Package 'gridify'

May 8, 2026

Type Package

Title Enrich Figures and Tables with Custom Headers and Footers and More

Version 0.7.7

Description A simple and flexible tool designed to create enriched figures and tables by providing a way to add text
around them through predefined or custom layouts.
Any input which is convertible to 'grob' is supported, like 'ggplot', 'gt' or 'flextable'.
Based on R 'grid' graphics, for more details see Paul Murrell (2018) <doi:10.1201/9780429422768>.

License Apache License 2.0

URL https://pharmaverse.github.io/gridify/

BugReports https://github.com/pharmaverse/gridify/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports grDevices, grid, methods

Suggests flextable (>= 0.8.0), ggplot2, gridGraphics, gt (>= 0.11.0), gtable, knitr, magrittr, rmarkdown, spelling, testthat (>= 3.0.0)

Collate grid_utils.R gridify-classes.R gridify-methods.R ansi_colour.R simple_layout.R complex_layout.R pharma_layout.R get_layouts.R layout_issues.R pagination_utils.R

VignetteBuilder knitr

Config/testthat/edition 3

Language en-GB

NeedsCompilation no

Author Maciej Nasinski [aut, cre],
Alexandra Wall [aut],
Sarah Robson [aut],
Pritish Dash [aut],

Jennifer Winick-Ng [aut],
Lily Nan [ctb],
Alphonse Kwizera [ctb],
Agota Bodoni [ctb],
Eilis Meldrum-Dolan [ctb],
Gary Cao [ctb],
UCB S.A., Belgium [cph, fnd]

Maintainer Maciej Nasinski <Maciej.Nasinski@ucb.com>

Repository CRAN

Date/Publication 2026-02-05 16:30:02 UTC

Contents

`complex_layout` *Complex Layout for a gridify object*

Description

This function creates a complex layout for a gridify object. The layout consists of six rows and three columns for headers, titles, notes and footnotes around the output.

Usage

```
complex_layout(
  margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
  global_gpar = grid::gpar(),
  background = grid::get.gpar()$fill,
  scales = c("fixed", "free")
)
```

Arguments

`margin`	A unit object specifying the margins around the output. Default is 10% of the output area on all sides.
`global_gpar`	A gpar object specifying the global graphical parameters. Must be the result of a call to `grid::gpar()`.
`background`	A string specifying the background fill colour. Default `grid::get.gpar()$fill` for a white background.
`scales`	A string, either `"free"` or `"fixed"`. By default, `"fixed"` ensures that text elements (titles, footers, etc.) retain a static height, preventing text overlap while maintaining a structured layout. However, this may result in different height proportions between the text elements and the output. The `"free"` option makes the row heights proportional, allowing them to scale dynamically based on the overall output size. This ensures that the text elements and the output maintain relative proportions.

Details

The layout consists of six rows for headers, titles, object (figure or table), notes, and footnotes. The object is placed in the fourth row.

- With `"free"` scales, the row heights are 5%, 5%, 5%, 70%, 5%, and 10% of the area, respectively.
- With `"fixed"` scales, the row heights are adjusted by the number of lines for all text elements around the object, with the remaining area occupied by the object. Note that reducing the output space will retain the space for all text elements, making the object appear smaller.

Value

A gridifyLayout object.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
```

```
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

Examples

```
complex_layout()

# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = complex_layout()
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_middle", "Middle Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("subtitle", "Subtitle") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer") %>%
  set_cell("footer_middle", "Middle Footer") %>%
  set_cell("footer_right", "Right Footer")

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
 layout = complex_layout(margin = grid::unit(c(t = 0.2, r = 0.2, b = 0.2, l = 0.2), units = "npc"))
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer")

gridify(
  object = gt::gt(head(mtcars)),
```

```
  layout = complex_layout(
    margin = grid::unit(c(t = 0.2, r = 0.2, b = 0.2, l = 0.2), units = "npc"),
    global_gpar = grid::gpar(col = "blue", fontsize = 18)
  )
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer")
```

`export_to` *Export gridify objects to a file*

Description

The `export_to()` function exports a `gridifyClass` object or a list of such objects to a specified file. Supported formats include PDF, PNG, TIFF and JPEG. For lists, if a single file name with a PDF file extension is provided, the objects are combined into a multi-page PDF; if a character vector with one file per object is provided, each object is written to its corresponding file. It is not possible to create multi-page PNG or JPEG files.

Usage

```
export_to(x, to, device = NULL, ...)

## S4 method for signature 'gridifyClass'
export_to(x, to, device = NULL, ...)

## S4 method for signature 'list'
export_to(x, to, device = NULL, ...)

## S4 method for signature 'ANY'
export_to(x, to, device = NULL, ...)
```

Arguments

`x`	A `gridifyClass` object or a list of `gridifyClass` objects.
`to`	A character string (or vector) specifying the output file name(s). The extension determines the output format.
`device`	a function for graphics device. By default a file name extension is used to choose a graphics device function. Default `NULL`
`...`	Additional arguments passed to the graphics device functions (`pdf()`, `png()`, `tiff()`, `jpeg()` or your custom one). Default width and height for each export type, respectively: • PDF: 11.69 inches x 8.27 inches

- PNG: 600 px x 400 px
- TIFF: 600 px x 400 px
- JPEG: 600 px x 400 px

Details

For PDF export, a new device is opened, the grid is printed using the object's custom print method, and then the device is closed. For PNG and JPEG, the device is opened, a new grid page is started, the grid is printed, and then the device is closed.

When exporting a list of objects:

- If `to` is a single PDF file (length is 1), the function creates a multi-page PDF.
- If a vector of file names (one per object) is provided, each gridify object is written to its corresponding file.

Value

No value is returned; the function is called for its side effect of writing output to a file.

Note

`gridify` objects can be saved directly in `.Rmd` and `.Qmd` documents, just like in the `gridify` package vignettes.

gt `pct()` issue

Using `pct()` to set the width of `gt` tables can be unreliable when exporting to PDF. It is recommended to use `px()` to set the width in pixels instead.

