Package 'marquee'

May 8, 2026

Title Markdown Parser and Renderer for R Graphics

Version 1.2.1

Description Provides the mean to parse and render markdown text with grid along with facilities to define the styling of the text.

License MIT + file LICENSE

URL https://marquee.r-lib.org, https://github.com/r-lib/marquee

BugReports https://github.com/r-lib/marquee/issues

Depends R (>= 4.1)

Imports cli, glue, grDevices, grid, jpeg, lifecycle, png, rlang (>= 1.1.0), S7, systemfonts (>= 1.2.0), textshaping (>= 1.0.0), utils, vctrs

Suggests ggplot2, gt, gtable, knitr, patchwork, ragg, rmarkdown, rsvg, testthat (>= 3.0.0)

LinkingTo cpp11

VignetteBuilder knitr

Config/build/compilation-database true

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-23

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation yes

Author Thomas Lin Pedersen [aut, cre] (ORCID: <https://orcid.org/0000-0002-5147-4711>), Martin Mitáš [aut] (Author of MD4C), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Thomas Lin Pedersen <thomas.pedersen@posit.co>

Repository CRAN

Date/Publication 2025-09-15 13:50:02 UTC

Contents

`classic_style` *Classic styling for markdown*

Description

This function facilitates construction of a complete style set based on the classic look of an HTML rendered markdown document. It contains style specifications for all the supported markdown elements as well as a `sub` and `sup` style that can be used for subscripts and superscript respectively. These are only accessible through custom spans (e.g. `H{.sub 2}O`) as markdown doesn't provide a syntax for these formats.

Usage

```
classic_style(
  base_size = 12,
  body_font = "",
  header_font = "",
  code_font = "mono",
  ...,
  ltr = TRUE
)
```

Arguments

`base_size`	The base font size for the text. All other sizing is based on this
`body_font`	The font family to use for body text
`header_font`	The font family to use for headers
`code_font`	The font family to use for code and code block text
`...`	Arguments passed on to `base_style`

`weight` The font weight to use. Can either be a number (`0`, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, or `900`) or a strings (`"undefined"`, `"thin"`, `"ultralight"`, `"light"`, `"normal"`, `"medium"`, `"semibold"`, `"bold"`, `"ultrabold"`, or `"heavy"`)

`italic` Should the font be slanted

`width` The font width to use. Can either be a number ("`0`, `1`‘, ‘`2`‘, ‘`3`‘, ‘`4`‘, ‘`5`‘, ‘`6`‘, ‘`7`‘, ‘`8`‘, or ‘`9`‘) or strings (‘`"undefined"`‘, ‘`"ultracondensed"`‘, ‘`"extracondensed"`‘, ‘`"condensed"`‘, ‘`"semicondensed"`‘, ‘`"normal"`‘, ‘`"semiexpanded"`‘, ‘`"expanded"`‘, ‘`"extraexpanded"`‘, or ‘`"ultraexpanded"`‘)

`features` A font_feature object specifying any OpenType font features to apply to the font

`color` Is the color of the font

`lineheight` The spacing between subsequent lines relative to the font size. Can be `relative()` in which case it is based on the parent lineheight.

`align` The alignment within the text. One of `"left"`, `"center"`, `"right"`, `"justified-left"`, `"justified-center"`, `"justified-right"`, or `"distributed"`

`tracking` Additional character spacing measured in 1/1000em. Can be `relative()` in which case it is based on the parent tracking.

`indent` The indentation of the first line in a paragraph measured in points. Can be `relative()` in which case it is based on the parent indent, `em()` in which case it is based on the font size in this style, or `rem()` in which case it is based on the font size of the body element.

`hanging` The indentation of all but the first line in a paragraph measured in points. Can be `relative()` in which case it is based on the parent hanging, `em()` in which case it is based on the font size in this style, or `rem()` in which case it is based on the font size of the body element.

`margin` The margin around the element, given as a call to `trbl()`. Margin refers to the area outside the box that text is placed in. If the element has a background, the margin area will not be colored.

`padding` The padding around the element, given as a call to `trbl()`. Padding refers to the distance between the text and the border of the box it will be drawn in. If the element has a background, the padding area will be colored.

`background` The color of the background fill. The background includes the padding but not the margin. Can be a solid color or a gradient or pattern made with `grid::linearGradient()`/`grid::radialGradient()`/`grid::pattern()`

`border` The color of the background stroke. The background includes the padding but not the margin

`border_width` The line width of the background stroke, given as a call to `trbl()`

`border_type` The linetype of the background stroke, given as an an `lty` compatible value (See the *Line Type Specification* section in par)

`border_radius` The corner radius of the background, given in points

`outline` The color of the outline stroke.

`outline_width` The line width of the outline stroke.

`outline_type` The linetype of the outline stroke, given as an an `lty` compatible value (See the *Line Type Specification* section in par)

`outline_join` The line join type for the outline. Either `"round"`, `"mitre"`, or `"bevel"`.

`outline_mitre` The mitre limit (relative distance between inner and outer corner at a join) if `outline_join = "mitre"`.

