GPL-3

Encoding UTF-8

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

RoxygenNote 7.1.1

VignetteBuilder knitr

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

NeedsCompilation no

Repository CRAN

Date/Publication 2021-07-18 23:10:02 UTC

R topics documented:

as.data.frame.yamlet	2
as.integer.classified	3

classified.default	4
decorate.character	5
decorate.data.frame	7
decorations.data.frame	8
desolve.decorated	9
ggplot.decorated	10
io_csv	12
io_table	13
mimic.default	14
modify.default	15
print.decorated_ggplot	17
read_yamlet	18
redecorate	19
resolve.decorated	20
write_yamlet	21
yamlet	22

Index	24
--------------	-----------

as.data.frame.yamlet *Coerce Yamlet to Data Frame*

Description

Coerces yamlet to data.frame. Columns are constructed in the order that attributes are encountered, beginning with top-level 'item' (default). Cell contents are calculated using `getOption('yamlet_cell_value',yamlet::cell_value)` to which is passed the cell-specific metadata as well as `sep` and `def`.

Usage

```
## S3 method for class 'yamlet'
as.data.frame(
  x,
  row.names = "item",
  optional = FALSE,
  sep = "\n",
  def = ": ",
  ...
)
```

Arguments

x	yamlet; see decorations and read_yamlet
row.names	a name for a column to hold top-level names, or NULL to represent these as row.names
optional	if TRUE and row.names is NULL, row.names will not be set
sep	separator for multiple items within an attribute

```
def          definition string: separator between items and their (preceding) names, if any
...         ignored
```

Value

```
data.frame
```

See Also

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_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)

file <- system.file(package = 'yamlet', 'extdata', 'quinidine.yaml')
file %>% read_yamlet %>% explicit_guide %>% as.data.frame
file <- system.file(package = 'yamlet', 'extdata', 'phenobarb.csv')

# phenobarb.yaml has conditional metadata that benefits
# from interpretation in the context of the data itself.
# thus, we
# * read the whole 'decorated' object (not just yaml),
# * resolve the 'guide' ambiguity,
# extract the best-guess decorations, and
# convert to data.frame.

file %>% io_csv %>% resolve %>% decorations %>% as.data.frame
```

as.integer.classified *Coerce Classified to Integer*

Description

Coerces classified to integer. Result is like `as.integer(as.numeric(x)) + offset` but has a `codelist` giving original values. If you need a simple integer, consider coercing first to numeric.

Usage

```
## S3 method for class 'classified'
as.integer(x, offset = 0L, ...)
```

Arguments

x classified, see [classified](#)
 offset an integer value to add to intermediate result
 ... ignored

Value

integer

See Also

Other classified: [\[.classified\(\)](#), [\[<-classified\(\)](#), [\[\[.classified\(\)](#), [\[\[<-classified\(\)](#),
[c.classified\(\)](#), [classified.data.frame\(\)](#), [classified.default\(\)](#), [classified\(\)](#), [unclassified.classified\(\)](#),
[unclassified.data.frame\(\)](#), [unclassified\(\)](#)

Examples

```
library(magrittr)
classified(c('knife', 'fork', 'spoon'))
classified(c('knife', 'fork', 'spoon')) %>% as.numeric
classified(c('knife', 'fork', 'spoon')) %>% as.integer
classified(c('knife', 'fork', 'spoon')) %>% as.integer(-1)
```

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 = levels,
  exclude = NA,
  ordered = is.ordered(x),
  nmax = NA,
  ...
)
```

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
...	ignored

Value

'classified' 'factor'

See Also

Other classified: [\[.classified\(\)](#), [\[<-.classified\(\)](#), [\[\[.classified\(\)](#), [\[\[<-.classified\(\)](#), [as.integer.classified\(\)](#), [c.classified\(\)](#), [classified.data.frame\(\)](#), [classified\(\)](#), [unclassified.classified\(\)](#), [unclassified.data.frame\(\)](#), [unclassified\(\)](#)

Examples

```
classified(1:3)
classified(1:3, levels = 4:6)
classified(1:3, levels = 1:3)
classified(1:3, labels = letters[1:3])
```

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\(\)](#), [redecorate\(\)](#)

Other interface: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [decorate.data.frame\(\)](#), [desolve.decorated\(\)](#), [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\(\)](#), [mimic.default\(\)](#), [modify.default\(\)](#), [promote.list\(\)](#), [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: `as.data.frame.yamlet()`, `classified.data.frame()`, `decorate.character()`, `desolve.decorated()`, `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()`, `mimic.default()`, `modify.default()`, `promote.list()`, `read_yamlet()`, `resolve.decorated()`, `selected.default()`, `write_yamlet()`