Examples

```
library(gridify)
library(magrittr)
library(ggplot2)

# Create a gridify object using a ggplot and a custom layout:

# Set text elements on various cells:
gridify_obj <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = pharma_layout_base(
    margin = grid::unit(c(0.5, 0.5, 0.5, 0.5), "inches"),
    global_gpar = grid::gpar(fontfamily = "serif", fontsize = 10)
  )
) %>%
  set_cell("header_left_1", "My Company") %>%
  set_cell("header_left_2", "<PROJECT> / <INDICATION>") %>%
  set_cell("header_left_3", "<STUDY>") %>%
  set_cell("header_right_1", "CONFIDENTIAL") %>%
  set_cell("header_right_2", "<Draft or Final>") %>%
  set_cell("header_right_3", "Data Cut-off: YYYY-MM-DD") %>%
```

```
  set_cell("output_num", "<Output> xx.xx.xx") %>%
  set_cell("title_1", "<Title 1>") %>%
  set_cell("title_2", "<Title 2>") %>%
  set_cell("title_3", "<Optional Title 3>") %>%
  set_cell("by_line", "By: <GROUP>, <optionally: Demographic parameters>") %>%
  set_cell("note", "<Note or Footnotes>") %>%
  set_cell("references", "<References:>") %>%
  set_cell("footer_left", "Program: <PROGRAM NAME>, YYYY-MM-DD at HH:MM") %>%
  set_cell("footer_right", "Page xx of nn") %>%
  set_cell("watermark", "DRAFT")

# Export a result to different file types

# Different file export formats require specific capabilities in your R installation.
# Use capabilities() to check which formats are supported in your R build.

# PNG
temp_png_default <- tempfile(fileext = ".png")
export_to(
  gridify_obj,
  to = temp_png_default
)

temp_png_custom <- tempfile(fileext = ".png")
export_to(
  gridify_obj,
  to = temp_png_custom,
  width = 2400,
  height = 1800,
  res = 300
)

# JPEG
temp_jpeg_default <- tempfile(fileext = ".jpeg")
export_to(
  gridify_obj,
  to = temp_jpeg_default
)

temp_jpeg_custom <- tempfile(fileext = ".jpeg")
export_to(
  gridify_obj,
  to = temp_jpeg_custom,
  width = 2400,
  height = 1800,
  res = 300
)

# TIFF
temp_tiff_default <- tempfile(fileext = ".tiff")
export_to(
  gridify_obj,
  to = temp_tiff_default
```

```
)

temp_tiff_custom <- tempfile(fileext = ".tiff")
export_to(
  gridify_obj,
  to = temp_tiff_custom,
  width = 2400,
  height = 1800,
  res = 300
)

# PDF
temp_pdf_A4 <- tempfile(fileext = ".pdf")
export_to(
  gridify_obj,
  to = temp_pdf_A4
)

temp_pdf_A4long <- tempfile(fileext = ".pdf")
export_to(
  gridify_obj,
  to = temp_pdf_A4long,
  width = 8.3,
  height = 11.7
)

# Use different pdf device - cairo_pdf
temp_pdf_A4long_cairo <- tempfile(fileext = ".pdf")
export_to(
  gridify_obj,
  to = temp_pdf_A4long_cairo,
  device = grDevices::cairo_pdf,
  width = 8.3,
  height = 11.7
)

# Multiple Objects - a list

gridify_list <- list(gridify_obj, gridify_obj)

temp_pdf_multipageA4 <- tempfile(fileext = ".pdf")
export_to(
  gridify_list,
  to = temp_pdf_multipageA4
)

temp_pdf_multipageA4long <- tempfile(fileext = ".pdf")
export_to(
  gridify_list,
  to = temp_pdf_multipageA4long,
  width = 8.3,
  height = 11.7
)
```

```
temp_png_multi <- c(tempfile(fileext = ".png"), tempfile(fileext = ".png"))
export_to(
  gridify_list,
  to = temp_png_multi
)

temp_png_multi_custom <- c(tempfile(fileext = ".png"), tempfile(fileext = ".png"))
export_to(
  gridify_list,
  to = temp_png_multi_custom,
  width = 800,
  height = 600,
  res = 96
)
```

`get_layouts` *Get the gridify layouts*

Description

Lists out all the layout functions exported from the gridify package.

Usage

```
get_layouts()
```

Value

A vector listing out the names of the layout functions from the gridify package.

See Also

`complex_layout()`, `simple_layout()`, `pharma_layout_base()`, `pharma_layout_A4()`, `pharma_layout_letter()`

Examples

```
get_layouts()
```

`gridify` *Create a gridify object*

Description

This function creates a gridify object, which represents an object with a specific layout and text elements around the output. The object can be a grob, ggplot2, gt, flextable, formula object. The layout can be a gridifyLayout object or a function that returns a gridifyLayout object.

Usage

```
gridify(object = grid::nullGrob(), layout, elements = list(), ...)
```

Arguments

`object`	A grob or ggplot2, gt, flextable, formula object. Default is `grid::nullGrob()`.
`layout`	A gridifyLayout object or a function that returns a gridifyLayout object. You can use predefined layouts; the `get_layouts()` function prints names of available layouts. You can create your own layout, please read `vignette("create_custom_layout", package = "gridify")` for more information.
`elements`	A list of text elements to fill the cells in the layout. Useful only in specific situations, please consider using `set_cell` method to set text elements around the output. Please note the elements list has to have a specific structure, please see the example.
`...`	Additional arguments.

Details

The elements argument is a list of elements to fill the cells, it can be used instead of or in conjunction with `set_cell`. Please access the vignettes for more information about gridify.

Value

A gridifyClass object.

Note

When setting your text within the `elements` argument, you can add new lines by using the new-line character, `\n`. The addition of `\n` may require setting a smaller `lineheight` argument in the `grid::gpar`. For all layouts with the default `scales = "fixed"`, the layout will automatically adjust to fit the new lines, ensuring no elements overlap.

