Package 'gridmicrotex'

August 24, 2026

Title Native 'LaTeX' Math Rendering for Grid Graphics

Version 0.1.1

Description Renders 'LaTeX' math equations as native R grid graphics objects (grobs) using the 'MicroTeX' 'C++' library as the layout engine. Produces resolution-independent vector output that works on any R graphics device, with no external 'LaTeX' installation required. Markdown labels and block documents that mix prose formatting with math are also rendered, for use with both 'grid' and 'ggplot2'.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements C++17, FreeType (>= 2.9), pkg-config, FriBidi (optional)

Depends R (>= 4.2.0)

LinkingTo Rcpp, systemfonts

Imports commonmark, grDevices, grid, Rcpp, systemfonts, tools, utils, xml2

Suggests ggplot2 (>= 4.0.0), grImport2, jpeg, knitr, png, ragg, rmarkdown, rsvg, S7, svglite, testthat (>= 3.0.0), textshaping, vdiffr

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/adayim/gridmicrotex, https://adayim.github.io/gridmicrotex/

BugReports https://github.com/adayim/gridmicrotex/issues

RoxygenNote 7.3.3

NeedsCompilation yes

Author Alim Dayim [aut, cre] (ORCID: <https://orcid.org/0000-0001-9998-7463>), Nano Michael [cph] (Author of included 'MicroTeX' library), Bundled math font authors [cph] (See inst/COPYRIGHTS for the full list of authors of the bundled math fonts.)

Maintainer Alim Dayim <ad938@cam.ac.uk>

Repository CRAN

Date/Publication 2026-08-24 07:30:20 UTC

Contents

`available_highlighters`

Syntax highlighting languages available

Description

Names accepted after an opening code fence in `markdown_box_grob`. Includes both the built-in grammars and any added with `register_highlighter`.

Usage

```
available_highlighters()
```

Details

Several common aliases also work and are not listed here, among them `py`, `sh`, `c++`, `yml`, `jl` and `tex`. A fence naming anything else renders as plain monospace text, without a warning.

Value

A character vector of language names, sorted.

See Also

register_highlighter

Examples

```
available_highlighters()
```

`available_math_fonts` *List available math fonts*

Description

Returns the names of all math fonts currently loaded by MicroTeX. These names can be passed to the `math_font` parameter of `latex_grob` and `grid.latex`.

Usage

```
available_math_fonts()
```

Value

A character vector of math font names.

Font pairing

The bundled math fonts have different styles. For a consistent look, pair them with a matching `fontfamily` in gp:

Math font	Style	Suggested text font
Lete Sans Math (`"lete"`, default)	Sans-serif	`"sans"`
STIX Two Math (`"stix"`)	Serif	`"serif"`

Additional math fonts can be loaded with `load_math_font`.

Examples

```
available_math_fonts()
```

check_math_fonts *Check math font status*

Description

Reports which **math** fonts are loaded and available for rendering: the MicroTeX version, the loaded math fonts, and whether the bundled font files are present.

Usage

```
check_math_fonts()
```

Details

Text fonts are not covered, because they are not registered here — they are resolved on demand by **systemfonts** from `gp$fontfamily`. Use `systemfonts::match_fonts()` to see what a text family resolves to.

Value

Invisibly returns the character vector of available math font names.

See Also

`available_math_fonts`, `load_math_font`

Examples

```
check_math_fonts()
```

define_macro *Define a user-level LaTeX macro*

Description

Registers a zero-argument shorthand that is expanded by text substitution before the expression reaches the MicroTeX parser. Useful for domain-specific notation (e.g. `\RR` for `\mathbb{R}`) you reuse across many plots.

Usage

```
define_macro(name, definition)

clear_macros(name = NULL)

list_macros()
```

Arguments

name	Macro name **without** the leading backslash. For clear_macros, the macro name to drop, or NULL (default) to clear all.
definition	LaTeX source the macro expands to.

Value

- define_macro: Invisibly returns NULL.
- clear_macros: Invisibly returns NULL.
- list_macros: A named character vector mapping macro names to their expansions. Empty if no macros are defined.

Choosing between this and \newcommand

MicroTeX also accepts \newcommand and plain-TeX \def written inside the expression itself, and those are the more capable form: they take up to nine arguments, which define_macro() does not — it substitutes text and nothing else.

```
# parameterised, but local to this one expression
grid.latex(r"(\def\norm#1{\left\lVert #1 \right\rVert}
              \norm{\vec{v}})")
```

What they cannot do is persist: the user-macro table is cleared at the start of every parse, so a \newcommand written in one call is gone by the next. That is the one thing define_macro() is for. Use \newcommand / \def for an abbreviation local to a single label, and define_macro() for notation you want available to every label in a script.

See Also

latex_grob, latex_options

Examples

```
define_macro("RR", "\\mathbb{R}")
define_macro("eps", "\\varepsilon")
grid::grid.newpage()
grid.latex("\\forall \\eps > 0, \\eps \\in \\RR")
clear_macros()
```

element_latex *A ggplot2 theme element for LaTeX text*

Description

Use this as a theme element for axis titles, axis labels, plot titles, or any other text element in a ggplot2 theme. The text string is parsed as LaTeX math and rendered via MicroTeX.

Usage

```
element_latex(
  math_font = "",
  fontsize = NULL,
  lineheight = 1.2,
  max_width = 0,
  input_mode = c("mixed", "math"),
  render_mode = c("typeface", "path"),
  ...
)
```

Arguments

math_font	Name of the math font to use (e.g., "stix"). Use "" (default) for Lete Sans Math, which pairs with R's default sans-serif text font. See available_math_fonts for loaded fonts.
fontsize	Convenience alias for size; when supplied, it is forwarded to ggplot2::element_text() as the text size in points. If NULL (default), the theme's inherited size is used.
lineheight	Multi-line height multiplier (default 1.2), matching grid::gpar() semantics.
max_width	Numeric maximum width in big points for automatic line wrapping. Use 0 (default) for no wrapping.
input_mode	How tex is interpreted before being parsed. "mixed" (default) wraps the input in \text{...} so the string reads as ordinary text and $...$ (or \(...\)) opens math mode, matching document-level LaTeX semantics. Useful for labels that arrive from external sources mixing prose and math without explicit \text{} markers. "math" is the classic MicroTeX behaviour — the whole string is treated as math, so unwrapped prose renders as spaced math italics. The default can be changed globally via latex_options(input_mode = "math"). See latex_wrap for details on the wrapping process.
render_mode	Character string: "typeface" (default) renders glyphs as native text using the math font, producing selectable/accessible text in PDF and SVG output. Bundled math fonts and any registered via load_math_font are read directly from their OTF files — no system-wide font install is required. Falls back to path mode automatically on devices that lack the R $\geq$ 4.3 glyph engine (e.g., the base pdf() device). For selectable PDF output, prefer cairo_pdf. "path" renders math symbols as filled vector paths (works on all devices but text is not selectable in PDF/SVG).

`...` Additional arguments passed to `ggplot2::element_text()` (e.g., `size`, `colour`, `hjust`).

Details

Dollar signs (`$...$`) in the label text are stripped automatically so that both `"\frac{a}{b}"` and `"$\frac{a}{b}$"` work.

This element is an S7 subclass of `ggplot2::element_text`, so it inherits all standard text properties (size, colour, hjust, etc.) from the theme and supports `merge_element()` correctly.