Other decorate: `as_decorated.default()`, `as_decorated()`, `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' 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.default\(\)](#), [as_decorated\(\)](#), [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, conc)
decorations(y, exclude_attr = NULL)
```

desolve.decorated *Desolve Guide for Decorated*

Description

Un-resolves explicit usage of default key 'guide' to implicit usage for decorated class. Simply calls [unclassified](#) followed by [implicit_guide](#).

Usage

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

Arguments

x object
 ... passed to [implicit_guide](#) and [classified](#)

Value

decorated

See Also

Other resolve: [desolve\(\)](#), [resolve.decorated\(\)](#), [resolve\(\)](#)

Other interface: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [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\(\)](#), [mimic.default\(\)](#), [modify.default\(\)](#), [promote.list\(\)](#), [read_yamlet\(\)](#), [resolve.decorated\(\)](#), [selected.default\(\)](#), [write_yamlet\(\)](#)

Examples

```
library(magrittr)
file <- system.file(package = 'yamlet', 'extdata', 'quinidine.csv')
x <- decorate(file)
x %>% resolve %>% decorations(Age, glyco, Race)
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 finer 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: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [decorate.character\(\)](#), [decorate.data.frame\(\)](#), [desolve.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\(\)](#), [mimic.default\(\)](#), [modify.default\(\)](#), [promote.list\(\)](#), [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_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'
```

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 codes are removed from codelist.

Usage

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

Arguments

x	vector-like
y	vector-like, same length as x
...	ignored arguments

Value

same class as x

See Also

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

Other interface: `as.data.frame.yamlet()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `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.list()`, `read_yamlet()`, `resolve.decorated()`, `selected.default()`, `write_yamlet()`

Examples

```
example(mimic.classified)
```

```
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

<code>x</code>	object
<code>...</code>	indicated columns, or name-value pairs
<code>.reserved</code>	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.default\(\)](#), [selected\(\)](#)

Other interface: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [decorate.character\(\)](#), [decorate.data.frame\(\)](#), [desolve.decorated\(\)](#), [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\(\)](#), [mimic.default\(\)](#), [promote.list\(\)](#), [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)

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

# distinguish data and environment
location <- 'environment'
x %>% modify(when = location) %>% decorations
x %>% modify(when = .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 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)
```

`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: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [decorate.character\(\)](#), [decorate.data.frame\(\)](#), [desolve.decorated\(\)](#), [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\(\)](#), [mimic.default\(\)](#), [modify.default\(\)](#), [promote.list\(\)](#), [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	<i>Redecorate a List-like Object</i>
------------	--------------------------------------

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\(\)](#)

Examples

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

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 [classified](#).

Usage

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

Arguments

x	object
...	passed to explicit_guide and classified

Value

decorated

See Also

Other resolve: [desolve.decorated\(\)](#), [desolve\(\)](#), [resolve\(\)](#)

Other interface: [as.data.frame.yamlet\(\)](#), [classified.data.frame\(\)](#), [decorate.character\(\)](#), [decorate.data.frame\(\)](#), [desolve.decorated\(\)](#), [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\(\)](#), [mimic.default\(\)](#), [modify.default\(\)](#), [promote.list\(\)](#), [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)
```

write_yamlet	<i>Write Yamlet</i>
--------------	---------------------

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 <code>writelnLines</code>
eol	end-of-line; passed to <code>writelnLines</code> as sep
useBytes	passed to <code>writelnLines</code>
default_keys	character: default keys for the first n anonymous members of each element
fileEncoding	if con is character, passed to <code>file</code> as encoding
...	passed to <code>as_yamlet</code>

Value

invisible character representation of yamlet (storage syntax)

See Also

`decorate.list`

Other interface: `as.data.frame.yamlet()`, `classified.data.frame()`, `decorate.character()`, `decorate.data.frame()`, `desolve.decorated()`, `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()`, `mimic.default()`, `modify.default()`, `promote.list()`, `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))
```

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 ]]
```

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.