See Also

`set_cell()`, `show_layout()`, print,gridifyClass-method, show,gridifyClass-method

Examples

```
library(magrittr)
object <- ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
  ggplot2::geom_point()
gridify(
  object = object,
  layout = simple_layout()
) %>%
  set_cell("title", "My Title", gpar = grid::gpar(fontsize = 30)) %>%
  set_cell("footer", "My Footer", gpar = grid::gpar(fontsize = 10))

gridify(
  gt::gt(head(mtcars)),
  layout = complex_layout(scales = "fixed")
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_middle", "Middle Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("subtitle", "Subtitle") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer") %>%
  set_cell("footer_middle", "Middle Footer") %>%
  set_cell("footer_right", "Right Footer")

# We encourage usage of set_cell but you can also use the elements argument
# to set text elements around the output.
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout(),
  elements = list(
    title = list(text = "My Title", gpar = grid::gpar(fontsize = 30)),
    footer = list(text = "My Footer", gpar = grid::gpar(fontsize = 10))
  )
)
```

---

gridifyCell *Create a gridifyCell*

Description

Function for creating a new instance of the gridifyCell class. Multiple gridifyCell objects are inputs for gridifyCells.

Usage

```
gridifyCell(
  row,
```

```
  col,
  text = character(0),
  mch = Inf,
  x = 0.5,
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  rot = 0,
  gpar = grid::gpar()
)
```

Arguments

row	A numeric value, span or a sequence specifying the range of occupied rows of the cell.
col	A numeric value, span or a sequence specifying the range of occupied columns of the cell.
text	A character value specifying the default text for the cell. Default `character(0)`.
mch	A numeric value specifying the maximum number of characters per line. The functionality is based on the `strwrap` function. By default, it avoids breaking up words and only splits lines when specified. Default `Inf`.
x	A numeric value specifying the x position of text in the cell. Default `0.5`.
y	A numeric value specifying the y position of text in the cell. Default `0.5`.
hjust	A numeric value specifying the horizontal position of the text in the cell, relative to the x value. Default `0.5`.
vjust	A numeric value specifying the vertical position of the text in the cell, relative to the y value. Default `0.5`.
rot	A numeric value specifying the rotation of the cell. Default `0`.
gpar	A `grid::gpar()` object specifying the graphical parameters of the cell. Default `grid::gpar()`.

Value

An instance of the gridifyCell class.

See Also

`gridifyCells()`, `gridifyLayout()`

Examples

```
cell <- gridifyCell(
  row = 1,
  col = 1:2,
  text = "Default cell text",
  mch = Inf,
  x = 0.5,
```

```
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  rot = 0,
  gpar = grid::gpar()
)
```

`gridifyCell-class` *gridifyCell class*

Description

Class for creating a cell used in a gridify layout.

Slots

`row` A numeric value, span or a sequence specifying the range of occupied rows of the cell.

`col` A numeric value, span or a sequence specifying the range of occupied columns of the cell.

`text` A character value specifying the default text for the cell.

`mch` A numeric value specifying the maximum number of characters per line. The functionality is based on the `strwrap` function. By default, it avoids breaking up words and only splits lines when specified.

`x` A numeric value specifying the x position of text in the cell.

`y` A numeric value specifying the y position of text in the cell.

`hjust` A numeric value specifying the horizontal position of the text in the cell, relative to the x value.

`vjust` A numeric value specifying the vertical position of the text in the cell, relative to the y value.

`rot` A numeric value specifying the rotation of the cell.

`gpar` A `grid::gpar()` object specifying the graphical parameters of the cell.

`gridifyCells` *Create a gridifyCells*

Description

Function for creating a new instance of the gridifyCells class. gridifyCells consists of multiple gridifyCell objects and is an input object for gridifyLayout.

Usage

```
gridifyCells(...)
```

Arguments

...	Arguments passed to the new function to create an instance of the gridifyCells class. Each argument should be the result of a call to gridifyCell.

Value

An instance of the gridifyCells class.

See Also

gridifyLayout()

Examples

```
cell1 <- gridifyCell(
  row = 1,
  col = 1,
  x = 0.5,
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  rot = 0,
  gpar = grid::gpar()
)
cell2 <- gridifyCell(
  row = 2,
  col = 2,
  x = 0.5,
  y = 0.5,
  hjust = 0.5,
  vjust = 0.5,
  rot = 0,
  gpar = grid::gpar()
)
cells <- gridifyCells(title = cell1, footer = cell2)
```

---

gridifyCells-class *gridifyCells class*

Description

Class for creating a list of cells in a gridify layout.

Slots

`cells` A list of cell objects.

gridifyClass-class *gridifyClass class*

Description

Class for creating a gridify object.

Slots

`object` A grob like object.

`layout` A gridifyLayout object.

`elements` A list of text elements, calls to `set_cell()`.

gridifyLayout *Create a gridifyLayout*

Description

Function for creating a new instance of the gridifyLayout class.

Usage

```
gridifyLayout(
  nrow,
  ncol,
  heights,
  widths,
  margin,
  global_gpar = grid::gpar(),
  background = grid::get.gpar()$fill,
  adjust_height = TRUE,
  object,
  cells
)
```

Arguments

`nrow`	An integer specifying the number of rows in the layout.
`ncol`	An integer specifying the number of columns in the layout.
`heights`	A call to `grid::unit()` specifying the heights of the rows.
`widths`	A call to `grid::unit()` specifying the widths of the columns.
`margin`	A `grid::unit()` specifying the margins around the object. Must be a vector of length 4, one element for each margin, with values in order for top, right, bottom, left.

global_gpar	A call to grid::gpar() specifying the global graphical parameters. Default is grid::gpar().
background	A string with background colour. Default grid::get.gpar()$fill.
adjust_height	A logical value indicating whether to automatically adjust the height of the object to make sure all of the text elements around the output do not overlap. This only applies for rows with height defined in cm, mm, inch or lines units. Default is TRUE.
object	A call to gridifyObject specifying the row and column location of the object.
cells	A call to gridifyCells listing out the text element cells required for the layout.

Value

A new instance of the gridifyLayout class.