`bullets` A vector of strings to use for bullets in unordered lists. `marquee_bullets` provides a selection

`underline` Should text be underlined

`strikethrough` Should text be strikethrough

`baseline` The baseline shift to apply to the text

`img_asp` The default aspect ratio for block level images if not provided by the image itself

`text_direction` The directional flow of the text. Either `"auto"` to let it be determined by the content of the text, or `"ltr"`/`"rtl"` to hard-code it to either left-to-right or right-to-left. This setting will not change the order of glyphs within a span of text, but rather whether consequtive blocks of text are laid out left-to-right or right-to-left. It also affects to which side indentation is applied as well as the meaning of `"auto"`, and `"justified-auto"` aligment.

`border_size` **[Deprecated]** Use `border_width` instead

`ltr` Is the style intended for left-to-right text? This affects list indentation and citation border

Value

A style set object

Examples

```
classic_style(16, "serif", "sans")
```

`element_marquee` *ggplot2 theme element supporting marquee syntax*

Description

This theme element is a drop-in replacement for `ggplot2::element_text()`. It works by integrating the various style settings of the element into the base style of the provided style set. If a margin is given, it is set on the body tag with `skip_inherit()`. The default width is `NA` meaning that it will span as long as the given text is, doing no line wrapping. You can set it to any unit to make it fit within a specific width. However, this may not work as expected with rotated text (you may get lucky). Note that you may see small shifts in the visuals when going from `element_text()` to `element_marquee()` as size reporting may differ between the two elements.

Usage

```
element_marquee(
  family = NULL,
  colour = NULL,
  size = NULL,
  hjust = NULL,
  vjust = NULL,
  angle = NULL,
  lineheight = NULL,
  color = NULL,
  margin = NULL,
  style = NULL,
  width = NULL,
  inherit.blank = FALSE
)
```

Arguments

family	The font family of the base style
colour, color	The font colour of the base style
size	The font size of the base style
hjust	Horizontal justification (in $[0, 1]$)
vjust	Vertical justification (in $[0, 1]$)
angle	Angle (in $[0, 360]$)
lineheight	The lineheight of the base style
margin	The margin for the body tag. As margins in element_text() doesn't rotate along with angle we follow this behavior here as well so that the right margin becomes the bottom margin when rotating the text 90 degrees and so forth.
style	A style set to base the rendering on
width	The maximum width of the text. See the description for some caveats for this
inherit.blank	Should this element inherit the existence of an element_blank among its parents? If TRUE the existence of a blank element among its parents will cause this element to be blank as well. If FALSE any blank parent element will be ignored when calculating final element state.

Value

An element_marquee object that can be used in place of element_text in ggplot2 theme specifications

Note

grid, which marquee, ggplot2, etc are build upon contains a bug that means that the height of a grob is calculated before the grob knows it's width. The result of this is that if the width of an element_marquee() is NULL (the default), the text may overflow its allocated space with an additional line. Unfortunately there is no great fix for this, other than eyeball the width it has available and pass that to the element (e.g. element_marquee(width = grid::unit(10, "cm")))

Examples

```
library(ggplot2)
p <- ggplot(mtcars) +
  geom_point(aes(mpg, disp)) +
  labs(title = "A {.red *marquee*} title\n* Look at this bullet list\n\n* great, huh?") +
  theme_gray(base_size = 6) +
  theme(title = element_marquee())

plot(p)

ggplot(mtcars) +
  geom_histogram(aes(x = mpg)) +
  labs(title =
"I put a plot in your title so you can plot while you title

![](p)

What more could you _possibly_ want?") +
  theme(title = element_marquee())
```

---

geom_marquee *Draw text formatted with marquee*

Description

The geom is an extension of `geom_text()` and `geom_label()` that allows you to draw richly formatted text in marquee-markdown format in your plot. For plain text it is a near-drop-in replacement for the above geoms except some sizing might be very slightly different. However, using this geom you are able to access the much more powerful font settings available in marquee, so even then it might make sense to opt for this geom.

Usage

```
geom_marquee(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  size.unit = "mm",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`size.unit` How the `size` aesthetic is interpreted: as millimetres (`"mm"`, default), points (`"pt"`), centimetres (`"cm"`), inches (`"in"`), or picas (`"pc"`).

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification

Details

Styling of the text is based on a style set with the exception that the standard aesthetics such as family, size, colour, fill, etc. are recognized and applied to the base tag style. The default style set (classic_style) can be changed using the style aesthetic which can take a vector of style sets so that each text can rely on it's own style if needed. As with `element_marquee()`, the `fill` aesthetic is treated differently and not applied to the base tag, but to the body tag as a `skip_inherit()` style so as to not propagate the fill.

Contrary to the standard text and label geoms, `geom_marquee()` takes a `width` aesthetic that can be used to turn on soft wrapping of text. The default value (`NA`) lets the text run as long as it want's (honoring hard breaks), but setting this to something else will instruct marquee to use at most that amount of space. You can use grid units to set it to an absolute amount. The default means that if the label contains no text at all (e.g. it is only an image tag) then the width is zero and the content disappears. In that case, you must provide a width.

Value

A ggplot2 layer that can be added to a plot

Examples

```
library(ggplot2)
# Standard use
```

```
p <- ggplot(mtcars, aes(wt, mpg))
p + geom_marquee(aes(label = rownames(mtcars)))