Value

An S7 object of class `element_latex`, inheriting from `ggplot2::element_text`.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(mtcars, aes(wt, mpg)) + geom_point() +
    labs(x = "$\\beta_1 \\cdot x + \\beta_0$") +
    theme(axis.title.x = element_latex())
}
```

`element_markdown` *A ggplot2 theme element for markdown text*

Description

Use as a theme element for axis titles, axis labels, plot titles or any other text element. The label is parsed as markdown — including `$...$` math — and rendered via MicroTeX.

Usage

```
element_markdown(
  math_font = "",
  fontsize = NULL,
  lineheight = 1.2,
  max_width = 0,
  render_mode = c("typeface", "path"),
  justify = FALSE,
  style = NA,
  width = NA,
  ...
)
```

Arguments

math_font	Name of the math font to use (e.g. "stix").
fontsize	Convenience alias for size; forwarded to ggplot2::element_text() as the text size in points. NULL (default) uses the theme's inherited size.
lineheight	Multi-line height multiplier (default 1.2).
max_width	Maximum width in big points for automatic line wrapping (default: 0, no wrapping).
render_mode	"typeface" (default) or "path".
justify	Logical; justify wrapped lines. Requires max_width.
style	A markdown_style object, CSS text, or the path to a .css file, applied to labels drawn through this theme element. NA, the default, means unset — the global latex_options(markdown_style =) applies instead. Only the properties that compile to LaTeX have an effect, unless the label is laid out as blocks (see *Block labels*). See md_style.
width	Wrapping measure for the label, as a unit. NA, the default, means unset: the label is sized to its content and does not wrap. unit(1, "npc") is the useful value for a plot title, whose cell really is the full plot width. See *Block labels*.
...	Additional arguments passed to ggplot2::element_text() (e.g. colour, hjust).

Details

This is an S7 subclass of ggplot2::element_text, so it inherits the standard text properties (size, colour, hjust, ...) from the theme and merges correctly with inherited theme entries.

Value

An S7 object of class element_markdown, inheriting from ggplot2::element_text.

Block labels

A label with real block structure — a heading, a list, a table, a rule, or more than one paragraph — is laid out by markdown_box_grob rather than flattened into one run, so list markers, indents and block spacing survive. That makes a title like this work:

```
labs(title = "## Findings\n\n- slope $\\beta_1$\n- *p* < 0.001")
```

The box's own background, border, padding and corner radius come from the stylesheet's body rule — style = "body \{ background: grey95; padding: 8px \}" — not from arguments here. See markdown_style.

Three details follow from how ggplot2 measures theme elements:

- **Wrapping is opt-in.** Without width the label is sized to its content, because ggplot2 asks an element for its height before placing it, when a relative width would resolve against the whole device rather than the element's cell.
- **Axis tick labels are never laid out as blocks**, whatever they contain, for the same reason.

- **A rotated label is never laid out as blocks.** The box cannot rotate, so a label with both blocks and a non-zero `angle` keeps the angle, is rendered as a single run, and warns. A `width` given with an angle becomes the run's wrapping measure instead.

`math_font`, `render_mode` and `justify` are not forwarded to the box; a block label takes those from `latex_options`.

Note that **ggtext** also exports a function called `element_markdown()`. If both packages are attached, the one loaded later wins; call `gridmicrotex::element_markdown()` explicitly to be unambiguous. The two are not interchangeable — ggtext renders HTML/CSS and images, this renders LaTeX math.

See Also

`markdown_grob`, `element_latex`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(mtcars, aes(wt, mpg)) + geom_point() +
    labs(x = "**weight** in $10^3$ lbs") +
    theme(axis.title.x = gridmicrotex::element_markdown())
}
```

`geom_latex` *A ggplot2 geom for LaTeX math labels*

Description

Renders LaTeX math expressions as native grid grobs within a ggplot2 plot. Each label is parsed and laid out by MicroTeX, producing resolution-independent vector output.

Usage

```
geom_latex(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fontsize = 11,
  math_font = "",
  lineheight = 1.2,
  max_width = 0,
  input_mode = c("mixed", "math"),
  render_mode = c("typeface", "path"),
  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 to layer.
fontsize	Default font size in points. Overridden by the size aesthetic if mapped.
math_font	Name of the math font to use (e.g., "stix").
lineheight	Multi-line height multiplier (default 1.2), matching grid::gpar() semantics.
max_width	Maximum width in big points for automatic line wrapping (default: 0, no wrapping).

input_mode — How tex is interpreted before being parsed. "mixed" (default) wraps the input in \text{...} so the string reads as ordinary text and $...$ (or \(...\)) opens math mode, matching document-level LaTeX semantics. Useful for labels that arrive from external sources mixing prose and math without explicit \text{} markers. "math" is the classic MicroTeX behaviour — the whole string is treated as math, so unwrapped prose renders as spaced math italics. The default can be changed globally via latex_options(input_mode = "math"). See latex_wrap for details on the wrapping process.

render_mode — Character string: "typeface" (default) renders glyphs as native text using the math font, producing selectable/accessible text in PDF and SVG output. Bundled math fonts and any registered via load_math_font are read directly from their OTF files — no system-wide font install is required. Falls back to path mode automatically on devices that lack the R $\geq$ 4.3 glyph engine (e.g., the base pdf() device). For selectable PDF output, prefer cairo_pdf. "path" renders math symbols as filled vector paths (works on all devices but text is not selectable in PDF/SVG).

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().

Value

A ggplot2 layer.

Aesthetics

geom_latex() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- label — LaTeX math string
- size — font size in points (default: 11)
- colour — text colour (default: "black")
- angle — rotation angle in degrees (default: 0)
- hjust — horizontal justification, 0–1 (default: 0.5)
- vjust — vertical justification, 0–1 (default: 0.5)
- alpha — transparency (default: 1)

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  df <- data.frame(
    x = 1:3, y = 1:3,
    eq = c("x^2", "\\frac{a}{b}", "\\sum_{i=1}^n x_i")
  )
  ggplot(df, aes(x, y, label = eq)) + geom_latex()

  # Use annotate() for single annotations (no legend, no data frame needed)
  ggplot(mtcars, aes(wt, mpg)) + geom_point() +
    annotate("latex", x = 4, y = 30,
             label = r"($\hat{y} = \beta_0 + \beta_1 x$)")
}
```

---

geom_markdown *A ggplot2 geom for markdown labels*

Description

Renders markdown labels – `**bold**`, `*italic*`, `` `code` ``, `~~strike~~` and `$math$` – as native grid grobs inside a plot. The markdown is converted to LaTeX and laid out by MicroTeX, so the output is resolution-independent vector graphics.