See Also

gridifyCells(), gridifyCell(), gridifyObject()

Examples

```
layout <- gridifyLayout(
  nrow = 3L,
  ncol = 1L,
  heights = grid::unit(c(0.15, 0.7, 0.15), "npc"),
  widths = grid::unit(1, "npc"),
  margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
  global_gpar = grid::gpar(),
  background = grid::get.gpar()$fill,
  adjust_height = FALSE,
  object = gridifyObject(row = 2, col = 1),
  cells = gridifyCells(
    title = gridifyCell(row = 1, col = 1),
    footer = gridifyCell(row = 3, col = 1)
  )
)

# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = layout
) %>%
  set_cell("title", "TITLE") %>%
  set_cell("footer", "FOOTER")

new_layout <- function(
    margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
    global_gpar = grid::gpar()) {
  gridifyLayout(
```

```
    nrow = 4L,
    ncol = 1L,
    heights = grid::unit(c(3, 0.5, 1, 3), c("cm", "cm", "null", "cm")),
    widths = grid::unit(1, "npc"),
    global_gpar = global_gpar,
    background = grid::get.gpar()$fill,
    margin = margin,
    adjust_height = FALSE,
    object = gridifyObject(row = 3, col = 1),
    cells = gridifyCells(
      title = gridifyCell(row = 1, col = 1, text = "Default Title"),
      subtitle = gridifyCell(row = 2, col = 1),
      footer = gridifyCell(row = 4, col = 1)
    )
  )
}
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = new_layout()
) %>%
  set_cell("subtitle", "SUBTITLE") %>%
  set_cell("footer", "FOOTER")
```

`gridifyLayout-class` *gridifyLayout class*

Description

Class for creating a layout for a gridify object.

Slots

`nrow` An integer specifying the number of rows in the layout.

`ncol` An integer specifying the number of columns in the layout.

`heights` A `grid::unit()` call specifying the heights of the rows.

`widths` A `grid::unit()` call specifying the widths of the columns.

`margin` A `grid::unit()` specifying the margins around the object.

`global_gpar` A `grid::gpar()` object specifying the global graphical parameters.

`background` A string with background colour.

`adjust_height` A logical value indicating whether to adjust the height of the object. Only applies for cells with height defined in cm, mm, inch or lines units.

`object` A grob object.

`cells` A list of cell objects.

gridifyObject *Create a gridifyObject*

Description

Function for creating a new instance of the gridifyObject class.

Usage

```
gridifyObject(row, col, height = 1, width = 1)
```

Arguments

row	A numeric value, span or sequence specifying the row position of the object.
col	A numeric value, span or sequence specifying the row position of the object.
height	A numeric value specifying the height of the object. Default is 1.
width	A numeric value specifying the width of the object. Default is 1.

Value

An instance of the gridifyObject class.

See Also

gridifyLayout()

Examples

```
object <- gridifyObject(row = 1, col = 1, height = 1, width = 1)
```

gridifyObject-class *gridifyObject class*

Description

Class for creating an object in a gridify layout.

Slots

row A numeric value, span or sequence specifying the row position of the object.

col A numeric value, span or sequence specifying the column position of the object.

height A numeric value specifying the height of the object.

width A numeric value specifying the width of the object.

`layout_issue` *Template for Layout Issues Note*

Description

Template for Layout Issues Note

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
```

```
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

`paginate_table` *Split a data frame into pages for multi-page tables*

Description

A lightweight utility function to split a data frame into pages based on the number of rows per page. This is useful when creating multi-page tables with `gridify`.

Usage

```
paginate_table(data, rows_per_page = NULL, split_by = NULL, fill_empty = NULL)
```

Arguments

`data`	A data frame to split into pages.
`rows_per_page`	Integer or NULL. The maximum number of rows per page. When used with `split_by`, groups larger than `rows_per_page` will be split into multiple pages. When used alone, splits the entire dataset by row count. At least one of `rows_per_page` or `split_by` must be provided.
`split_by`	Character string or NULL. Name of a column in `data` to split by. Each unique value in this column starts a new page. Can be combined with `rows_per_page` to further split large groups. At least one of `rows_per_page` or `split_by` must be provided.

`fill_empty` Character string or NULL. When provided, fills incomplete pages with empty rows to match the target row count. Default is `NULL` (no filling). Providing a value (e.g., `"|"`, `""`, or `"–"`) automatically enables filling and uses that value for all cells in the empty rows. This helps maintain consistent vertical positioning across all pages. The target row count is the maximum page size across all pages.

Details

This is a simple utility to help with the common task of paginating large tables. After splitting the data, you can use a loop to create multiple `gridify` objects and export them as a multi-page PDF or separate image files.

The function does not perform the gridify conversion itself - it only prepares the data. This keeps the package lightweight and flexible.

Value

A list of data frames, one for each page. When `split_by` is used, the list is named with the group values. If a group spans multiple pages (when combined with `rows_per_page`), multiple list elements will have the same name. When only `rows_per_page` is used, returns an unnamed list.

Note

This function is designed to work with data frames. It is suited especially for use with gt package.

See Also

`gridify()`, `export_to()`

Examples

```
# Basic usage - split mtcars into pages of 10 rows
pages <- paginate_table(mtcars, rows_per_page = 10)
length(pages) # Number of pages

# With filled last page for consistent positioning
pages_filled <- paginate_table(mtcars, rows_per_page = 10, fill_empty = "-")
nrow(pages_filled[[1]]) # 10 rows
nrow(pages_filled[[length(pages_filled)]]) # Also 10 rows (filled with empty rows)

# With empty string fill
pages_empty <- paginate_table(mtcars, rows_per_page = 10, fill_empty = " ")

# Without filling (default)
pages_no_fill <- paginate_table(mtcars, rows_per_page = 10)

# Split by a grouping column
pages_by_cyl <- paginate_table(mtcars, split_by = "cyl")
length(pages_by_cyl) # 3 pages (one for each cylinder count: 4, 6, 8)
names(pages_by_cyl) # "4", "6", "8" - named with group values
```

```
# Split by column with filling to match maximum page size
pages_by_cyl_filled <- paginate_table(mtcars, split_by = "cyl", fill_empty = "|")
sapply(pages_by_cyl_filled, nrow) # All pages have same number of rows
names(pages_by_cyl_filled) # "4", "6", "8"

# Combine split_by and rows_per_page: split by cylinder, then by 5 rows
pages_combined <- paginate_table(mtcars, split_by = "cyl", rows_per_page = 5)
# Groups with more than 5 rows will be split into multiple pages
names(pages_combined) # e.g., "4", "6", "8", "8", "8" (8 cylinder group split into 3 pages)

# With filling for combined approach
pages_combined_filled <- paginate_table(
  mtcars,
  split_by = "cyl",
  rows_per_page = 5,
  fill_empty = "-"
)


library(gridify)
library(gt)
# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

# Regular Example with gt

pages <- paginate_table(mtcars, rows_per_page = 10, fill_empty = " ")

row_height_pixels <- 10
font_size <- 12
font_type <- "serif"

# Create gridify objects for each page
gridify_list <- lapply(seq_along(pages), function(page) {
  gt_table <- gt::gt(pages[[page]]) %>%
    gt::tab_options(
      table.width = gt::pct(80),
      data_row.padding = gt::px(row_height_pixels),
      table.font.size = font_size,
      table.font.names = font_type
    )

  gridify(
    gt_table,
    layout = pharma_layout_A4(global_gpar = grid::gpar(fontfamily = font_type))
  ) %>%
    set_cell("title_1", "My Multi-Page Table") %>%
    set_cell("footer_right", paste("Page", page, "of", length(pages)))
})

# Export as multi-page PDF
temp_my_multipage_table_gt_simple <- tempfile(fileext = ".pdf")
export_to(gridify_list, temp_my_multipage_table_gt_simple)
```

```
# By var Example with gt

pages <- paginate_table(mtcars, split_by = "cyl")

row_height_pixels <- 10
font_size <- 12
font_type <- "serif"

# Create gridify objects for each page
gridify_list <- lapply(seq_along(pages), function(page) {
  gt_table <- gt::gt(pages[[page]]) %>%
    gt::tab_options(
      table.width = gt::pct(80),
      data_row.padding = gt::px(row_height_pixels),
      table.font.size = font_size,
      table.font.names = font_type
    )

  gridify(
    gt_table,
    layout = pharma_layout_A4(global_gpar = grid::gpar(fontfamily = font_type))
  ) %>%
    set_cell("title_1", "My Multi-Page Table") %>%
    set_cell("by_line", sprintf("cyl is equal to %s", names(pages)[page])) %>%
    set_cell("footer_right", paste("Page", page, "of", length(pages)))
})

# Export as multi-page PDF
temp_my_multipage_table_gt_by <- tempfile(fileext = ".pdf")
export_to(gridify_list, temp_my_multipage_table_gt_by)
```

