Package 'munch'

April 16, 2026

Title Rich Inline Text for 'grid' Graphics and 'flextable'

Version 0.0.2

Description Renders rich inline text (bold, italic, code, links, images) in grid graphics and 'ggplot2', from markdown or 'flextable' chunks. Provides grobs, theme elements, and geometry layers for styled text rendering. Only works with graphics devices that support 'systemfonts', such as those provided by 'ragg', 'svglite', or 'ggiraph'. The 'cairo_pdf' device is also supported when fonts are installed at the system level.

License MIT + file LICENSE

URL https://ardata-fr.github.io/munch/

BugReports https://github.com/ardata-fr/munch/issues

Imports cli, commonmark, flextable, patchwork, gdtools, ggplot2, grid, methods, systemfonts, xml2

Suggests doconv, ggiraph, knitr, magick, ragg, rmarkdown, rsvg, svglite, testthat (>= 3.0.0), withr

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author David Gohel [aut, cre], ArData [cph]

Maintainer David Gohel <david.gohel@ardata.fr>

Repository CRAN

Date/Publication 2026-04-16 09:10:02 UTC

Contents

`munch-package` *munch: Rich Inline Text for Grid Graphics and Flextable*

Description

The 'munch' package renders rich inline text (bold, italic, code, links, images) in grid graphics and 'ggplot2', from markdown or 'flextable' chunks.

For 'ggplot2' integration, use `element_md()` and `element_chunks()` in theme elements, or `geom_text_md()` and `geom_label_md()` for annotations.

Advanced usage

Use `md_grob()` to convert markdown text to a grid grob, or `chunks_grob()` to convert 'flextable' paragraph objects (for superscripts, subscripts, and custom colors not available in markdown).

Limitations

Only graphics devices that support 'systemfonts' are compatible: 'ragg', 'svglite', or 'ggiraph'. The `cairo_pdf()` device is also supported when fonts are installed at the system level. Standard R devices (`pdf()`, `png()`, `jpeg()`) are not compatible but can still produce correct output if fonts are made available to the device (e.g. via 'extrafont'). See the "Getting started" vignette for details.

LaTeX and MathJax equations are not supported. For simple scientific notation, use `as_sup()` and `as_sub()` from 'flextable'.

Only inline formatting is supported: bold, italic, code, strikethrough, links, and images. Block elements such as headings, bullet lists, and numbered lists are not available.

Options

`munch.skip_device_check` If set to `TRUE`, the device compatibility warning is suppressed. Use this when you have ensured that the fonts used by 'munch' are available to the active graphics device (e.g. via 'extrafont' for `pdf()`). Set with `options(munch.skip_device_check = TRUE)`.

Author(s)

Maintainer: David Gohel <david.gohel@ardata.fr>

Other contributors:

- ArData [copyright holder]

See Also

Useful links:

- https://ardata-fr.github.io/munch/
- Report bugs at https://github.com/ardata-fr/munch/issues

---

as_paragraph_md *Convert Markdown to a Paragraph Object*

Description

Converts a markdown string to a paragraph object compatible with flextable. This allows using markdown syntax to format text in flextable cells.

Usage

```
as_paragraph_md(x, font_family_mono = "mono")
```

Arguments

x	A character string containing markdown.
font_family_mono	Font family to use for inline code. Defaults to "mono".

Details

Supported markdown syntax:

- **bold** or __bold__
- *italic* or _italic_
- `code` (monospace)
- ~~strikethrough~~
- [link](url) (underlined and colored)
- ![alt](path) (images)
- Line breaks with \n\n (paragraph) or two spaces + \n (hard break)

Value

A paragraph object (class "paragraph") that can be used with flextable::mk_par() or flextable::as_flextable().

Examples

```
library(flextable)

ft <- flextable(head(iris, 3))
ft <- mk_par(
  ft,
  j = 1,
  part = "body",
  value = as_paragraph_md("**Column1** has *values*")
)
ft

# With inline code
ft <- mk_par(
  ft,
  j = 2,
  part = "body",
  value = as_paragraph_md("Use `Sepal.Width` column")
)
ft
```

---

chunks_grob *Create a Grob from flextable Chunks*

Description

Creates a grob directly from flextable chunk data.frames (created with `flextable::as_paragraph()`), bypassing markdown parsing.