Usage

```
geom_markdown(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  fontsize = 11,
  math_font = "",
  lineheight = 1.2,
  max_width = 0,
  render_mode = c("typeface", "path"),
  justify = FALSE,
  style = NULL,
  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 to layer.
fontsize	Default font size in points. Overridden by the size aesthetic if mapped.
math_font	Name of the math font to use (e.g. "stix").
lineheight	Multi-line height multiplier (default 1.2), matching grid::gpar() semantics.
max_width	Maximum width in big points for automatic line wrapping (default: 0, no wrapping).
render_mode	"typeface" (default) or "path"; see latex_grob.
justify	Logical; justify wrapped lines. Requires max_width. See latex_grob.

style	A markdown_style object, CSS text, or the path to a .css file, applied to every label this layer draws. NULL falls back to latex_options(markdown_style =). Only the properties that compile to LaTeX apply here — a label has no block layout, so margins and padding are ignored. See md_style.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, they are removed silently.
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().

Value

A ggplot2 layer.

Aesthetics

geom_markdown() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- label — markdown string
- size — font size in points (default: 11)
- colour — text colour (default: "black")
- angle — rotation angle in degrees (default: 0)
- hjust — horizontal justification, 0-1 (default: 0.5)
- vjust — vertical justification, 0-1 (default: 0.5)
- alpha — transparency (default: 1)

See Also

markdown_grob, geom_latex

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  df <- data.frame(
    x = 1:3, y = 1:3,
    lab = c("**bold**", "*slope* $\\beta_1$", "`code` and $x^2$")
  )
  ggplot(df, aes(x, y, label = lab)) + geom_markdown()
}
```

grobMark *Look up a named anchor inside a LaTeX grob*

Description

Resolves a `\mark{name}` that was placed inside the LaTeX source to a pair of **grid** units in the grob's parent viewport. The returned units already account for the grob's viewport position and `hjust`/`vjust`, so you can pass them directly to grid drawing functions to anchor other graphics on parts of the formula.

Usage

```
grobMark(grob, name)
```

Arguments

`grob`	A `latexgrob` returned by `latex_grob`.
`name`	The mark name (the argument to `\mark{...}`).

Value

A list with elements `x` and `y`, each a `unit`. Mark coordinates are evaluated in the grob's parent viewport. They assume an unrotated grob: with `rot != 0` the returned position does not account for the rotation.

See Also

`latex_grob`

Examples

```
g <- latex_grob(r"($a\mark{eq}^2 = b + c^2$)",
                x = grid::unit(0.5, "npc"),
                y = grid::unit(0.5, "npc"))
grid::grid.newpage(); grid::grid.draw(g)
mk <- grobMark(g, "eq")
grid::grid.points(mk$x, mk$y, pch = 19,
                  gp = grid::gpar(col = "red"))
```

`latex_cache_limit` *Set the maximum number of entries kept in the LaTeX layout cache*

Description

The cache stores parsed layout information for recently rendered LaTeX expressions, keyed by the expression and relevant rendering parameters (font, size, macros, etc.). This speeds up repeated rendering of the same expressions, especially in loops or interactive sessions. The default limit is 512 entries, which should be sufficient for most use cases. When the limit is exceeded, the least recently used entries are automatically evicted.

Usage

```
latex_cache_limit(n = 512L)

latex_cache_clear()

latex_cache_info()
```

Arguments

`n`	Non-negative integer cache capacity. Default is 512. Set to `0` to disable caching.

Value

- `latex_cache_limit`: Invisibly returns the previous limit.
- `latex_cache_clear`: Invisibly returns `NULL`.
- `latex_cache_info`: A list with elements `size` (entries currently stored), `max_size`, `hits`, and `misses`.

See Also

`latex_grob`, `latex_options`

Examples

```
latex_cache_limit(256)
grid.latex("e^{i\\pi} + 1 = 0")
latex_cache_info()
latex_cache_clear()
```

latex_dims *Get dimensions of a LaTeX expression*

Description

Get dimensions of a LaTeX expression

Usage

```
latex_dims(
  tex,
  math_font = "",
  max_width = 0,
  tex_style = "",
  input_mode = c("mixed", "math"),
  render_mode = c("typeface", "path"),
  justify = FALSE,
  line_break = c("greedy", "optimal"),
  gp = grid::gpar()
)
```

Arguments

tex	Character string of LaTeX math code.
math_font	Name of the math font to use (e.g., "stix"). Use "" (default) for Lete Sans Math, which pairs with R's default sans-serif text font. See available_math_fonts for loaded fonts.
max_width	Numeric maximum width in big points for automatic line wrapping. Use 0 (default) for no wrapping.
tex_style	Character: TeX style override. One of "" (default; let the parser decide), "display", "text", "script", or "scriptscript". See latex_grob for the semantics of each value.
input_mode	How tex is interpreted before being parsed. "mixed" (default) wraps the input in \text{...} so the string reads as ordinary text and $...$ (or \(...\)) opens math mode, matching document-level LaTeX semantics. Useful for labels that arrive from external sources mixing prose and math without explicit \text{} markers. "math" is the classic MicroTeX behaviour — the whole string is treated as math, so unwrapped prose renders as spaced math italics. The default can be changed globally via latex_options(input_mode = "math"). See latex_wrap for details on the wrapping process.
render_mode	Character string: "typeface" (default) renders glyphs as native text using the math font, producing selectable/accessible text in PDF and SVG output. Bundled math fonts and any registered via load_math_font are read directly from their OTF files — no system-wide font install is required. Falls back to path mode automatically on devices that lack the R $\geq$ 4.3 glyph engine (e.g., the

base pdf() device). For selectable PDF output, prefer cairo_pdf. "path" renders math symbols as filled vector paths (works on all devices but text is not selectable in PDF/SVG).

justify — Logical. When TRUE, wrapped text is stretched at its interword spaces so every line but the last fills max_width exactly. Requires max_width: it acts on the lines the wrapper produces, so it does nothing on its own. FALSE (default) leaves the right edge ragged, matching R's own text drawing. In a narrow column, expect wide word spaces; mark the words that may break with \- to tighten them.

line_break — How lines are chosen when wrapping. "greedy" (default) fills each line as far as it will go and never reconsiders. "optimal" chooses the breaks together so the paragraph as a whole reads best, in the spirit of Knuth-Plass: pulling one word down early can improve every later line, which a greedy pass cannot see. Requires max_width, and costs a little more layout time.

gp — Graphical parameters (see gpar). Common entries: col (formula foreground), fontfamily (text font), fontsize / cex (formula size), and lineheight (multiline spacing). See latex_grob for how each of these flows through MicroTeX.

Value

A list with the following elements:

- width, height, depth: grid unit objects in big points. height is total height (ascent + descent).
- baseline: grid unit object giving the baseline position measured in big points from the *bottom* of the bounding box. Equivalent to height - depth for single-line formulas. Useful for aligning a formula's baseline with surrounding text.
- is_split: logical; TRUE if the formula was wrapped across multiple lines (only possible when max_width > 0).

Examples

```
latex_dims("\\frac{a}{b}")
```

---

latex_grob *Create a grid grob from a LaTeX expression*

Description

Parses a LaTeX math expression and returns a grid grob object that renders the formula using native grid graphics primitives. The grob supports standard grid queries such as grobWidth(), grobHeight(), grobX(), and grobY().

A convenience wrapper that creates a latex_grob and immediately draws it on the current device via grid.draw.