`pharma_layout` *Pharma Layouts*

Description

The `pharma_layout` functions define structured layouts for positioning text elements (titles, subtitles, footnotes, captions, etc.) around the outputs. These layouts ensure consistency in pharmaceutical reporting across different output formats, including A4 and letter paper sizes.

Available Layouts

- `pharma_layout_base()`: The base function for pharma layouts.
- `pharma_layout_A4()`: Layout specifically designed for A4 paper size.
- `pharma_layout_letter()`: Layout specifically designed for letter paper size.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
          # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
```

```
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

See Also

`pharma_layout_base()`, `pharma_layout_A4()`, `pharma_layout_letter()`

`pharma_layout_A4` *Pharma Layout (A4) for a gridify object*

Description

This function sets up the general structure for positioning the text elements for pharma layouts using the A4 paper size.

Usage

```
pharma_layout_A4(global_gpar = NULL, background = grid::get.gpar()$fill)
```

Arguments

`global_gpar`	A list specifying global graphical parameters to change in the layout. Default is NULL, however the defaults in the layout, inherited from `pharma_layout_base()`, are: `fontfamily = "Serif", fontsize = 9` and `lineheight = 0.95`, which can be overwritten alongside other graphical parameters found by `grid::get.gpar()`.
`background`	A character string specifying the background fill colour. Default `grid::get.gpar()$fill` for a white background.

Details

The margins for the A4 layout are:

- top = 1 inch
- right = 1.69 inches
- bottom = 1 inch
- left = 1 inch

The `pharma_layout_base()` function is used to set up the general layout structure, with these specific margins applied for the A4 format.

Value

A `gridifyLayout` object with the structure defined for A4 paper size.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
```

```
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

See Also

pharma_layout, `pharma_layout_base()`, `pharma_layout_letter()`

Examples

```
pharma_layout_A4()
# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)
# Example with all cells filled out
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = pharma_layout_A4()
) %>%
  set_cell("header_left_1", "My Company") %>%
  set_cell("header_left_2", "<PROJECT> / <INDICATION>") %>%
  set_cell("header_left_3", "<STUDY>") %>%
  set_cell("header_right_1", "CONFIDENTIAL") %>%
  set_cell("header_right_2", "<Draft or Final>") %>%
  set_cell("header_right_3", "Data Cut-off: YYYY-MM-DD") %>%
  set_cell("output_num", "<Output> xx.xx.xx") %>%
  set_cell("title_1", "<Title 1>") %>%
  set_cell("title_2", "<Title 2>") %>%
  set_cell("title_3", "<Optional Title 3>") %>%
  set_cell("by_line", "By: <GROUP>, <optionally: Demographic parameters>") %>%
  set_cell("note", "<Note or Footnotes>") %>%
  set_cell("references", "<References:>") %>%
  set_cell("footer_left", "Program: <PROGRAM NAME>, YYYY-MM-DD at HH:MM") %>%
  set_cell("footer_right", "Page xx of nn") %>%
  set_cell("watermark", "DRAFT")
```

pharma_layout_base *Base Function for Pharma Layouts*

Description

This function sets up the general structure for positioning text elements for pharma layouts. It defines the layout with specified margins, global graphical parameters, and height adjustment. The layout includes cells for headers, titles, footers, and optional elements like watermarks.

Usage

```
pharma_layout_base(
  margin = grid::unit(c(t = 1, r = 1, b = 1, l = 1), units = "inches"),
  global_gpar = NULL,
  background = grid::get.gpar()$fill,
  adjust_height = TRUE
)
```

Arguments

margin	A `grid::unit` object defining the margins of the layout (top, right, bottom, left) in inches. Default is `grid::unit(c(1, 1, 1, 1), "inches")`.
global_gpar	A list specifying global graphical parameters to change in the layout. Default is NULL, however the defaults in the layout are: `fontfamily = "Serif", fontsize = 9` and `lineheight` which can be overwritten alongside other graphical parameters found by `grid::get.gpar()`.
background	A string specifying the background fill colour. Default `grid::get.gpar()$fill` for a white background.
adjust_height	A logical value indicating whether to adjust the height of the layout. Default is `TRUE`.

Details

This function is primarily used internally by other layout functions such as `pharma_layout_A4()` and `pharma_layout_letter()` to create specific layouts.