Usage

```
chunks_grob(
  chunks,
  x = 0.5,
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  width = NULL,
  line_spacing = 1.2,
  img_baseline_ratio = 0.15,
  name = NULL
)
```

Arguments

chunks	A chunk data.frame from flextable (e.g., from `flextable::as_paragraph()`).
x	X position (0-1 in npc units).
y	Y position (0-1 in npc units).

hjust	Horizontal justification.
vjust	Vertical justification.
width	Maximum width in inches (NULL for no wrapping).
line_spacing	Line height multiplier.
img_baseline_ratio	Controls vertical alignment of inline images. Specifies the proportion of the image height below the text baseline: `0` = bottom at baseline, `0.15` = default, `0.35` = centered on text, `0.5` = centered on baseline.
name	Grob name.

Value

A grob of class "mdGrob".

Examples

```
#

library(grid)
library(flextable)
library(ggiraph)

library(gdtools)
font_set_liberation()
flextable::set_flextable_defaults(
  font.size = 12,
  font.family = "Liberation Sans"
)

chunks <- as_paragraph(
  as_chunk("E = mc"),
  as_sup("2")
)
gr <- chunks_grob(chunks)

x <- girafe(
  width_svg = 1,
  height_svg = 1,
  bg = "wheat",
  options = list(opts_sizing(rescale = FALSE)),
  code = {
    grid.newpage()
    grid.draw(gr)
  },
  dependencies = list(
    liberationsansHtmlDependency()
  )
)

x
```

```
init_flextable_defaults()
```

`default_text_props` *Default Text Properties for Markdown Grobs*

Description

Returns a list of text properties used for rendering markdown text. Default values are taken from `flextable::get_flextable_defaults()`, allowing integration with flextable's theming system while providing a standalone API for munch.

Usage

```
default_text_props(
  font_size = NULL,
  font_family = NULL,
  font_color = NULL,
  code_font_family = NULL,
  line_spacing = NULL,
  img_baseline_ratio = NULL
)
```

Arguments

`font_size`	Font size in points.
`font_family`	Font family name for regular text.
`font_color`	Text color.
`code_font_family`	Font family name for inline code (backticks). Defaults to `"mono"`.
`line_spacing`	Line height multiplier.
`img_baseline_ratio`	Controls vertical alignment of inline images. Specifies the proportion of the image height that appears below the text baseline: • `0`: image bottom at baseline • `0.15`: (default) similar to text descent proportions • `0.35`: image centered on text vertical center • `0.5`: image centered on baseline

Details

When arguments are NULL (the default), values are inherited from flextable::get_flextable_defaults(). This allows users to either:

1. Use munch independently by specifying all properties:

```
props <- default_text_props(font_size = 12, font_family = "serif")
md_grob("**bold**", text_props = props)
```

2. Use flextable's theming system:

```
flextable::set_flextable_defaults(font.size = 14)
md_grob("**bold**")
```

Value

A list of class "text_props" with elements:

- font.size: Font size in points
- font.family: Font family name
- font.color: Text color
- code.font.family: Font family for code
- line_spacing: Line height multiplier
- img_baseline_ratio: Image baseline alignment ratio

Some element names use dot separators (font.size, font.family, etc.) for compatibility with 'flextable' conventions. Function parameters and newer elements use snake_case (line_spacing, img_baseline_ratio) which is the preferred convention going forward.

Examples

```
library(grid)


# Get current defaults
default_text_props()

# Override specific properties
default_text_props(font_size = 14, font_color = "navy")

library(gdtools)
font_set_liberation()

# Specify a monospace font for inline code
props <- default_text_props(
  font_family = "Liberation Sans",
  code_font_family = "Liberation Mono"
)

if (require(ggiraph, quietly = TRUE)) {
  x <- girafe(
```

```
    width_svg = 2,
    height_svg = 1,
    bg = "wheat",
    options = list(opts_sizing(rescale = FALSE)),
    code = {
      grid.newpage()
      gr <- md_grob("Text with `inline code`", text_props = props)
      grid.draw(gr)
    },
    dependencies = list(
      liberationsansHtmlDependency(),
      liberationmonoHtmlDependency()
    )
  )

  print(x)
}
```

`element_chunks` *Chunks Theme Element for 'ggplot2'*

Description

A theme element that renders 'flextable' chunks directly in plot titles, subtitles, captions, and other text positions. This allows using superscripts, subscripts, and other formatting not available in markdown.

