

Package: pnpmisc (via r-universe)

October 13, 2026

Type Package

Title Utilities for Print-and-Play Board Games

Version 0.2.1

URL <https://github.com/trevorld/pnpmisc>,
<https://trevorldavis.com/R/pnpmisc/dev/>

BugReports <https://github.com/trevorld/pnpmisc/issues>

Description Functions for creating and editing PDF files for print-and-play board games with a consistent interface in which the input filename is the first argument, the output filename is the second, and all other arguments are named. Includes functions to resize, scale, rotate, and combine PDF pages, add crop marks, crosshairs, and origami fold guides, and create plastic storage box jackets and origami card wallets. Also provides layout functions for splitting pages into individual component images and reassembling them. Wraps PDF functionality from 'qpdf', 'pdftools', and 'xmpdf' and optionally uses 'ghostscript', 'pdfxup', and 'bittermelon' for additional functionality.

Depends R (>= 4.2)

Imports grid, grDevices, pdftools, qpdf (>= 1.4.1), rlang, stats, tools, utils

Suggests bittermelon (>= 2.2.0-1), gridpattern, jpeg, piecepackr (>= 1.14.0-6), png, marquee, testthat (>= 3.0.0), xmpdf

SystemRequirements Several features require `ghostscript`. To make N-up pdf documents you'll also need `pdfxup`.

License MIT + file LICENSE

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libjpeg-dev libssl-dev libpoppler-cpp-dev poppler-data

Repository <https://trevorld.r-universe.dev>

Date/Publication 2026-08-25 21:13:33 UTC

RemoteUrl <https://github.com/trevorld/pnpmisc>

RemoteRef HEAD

RemoteSha 8853c3088cc9a2cd589e31d8372821889be598a6

Contents

bm_create_pdf	3
bm_crop_layout	4
bm_rm_crosshairs	5
bm_split_layout	6
fullGrob	7
grid_add_cropmarks	8
grid_add_crosshairs	9
grid_add_layout	10
grid_add_lines	11
grid_add_origami_bb	12
grid_add_rects	13
layout_grid	14
layout_name_fn	15
layout_octavo	15
layout_preset	17
ls_temp_pdfs	18
pdf_add_cropmarks	19
pdf_add_crosshairs	20
pdf_add_lines	21
pdf_add_origami	21
pdf_add_overlay	22
pdf_add_rects	23
pdf_append_blank	24
pdf_apply	25
pdf_change_paper	27
pdf_clean	29
pdf_combine	29
pdf_create_blank	31
pdf_create_jacket	32
pdf_create_wallet	35
pdf_gs	37
pdf_orientation	38
pdf_pages	38
pdf_paper	39
pdf_pdfxup	40
pdf_rasterize	42
pdf_render_bm_list	43

<i>bm_create_pdf</i>	3
pdf_render_bm_pixmap	44
pdf_render_raster	45
pdf_rep	46
pdf_rm_crosshairs	47
pdf_scale	48
pdf_set_bookmarks	49
pdf_width	50
zip_extract_bm_pixmap	51
Index	52

Examples

```
f1 <- pdf_create_blank(paper = "a4", orientation = "landscape")
f2 <- pdf_create_blank(length = 4L)
unlink(f1)
unlink(f2)
```

bm_crop_layout	<i>Crop out a component from a print-and-play layout</i>
----------------	--

Description

bm_crop_layout() crops out a print-and-play component from a layout.

Usage

```
bm_crop_layout(
  page,
  ...,
  layout = "button_shy_cards",
  row = 1L,
  col = 1L,
  bleed = FALSE,
  name = NULL
)
```

Arguments

page	A bittermelon::bm_pixmap() object representing a print-and-play layout page. Often the output from pdf_render_bm_pixmap() .
...	Ignored for now.
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
row, col	The row and col of the component in the layout (integers).
bleed	Include the bleed (if available).
name	Instead of row and col can instead use the name of the component in the layout (string).

Value

A [bittermelon::bm_pixmap\(\)](#) object.

See Also

[bm_split_layout\(\)](#) to extract all components from the layout

Examples

```
## Not run: # User not expected to have this PDF file
if (requireNamespace("bittermelon", quietly = TRUE)) {
  input <- "A Nice Cuppa - PNP.pdf"
  page <- pdf_render_bm_pixmap(input, page = 4L, dpi = 75)
  card <- bm_crop_layout(page, layout = "button_shy_cards", row = 1L, col = 1L)
  grid::grid.raster(card)
}

## End(Not run)
```

bm_rm_crosshairs	<i>Remove crosshairs from a raster object</i>
------------------	---

Description

bm_rm_crosshairs() removes crosshairs from a raster object.