# Make use of more powerful font features (note, result may depend on fonts
# installed on the system)
p + geom_marquee(
  aes(label = rownames(mtcars)),
  style = classic_style(weight = "thin", width = "condensed")
)

# Turn on line wrapping
p + geom_marquee(aes(label = rownames(mtcars)), width = unit(2, "cm"))

# Style like label
label_style <- modify_style(
  classic_style(margin = trbl(0)),
  "body",
  padding = skip_inherit(trbl(4)),
  border = "black",
  border_width = skip_inherit(trbl(1)),
  border_radius = 3
)
p + geom_marquee(aes(label = rownames(mtcars), fill = gear), style = label_style)

# Use markdown to style the text
red_bold_names <- sub("(\\w+)", "{.red **\\1**}", rownames(mtcars))
p + geom_marquee(aes(label = red_bold_names))
```

---

guide_marquee *Marquee subtitle guide*

Description

This legend appears similar to a subtitle and uses marquee syntax to typeset the text and interpolate legend glyphs.

Usage

```
guide_marquee(
  title = ggplot2::waiver(),
  style = marquee::style(background = NA),
  detect = FALSE,
  width = NULL,
  theme = NULL,
  position = "top",
  override.aes = list(),
  order = 1
)
```

Arguments

title	A single character string indicating the text to display. If NULL the title is not shown. If waiver() (default), the name of the scale or the name specified in labs() is used for the tyle.
style	Either a style_set to override style sets inherited from the theme, or a style for styling the labels specifically. For colour or fill scales, the color, background and border style properties are overridden when set as NULL, see examples.
detect	Either FALSE to typeset entirely through syntax or TRUE to automatically detect labels and apply.
width	The width of the textbox. If NULL it will take up the width of the panel, but due to issues with grid this might lead to an erroneous height calculation
theme	A theme object to style the guide individually or differently from the plot's theme settings. The theme argument in the guide partially overrides, and is combined with, the plot's theme. Arguments that apply to a single legend are respected, most of which have the legend-prefix. Arguments that apply to combined legends (the legend box) are ignored, including legend.position, legend.justification.*, legend.location and legend.box.*.
position	A character string indicating where the legend should be placed relative to the plot panels. One of "top", "right", "bottom", "left", or "inside".
override.aes	A list specifying aesthetic parameters of the legend keys. See details and examples in ?guide_legend.
order	positive integer less than 99 that specifies the order of this guide among multiple guides. This controls the order in which multiple guides are displayed, not the contents of the guide itself. If 0 (default), the order is determined by a secret algorithm.

Value

A GuideMarquee object that can be passed to the guides() function or used as the guide argument in a scale.

Text formatting

In addition to standard marquee syntax, there is additional syntax to make building a guide easier. In the text below, n marks the n-th break in the scale, label represents any of the scale's labels and foo represents arbitrary text.

- <<n>> or <<label>> can be used to insert key glyphs into the text.
- or can also be used to insert key glyphs into the text.
- {.n foo} or {.label foo} applies the style argument to foo, including recoloring when the guide represents a colour or fill scale.
- !!n or !!label translates to {.label label} to insert the label verbatim with the application of the style argument.

Examples

```
library(ggplot2)
# A standard plot
base <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

# Using key glyphs
base + aes(shape = drv) +
  scale_shape_discrete(
    # Same as using <<1>>, <<2>> and <<3>>,
    # or ![](1), ![](2) and ![](3)
    # or ![](4), ![](f) and ![](r)
    name = "Cars with four wheel <<4>>, forward <<f>> or reverse <<r>> drive.",
    guide = "marquee"
  )

# Recolouring text
base <- base +
  aes(colour = drv) +
  labs(
    colour = "Cars with {.4 four wheel}, {.f forward} or {.r reverse} drive."
  )
base + guides(colour = "marquee")

# Adjust display of labels
st <- style(weight = "bold", italic = TRUE, background = NA)
base + guides(colour = guide_marquee(style = st))

# Using background instead of text colour by setting it to NULL
st <- style(color = "black", background = NULL)
base + guides(colour = guide_marquee(style = st))

# Customising style of each label through style sets
# Note: tag names must be universal per `vctrs::vec_as_names` and
# prefixed with `lab_`.
st <- classic_style()
st <- modify_style(st, tag = "lab_f", background = NULL, color = "black")
st <- modify_style(st, tag = "lab_r", border_width = trbl(1),
                   color = "black", background = NA)
base + guides(colour = guide_marquee(style = st))

# Alternatively:
base + guides(colour = "marquee") +
  theme(plot.subtitle = element_marquee(style = st))

# Splicing in labels by number (!!2) or label (!!subcompact)
base + aes(colour = class) +
  labs(colour = "Cars including !!2 and !!subcompact vehicles") +
  guides(colour = "marquee")

# Using automatic detection
base + aes(colour = class) +
```

```
  labs(colour = "Cars including suv and minivan vehicles") +
  guides(colour = guide_marquee(detect = TRUE))
```

---

ink *Make justifications relative to the ink extent of the text*

Description

Marquee measures the extent of the box around text with bearings, that is, the height of the string "mean" is the same as the height of the string "median", despite the latter having a "d" extending upwards. This makes it easier to justification text irrespective of the glyphs used to render it. However, if you want alignment to be relative to the "tight" box around the text (the bounding box of where ink has been placed), you can use the `ink()` function to inform marquee of your intend. In general the effect is often minuscule for horizontal justifications but can have a big effect on vertical justification depending on the presence of ascenders and descenders in the rendered glyphs.

Usage