Usage

```
element_chunks(
  chunks,
  hjust = NULL,
  vjust = NULL,
  angle = NULL,
  lineheight = NULL,
  size = NULL,
  margin = NULL,
  debug = FALSE,
  inherit.blank = FALSE
)
```

Arguments

chunks	A paragraph object created with `flextable::as_paragraph()` containing chunks such as `flextable::as_chunk()`, `flextable::as_sup()`, `flextable::as_sub()`, etc.
hjust	Horizontal justification (0-1).
vjust	Vertical justification (0-1).

angle	Text rotation angle in degrees.
lineheight	Line height multiplier.
size	font size
margin	Margins around the text (use ggplot2::margin()).
debug	Show debug rectangles.
inherit.blank	Should this element inherit blank elements.

Value

An element of class c("element_chunks", "element_text", "element").

Examples

```
library(ggplot2)
library(flextable)
library(ggiraph)

library(gdtools)
font_set_liberation()
flextable::set_flextable_defaults(
  font.size = 12,
  font.family = "Liberation Sans"
)

# Caption with subscript (H2O)
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_minimal(base_family = "Liberation Sans") +
  theme(
    plot.caption = element_chunks(
      as_paragraph(
        as_chunk("Chemical formula: "),
        as_chunk("H"),
        as_sub("2"),
        as_chunk("O")
      )
    )
  )

x <- girafe(
  ggobj = p,
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)
x

# Title with superscript
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
```

```
  theme_minimal(base_family = "Liberation Sans") +
  theme(
    plot.title = element_chunks(
      as_paragraph(
        as_chunk("Model fit: R"),
        as_sup("2"),
        as_chunk(" = 0.87")
      )
    )
  )

x <- girafe(
  ggobj = p,
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)

x
```

---

element_md *Markdown Theme Element for ggplot2*

Description

A theme element that renders text labels as markdown. Use in place of ggplot2::element_text() for plot titles, subtitles, captions, axis labels, etc.

Usage

```
element_md(
  family = NULL,
  code_font_family = NULL,
  face = NULL,
  colour = NULL,
  size = NULL,
  hjust = NULL,
  vjust = NULL,
  angle = NULL,
  lineheight = NULL,
  color = NULL,
  margin = NULL,
  debug = FALSE,
  inherit.blank = FALSE
)
```

Arguments

family	Font family.
code_font_family	Font family for inline code. Defaults to "mono".
face	Font face ("plain", "bold", "italic", "bold.italic").
colour, color	Text color.
size	Font size in points.
hjust	Horizontal justification (0-1).
vjust	Vertical justification (0-1).
angle	Text rotation angle in degrees.
lineheight	Line height multiplier.
margin	Margins around the text (use ggplot2::margin()).
debug	Show debug rectangles.
inherit.blank	Should this element inherit blank elements.

Value

An element of class c("element_md", "element_text", "element").

Examples

```
library(ggplot2)

library(gdtools)
font_set_liberation()
flextable::set_flextable_defaults(
  font.size = 12,
  font.family = "Liberation Sans"
)

p1 <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  labs(title = "**Bold title** with *italic*") +
  theme(plot.title = element_md())

girafe(
  ggobj = p1,
  dependencies = list(
    liberationsansHtmlDependency()
  )
)

p2 <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  labs(
    title = "**MPG** vs *Weight*",
    caption = "Source: `mtcars` dataset"
```

```
  ) +
  theme(
    plot.title = element_md(size = 16),
    plot.caption = element_md(size = 9)
  )
girafe(
  ggobj = p2,
  check_fonts_dependencies = TRUE
)
```

---

geom_label_md *Draw Markdown Text with a Background Box in ggplot2*

Description

A ggplot2 geom for rendering markdown-formatted text with a background rectangle, similar to ggplot2::geom_label(). Supports basic markdown formatting (bold, italic, code, strikethrough, links, images).