Usage

```
bm_rm_crosshairs(x, ..., layout = "poker_3x2_bleed")
```

Arguments

x	A raster object coercible to a <code>bittermelon::bm_pixmap()</code> object.
...	Ignored for now.
layout	Either a layout preset name in <code>layout_names()</code> or a data frame with layout data (as returned by <code>layout_grid()</code>).

Details

- In order to work the PnP layout needs a solid color bleed.
- The default layout supports **Galdor's Grip** (PnP letter size v1).

Value

A `bittermelon::bm_pixmap()` object.

See Also

`pdf_rm_crosshairs()` to remove crosshairs from a pdf.

Examples

```
if (requireNamespace("bittermelon", quietly = TRUE) &&
    requireNamespace("piecepackr", quietly = TRUE) &&
    utils::packageVersion("piecepackr") >= "1.14.0-5") {
  f1 <- pdf_create_blank(length = 1L, width = 11, height = 8.5)
  f2 <- pdf_add_crosshairs(f1, pages = "all",
                          layout = "poker_3x2_bleed")
  page <- pdf_render_bm_pixmap(f2, page = 1L, dpi = 75)
  page2 <- bm_rm_crosshairs(page, layout = "poker_3x2_bleed")
  unlink(f1)
  unlink(f2)
}
```

bm_split_layout	<i>Extract components from a print-and-play layout</i>
-----------------	--

Description

bm_split_layout() extracts print-and-play components from a layout.

Usage

```
bm_split_layout(page, ..., layout = "button_shy_cards", bleed = FALSE)
```

Arguments

page	A <code>bittermelon::bm_pixmap()</code> object representing a print-and-play layout page. Often the output from <code>pdf_render_bm_pixmap()</code> .
...	Ignored for now.
layout	Either a layout preset name in <code>layout_names()</code> or a data frame with layout data (as returned by <code>layout_grid()</code>).
bleed	Include the bleed (if available).

Value

A `bittermelon::bm_list()` of `bittermelon::bm_pixmap()` objects.

See Also

`bm_crop_layout()` to just crop a single component

Examples

```
## Not run: # User not expected to have this PDF file
if (requireNamespace("bittermelon", quietly = TRUE)) {
  input <- "A Nice Cuppa - PNP.pdf"
  page <- pdf_render_bm_pixmap(input, page = 4L, dpi = 75)
  cards <- bm_split_layout(page, layout = "button_shy_cards")
  grid::grid.raster(cards[[1L]])
}

## End(Not run)
```

fullGrob

A rectangular grob filled with a color, image, or pattern.

Description

fullGrob() is a wrapper around `grid::rectGrob()` for a single rectangle grob filled with a color, bitmap, grob, or pattern. In particular grob or bitmap fills will be automatically converted to a `grid::pattern()` fill by `as_fill()`. `grid.full()` creates a fullGrob() and draws it to the screen.

Usage

```
fullGrob(
  fill = "transparent",
  ...,
  width = NULL,
  height = NULL,
  vp = NULL,
  name = NULL
)

grid.full(...)

as_fill(fill = "transparent", ..., width = NULL, height = NULL)
```

Arguments

fill	A color string, a <code>grid::grob()</code> , a <code>grid::pattern()</code> , or a bitmap like a <code>bittermelon::bm_pixmap()</code> or raster object.
...	Ignored for now.
width, height	If fill is a bitmap object then these will be passed to <code>grid::rasterGrob()</code> .
vp, name	Passed to <code>grid::rectGrob()</code> .

Value

fullGrob() and grid.full() return `grid::rectGrob()` object. As a side effect `grid.full()` draws to the active graphics device. `as_fill()` returns a supported grid fill object such as a color string or a `grid::pattern()` object.

Examples

```
grob <- fullGrob(grid::radialGradient())
grid::grid.draw(grob)
```

grid_add_cropmarks	<i>Draw crop marks around components</i>
--------------------	--

Description

`grid_add_cropmarks()` adds crop marks to the edges of components of a print-and-play layout.

Usage

```
grid_add_cropmarks(
  ...,
  layout = "poker_3x3",
  bleed = NULL,
  gp = gpar(),
  cm_width = unit(0.25, "mm"),
  cm_length = unit(0.125, "in")
)
```

Arguments

...	Passed to <code>piecepackr::grid.cropmark()</code> .
layout	Either a layout preset name in <code>layout_names()</code> or a data frame with layout data (as returned by <code>layout_grid()</code>).
bleed	Passed to <code>piecepackr::grid.cropmark()</code> . If NULL defaults to <code>max(max(layout\$bleed), 0.125)</code> .
gp, cm_width, cm_length	Passed to <code>piecepackr::grid.cropmark()</code> .