Usage

```
latex_grob(
  tex,
  x = grid::unit(0.5, "npc"),
  y = grid::unit(0.5, "npc"),
  default.units = "npc",
  hjust = 0.5,
  vjust = 0.5,
  rot = 0,
  math_font = "",
  max_width = 0,
  tex_style = "",
  input_mode = c("mixed", "math"),
  render_mode = c("typeface", "path"),
  justify = FALSE,
  line_break = c("greedy", "optimal"),
  debug = FALSE,
  name = NULL,
  gp = grid::gpar()
)

grid.latex(tex, ...)
```

Arguments

tex	Character string of LaTeX math code.
x, y	Position in grid coordinates.
default.units	Units for x, y if given as numeric.
hjust, vjust	Horizontal/vertical justification. Accepts the usual numeric values in [0, 1]. As a convenience, hjust also accepts the strings "left"/"bbleft", "center"/"centre"/ "middle"/"bbcentre", and "right"/"bbright"; vjust accepts "bottom", "center"/"centre"/"middl "top", and "baseline". "baseline" aligns the formula's math baseline with the anchor point — handy for placing a formula in flowing text.
rot	Rotation angle in degrees, counter-clockwise (default: 0). Matches the rot parameter of textGrob.
math_font	Name of the math font to use (e.g., "stix"). Use "" (default) for Lete Sans Math, which pairs with R's default sans-serif text font. See available_math_fonts for loaded fonts.
max_width	Numeric maximum width in big points for automatic line wrapping. Use 0 (default) for no wrapping.
tex_style	Character: TeX style override. One of "" (default; let the parser decide), "display", "text", "script", or "scriptscript". See latex_grob for the semantics of each value.
input_mode	How tex is interpreted before being parsed. "mixed" (default) wraps the input in \text{...} so the string reads as ordinary text and $...$ (or \(...\))

opens math mode, matching document-level LaTeX semantics. Useful for labels that arrive from external sources mixing prose and math without explicit `\text{}` markers. `"math"` is the classic MicroTeX behaviour — the whole string is treated as math, so unwrapped prose renders as spaced math italics. The default can be changed globally via `latex_options(input_mode = "math")`. See `latex_wrap` for details on the wrapping process.

`render_mode` Character string: `"typeface"` (default) renders glyphs as native text using the math font, producing selectable/accessible text in PDF and SVG output. Bundled math fonts and any registered via `load_math_font` are read directly from their OTF files — no system-wide font install is required. Falls back to path mode automatically on devices that lack the R $\geq$ 4.3 glyph engine (e.g., the base `pdf()` device). For selectable PDF output, prefer `cairo_pdf`. `"path"` renders math symbols as filled vector paths (works on all devices but text is not selectable in PDF/SVG).

`justify` Logical. When `TRUE`, wrapped text is stretched at its interword spaces so every line but the last fills `max_width` exactly. Requires `max_width`: it acts on the lines the wrapper produces, so it does nothing on its own. `FALSE` (default) leaves the right edge ragged, matching R's own text drawing. In a narrow column, expect wide word spaces; mark the words that may break with `\-` to tighten them.

`line_break` How lines are chosen when wrapping. `"greedy"` (default) fills each line as far as it will go and never reconsiders. `"optimal"` chooses the breaks together so the paragraph as a whole reads best, in the spirit of Knuth-Plass: pulling one word down early can improve every later line, which a greedy pass cannot see. Requires `max_width`, and costs a little more layout time.

`debug` Logical; if `TRUE`, draws diagnostic overlays on the grob — the full bounding box (dashed gray), the baseline (solid red), the depth line (dashed gray), and a small dot at each MicroTeX draw record's origin. Useful for checking positioning and diagnosing vertical alignment.

`name` Optional grob name.

`gp` Graphical parameters (see `gpar`). Common entries: `col` (formula foreground), `fontfamily` (text font), `fontsize` / `cex` (formula size), and `lineheight` (multi-line spacing). See `latex_grob` for how each of these flows through MicroTeX.

`...` Additional arguments passed to `latex_grob`.

Details

Controlling TeX style with `tex_style`:

`tex_style` selects the size-and-spacing regime MicroTeX applies to the whole expression. It changes the *style* (display vs. text), not the font size — size is always set via `gp$fontsize` / `gp$cex`; style-dependent shrinking (for `"script"` and `"scriptscript"`) is applied on top of that size.

- `""` (default): let the parser choose based on the delimiters in `tex`. Inline delimiters (single `$`, or `\(...\)`) produce `"text"` style; display delimiters (double `$$`, or `\[...\]`) produce `"display"` style. If the string has no delimiters, MicroTeX defaults to `"text"` style.

- `"display"`: force display style. Large operators (`\sum`, `\int`, `\prod`) render at their full size, limits are placed above/below rather than as subscripts/superscripts, and fractions use full-size numerators and denominators. Useful when you want a display-style equation inline in a label, legend, or `element_latex()` title.
- `"text"`: force text (inline) style. Big operators shrink to their inline size and limits attach as scripts. The right choice for formulas embedded in a line of prose.
- `"script"`: force script style — the size normally used for first-level subscripts and superscripts. Produces a smaller, tighter layout; mainly useful for callouts or sub-labels where a compact equation is wanted.
- `"scriptscript"`: force scriptscript style — the smallest style, used by TeX for doubly-nested scripts. Rarely needed on its own; primarily for very dense annotations.

`tex_style` applies to the entire expression. To override the style of a sub-expression from within `tex`, use the inline TeX commands `\displaystyle`, `\textstyle`, `\scriptstyle`, or `\scriptscriptstyle`.

Graphical parameters (`gp`):

- `col`: default foreground color for the formula. Individual elements can still be overridden with an inline `\textcolor` command in the LaTeX string.
- `fontfamily`: controls the font of text inside `\text` and `\mbox` blocks. For example, `gpar(fontfamily = "serif")` renders `\text` content in R's serif family. Any font available to R's graphics system works — base families (`"sans"`, `"serif"`, `"mono"`) as well as fonts registered via **showtext** or **systemfonts**. Math symbols always use the selected math font (see `math_font`). Bold/italic text is controlled from within the LaTeX source (`\textbf{}`, `\textit{}`, `\bf`, ...), not via `gp$fontface` — MicroTeX needs the style at layout time to size each run correctly, so a `gpar()`-level face is not consulted.
  `fontfamily` *also* drives MicroTeX's layout metrics for non-math text: the matching system font is resolved via **systemfonts**, a minimal metrics file is generated on first use and cached under `tools::R_user_dir("gridmicrotex", "cache")`, so MicroTeX's spacing of `\text` blocks stays in sync with what **grid** actually draws. When `fontfamily` is unset, the R default (`"sans"`) is used. No manual font loading is required for text fonts; `load_math_font()` remains only for adding custom **math** fonts.
- `fontsize` / `cex`: formula size is `fontsize * cex` big points (default 20 * 1). Both math and text scale together. The effective size is baked into the parsed layout, so downstream viewports that inherit `cex` will not re-scale the grob (matching `textGrob` semantics when `gp` is set explicitly).
- `lineheight`: controls multi-line spacing (default 1.2). The inter-line gap is `(lineheight - 1) * fontsize` big points.

LaTeX document-level wrappers:

The parser accepts raw output from `print.xtable()`, `knitr::kable()`, and similar functions that emit complete `tabular` LaTeX. The following document-level constructs are recognized and rewritten silently before the input reaches MicroTeX:

Removed (no visual effect):

- `%`-to-end-of-line comments (escaped `\%` is preserved)
- preamble: `\documentclass[...]{...}`, `\usepackage[...]{...}`, `\begin{document}` / `\end{document}`
- title metadata: `\maketitle`, `\title{...}`, `\author{...}`

- cross-reference labels: `\label{...}`
- float wrappers: `\begin{table}` / `\end{table}`, `\begin{figure}` / `\end{figure}` (and starred variants)
- layout scopes: `\centering`, `\raggedright`, `\raggedleft`, `\flushleft`, `\flushright`

Rewritten:

- booktabs rules: `\toprule`, `\midrule`, `\bottomrule`, `\cmidrule` are mapped to `\hline`. The optional column-range and trim arguments of `\cmidrule` are discarded (MicroTeX has no concept of partial-column rules).
- `\caption[short]{X}` is extracted as `\text{X}\\` at its source position, so a caption written after `\includegraphics` renders below the figure and one written before a `tabular` renders above the table. Full LaTeX instead positions the caption by float type regardless of source order, and numbers it from a counter; there is no counter here. Wrap the figure and its caption in `\begin{array}{c}...\end{array}` to centre them on each other (`\centering` is dropped — a grob has no page to centre against).
- `\graphicspath{{dir/}}` and `\DeclareGraphicsExtensions{...}` are consumed rather than typeset; the former's directories are searched.