Value

A `gridifyLayout` object that defines the general structure and parameters for a pharma layout.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

See Also

pharma_layout, `pharma_layout_A4()`, `pharma_layout_letter()`

Examples

```
# Create a general pharma layout with default settings
pharma_layout_base()

library(magrittr)
# Customize margins and global graphical parameters and fill all cells
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = pharma_layout_base(
    margin = grid::unit(c(0.5, 0.5, 0.5, 0.5), "inches"),
    global_gpar = list(col = "blue", fontsize = 10)
  )
) %>%
  set_cell("header_left_1", "My Company") %>%
  set_cell("header_left_2", "<PROJECT> / <INDICATION>") %>%
  set_cell("header_left_3", "<STUDY>") %>%
  set_cell("header_right_1", "CONFIDENTIAL") %>%
  set_cell("header_right_2", "<Draft or Final>") %>%
  set_cell("header_right_3", "Data Cut-off: YYYY-MM-DD") %>%
  set_cell("output_num", "<Output> xx.xx.xx") %>%
  set_cell("title_1", "<Title 1>") %>%
  set_cell("title_2", "<Title 2>") %>%
  set_cell("title_3", "<Optional Title 3>") %>%
  set_cell("by_line", "By: <GROUP>, <optionally: Demographic parameters>") %>%
  set_cell("note", "<Note or Footnotes>") %>%
  set_cell("references", "<References:>") %>%
  set_cell("footer_left", "Program: <PROGRAM NAME>, YYYY-MM-DD at HH:MM") %>%
  set_cell("footer_right", "Page xx of nn") %>%
  set_cell("watermark", "DRAFT")
```

---

pharma_layout_letter *Pharma Layout (Letter) for a gridify object*

Description

This function sets up the general structure for positioning the text elements for pharma layouts using the letter paper size.

Usage

```
pharma_layout_letter(global_gpar = NULL, background = grid::get.gpar()$fill)
```

Arguments

global_gpar	A list specifying global graphical parameters to change in the layout. Default is NULL, however the defaults in the layout, inherited from `pharma_layout_base()`, are: `fontfamily = "Serif", fontsize = 9` and `lineheight = 0.95`, which can be overwritten alongside other graphical parameters found by `grid::get.gpar()`.
background	A character string specifying the background fill colour. Default `grid::get.gpar()$fill` for a white background.

Details

The margins for the letter layout are:

- top = 1 inch
- right = 1 inch
- bottom = 1.23 inches
- left = 1 inch

The `pharma_layout_base()` function is used to set up the general layout structure, with these specific margins applied for the letter format.

Value

A `gridifyLayout` object with the structure defined for letter paper size.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

See Also

pharma_layout, `pharma_layout_base()`, `pharma_layout_A4()`

Examples

```
pharma_layout_letter()
```

```
library(magrittr)
# Example with all cells filled out
gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = pharma_layout_letter()
) %>%
  set_cell("header_left_1", "My Company") %>%
  set_cell("header_left_2", "<PROJECT> / <INDICATION>") %>%
  set_cell("header_left_3", "<STUDY>") %>%
  set_cell("header_right_1", "CONFIDENTIAL") %>%
  set_cell("header_right_2", "<Draft or Final>") %>%
  set_cell("header_right_3", "Data Cut-off: YYYY-MM-DD") %>%
  set_cell("output_num", "<Output> xx.xx.xx") %>%
  set_cell("title_1", "<Title 1>") %>%
  set_cell("title_2", "<Title 2>") %>%
  set_cell("title_3", "<Optional Title 3>") %>%
  set_cell("by_line", "By: <GROUP>, <optionally: Demographic parameters>") %>%
  set_cell("note", "<Note or Footnotes>") %>%
  set_cell("references", "<References:>") %>%
  set_cell("footer_left", "Program: <PROGRAM NAME>, YYYY-MM-DD at HH:MM") %>%
  set_cell("footer_right", "Page xx of nn") %>%
  set_cell("watermark", "DRAFT")
```

`print,gridifyClass-method` *Print method for gridifyClass*

Description

Method for printing a gridifyClass object. Prevents the `show` method from being triggered.

Usage

```
## S4 method for signature 'gridifyClass'
print(x, ...)
```

Arguments

`x`	A gridifyClass object.
`...`	Additional arguments. Not yet in use.

Value

Invisibly a grid call used to draw the object.

See Also

gridify(), set_cell()

Examples

```
# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
) %>%
  set_cell("title", "TITLE")

print(g)

# grid call is returned when printed to a variable
gg <- print(g)
# unevaluated grid code
gg
# evaluate the code
grid::grid.draw(eval(gg, envir = attr(gg, "env")))
# or
OBJECT <- attr(gg, "env")[["OBJECT"]]
grid::grid.draw(eval(gg))
```

---

set_cell *Add text elements to a gridify cell*

Description

This function sets a text element for a specific cell in a gridify object. The element can be positioned and rotated as desired, and its graphical parameters can be customized.

Usage

```
set_cell(
  object,
  cell,
  text,
  mch = NULL,
  x = NULL,
  y = NULL,
  hjust = NULL,
  vjust = NULL,
  rot = NULL,
  gpar = NULL,
```

```
  ...
)

## S4 method for signature 'gridifyClass'
set_cell(
  object,
  cell,
  text,
  mch = NULL,
  x = NULL,
  y = NULL,
  hjust = NULL,
  vjust = NULL,
  rot = NULL,
  gpar = NULL,
  ...
)
```

Arguments

object	A gridifyClass object. (See note)
cell	A single character string specifying the name of the cell.
text	A single character string specifying the text of the element. When setting your string within the `text` argument, you can add new lines by using the newline character, `\n`.
mch	A positive numeric value specifying the maximum number of characters per line. The functionality is based on the `strwrap()` function. By default, it avoids breaking up words and only splits lines when specified.
x	A numeric value specifying the x (horizontal) location of the text element in the cell. Takes values between 0 and 1; 0 places the text element at the left side of the cell and 1 at the right side.
y	A numeric value specifying the y (vertical) location of the text element in the cell. Takes values between 0 and 1; 0 places the text element at the bottom of the cell and 1 at the top.
hjust	A numeric value specifying which part of the text element lines up with the x value. Adjusting this value changes how the text element is positioned horizontally relative to the x coordinate specified before. Takes values between 0 and 1; 0 aligns the left side of the text element with the x coordinate and 1 aligns the right side.
vjust	A numeric value specifying which part of the text element lines up with the y value. Adjusting this value changes how the text element is positioned vertically relative to the y coordinate specified before. Takes values between 0 and 1; 0 aligns the bottom of the text element with the y coordinate and 1 aligns the top.
rot	A numeric value specifying the rotation of the text element anticlockwise from the x-axis.
gpar	A `grid::gpar()` object specifying the graphical parameters of the text element.
...	Additional arguments.