Usage

```
geom_label_md(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  width = NA,
  code_font_family = NULL,
  label.padding = unit(0.25, "lines"),
  label.r = unit(0.15, "lines"),
  label.size = 0.25,
  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.

width — Maximum width for text wrapping in inches. NA (default) means no wrapping. Use "auto" to wrap at the plot panel edge: the wrap width is computed from each label's x position and hjust so that text stays within the panel.

code_font_family — Font family for inline code (backticks). Defaults to "mono" when NULL.

label.padding — Amount of padding around label (defaults to 0.25 lines).

label.r — Radius of rounded corners (defaults to 0.15 lines).

label.size — Size of label border in mm.

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, e.g. annotation_borders().

Details

The geom supports the same markdown syntax as geom_text_md().

The fill aesthetic controls the background color of the label box, while colour controls both the text color and the border color.

Value

A ggplot2 layer that can be added to a plot.

Aesthetics

geom_label_md() understands the following aesthetics (required in bold):

- x
- y
- label
- alpha
- angle
- colour
- family

- fill
- hjust
- lineheight
- size
- vjust

Examples

```
library(ggplot2)

library(gdtools)
font_set_liberation()

# Basic usage
df <- data.frame(
  x = 1:3,
  y = 1:3,
  label = c("**Bold**", "*Italic*", "`Code`")
)

if (require(ggiraph, quietly = TRUE)) {
  x <- girafe(
    ggplot(df, aes(x, y, label = label)) +
      geom_label_md(),
    options = list(opts_sizing(rescale = FALSE)),
    dependencies = list(
      liberationsansHtmlDependency(),
      liberationmonoHtmlDependency()
    )
  )

  print(x)

  x <- girafe(
    # With custom fill
    ggplot(df, aes(x, y, label = label)) +
      geom_label_md(fill = "lightyellow"),
    options = list(opts_sizing(rescale = FALSE)),
    dependencies = list(
      liberationsansHtmlDependency(),
      liberationmonoHtmlDependency()
    )
  )

  print(x)

  x <- girafe(
    # Annotation with statistics
    ggplot(mtcars, aes(mpg, wt)) +
      geom_point() +
      geom_smooth(method = "lm", se = FALSE) +
```

```
    geom_label_md(
      label = "**R\u00B2 = 0.75**\n\n*p < 0.001*",
      x = 30,
      y = 5,
      hjust = 1,
      vjust = 1,
      fill = "white"
    ),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)

print(x)
}
```

---

geom_text_md *Draw Markdown Text in ggplot2*

Description

A ggplot2 geom for rendering markdown-formatted text. This geom is similar to ggplot2::geom_text() but supports basic markdown formatting (bold, italic, code, strikethrough, links, images).

Usage

```
geom_text_md(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  width = NA,
  code_font_family = NULL,
  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.

`width` Maximum width for text wrapping in inches. `NA` (default) means no wrapping. Use `"auto"` to wrap at the plot panel edge: the wrap width is computed from each label's `x` position and `hjust` so that text stays within the panel.

`code_font_family` Font family for inline code (backticks). Defaults to `"mono"` when `NULL`.

`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, e.g. `annotation_borders()`.

Details

The geom supports the following markdown syntax:

- `**bold**` or `__bold__`
- `*italic*` or `_italic_`
- `` `code` `` (monospace)
- `~~strikethrough~~`
- `[link](url)` (underlined and colored)
- `![alt](path)` (images)

Text properties (font size, family, color) are controlled by the `size`, `family`, and `colour` aesthetics respectively.

Value

A ggplot2 layer that can be added to a plot.

Aesthetics

`geom_text_md()` understands the following aesthetics (required in bold):