Images:

- `\includegraphics[opts]{file}` draws a PNG, JPEG or SVG inline. The starred form is accepted and behaves identically. `width`, `height` and `scale` take any LaTeX length (`\textwidth` resolves against `max_width`, and without one falls back to the file's own size with a warning); `scale` multiplies whatever `width`/`height` settled on, and `keepaspectratio` fits inside them instead of stretching to fill. `angle` rotates the figure and grows the surrounding box to the rotated bounds, as `\rotatebox` does. `origin`, `trim`, `clip` and `viewport` are parsed but not applied, and warn once so the difference is not silent; so does a length that cannot be read, or one that sizes the figure to nothing.
- The extension may be omitted, as in LaTeX: `{plots/fig}` finds `plots/fig.svg`, then `.png`, `.jpg`, `.jpeg`.
- An SVG is drawn as real vector and stays sharp at any output resolution; a bitmap does not, and warns when it would be shown below 150 dpi. PDF and EPS are not supported — save the figure as SVG instead. A file that cannot be read — missing, unsupported, or an SVG with no `rsvg` installed — warns and draws its name rather than disappearing.

Anything not in this list is passed to MicroTeX unchanged. An unknown command is not an error: MicroTeX typesets its name in red, which makes unsupported markup easy to spot in the output.

Parallelism:

The MicroTeX engine keeps mutable C++ state for font caching and text measurement. Rendering is safe single-threaded and under separate-process backends such as `future::plan(multisession)`. It is *not* safe under forked backends (`parallel::mclapply()`, `future::plan(multicore)`) on Unix, because forked workers share that state without synchronisation. Use a socket/multisession backend instead.

Value

A `grid` grob of class `"latexgrob"`.

Invisibly returns the grob.

See Also

grid.latex, latex_dims, geom_latex, available_math_fonts, latex_wrap, latex_options

Examples

```
g <- latex_grob(r"($\fcolorbox{red}{yellow}{\frac{a}{b}}$)",
                x = grid::unit(0.3, "npc"),
                y = grid::unit(0.3, "npc"),
                gp = grid::gpar(fontsize = 30))
grid::grid.draw(g)
# Red formula
grid::grid.draw(latex_grob("$x^{2}$",
                           x = grid::unit(0.3, "npc"),
                           y = grid::unit(0.8, "npc"),
                           gp = grid::gpar(col = "red")))

                           # Rotated formula
grid::grid.draw(latex_grob(r"($\colorbox{BurntOrange}{x^{2}} + y^{2}$)",
                           x = grid::unit(0.6, "npc"),
                           y = grid::unit(0.3, "npc"),
                           gp = grid::gpar(fontsize = 24),
                           rot = 45))

grid.latex(r"($\textcolor{red}{x^{2}} + y^{2} = z^{2}$)",
           x = grid::unit(0.6, "npc"),
           y = grid::unit(0.8, "npc"),)
```

`latex_options` *Set or query package-wide LaTeX rendering defaults*

Description

A single entry point for project-wide defaults used by latex_grob, grid.latex, latex_dims, and latex_tree. Options set here are applied only when the corresponding argument is *not* supplied at the call site, so explicit arguments always win.

Usage

```
latex_options(
  math_font = NULL,
  render_mode = NULL,
  tex_style = NULL,
  input_mode = NULL,
  justify = NULL,
  line_break = NULL,
  markdown_style = NULL
)

reset_latex_options()
```

Arguments

math_font	Math font name or alias (see available_math_fonts).
render_mode	Either "typeface" or "path".
tex_style	TeX style override. One of "" (let the parser decide), "display", "text", "script", or "scriptscript". "display" forces large operators with limits placed over/under, useful for inline labels that should still look like display equations.
input_mode	How the input string is interpreted before being handed to MicroTeX. "mixed" (default) wraps the string in \text{...} so it reads as ordinary text, with $...$ (and \(...\)) opening math mode — the document-level LaTeX convention. Useful when consuming labels from other packages that mix prose and math without explicit \text{} markers. "math" treats the whole string as math — the classic MicroTeX behaviour, where letters render as math italics and unwrapped prose looks wrong.
justify	Logical. When TRUE, wrapped text is stretched at its interword spaces so every line but the last fills max_width exactly. Has no effect without max_width, since it acts on the lines the wrapper produces. FALSE (default) leaves the right edge ragged, matching R's own text drawing. In a narrow column, justifying alone opens noticeably wide word spaces; mark the words that may break with \-.
line_break	How lines are chosen when wrapping. "greedy" (default) fills each line as far as it will go and never reconsiders. "optimal" chooses the breaks together so the paragraph as a whole reads best, in the spirit of Knuth-Plass: pulling one word down early can improve every later line, which a greedy pass cannot see. Requires max_width, and costs a little more layout time.
markdown_style	Default style for markdown_grob and markdown_box_grob: a markdown_style object, CSS text, or a path to a .css file.

Details

Calling latex_options() with no arguments returns the current settings (a list whose NULL entries mean "use the built-in default"). Supply one or more named arguments to update them.

Font size and line spacing are controlled via gp parameters (fontsize, cex, lineheight) at the grob level — see latex_grob.

Value

Invisibly returns the previous settings (a list). With no arguments, returns the current settings visibly.

See Also

available_math_fonts, latex_grob

Examples

```
latex_options(math_font = "stix", render_mode = "typeface")
grid.latex("\\sum_{i=1}^{n} i^{2}", gp = grid::gpar(fontsize = 14))
reset_latex_options()
```

`latex_tree` *Inspect the parsed layout of a LaTeX expression*

Description

Returns the raw draw-record table produced by MicroTeX's layout pass together with the bounding-box metadata. Useful for debugging alignment issues, building custom grobs on top of the layout, or counting glyphs/paths/rules in a formula.

Usage

```
latex_tree(
  tex,
  math_font = "",
  max_width = 0,
  tex_style = "",
  input_mode = c("mixed", "math"),
  render_mode = c("typeface", "path"),
  gp = grid::gpar()
)
```

Arguments

`tex`	Character string of LaTeX math code.
`math_font`	Name of the math font to use (e.g., `"stix"`). Use `""` (default) for Lete Sans Math, which pairs with R's default sans-serif text font. See `available_math_fonts` for loaded fonts.
`max_width`	Numeric maximum width in big points for automatic line wrapping. Use `0` (default) for no wrapping.
`tex_style`	Character: TeX style override. One of `""` (default; let the parser decide), `"display"`, `"text"`, `"script"`, or `"scriptscript"`. See `latex_grob` for the semantics of each value.
`input_mode`	How `tex` is interpreted before being parsed. `"mixed"` (default) wraps the input in `\text{...}` so the string reads as ordinary text and `$...$` (or `\(...\)`) opens math mode, matching document-level LaTeX semantics. Useful for labels that arrive from external sources mixing prose and math without explicit `\text{}` markers. `"math"` is the classic MicroTeX behaviour — the whole string is treated as math, so unwrapped prose renders as spaced math italics. The default can be changed globally via `latex_options(input_mode = "math")`. See `latex_wrap` for details on the wrapping process.
`render_mode`	Character string: `"typeface"` (default) renders glyphs as native text using the math font, producing selectable/accessible text in PDF and SVG output. Bundled math fonts and any registered via `load_math_font` are read directly from their OTF files — no system-wide font install is required. Falls back to path mode automatically on devices that lack the R $\geq$ 4.3 glyph engine (e.g., the

base pdf() device). For selectable PDF output, prefer cairo_pdf. "path" renders math symbols as filled vector paths (works on all devices but text is not selectable in PDF/SVG).

gp Graphical parameters (see gpar). Common entries: col (formula foreground), fontfamily (text font), fontsize / cex (formula size), and lineheight (multi-line spacing). See latex_grob for how each of these flows through MicroTeX.