Details

`set_cell()` can also make minor adjustments to the positioning of the text elements in the layout.

If the existing layouts generally meet your needs and you only require additional lines in certain cells, there is no need to create a new layout. By using the newline character, `\n`, within your text, you can add as many new lines as desired. For all layouts with the default `scales = "fixed"`, the layout will automatically adjust to fit the new lines, ensuring no elements overlap.

For applying more substantial changes to a layout or when applying adjustments across multiple objects and projects, it is recommended to create a custom layout instead. This will promote reproducibility and consistency across projects. See `vignette("create_custom_layout", package = "gridify")` for more information on how to create a custom layout.

Value

The gridifyClass object with the added text element.

Note

The `object` argument has to be passed directly only when adding `set_cell()` after a `gridify` object has already been defined. We do NOT need to pass the `object` directly when using pipes. See first example.

See Also

`gridify()`

Examples

```
# using set_cell() without the pipe operator
object <- ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
  ggplot2::geom_line()

g <- gridify(object = object, layout = simple_layout())
g <- set_cell(g, "title", "TITLE")
g

# using set_cell() with the pipe operator
# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE")

# using multiple lines in set_cell()
gridify(object, layout = simple_layout()) %>%
  set_cell(cell = "title", text = "THIS IS THE MAIN TITLE\nA Second Title\nSubtitle") %>%
  set_cell(
    cell = "footer", text = "This is a footer.\nWe can have multiple lines here as well.",
    x = 0, hjust = 0
  )
```

```
# using mch in set_cell()
long_footer_string <- paste0(
  "This is a footer. We can have a long description here.",
  "We can have another long description here.",
  "We can have another long description here."
)
gridify(object, layout = simple_layout()) %>%
  set_cell(
    cell = "footer", long_footer_string, mch = 60, x = 0, hjust = 0
  )

# using the location and alignment arguments
# the left side of the text is on the left side of the cell
gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE", x = 0, hjust = 0)

# the right side of the text is on the right side of the cell
gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE", x = 1, hjust = 1)

# the right side of the text is 30% from the right side of the cell
gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE", x = 0.7, hjust = 1)

# using the rotation argument
gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE", x = 0.7, rot = 45)

# using the graphical parameters argument
gridify(object = object, layout = simple_layout()) %>%
  set_cell("title", "TITLE", x = 0.7, rot = 45, gpar = grid::gpar(fontsize = 20)) %>%
  set_cell("footer", "FOOTER", x = 0.2, y = 1, gpar = grid::gpar(col = "blue"))
```

`show,gridifyClass-method`

Show method for gridifyClass

Description

Method for showing a gridifyClass object.

Usage

```
## S4 method for signature 'gridifyClass'
show(object)
```

Arguments

`object`	A gridifyClass object.

Value

The object with all the titles, subtitles, footnotes, and other text elements around the output is printed in the graphics device. A list is also printed to the console containing:

- the dimensions of the layout
- where the object is located in the layout
- the size of the margin
- any global graphical parameters
- the list of elements cells and if they are filled or empty

See Also

gridify()

Examples

```
g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = complex_layout()
)
show(g)
```

`show,gridifyLayout-method`

Show method for gridifyLayout

Description

Method for showing a gridifyLayout object. It prints the names of the cells in the layout.

Usage

```
## S4 method for signature 'gridifyLayout'
show(object)
```

Arguments

`object`	A gridifyLayout object.

Value

The list of cells defined in the layout.

See Also

gridifyLayout()

Examples

```
show(complex_layout())
```

---

show_cells *Show the cells in a gridify object or layout*

Description

Method for showing the cells of a gridifyClass or gridifyLayout object. It prints the names of the cells and for gridifyClass it indicates whether each cell is filled or empty.

Usage

```
show_cells(object)

## S4 method for signature 'gridifyClass'
show_cells(object)

## S4 method for signature 'gridifyLayout'
show_cells(object)
```

Arguments

object	A gridifyClass or gridifyLayout object.

Value

A print out of the available cells and for gridifyClass indicates whether each cell is filled or empty.

Examples

```
show_cells(complex_layout())

# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
) %>%
  set_cell("title", "TITLE")

show_cells(g)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
```

```
  layout = complex_layout()
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer") %>%
  set_cell("footer_right", "Right Footer")

show_cells(g)
```

---

show_layout *Show the layout in a given gridify object or layout*

Description

Method for showing the layout of a gridifyClass or gridifyLayout object. It prints the layout of the object, including the number of rows and columns and the heights and widths of the rows and columns in the graphics device.

Usage

```
show_layout(x)

## S4 method for signature 'gridifyClass'
show_layout(x)

## S4 method for signature 'gridifyLayout'
show_layout(x)
```

Arguments

x	A gridifyClass or gridifyLayout object.

Value

An object showing the layout, including the widths and heights of all the rows and columns.

Note

When using `show_layout()`, not all lines are initially visible. Some lines may be assigned zero space and are dynamically updated to have more space once the text is added.

See Also

`gridify()`, `simple_layout()`, `complex_layout()`, pharma_layout, `pharma_layout_base()`, `pharma_layout_A4()`, `pharma_layout_letter()`

Examples

```
g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
)
show_layout(g)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = complex_layout()
)
show_layout(g)

show_layout(simple_layout())
show_layout(complex_layout())

# Example with a custom layout

custom_layout <- gridifyLayout(
  nrow = 3L,
  ncol = 2L,
  heights = grid::unit(c(0.15, 0.7, 0.15), "npc"),
  widths = grid::unit(c(0.5, 0.5), "npc"),
  margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
  global_gpar = grid::gpar(),
  adjust_height = FALSE,
  object = gridifyObject(row = 2, col = 1),
  cells = gridifyCells(header = gridifyCell(
    row = 1,
    col = 1,
    x = 0.5,
    y = 0.5,
    hjust = 0.5,
    vjust = 0.5,
    rot = 0,
    gpar = grid::gpar()
  ), footer = gridifyCell(
    row = 2,
    col = 2,
    x = 0.5,
    y = 0.5,
    hjust = 0.5,
    vjust = 0.5,
    rot = 0,
    gpar = grid::gpar()
  ))
)

show_layout(custom_layout)
```

show_spec *Show the layout specifications of a gridifyClass or gridifyLayout*

Description

Method for showing the specifications of the layout in a gridifyClass or gridifyLayout object, including, but not limited to:

- Layout dimensions
- Heights of rows
- Widths of columns
- Margins
- Graphical parameters defined in the layout.
- Default specs per cell.