- `x`
- `y`

- label
- alpha
- angle
- colour
- family
- hjust
- lineheight
- size
- vjust

Examples

```
#
library(ggplot2)

library(gdtools)
font_set_liberation()

# ------------------------------------------------------------------------

# Basic usage with markdown formatting
# ------------------------------------------------------------------------
df <- data.frame(
  x = 1:3,
  y = 1:3,
  label = c("**Bold**", "*Italic*", "`Code`")
)

x <- girafe(
  ggplot(df, aes(x, y, label = label)) +
    geom_text_md(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)
x

# ------------------------------------------------------------------------
# Customizing size and colour aesthetics
# ------------------------------------------------------------------------
df <- data.frame(
  x = c(1, 2, 3),
  y = c(1, 2, 3),
  label = c(
    "**Small** text",
    "**Medium** text",
    "**Large** text"
```

```
  ),
  size = c(3, 5, 7),
  color = c("steelblue", "darkgreen", "firebrick")
)

x <- girafe(
  ggplot(df, aes(x, y, label = label, size = size, colour = color)) +
    geom_text_md(width = 1) +
    scale_size_identity() +
    scale_colour_identity() +
    scale_x_continuous(expand = c(.15, .15)) +
    theme_minimal(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Using different font families
# ------------------------------------------------------------------
df <- data.frame(
  x = c(1, 1, 1),
  y = c(3, 2, 1),
  label = c(
    "**Sans-serif** font (*default*)",
    "**Serif** font (*elegant*)",
    "**Monospace** font (`code style`)"
  ),
  family = c("Liberation Sans", "Liberation Serif", "Liberation Mono")
)

x <- girafe(
  ggplot(df, aes(x, y, label = label, family = family)) +
    geom_text_md(hjust = 0) +
    xlim(0.5, 3) +
    theme_void(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationserifHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Text rotation with angle aesthetic
# ------------------------------------------------------------------
df <- data.frame(
  x = c(1, 2, 3, 4),
```

```
  y = c(1, 1, 1, 1),
  label = "**Rotated** *text*",
  angle = c(0, 45, 90, -45)
)

x <- girafe(
  ggplot(df, aes(x, y, label = label, angle = angle)) +
    geom_text_md(size = 4) +
    scale_x_continuous(expand = c(.15, .15)) +
    ylim(0.5, 1.5) +
    theme_minimal(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Justification with hjust and vjust
# ------------------------------------------------------------------
df <- data.frame(
  x = rep(2, 9),
  y = rep(2, 9),
  hjust = rep(c(0, 0.5, 1), 3),
  vjust = rep(c(0, 0.5, 1), each = 3),
  label = paste0(
    "**h=",
    rep(c(0, 0.5, 1), 3),
    "**\n*v=",
    rep(c(0, 0.5, 1), each = 3),
    "*"
  )
)

x <- girafe(
  ggplot(df, aes(x, y, label = label, hjust = hjust, vjust = vjust)) +
    geom_point(color = "red", size = 3) +
    geom_text_md(size = 3) +
    facet_grid(vjust ~ hjust, labeller = label_both) +
    xlim(1, 3) +
    ylim(1, 3) +
    theme_minimal(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Text wrapping with width parameter
# ------------------------------------------------------------------
```

```
df <- data.frame(
  x = 1,
  y = 1,
  label = paste0(
    "This is a **long text** that will be *wrapped* ",
    "automatically when it exceeds the specified `width`."
  )
)

x <- girafe(
  ggplot(df, aes(x, y, label = label)) +
    geom_text_md(width = 2, hjust = 0) +
    xlim(0.5, 3) +
    theme_void(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Alpha transparency
# ------------------------------------------------------------------
df <- data.frame(
  x = 1:4,
  y = rep(1, 4),
  label = "**Text**",
  alpha = c(1, 0.7, 0.4, 0.2)
)

x <- girafe(
  ggplot(df, aes(x, y, label = label, alpha = alpha)) +
    geom_text_md(size = 8) +
    scale_alpha_identity() +
    theme_minimal(),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)
x

# ------------------------------------------------------------------
# Practical example: Annotated scatter plot
# ------------------------------------------------------------------
x <- girafe(
  ggplot(mtcars, aes(mpg, wt)) +
    geom_point(colour = "steelblue", size = 2) +
    geom_smooth(method = "lm", se = FALSE, colour = "firebrick") +
    geom_text_md(
      label = "**R\u00B2 = 0.75**\n*p < 0.001*\n\n`lm(wt ~ mpg)`",
```

```
      x = 32,
      y = 5,
      hjust = 1,
      vjust = 1,
      size = 3.5,
      colour = "gray30"
    ) +
    labs(
      title = "MPG vs Weight",
      x = "Miles per Gallon",
      y = "Weight (1000 lbs)"
    ) +
    theme_minimal(base_family = "Liberation Sans"),
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency(),
    liberationmonoHtmlDependency()
  )
)
x
```

`md_grob` *Create a Grob from Markdown Text*

Description

Parses markdown text and creates a grid grob that can be used in ggplot2 with `annotation_custom()` or directly with grid.