Value

A list with class "latex_tree" containing:

records Data frame of draw records (one row per glyph, path, line, rect, or text block). Columns include type, x, y, glyph, font_size, color, text, codepoint, font_file.

bbox Named numeric vector with width, height, depth, baseline (all in big points).

tex The (macro-expanded) input string.

render_mode Rendering mode used for the layout.

See Also

latex_grob, latex_dims

Examples

```
tree <- latex_tree("\\frac{a}{b}")
print(tree)
head(tree$records)
```

---

latex_wrap *Wrap standard text for math-first LaTeX renderers*

Description

Parses character strings to safely isolate standard natural language from LaTeX math environments. Standard text is wrapped in \text{} blocks, while equations, display math, and specific LaTeX environments are preserved verbatim. This is heavily optimized for passing mixed-content strings (like plot titles or axis labels) to pure-math typesetting engines like MicroTex. The conversion is not perfect, but it should handle most common cases without user intervention.

Usage

```
latex_wrap(tex, input_mode = c("mixed", "math"))
```

Arguments

tex	character. The string or vector of strings to be processed.
input_mode	character. A length-one character vector dictating the parsing strategy. If "mixed" (default), the string is tokenized and text is wrapped. If "math", the parser is bypassed and the string is returned unmodified, assuming the user has provided a pure math equation.

Details

latex_wrap() operates as a state-machine tokenizer to ensure that valid LaTeX math is not corrupted by the text-wrapping process. It features:

- **Delimiter Preservation**: Standard inline ($, \() and block ($$, \[) math delimiters are recognized and preserved.
- **Environment Tracking**: Complex nested environments (e.g., \begin{matrix}) are safely extracted and bypassed.
- **Newline Conversion**: R newline characters (\n) occurring outside of math environments are automatically converted to LaTeX line breaks (\\) inside the \text{} wrapper.
- **Literal Escapes**: Escaped LaTeX literals (e.g., \$, \%, \#) are safely passed into the \text{} block without triggering math modes. The escape character for \$ is automatically resolved for MicroTex compatibility.

Value

A character vector of the same length as tex, formatted for math-mode LaTeX rendering.

Examples

```
# "mixed" mode (default) safely wraps text and preserves inline math
latex_wrap(r"(The equation \(E=mc^2\) is famous)")

# "mixed" mode handles user-escaped characters seamlessly
latex_wrap(r"(Cost: \$100 for $x$ items)")

# "mixed" mode converts R newlines to stacked text blocks
latex_wrap(r"(Line 1\nLine 2)")

# "math" mode returns the string completely unmodified
latex_wrap(r"(\frac{\alpha}{\beta})", input_mode = "math")
```

---

load_math_font *Load a math font from an OTF file*

Description

Loads an OTF/TTF **math** font — one carrying an OpenType MATH table — into MicroTeX's internal font registry. The MATH table is parsed directly in C++ and the required metrics are synthesised on the fly. You can download a free math font such as Latin Modern Math (the LaTeX default) and load it for math rendering.

Usage

```
load_math_font(otf_path)
```

Arguments

otf_path	Path to the OTF/TTF font file.

Details

The font is also registered with the **systemfonts** package so it can be selected for surrounding plot text via `gp = gpar(fontfamily = "...")` without being installed system-wide.

Plain **text** fonts — those used inside `\text{}` blocks — need no loading at all. They are resolved automatically by **systemfonts** from `gp$fontfamily`, or per run with `\gmfontfamily{}{}`.

Value

Invisibly returns `NULL`.

See Also

`available_math_fonts`, `check_math_fonts`, `latex_options`, `latex_grob`

Examples

```
# Load a math font from a local OTF file. Here we point at the
# bundled STIX font so the example is self-contained and loaded.
# You don't need to load the bundled fonts to use them — they're registered
# with systemfonts on first render — but this shows how to load a custom font.
# in practice you would pass the path to any OTF with an OpenType MATH table.
otf <- system.file("fonts", "STIXTwoMath-Regular.otf",
                   package = "gridmicrotex")
load_math_font(otf)
available_math_fonts()
```

---

markdown_box_grob *Render a markdown document as a boxed grid grob*

Description

Lays markdown out as a block document — headings, paragraphs, lists (including GFM task lists), block quotes, code blocks, tables, horizontal rules and images — inside an optional padded, filled and bordered box. Prose wraps to the requested width, and `$...$` math is typeset by MicroTeX as usual. All the inline formatting `markdown_grob` understands, including the inline HTML subset, works inside every block.

Usage

```
markdown_box_grob(
  md,
  x = grid::unit(0.5, "npc"),
  y = grid::unit(0.5, "npc"),
  width = grid::unit(1, "npc"),
  height = NULL,
  hjust = 0.5,
  vjust = 0.5,
  halign = 0,
  valign = 1,
  padding = NULL,
  margin = NULL,
  box_gp = NULL,
  r = NULL,
  style = NULL,
  name = NULL,
  gp = grid::gpar(),
  vp = NULL
)
```

Arguments

`md`	Character string of markdown.
`x`, `y`	Position of the box in the parent viewport.
`width`	Width of the box, including `margin`. `NULL` sizes the box to its content, so nothing wraps — useful where the available width is not known, as in a ggplot2 theme element.
`height`	Fixed height, or `NULL` (default) to take whatever height the content needs.
`hjust`, `vjust`	Justification of the whole box about `x` and `y`.
`halign`	Horizontal alignment of blocks within the box: `0` left (default), `0.5` centred, `1` right.
`valign`	Vertical alignment of the content when `height` leaves room to spare: `1` top (default), `0` bottom.
`padding`, `margin`	A `unit` of length 1 or 4 giving top, right, bottom and left. Padding is inside the box, margin outside it. `NULL` (default) takes them from the stylesheet's `body` rule, and is zero if that says nothing.
`box_gp`	Graphical parameters for the box itself, e.g. `gpar(fill = "grey95", col = "black")`. `NULL` (default) takes the fill from `body { background }` and the border from `body { border }`, and draws no box if neither is set.
`r`	Corner radius; a non-zero value draws a rounded box. `NULL` (default) takes it from `body { border-radius }`.
`style`	Appearance of the blocks: a `markdown_style` object, CSS text, or a path to a `.css` file. `NULL` (default) uses `latex_options("markdown_style")` if set, and the built-in defaults otherwise.

name	Optional grob name.
gp	Graphical parameters for the text. fontsize also sets the scale for block spacing and list indentation, and cex multiplies it as elsewhere in **grid**.
vp	Optional viewport. Supplying one replaces the viewport built from x, y, width, height, hjust and vjust, so those are then ignored.

Details

Where markdown_grob flattens everything into a single run, this stacks one grob per block. That is what makes headings, list indentation, block-quote rules and background fills possible: MicroTeX has no concept of any of them, and its line breaking does not reach inside the cells it uses for list and table layout.