```
ink(x = numeric(), use_ink = TRUE)
```

Arguments

x	A string giving a valid justification or a numeric between 0 and 1
use_ink	Should the values be relative to the ink extend. Will be recycled to the length of x

Value

A `marquee_ink` vector

Examples

```
# Plot to illustrate the difference in vertical alignment
library(grid)
grid.newpage()
grid.draw(
  marquee_grob(
    c("### Textbox justification (default)",
      "### Bounding box justification (using `ink()`)"),
    x = 0.5,
    y = c(0.95, 0.45),
    hjust = 0.5,
    width = NA
  )
)

# Standard justification
grid.draw(
```

```
  marquee_grob(
    "mean",
    x = 0.5,
    y = 0.75,
    hjust = "right",
    vjust = 0.5,
    width = NA
  )
)
grid.draw(
  marquee_grob(
    "median",
    x = 0.5,
    y = 0.75,
    hjust = "left",
    vjust = 0.5,
    width = NA
  )
)

# Justification using `ink()`
grid.draw(
  marquee_grob(
    "mean",
    x = 0.5,
    y = 0.25,
    hjust = "right",
    vjust = ink(0.5),
    width = NA
  )
)
grid.draw(
  marquee_grob(
    "median",
    x = 0.5,
    y = 0.25,
    hjust = "left",
    vjust = ink(0.5),
    width = NA
  )
)
```

`marquee_glue` *Marquee-aware string interpolation*

Description

If you want to create your markdown programmatically you'd probably want to use some sort of string interpolation such as `glue()`. However, the custom span syntax of marquee interferes with the standard interpolation syntax of glue. This function let's you use both together.

Usage

```
marquee_glue(
  ...,
  .sep = "",
  .envir = parent.frame(),
  .open = "{",
  .close = "}",
  .na = "NA",
  .null = character(),
  .comment = character(),
  .literal = FALSE,
  .transformer = NULL,
  .trim = TRUE
)

marquee_glue_data(
  .x,
  ...,
  .sep = "",
  .envir = parent.frame(),
  .open = "{",
  .close = "}",
  .na = "NA",
  .null = character(),
  .comment = character(),
  .literal = FALSE,
  .transformer = NULL,
  .trim = TRUE
)
```

Arguments

`...`	[`expressions`] Unnamed arguments are taken to be expression string(s) to format. Multiple inputs are concatenated together before formatting. Named arguments are taken to be temporary variables available for substitution. For `glue_data()`, elements in `...` override the values in `.x`.
`.sep`	[`character(1)`: `""`] Separator used to separate elements.
`.envir`	[`environment`: `parent.frame()`] Environment to evaluate each expression in. Expressions are evaluated from left to right. If `.x` is an environment, the expressions are evaluated in that environment and `.envir` is ignored. If `NULL` is passed, it is equivalent to `emptyenv()`.
`.open`	[`character(1)`: `'\{'`] The opening delimiter. Doubling the full delimiter escapes it.
`.close`	[`character(1)`: `'\}'`] The closing delimiter. Doubling the full delimiter escapes it.

`.na` [character(1): 'NA']
Value to replace NA values with. If NULL missing values are propagated, that is an NA result will cause NA output. Otherwise the value is replaced by the value of .na.

`.null` [character(1): 'character()']
Value to replace NULL values with. If character() whole output is character(). If NULL all NULL values are dropped (as in paste0()). Otherwise the value is replaced by the value of .null.

`.comment` [character(1): '#']
Value to use as the comment character.

`.literal` [boolean(1): 'FALSE']
Whether to treat single or double quotes, backticks, and comments as regular characters (vs. as syntactic elements), when parsing the expression string. Setting .literal = TRUE probably only makes sense in combination with a custom .transformer, as is the case with glue_col(). Regard this argument (especially, its name) as experimental.

`.transformer` [function]
A function taking two arguments, text and envir, where text is the unparsed string inside the glue block and envir is the execution environment. A .transformer lets you modify a glue block before, during, or after evaluation, allowing you to create your own custom glue()-like functions. See vignette("transformers") for examples.

`.trim` [logical(1): 'TRUE']
Whether to trim the input template with trim() or not.

`.x` [listish]
An environment, list, or data frame used to lookup values.

Details

If you choose a different set of delimiters than "{" and "}" for the interpolation the functions will call the equivalent glue functions directly. However, if you keep the defaults, the functions will use a custom transformer that will make sure to keep the marquee custom span notation. You can both interpolate the content of the span, as well as the span class (see examples)

Value

A character vector

Examples

```
# standard use
red_text <- "this text will be red"
marquee_glue("This will be black and {.red {red_text}}!")

# if the span is not valid it will be treated as standard glue interpolation
try(
  marquee_glue("This will be black and {.red}!")
)
```

```
# You can interpolate the tag name as well
col <- "green"
marquee_glue("This will be black and {.{col} this text will be {col}}!")