Usage

```
show_spec(object)

## S4 method for signature 'gridifyLayout'
show_spec(object)

## S4 method for signature 'gridifyClass'
show_spec(object)
```

Arguments

object	A gridifyClass or gridifyLayout object.

Value

A print out of the specifications of a gridifyClass or gridifyLayout object.

Examples

```
show_spec(complex_layout())

# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
) %>%
  set_cell("title", "TITLE")
```

```
show_spec(g)

g <- gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = complex_layout()
) %>%
  set_cell("header_left", "Left Header") %>%
  set_cell("header_right", "Right Header") %>%
  set_cell("title", "Title") %>%
  set_cell("note", "Note") %>%
  set_cell("footer_left", "Left Footer") %>%
  set_cell("footer_right", "Right Footer")

show_spec(g)
```

`simple_layout` *Simple Layout for a gridify object*

Description

Creates a simple layout only containing two text element cells: a title and a footer.

Usage

```
simple_layout(
  margin = grid::unit(c(t = 0.1, r = 0.1, b = 0.1, l = 0.1), units = "npc"),
  global_gpar = grid::gpar(),
  background = grid::get.gpar()$fill,
  scales = c("fixed", "free")
)
```

Arguments

`margin`	A unit object specifying the margins around the output. Default is 10% of the output area on all sides.
`global_gpar`	A gpar object specifying the global graphical parameters. Must be the result of a call to `grid::gpar()`.
`background`	A string specifying the background fill colour. Default `grid::get.gpar()$fill` for a white background.
`scales`	A string, either `"free"` or `"fixed"`. By default, `"fixed"` ensures that text elements (titles, footers, etc.) retain a static height, preventing text overlap while maintaining a structured layout. However, this may result in different height proportions between the text elements and the output. The `"free"` option makes the row heights proportional, allowing them to scale dynamically based on the overall output size. This ensures that the text elements and the output maintain relative proportions.

Details

The layout consists of three rows, one each for the title, output, and footer.
The heights of the rows in simple_layout with `"free"` scales are 15%, 70% and 15% of the area respectively.
The heights of the rows in simple_layout with `"fixed"` scales are adjusted n number of lines for all text elements around the output and the rest of the area taken up by the output.
Please note that as output space is reduced, all text elements around the output retain their space which makes the output appear smaller.

Value

A gridifyLayout object.

Note

The Font Issue Information:

Changes to the fontfamily may be ignored by some devices, but is supported by PostScript, PDF, X11, Windows, and Quartz. The fontfamily may be used to specify one of the Hershey Font families (e.g., HersheySerif, serif), and this specification will be honoured on all devices.

If you encounter this warning, you can register the fonts using the `extrafont` package:

```
library(extrafont)
font_import()
loadfonts(device = 'all')
```

If you still see the warning while using RStudio, try changing the graphics backend.

Negative Dimensions Issues:

grobs from the grid package and ggplot2 objects (when converted to grobs by gridify) may appear distorted in the output if there is insufficient space in the window, caused by negative dimensions. This should be resolved. However, if this is affecting your layout, please increase your window size or only use static heights/widths for custom layouts.

The negative dimensions are caused by the way grid handles `null` and `npc` heights/widths so if some dimensions are static, then the `npc` or `null` values may cause unexpected behaviour when the window size is too small. It was resolved by setting a minimum size of the object in the gridify object to 1 inch for each dimension.

The following example demonstrates this behaviour Try resizing your window:

```
library(grid)
library(ggplot2)
grid.newpage()
object <- ggplot2::ggplotGrob(ggplot(mtcars, aes(mpg, wt)) + geom_line())
grid::grid.draw(
  grid::grobTree(
    grid::grobTree(
      grid::editGrob(
        object,
```

```
        vp = grid::viewport(
        # height = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch")),
         # width = grid::unit.pmax(grid::unit(1, "npc"), grid::unit(1, "inch"))
        )
      ),
      vp = grid::viewport(
        layout.pos.row = 2,
        layout.pos.col = 1:3
      )
    ),
    vp = grid::viewport(
      layout = grid::grid.layout(
        nrow = 3,
        ncol = 3,
        heights = grid::unit(c(9, 1, 9), c("cm", "null", "cm"))
      )
    )
  )
)
```

gt Font Size Issue:

When specifying font sizes, the `gt` package interprets values as having the unit pixels (`px`), whilst the `grid` package, on which `gridify` is built, assumes points (`pt`). As a result, even if you set the font sizes in both `gt` and `gridify` (using `grid::gpar()`) to the same number, they may still appear different. To convert point size to pixel size, multiply the point size by `96 / 72`.

Examples

```
simple_layout()

# (to use |> version 4.1.0 of R is required, for lower versions we recommend %>% from magrittr)
library(magrittr)

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
) %>%
  set_cell("title", "TITLE") %>%
  set_cell("footer", "FOOTER")

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout(
    margin = grid::unit(c(5, 2, 2, 2), "cm"),
    global_gpar = grid::gpar(fontsize = 20, col = "blue")
  )
) %>%
  set_cell("title", "TITLE") %>%
```

```
  set_cell("footer", "FOOTER")

gridify(
  object = ggplot2::ggplot(data = mtcars, ggplot2::aes(x = mpg, y = wt)) +
    ggplot2::geom_line(),
  layout = simple_layout()
) %>%
  set_cell("title", "TITLE\nSUBTITLE", x = 0.7, gpar = grid::gpar(fontsize = 12)) %>%
  set_cell("footer", "FOOTER", x = 0.5, y = 0.5, gpar = grid::gpar())
```

Index