Two consequences worth knowing. Table cells and code lines are not wrapped, so a wide table overflows rather than reflowing. And list items are stacked here rather than handed to MicroTeX's itemize, which is what gives them a proper hanging indent.

An image on a line of its own is drawn as a raster, scaled to fit the column but never enlarged past its natural size (pixels are read at 96 dpi). PNG needs the **png** package and JPEG needs **jpeg**, both *Suggests*: when the reader is not installed, the file is missing, or the format is anything else, the image degrades to its alt text. An image *within* a sentence stays inline, where only its alt text survives.

The layout is computed at draw time, so an open device is required — which is what lets a relative width and the measured height of the text resolve against the viewport the box is actually drawn in.

Value

A markdownbox gTree.

Styling

Appearance comes from a small CSS cascade. style sets the house style for the whole document, by tag:

```
markdown_box_grob(md, style = markdown_style(
  h1         = md_style(color = "steelblue", font_size = 2),
  blockquote = md_style(border_left = "3px solid grey60")
))
```

The same thing written as CSS, which style also takes directly, as text or as the path to a .css file:

```
markdown_box_grob(md, style = "
  h1 { color: steelblue; font-size: 2rem }
  blockquote { border-left: 3px solid grey60 }
")
```

To style one chunk rather than every block of a kind, wrap it in a carrying a class or a style. **Leave blank lines around the tags** — that is what makes CommonMark parse the markdown between them instead of treating the whole thing as raw HTML:

```
<div class="note">

## This heading only

</div>
```

Inline runs take `class` as well as `style` on a `<span>`. See `markdown_style` for the tag names, the supported properties and how the cascade resolves.

See Also

`markdown_grob`, `latex_grob`

Examples

```
md <- paste(
  "# Results", "",
  "The slope is $\\beta_1$ with *p* < 0.001.", "",
  "- first point", "- second point",
  sep = "\n"
)
grid::grid.newpage()
grid::grid.draw(markdown_box_grob(
  md,
  width = grid::unit(4, "in"),
  padding = grid::unit(8, "pt"),
  box_gp = grid::gpar(fill = "grey95", col = "grey40")
))
```

`markdown_grob` *Render markdown as a grid grob*

Description

Parses a markdown string and returns a grid grob, so plot labels can mix ordinary prose formatting with real LaTeX math. Inline markdown (`**bold**`, `*italic*`, `` `code` ``, `~~strike~~`) is translated to the equivalent LaTeX commands, while `$...$` (and `\(...\)`, `$$...$$`, `\[...\]`) math spans are passed through to MicroTeX byte-for-byte.

Usage

```
markdown_grob(md, style = NULL, ...)

grid.markdown(md, ...)
```

Arguments

md	Character string of markdown.
style	Appearance of the text: a markdown_style object, CSS text, or a path to a .css file. NULL (default) uses latex_options("markdown_style") if set, and the built-in defaults otherwise. Only the properties md_style marks as *inline* apply here — there is no block layout in a single run for a margin, an indent or an alignment to act on, so those are ignored. markdown_box_grob honours them all.
...	Passed to latex_grob — e.g. x, y, hjust, vjust, rot, max_width, gp.

Details

Markdown and LaTeX disagree about several characters — most importantly \, which CommonMark treats as an escape. Math spans are therefore hidden from the markdown parser before it runs and restored afterwards, so constructs like $\begin{matrix}a\\b\end{matrix}$ survive intact.

That hiding uses three private-use codepoints (U+E000, U+E001, U+E002) as markers, so those three characters are *removed* from the input. They are unassigned in Unicode, but icon fonts such as Nerd Fonts do put real glyphs there: if your text contains one it will be dropped rather than drawn. The alternative is worse — a pasted marker would be spliced together with a math span on the way back out and silently duplicate a formula.

GFM has no markdown syntax for colour, underline, super/subscript, highlight or size, so — as in CommonMark, and as **ggtext** does — these come from inline HTML. Each tag renders as HTML's own default rendering prescribes:

tag	effect
<b>,	bold
<i>, , <cite>, <dfn>, <var>, <address>	italic
<code>, <kbd>, <samp>, <tt>	monospace
<u>, <ins>	underline
<s>, <del>, <strike>	strikethrough
<sub>, <sup>	sub / superscript
<mark>	yellow highlight
<small>, <big>	smaller / larger
<q>	wrapped in quotation marks
 	line break
	see below

A style attribute is read for color (any R colour name, the nine CSS names R lacks — crimson, teal, rebeccapurple and friends — #rgb, #rrggbb or rgb(); note that green, gray, grey, maroon and purple keep their R values, not their CSS ones), text-decoration (underline, line-through), font-size (pt, px, in, cm, mm, em, rem, %, smaller, larger) and font-family. Any other property is ignored.

font-family takes the CSS generics monospace, sans-serif and serif, or any font name; a fallback list resolves to its first entry. The name is handed to gp$fontfamily, so *the device resolves it*: **ragg** and **svglite** see any installed family plus anything registered with systemfonts::register_font(), cairo devices see installed families, and base pdf() sees only what pdfFonts() declares — a named

family will not resolve there. An unavailable font falls back silently, as it does for `gpar(fontfamily=)`. A font file that is not installed system-wide is used by registering it first:

```
systemfonts::register_font(name = "MyFont", plain = "MyFont.otf")
```

`load_math_font` is *not* the function for this — it registers *math* fonts with MicroTeX, which is a different mechanism.

A span's own family wins over `gp$fontfamily`, but the width of the spaces *between* its words still comes from `gp$fontfamily`; set both to the same family if that shows.

Tags nest and combine freely with markdown. Any other tag — and all block-level HTML — is dropped, keeping the text inside it, which is also what a browser shows for the ones (`<a>`, `<abbr>`, `<span>` without a style) that have no default rendering.

Not every markdown feature has a MicroTeX equivalent. Links keep their text and drop the destination, and images keep their alt text.

Everything is flattened into a single run here, with paragraphs joined by line breaks: there is no block layout, so indentation, list markers and block-quote rules need `markdown_box_grob`. A heading still takes the size and weight its `style` gives it, since those compile to LaTeX commands rather than to layout.

Value

A `latexgrob`, as returned by `latex_grob`.

See Also

`latex_grob`, `latex_wrap`

Examples

```
grid::grid.newpage()
grid.markdown("The **fitted** slope is $\\beta_1$, *p* < 0.001")
```

`markdown_style` *A style for markdown rendering*

Description

Builds the style used by `markdown_box_grob` and `markdown_grob`: a small CSS cascade over the markdown tags. One constructor covers creating a style, starting from a preset, and extending an existing one. For the properties themselves, and which are honoured where, see `md_style`.

Usage

```
markdown_style(base = NULL, css = NULL, ...)
```

Arguments

base	NULL (default) for the built-in defaults, the name of a bundled preset such as "github", or an existing markdown_style to extend.
css	A stylesheet: CSS text, or a path to a .css file. Read as a file when it names one.
...	Named tag overrides, each an md_style. Use a leading dot for a class, e.g. .note = md_style(...).

Details

Tags are named as in HTML, so a stylesheet reads the way a CSS author expects: body (the document root, which every other tag inherits from — and which also styles the box itself, see md_style), p, h1 ... h6, ul, ol, li, blockquote, pre (a code block), code (an inline code span), strong and em (what markdown's ** and * produce), table, tr, td, th, hr, img, a (a link), math (a paragraph that is nothing but $$...$$), footnote, div and span.