# Tag name interpolation must follow a `.` or a `#` as these identify the
# bracket pair as a custom span class
col <- ".yellow"
# This is not what you want probably
marquee_glue("This will be black and {{col} this text will be {col}}!")

# Tag interpolation should also interpolate the full tag and be followed by
# a space in order to be valid
part <- "l"
marquee_glue("This will be black and {.ye{part}low this text will be {col}}!")
try(
  marquee_glue("This will be black and {.{part}avender this text will be {col}}!")
)
```

`marquee_grob` *Construct a grob rendering one or more markdown texts*

Description

This is the main function of marquee. It takes a vector of markdown strings, parses them with the provided style, and returns a grob capable of rendering the parsed text into rich text and (possibly) images. See `marquee_parse()` for more information about how markdown is parsed and see details below for further information on how rendering proceeds.

Usage

```
marquee_grob(
  text,
  style = classic_style(),
  ignore_html = TRUE,
  force_body_margin = FALSE,
  x = 0,
  y = 1,
  width = NULL,
  default.units = "npc",
  hjust = "left",
  vjust = "top",
  angle = 0,
  vp = NULL,
  name = NULL
)
```

Arguments

text	Either a character vector or a marquee_parsed object as created by marquee_parse()
style	A style set such as classic_style() that defines how the text should be rendered
ignore_html	Should HTML code be removed from the output
force_body_margin	Should the body margin override margin collapsing calculations. See Details.
x, y	The location of the markdown text in the graphics. If numeric it will be converted to units using default.units
width	The width of each markdown text. If numeric it will be converted to units using default.units. NULL is equivalent to the width of the parent container. NA uses the width of the text as the full width of the grob and will thus avoid any soft breaking of lines.
default.units	A string giving the default units to apply to x, y, and width
hjust	The horizontal justification of the markdown with respect to x. Can either be a numeric or one of "left", "left-ink", "center", "center-ink", "right-ink", or "right"
vjust	The vertical justification of the markdown with respect to y. Can either be a numeric or one of "bottom", "bottom-ink", "last-line", "center", "center-ink", "first-line", "top-ink", "top"
angle	The angle of rotation (in degrees) around x and y
vp	An optional viewport to assign to the grob
name	The name for the grob. If NULL a unique name will be generated

Value

A grob of class marquee

Rendering

marquee is first and foremost developed with the new 'glyph' rendering features in 4.3.0 in mind. However, not all graphics devices supports this, and while some might eventually do, it is quite concievable that some never will. Because of this, marquee has a fallback where it will render text as a mix of polygons and rasters (depending on the font in use) if the device doesn't report 'glyphs' capabilities. The upside is that it works (almost) everywhere, but the downside is that the fallback is much slower and with poorer visual quality. Because of this it is advisable to use a modern graphics device with glyphs support if at all possible.

Rendering style

The rendering more or less adheres to the styling provided by marquee_parse(), but has some intricacies as detailed below:

Tight lists

If a list is tight, the bottom margin of each li tag will be set so the spacing matches the lineheight. Further, the top margin will be set to 0.

Block images

In markdown, image tags are span elements so they can be placed inline. However, if an image tag is the only thing that is contained inside a p tag marquee determines that it should be considered a block element. In that case, the parent p element inherits the styling from the image element so that the image can e.g. adhere to `align` properties, or provide their own padding.

Horizontal rulers

These elements are rendered as an empty block. The standard style sets a bottom border size and no size for the other sides.

Margin collapsing

Margin calculations follows the margin collapsing rules of HTML. Read more about these at mdn. Margin collapsing means that elements with margin set to 0 might end up with a margin. Specifically for the body element this can be a problem if you want to enforce a tight box around your text. Because of this the `force_body_margin` argument allows you to overwrite the margins for the body element with the original values after collapsing has been performed.

Underline and strikethrough

Underlines are placed according to the font specification. Strikethrough are placed 0.3em above the baseline. The width of the line is set according to the fonr specification for underline width, both for underline and strikethrough. It inherits the color of the text.

Spans with background

Consecutive spans with the same background and border settings are merged into a single rectangle. The padding of the span defines the size of the background.

Bullet position

Bullets are placed, right-aligned, 0.25em to the left of the first line in the li element if the text direction is ltr. For rtl text it is placed, left-aligned, 0.25 em to the right of the first line.

Border with border radius

If borders are not the same on all sides they are drawn one by one. In this case the border radius is ignored.

Image rendering

The image tag can be used to place images. There are support for both png, jpeg, and svg images. If the path instead names a grob, ggplot, or patchwork object then this is rendered instead. If the file cannot be read, if it d oesn't exist, or if the path names an object that is not a grob, ggplot or patchwork, a placeholder is rendered in it's place (black square with red cross).

Image sizing

There is no standard in markdown for specifying the size of images. By default, block-level images fill the width of it's container and maintain it's aspect ratio. Inline images have a default width of 0.65em and a height matching the aspect ration.

However, if you wish to control sizing, you can instead provide the image as a grob with a viewport with fixed dimensions, in which case this will be used as long as the width doesn't exceed the width of the container (in which case it will get downsized). If a rastergrob is provided without absolute sizing, the aspect ratio will match the raster, otherwise the aspect ratio will be taken from the styling of the element (defaults to 1.65)

Table rendering

While marquee does not support the extended table syntax for markdown it does allow you to include tables in the output. It does so by supporting gt objects as valid paths in image tags in the same way as ggplots etc. This meeans that you can style your tables any way you wish and with the full power of gt, which is much more flexible than the markdown table syntax.

Textbox justification

The justification options exceeds the classic ones provided by grid. While numeric values are available as always, the number of possible text values are larger. Horizontal justification add `"left-ink"`, `"center-ink"`, and `"right-ink"` which uses the left-most and right-most positioned glyph (or halfway between them) as anchors. Vertical justification has the equivalent `"bottom-ink"`, `"center-ink"`, and `"top-ink"` anchors, but also `"first-line"` and `"last-line"` which sets the anchor at the baseline of the first or last line respectively.

`marquee_parse` *Parse a text as marquee*

Description

marquee uses an extension of CommonMark with no support for HTML code (it is rendered verbatim). The focus is to allow easy formatting of text for graphics, rather than fully fledged typesetting. See *marquee syntax* for more about the format.

Usage