Usage

```
md_grob(
  md_text,
  x = 0.5,
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  width = NULL,
  line_spacing = 1.2,
  text_props = NULL,
  name = NULL
)
```

Arguments

`md_text`	Character string with markdown formatting.
`x`	X position (0-1 in npc units, default 0.5).

y	Y position (0-1 in npc units, default 0.5).
hjust	Horizontal justification (0=left, 0.5=center, 1=right).
vjust	Vertical justification (0=bottom, 0.5=center, 1=top).
width	Maximum width in inches (NULL for no wrapping).
line_spacing	Line height multiplier (default 1.2).
text_props	Text properties created with default_text_props(). If NULL, uses current defaults from flextable::get_flextable_defaults(). Use img_baseline_ratio in text_props to control image vertical alignment.
name	Grob name (optional).

Value

A grob of class "mdGrob".

Examples

```
library(grid)

library(gdtools)
font_set_liberation()
flextable::set_flextable_defaults(
  font.size = 12,
  font.family = "Liberation Sans"
)

if (require(ggiraph, quietly = TRUE)) {
  x <- girafe(
    width_svg = 3,
    height_svg = 1,
    bg = "wheat",
    options = list(opts_sizing(rescale = FALSE)),
    code = {
      # Simple markdown (uses flextable defaults)
      gr <- md_grob("This is **bold** and *italic* text.")
      grid.newpage()
      grid.draw(gr)
    },
    dependencies = list(
      liberationsansHtmlDependency()
    )
  )

  print(x)

  x <- girafe(
    width_svg = 3,
    height_svg = 1,
    bg = "wheat",
    options = list(opts_sizing(rescale = FALSE)),
    code = {
```

```
    # Customize with text_props (standalone, no flextable dependency)
    props <- default_text_props(font_size = 14, font_color = "navy")
    gr2 <- md_grob("Code example: `print(x)`", text_props = props)
    grid.newpage()
    grid.draw(gr2)
  },
  dependencies = list(
    liberationsansHtmlDependency()
  )
)

print(x)
}
```

`munch_annotation` *Patchwork Annotation with Rich Text*

Description

A replacement for `patchwork::plot_annotation()` that supports `element_chunks()` and `element_md()` for the title, subtitle, and/or caption of a combined patchwork plot.

Each text argument accepts:

- An `element_chunks()` or `element_md()` object — rendered as rich text with auto-wrapping; all formatting options (hjust, lineheight, margin, etc.) are set inside the element call.
- A bare `flextable::as_paragraph()` object — wrapped in `element_chunks()` with default settings.
- A plain character string — passed through unchanged.
- `NULL` — element omitted.

Patchwork only renders an annotation element when a non-NULL string is provided to `patchwork::plot_annotation()`. For element/chunks-based inputs, `munch_annotation` supplies an invisible sentinel string so patchwork creates the layout row, while `element_chunks()` or `element_md()` handles the actual rendering using the stored content (the label is ignored).

Usage

```
munch_annotation(
  caption = NULL,
  title = NULL,
  subtitle = NULL,
  tag_levels = NULL,
  tag_prefix = NULL,
  tag_suffix = NULL,
  tag_sep = NULL,
  theme = NULL
)
```

Arguments

caption	An element_chunks(), element_md(), flextable::as_paragraph() object, a plain character string, or NULL.
title	Same as caption.
subtitle	Same as caption.
tag_levels, tag_prefix, tag_suffix, tag_sep	Passed through to patchwork::plot_annotation().
theme	Additional ggplot2::theme() elements merged on top of any element themes derived from title, subtitle, or caption.

Value

An object of class c("munch_annotation", "gg") that can be added to a patchwork object with +.

Examples

```
library(ggplot2)
library(patchwork)
library(flextable)

library(gdtools)
font_set_liberation()
set_flextable_defaults(
  font.size = 12,
  font.family = "Liberation Sans"
)
p1 <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
p2 <- ggplot(mtcars, aes(hp, wt)) + geom_point()

caption_chunks <- as_paragraph(
  as_chunk("R"), as_sup("2"), as_chunk(" = 0.75")
)

plot_obj <- wrap_plots(p1, p2) +
  munch_annotation(
    title   = "A plain string title",
    caption = element_chunks(caption_chunks, lineheight = 1.5, hjust = 0)
  )
x <- girafe(
  plot_obj,
  options = list(opts_sizing(rescale = FALSE)),
  dependencies = list(
    liberationsansHtmlDependency()
  )
)
x
```

Index