The table tags nest as they do in HTML: tr, td and th inherit through table, so table { color: } reaches the cells. background on tr fills the row, on td/th the individual cell.

Four things can style a document, and they resolve in CSS's own order: the built-in defaults, then a type selector (h1), then a class selector (.note), then an inline style attribute. Ties are broken by document order, so a later rule wins. Inheritable properties (color, the font-* family, line-height, text-align) fall through into nested containers, so blockquote { color: grey40 } greys everything quoted; box properties (margins, padding, borders) do not.

The supported CSS is a deliberately small subset: type selectors, class selectors and selector lists (h1, h2). Combinators, pseudo-classes, attribute selectors, #id and at-rules are skipped rather than raised, so an existing stylesheet can be handed over and the parts that apply still take effect.

Value

An object of class gridmicrotex_markdown_style.

See Also

md_style, markdown_box_grob

Examples

```
markdown_style()
markdown_style("github", h1 = md_style(color = "firebrick"))
markdown_style(css = "h1 { color: steelblue } .note { padding-left: 2em }")
```

`md_style` *Declarations for one markdown tag*

Description

A set of CSS declarations, for use as a named argument to `markdown_style`. Argument names are the CSS property names with underscores in place of hyphens, so `font_size` sets `font-size`.

Usage

```
md_style(...)
```

Arguments

`...`	Named declarations.

Details

Lengths accept three forms: a bare number is `rem`, a multiple of the body font size (`font_size = 2.5`); a string is whatever CSS says it is (`"2.5em"`, `"12pt"`, `"150%"`); and a `unit` is absolute.

These are the supported properties, and where each one has an effect. *Inline* means it also works on a `<span>` and in `markdown_grob`, which has no block layout; *block* means it needs `markdown_box_grob`.

property	scope	notes
`color`	inline + block	
`font_size`	inline + block	
`font_family`	inline + block	
`font_weight`	inline + block	prose blocks only, see below
`font_style`	inline + block	prose blocks only, see below
`text_decoration`	inline + block	`underline`, `overline`, `line-through`
`background`	inline + block	a fill behind the text
`border`	inline	a frame; the inset is fixed
`border_style`	inline	only `double`
`border_radius`	inline	rounds the frame
`box_shadow`	inline	
`visibility`	inline	`hidden` keeps the space
`vertical_align`	inline	`super`, `sub`, or a length
`transform`	inline	`rotate()`, `scale()`, `scaleX(-1)`
`line_height`	block	unitless, as `gpar()` wants it
`margin_top`, `margin_bottom`	block	margins do not collapse
`margin_left`, `margin_right`	block	
`padding_left`, `padding_right`	block	
`padding_top`, `padding_bottom`	block	
`margin`, `padding`	block	the CSS shorthand: one to four lengths, in CSS's order
`text_align`	block	`left`, `center`, `right`
`border_left`	block	the `blockquote` bar
`border_top`	block	the `hr` rule

border_bottom	tr	a rule under each table row
border_color	table	colour of the table's rules
table_layout	table	fixed divides the width between the columns so a wide table wraps
height	block	the band an hr sits in
bullet	ul	raw LaTeX for the marker glyph
marker_gap	ul, ol	marker to text

font_size also accepts CSS's keywords — xx-small through xx-large, plus smaller and larger — taken from the \tiny..\Huge ladder MicroTeX implements.

The body rule styles the box itself. On any other tag, background, border, border_radius, padding and margin apply to that block. On body they apply to the whole markdown_box_grob — its fill, its frame, its corner radius, and the space inside and outside it. That is the only way to give a element_markdown title a background, since the theme element takes no box arguments of its own:

```
body { background: grey95; padding: 8px;
       border: 1px solid grey60; border-radius: 4px }
```

An explicit box_gp, padding, margin or r argument to markdown_box_grob() wins over the rule, the way an inline style wins in CSS.

Anything else is an error — unlike a pasted stylesheet, where an unknown property is ignored the way a browser ignores it.

One limitation worth knowing. font_weight and font_style apply to blocks whose content is prose — paragraphs, headings, list items, block quotes, table cells and s. They do *not* apply to pre or an image's alt text, which build their own LaTeX and impose their own font handling. This is a MicroTeX constraint rather than a choice: \text{} resets the font style, so emphasis has to be decided when the content is generated, not wrapped around it afterwards.

What cannot be styled at all. There is no small-caps (\textsc is not a MicroTeX command), no font-variant-numeric, no right-to-left or bidirectional text, and no padding inside an inline border — MicroTeX has no \fboxsep, so that inset is fixed.

Value

An object of class gridmicrotex_md_style.

See Also

markdown_style, markdown_box_grob

Examples

```
md_style(color = "steelblue", font_size = 2.5, margin_top = 1.2)
```

`register_highlighter` *Add a syntax highlighting grammar*

Description

Registers a KDE/Kate syntax definition so that fenced code blocks tagged with `lang` are highlighted by `markdown_box_grob`. Ten languages are built in — see `available_highlighters` — and this is how to add another.

Usage

```
register_highlighter(lang, file)
```

Arguments

`lang`	Language name, as written after the opening fence. Case is ignored. Registering a name that is already built in replaces it.
`file`	Path to a KDE syntax XML grammar.

Details

The grammar is an XML file in the format used by Kate, KDevelop and, through **skylighting**, Pandoc. The simplest way to write one is to copy a built-in grammar and edit it:

```
file.copy(system.file("highlight", "python.xml",
                      package = "gridmicrotex"),
          "mylang.xml")
```

Because the format is KDE's, the several hundred definitions upstream are a useful *reference* when writing your own — one XML file per language, catalogued at `https://kate-editor.org/syntax/`, repository at `https://invent.kde.org/frameworks/syntax-highlighting`.

Many load and work directly: of twenty sampled, thirteen registered, including `python` (249 contexts), `css`, `yaml`, `makefile` and `go`. The seven that did not — `bash`, `ruby`, `perl`, `rust`, `lua`, `javascript` and `markdown` — all use *dynamic* rules, which substitute part of the match into a later pattern and are not implemented.

Cross-language includes (`##Alerts`, `##Doxygen` and friends) are skipped rather than followed. Those only add TODO/FIXME marks *inside* a comment, so the comment is still a comment; the marks are all that is lost.

They also carry their own licences, mostly GPL or LGPL, which is why none is bundled with this MIT-licensed package. Borrowing from one locally is your own decision; redistributing it is subject to its licence.

Colours come from the `defStyleNum` of each `<itemData>`, which maps onto the same CSS class names **knitr** and Pandoc use (`kw` for a keyword, `co` for a comment, `st` for a string, and so on), so a grammar needs no colour information of its own — restyle with `markdown_style`.

At each position the current context's rules are tried in document order and the first to match wins. Contexts, `IncludeRules`, `lookAhead` and `fallthroughContext` all work, and the context stack carries across lines, so a string or block comment may span them.

A rule this engine cannot represent is skipped when it would have stayed in the current context — costing colour on the text it matched and nothing more — but *refuses the grammar* when it would have switched, because a lost context switch leaves the machine in the wrong state and paints the rest of the file wrongly. The refusal names the construct.

Value

Invisibly, `lang`.

See Also

`available_highlighters`, `markdown_box_grob`, `markdown_style`

Examples

```
# Registering a grammar under a name of your own.
f <- system.file("highlight", "python.xml", package = "gridmicrotex")
register_highlighter("mypython", f)
"mypython" %in% available_highlighters()
```

Index