```
marquee_parse(text, style = classic_style(), ignore_html = TRUE)
```

Arguments

`text`	A character string. The core quality of markdown is that any text is valid markdown so there is no restrictions on the content
`style`	A style set such as `classic_style()` that defines how the text should be rendered
`ignore_html`	Should HTML code be removed from the output

Value

A data frame describing the various tokens of the text and the style to apply to them. The output is mainly meant for programmatic consumption such as in `marquee_grob()`

marquee tags

marquee tokenizes the input text into blocks and spans. It recognises the following tags:

Block tags

`body` is the parent tag of a markdown document. It never contains any text itself, only other blocks.

`ul` is an unordered list. It contains a number of `li` children

`ol` is an ordered list. It contains a number of `li` children

`li` is a list element. If the list is tight it contains text directly inside of it. If not, text are placed inside child `p` blocks

`hr` is a horizontal line, spanning the width of the parent block. For styling, the bottom border size is used when rendering

`h1-h6` are headings at different levels

`cb` is a code block. Text inside code blocks are rendered verbatim, i.e. it cannot contain any children

`p` is a standard paragraph block. Text separated by two line-ends are separated into separate paragraphs

`qb` is a quote block. It may contain children

Span tags

`em` is an emphasized text span. Often this means italicizing the text, but it is ultimately up to the renderer

`str` is strong text, often rendered with bold text

`a` is a link text. While marquee rendering doesn't allow for links, it can still be rendered in a particular way

`code` is text rendered as code. Often this uses a monospaced font. Text inside this span is rendered verbatim

`u` is text that should be underlined

`del` is text that should have strikethrough

custom spans is a marquee specific extension to the syntax that allows you to make up tags on the fly. See the section on marquee syntax for more.

marquee syntax

marquee uses md4c which is a fully CommonMark compliant markdown parser. CommonMark is an effort to create an internally coherent markdown specification, something that was missing from the original markdown description. If you are used to writing markdown, you are used to CommonMark. Below is a list of notable additions or details about the specific way marquee handles CommonMark

Underlines and strikethrough

While not part of the basic CommonMark spec, underline and strikethrough are supported by marquee using _ and ~ (e.g. `_underline this_` and `~this was an error~`).

Images

Image tags (`![image title](path/to/image)`) are supported, but the title is ignored. The path is returned as the token text.

HTML

HTML tags are ignored, i.e. they are rendered verbatim. This is not that different from classic markdown rendering except that people often convert markdown to HTML where these tags suddenly have meaning. They do not carry any special significance when rendered with marquee

Custom tags

While markdown provides most of what is necessary for standard text markup, there are situations, especially in visualisation, where we need something more. Often users reach for inline HTML spans for that, but since HTML is fully ignored in marquee this is not an option. Further, adding in HTML decreases readability of the unformatted text a lot.

With marquee you can create a custom span using the `{.tag <some text>}` syntax, e.g. `{.sm small text}` to wrap "small text" in the `sm` tag. You can alternatively use `{#tag <some text>}` for the same effect. The only difference is that in the former syntax the `.` is stripped from the tag name, whereas in the latter the `#` remains part of the name. See the Styling section for the primal use of the latter syntax.

Styling

During parsing, each token is assigned a style based on the provided style set. The styling is cascading, but without the intricacies of CSS. A child element inherits the styling of it's parent for the options that are set to `NULL` in the style matching the child tag. Any style element that are `relative()` are computed based on the value of the parent style element. `em()` elements are resolved based on the size element of the child style, and `rem()` elements are resolved using the size element of the `body` style. If a style is not provided for the tag, it fully inherits the style of it's parent.

Automatic coloring Recognizing that the primary use for custom tags may be to change the color of some text, marquee provides a shortcut for this. If a style is not found for the tag in the provided style set, marquee will check if the tag matches a valid color (i.e. a string from `grDevices::colors()`, or a valid hex string, e.g. `#53f2a9`). If it is a valid color it will set this as the font color of the style. This means that parsing `"Color {.red this} red"` automatically sets the color of `"this"` to red, even if no style is provided for the `red` tag. Likewise, parsing `"Color {#00FF00 me} green"` will automatically set the color of `"me"` to #00FF00 (fully saturated green).

Automatic sizing Like the automatic coloring described above, marquee also offers a shortcut for changing the size of text on the fly. Any class consisting solely of numbers will (if the class is not explicitly defined) by considered a text sizing class. So, parsing `"This {.50 text} is big"` automatically sets the font size of `"text"` to `50`.

Additional parsing information

Apart from splitting the text up into tokens, `marquee_parse()` also provides some additional information useful for rendering the output in the expected way. The `id` column refers the tokens back to the original input text, the `block` relates tokens together into blocks. Block elements increment the block count when they are entered, and decrement it when they are excited. The `type` column provides the type of the block. The `indentation` column provides the node level in the tree. A child block will increase the indentation for as long as it is active. `ol_index` provides the number associated with the ordered list element. `tight` indicates whether the list is tight (i.e. it was provided with no empty lines between list elements). The `ends` column indicate until which row in the output the tag is active (i.e. the tag is closed after the row indicated by the value in this column).

Examples