Index

- * **classified**
 - as.integer.classified, 3
 - classified.default, 4
- * **decorated ggplot**
 - ggplot.decorated, 10
 - print.decorated_ggplot, 17
- * **decorate**
 - decorate.character, 5
 - decorate.data.frame, 7
 - decorations.data.frame, 8
 - redecorate, 19
- * **interface**
 - as.data.frame.yamlet, 2
 - decorate.character, 5
 - decorate.data.frame, 7
 - desolve.decorated, 9
 - ggplot.decorated, 10
 - mimic.default, 14
 - modify.default, 15
 - read_yamlet, 18
 - resolve.decorated, 20
 - write_yamlet, 21
- * **io**
 - io_csv, 12
 - io_table, 13
- * **mimic**
 - mimic.default, 14
- * **modify**
 - modify.default, 15
- * **resolve**
 - desolve.decorated, 9
 - resolve.decorated, 20
- [.classified, 4, 5
- [[.classified, 4, 5

- as.csv, 12
- as.data.frame, 10
- as.data.frame.yamlet, 2, 6, 8, 10, 11, 15, 16, 18, 20, 21
- as.integer.classified, 3, 5

- as_decorated, 6, 8–10, 19
- as_decorated.default, 6, 8, 9, 19
- as_yamlet, 18, 21
- as_yamlet.character, 6

- c.classified, 4, 5
- classified, 4, 5, 9, 20
- classified.data.frame, 3–6, 8, 10, 11, 15, 16, 18, 20, 21
- classified.default, 4, 4

- decorate, 6, 8–10, 19
- decorate.character, 3, 5, 8–11, 15, 16, 18–22
- decorate.data.frame, 3, 6, 7, 9–11, 15, 16, 18–21
- decorate.list, 6–9, 19, 21
- decorations, 2, 6, 8, 9, 19
- decorations.data.frame, 6, 8, 8, 19
- desolve, 10, 20
- desolve.decorated, 3, 6, 8, 9, 11, 15, 16, 18, 20, 21

- eval_tidy, 15
- explicit_guide, 20

- factor, 4, 5
- file, 21

- ggplot, 10
- ggplot.decorated, 3, 6, 8, 10, 10, 15–18, 20, 21
- ggplot_build.decorated_ggplot, 11, 17
- ggready, 10

- implicit_guide, 9
- io_csv, 12, 14
- io_csv.character, 3, 6, 8, 10, 11, 13–16, 18, 20, 21
- io_csv.data.frame, 3, 6, 8, 10, 11, 13–16, 18, 20, 21

`io_res`, [13](#), [14](#)
`io_res.character`, [3](#), [6](#), [8](#), [10](#), [11](#), [13–16](#), [18](#),
[20](#), [21](#)
`io_table`, [13](#), [13](#)
`io_table.character`, [3](#), [6](#), [8](#), [10](#), [11](#), [13–16](#),
[18](#), [20](#), [21](#)
`io_table.data.frame`, [3](#), [6](#), [8](#), [10](#), [11](#), [13–16](#),
[18](#), [20](#), [21](#)
`io_yamlet`, [13](#), [14](#)
`io_yamlet.character`, [3](#), [6](#), [8](#), [10](#), [11](#), [13–16](#),
[18](#), [20](#), [21](#)
`io_yamlet.data.frame`, [3](#), [6](#), [8](#), [10](#), [11](#),
[13–16](#), [18](#), [20](#), [21](#)
`io_yamlet.yamlet`, [13](#), [14](#)
`is_parseable.default`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#),
[16](#), [18](#), [20](#), [21](#)

`mimic`, [15](#)
`mimic.classified`, [15](#)
`mimic.default`, [3](#), [6](#), [8](#), [10](#), [11](#), [14](#), [16](#), [18](#), [20](#),
[21](#)
`modify`, [16](#)
`modify.default`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [15](#), [18](#),
[20](#), [21](#)

`named`, [16](#)

`print.decorated_ggplot`, [10](#), [11](#), [17](#)
`print.ggplot`, [17](#)
`promote.list`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [16](#), [18](#), [20](#),
[21](#)

`read.table`, [13](#)
`read_yamlet`, [2](#), [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [16](#), [18](#), [20](#),
[21](#)
`redecorate`, [6](#), [8](#), [9](#), [19](#)
`resolve`, [10](#), [20](#)
`resolve.decorated`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [16](#),
[18](#), [20](#), [21](#)

`select`, [9](#)
`selected`, [16](#)
`selected.default`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [16](#), [18](#),
[20](#), [21](#)

`unclassified`, [4](#), [5](#), [9](#)
`unclassified.classified`, [4](#), [5](#)
`unclassified.data.frame`, [4](#), [5](#)

`write.table`, [13](#)
`write_yamlet`, [3](#), [6](#), [8](#), [10](#), [11](#), [15](#), [16](#), [18](#), [20](#),
[21](#)
`writelnLines`, [21](#)
`yamlet`, [22](#)