```
marquee_parse("# Header of the example\nSome body text", classic_style())
```

`marquefy_theme` *Convert all text elements in a theme to marquee elements*

Description

While `element_marquee()` should behave similar to `ggplot2::element_text()` when used on plain text (i.e. text without any markdown markup), the reality can be different. This is because the text shaping engine used by marquee (`textshaping::shape_text()`) may differ from the one used by the graphics device (which is responsible for laying out text in `element_text()`). Differences can range from slight differences in letter spacing to using a different font altogether (this is because the font keywords `""`, `"sans"`, `"serif"`, `"mono"`, and `"symbol"` may be mapped to different fonts depending on the shaper). One way to handle this is to provide an explicit font name for the elements, but alternatively you can use this function to convert all text elements in a theme to `element_marquee()`

Usage

```
marquefy_theme(theme)
```

Arguments

`theme`	A (complete) ggplot2 theme

Value

`theme` with all text elements substituted for marquee elements

Examples

```
library(ggplot2)
ggplot(mtcars) +
  geom_point(aes(disp, mpg)) +
  ggtitle("How about that") +
  marquefy_theme(theme_gray())
```

style *Create a style specification for a single tag*

Description

`style()` constructs a `marquee_style` object specifying the styling for a single tag. The meaning of `NULL` is to inherit the value from the parent element. It follows that top parent (the body element), must have values for all it's options. The `base_style()` constructor is a convenient constructor for a style with sensible defaults for all it's options.

Usage

```
style(
  family = NULL,
  weight = NULL,
  italic = NULL,
  width = NULL,
  features = NULL,
  size = NULL,
  color = NULL,
  lineheight = NULL,
  align = NULL,
  tracking = NULL,
  indent = NULL,
  hanging = NULL,
  margin = NULL,
  padding = NULL,
  background = NULL,
  border = NULL,
  border_width = NULL,
  border_type = NULL,
  border_radius = NULL,
  outline = NULL,
  outline_width = NULL,
  outline_type = NULL,
  outline_join = NULL,
  outline_mitre = NULL,
  bullets = NULL,
  underline = NULL,
  strikethrough = NULL,
  baseline = NULL,
  img_asp = NULL,
  text_direction = NULL,
  border_size = deprecated()
)

base_style(
```

```
  family = "",
  weight = "normal",
  italic = FALSE,
  width = "normal",
  features = systemfonts::font_feature(),
  size = 12,
  color = "black",
  lineheight = 1.6,
  align = "auto",
  tracking = 0,
  indent = 0,
  hanging = 0,
  margin = trbl(0, 0, rem(1)),
  padding = trbl(0),
  background = NA,
  border = NA,
  border_width = trbl(0),
  border_type = "solid",
  border_radius = 0,
  outline = NA,
  outline_width = 1,
  outline_type = "solid",
  outline_join = "round",
  outline_mitre = 10,
  bullets = marquee_bullets,
  underline = FALSE,
  strikethrough = FALSE,
  baseline = 0,
  img_asp = 1.65,
  text_direction = "auto",
  border_size = deprecated()
)
```

Arguments

family	The name of the font family to use
weight	The font weight to use. Can either be a number (0, 100, 200, 300, 400, 500, 600, 700, 800, or 900) or a strings ("undefined", "thin", "ultralight", "light", "normal", "medium", "semibold", "bold", "ultrabold", or "heavy")
italic	Should the font be slanted
width	The font width to use. Can either be a number (‘0‘, ‘1‘, ‘2‘, ‘3‘, ‘4‘, ‘5‘, ‘6‘, ‘7‘, ‘8‘, or ‘9‘) or strings (‘"undefined"‘, ‘"ultracondensed"‘, ‘"extracondensed"‘, ‘"condensed"‘, ‘"semicondensed"‘, ‘"normal"‘, ‘"semiexpanded"‘, ‘"expanded"‘, ‘"extraexpanded"‘, or ‘"ultraexpanded"‘)
features	A font_feature object specifying any OpenType font features to apply to the font
size	The size of the font in points. Can be relative() or em() in which case it is based on the parent font size (for size these are equivalent) or rem() in which case it is based on the font size of the body element.

color	Is the color of the font
lineheight	The spacing between subsequent lines relative to the font size. Can be relative() in which case it is based on the parent lineheight.
align	The alignment within the text. One of "left", "center", "right", "justified-left", "justified-center", "justified-right", or "distributed"
tracking	Additional character spacing measured in 1/1000em. Can be relative() in which case it is based on the parent tracking.
indent	The indentation of the first line in a paragraph measured in points. Can be relative() in which case it is based on the parent indent, em() in which case it is based on the font size in this style, or rem() in which case it is based on the font size of the body element.
hanging	The indentation of all but the first line in a paragraph measured in points. Can be relative() in which case it is based on the parent hanging, em() in which case it is based on the font size in this style, or rem() in which case it is based on the font size of the body element.
margin	The margin around the element, given as a call to trbl(). Margin refers to the area outside the box that text is placed in. If the element has a background, the margin area will not be colored.
padding	The padding around the element, given as a call to trbl(). Padding refers to the distance between the text and the border of the box it will be drawn in. If the element has a background, the padding area will be colored.
background	The color of the background fill. The background includes the padding but not the margin. Can be a solid color or a gradient or pattern made with grid::linearGradient()/grid::rad
border	The color of the background stroke. The background includes the padding but not the margin
border_width	The line width of the background stroke, given as a call to trbl()
border_type	The linetype of the background stroke, given as an an lty compatible value (See the *Line Type Specification* section in par)
border_radius	The corner radius of the background, given in points
outline	The color of the outline stroke.
outline_width	The line width of the outline stroke.
outline_type	The linetype of the outline stroke, given as an an lty compatible value (See the *Line Type Specification* section in par)
outline_join	The line join type for the outline. Either "round", "mitre", or "bevel".
outline_mitre	The mitre limit (relative distance between inner and outer corner at a join) if outline_join = "mitre".
bullets	A vector of strings to use for bullets in unordered lists. marquee_bullets provides a selection
underline	Should text be underlined
strikethrough	Should text be strikethrough
baseline	The baseline shift to apply to the text
img_asp	The default aspect ratio for block level images if not provided by the image itself

text_direction The directional flow of the text. Either "auto" to let it be determined by the content of the text, or "ltr"/"rtl" to hard-code it to either left-to-right or right-to-left. This setting will not change the order of glyphs within a span of text, but rather whether consequtive blocks of text are laid out left-to-right or right-to-left. It also affects to which side indentation is applied as well as the meaning of "auto", and "justified-auto" aligment.

border_size **[Deprecated]** Use border_width instead

Value

A marquee_style object

Examples

```
# A partial style
style(color = "red", underline = TRUE)

# Full style
base_style()
```

---

style-helpers *Helpers for defining styles*

Description

marquee provides a small set of helpers for constructing the needed styles. relative() specifies a numeric value as relative to the value of the parent style by a certain factor, e.g. a font size of relative(0.5) would give a style a font size half of it's parent. em() specify a numeric value as relative to the font size of the current style. If the font size is 12, and indent is set to em(2), then the indent will be equivalent to 24. rem() works like em() but rather than using the font size of the current style it uses the font size of the root style (which is the body element). trbl() helps you construct styles that refers to sides of a rectangle (margin, padding, and border size). The function names refers to the order of the arguments (top, right, bottom, left). skip_inherit() tells the style inheritance to ignore this value and look for the value one above in the stack. marquee_bullets is just a character vector with 6 sensible bullet glyphs for unordered lists.

Usage

```
relative(x)

em(x)

rem(x)

trbl(top = NULL, right = top, bottom = top, left = right)

skip_inherit(x)
```

```
marquee_bullets
```

Arguments

x	A decimal number. If a vector is provided only the first element will be used
top, right, bottom, left	Values for the sides of the rectangles. Either numbers or modifiers (relative, em, or rem)

Format

An object of class character of length 6.

Value

Objects of the relevant class

Examples

```
relative(0.35)

em(2)

rem(1.2)

# Argument default means it recycles like CSS if fewer values are specified
trbl(6, em(1.5))

skip_inherit("sans")

marquee_bullets
```

---

style_set *Create or modify a style set that describes a full markdown text*

Description

A style set contains information on how to style the various tags in a markdown text. While it is not necessary to provide a style for all tags (it will just inherit the parent if missing), it is required to provide a complete style for the body tag so an option is avialable through inheritance for all tags and all style options. It can often be easier to derive a new style set from an existing one rather than building one from scratch.

Usage

```
style_set(...)

modify_style(x, tag, ...)

remove_style(x, tag)
```

Arguments

...	Named arguments providing a style for the specific tags. For `modify_style()` a number of style options to change. If the first argument is a marquee style it will overwrite the tag and subsequent arguments are ignored. This only holds if `x` is a style set.
x	A style or style set to modify
tag	The name of the tag to modify or remove if `x` is a style set. Tags are internally all lowercase and `tag` will be converted to lowercase before matching

Value

A `marquee_style_set` object

Examples

```
# Create a style
s_set <- style_set(base = base_style(), p = style(indent = em(2)))

# Modify an existing tag
modify_style(s_set, "p", size = 16)

# Add a new tag, supplying a full style object
modify_style(s_set, "str", style(weight = "bold"))

# Same as above, but style object created implicitly
modify_style(s_set, "str", weight = "bold")

# Remove a tag style
remove_style(s_set, "p")
```

Index