Details

- This function draws in **inches** so make sure your graphics device is "big" enough.

Value

NULL invisibly. As a side effect draws crop marks to the active graphics device.

See Also

[pdf_add_cropmarks\(\)](#), [piecepackr::grid.cropmark\(\)](#)

Examples

```
if (requireNamespace("piecepackr", quietly = TRUE) &&
    utils::packageVersion("piecepackr") >= "1.14.0-6") {
  grid::grid.newpage()
  vp <- grid::viewport(width=8.5, height=11, default.units="in",
                       x=0.5, y=0.5, just=c("left", "bottom"))
  grid::pushViewport(vp)
  grid_add_rects(layout = "poker_3x3")
  grid_add_cropmarks(layout = "poker_3x3")
  grid::popViewport()
}
```

`grid_add_crosshairs` *Draw crosshairs at component corners*

Description

`grid_add_crosshairs()` draws crosshairs at the corners of components of a print-and-play layout.

Usage

```
grid_add_crosshairs(
  ...,
  layout = "poker_3x3",
  gp = gpar(),
  ch_width = unit(1/6, "in"),
  ch_grob = NULL
)
```

Arguments

`...` Passed to [piecepackr::grid.crosshair\(\)](#).

`layout` Either a layout preset name in [layout_names\(\)](#) or a data frame with layout data (as returned by [layout_grid\(\)](#)).

`gp, ch_width, ch_grob` Passed to [piecepackr::grid.crosshair\(\)](#). `ch_grob` defaults to [piecepackr::squaresCrosshairGro](#)

Details

- This function draws in **inches** so make sure your graphics device is "big" enough.

Value

NULL invisibly. As a side effect draws crosshairs to the active graphics device.

Examples

```

if (requireNamespace("piecepackr", quietly = TRUE) &&
    utils::packageVersion("piecepackr") >= "1.14.0-5") {
  grid::grid.newpage()
  vp <- grid::viewport(width=8.5, height=11, default.units="in",
                       x=0.5, y=0.5, just=c("left", "bottom"))
  grid::pushViewport(vp)
  grid_add_crosshairs(layout = "poker_3x3")
  grid::popViewport()
}

```

grid_add_layout

*Draw images to a layout with grid***Description**

grid_add_layout() draws images to a layout in grid.

Usage

```
grid_add_layout(images, ..., layout = "poker_3x3")
```

Arguments

images	A named list of images with all the names in the name column of the layout argument. Currently supports raster objects supported by <code>bittermelon</code> and <code>grid</code> objects.
...	Ignored for now.
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).

Value

NULL invisibly. As a side effect draws images to the active graphics device.

See Also

[piecepackr::pmap_piece\(\)](#) to draw pieces to a layout represented by a [piecepackr::pp_cfg\(\)](#) object.

Examples

```

if (requireNamespace("bittermelon", quietly = TRUE)) {
  crops <- bittermelon::farming_crops_16x16()$corn
  layout <- data.frame(
    name = names(crops),
    x = 0.1 + rep(c(1, 3, 5), 2),
    y = 0.1 + rep(c(3, 1), each = 3),

```

```

      width = 2,
      height = 2,
      angle = 0
    )
    grid::grid.newpage()
    grid_add_layout(crops, layout = layout)
    grid_add_rects(layout = layout, gp = grid::gpar(lwd = 4))
  }

```

grid_add_lines

Draw lines along component edges

Description

grid_add_lines() draws lines along the components of a print-and-play layout.

Usage

```
grid_add_lines(..., layout = "poker_3x3", gp = gpar())
```

Arguments

...	Ignored for now.
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
gp	Passed to grid::grid.segments() .

Details

- This function draws in **inches** so make sure your graphics device is "big" enough.

Value

NULL invisibly. As a side effect draws lines to the active graphics device.

See Also

[grid::grid.segments\(\)](#)

Examples

```

grid::grid.newpage()
vp <- grid::viewport(width=8.5, height=11, default.units="in",
                     x=0.0, y=0.0, just=c("left", "bottom"))
grid::pushViewport(vp)
grid_add_lines(layout = "poker_3x3",
              gp = grid::gpar(lty = "dashed", col = "grey"))
grid::popViewport()

```

grid_add_origami_bb *Draw origami fold symbols*

Description

grid_add_origami() draws origami fold symbols for a box jacket. grid_add_origami_bb() draws origami fold symbols optimized for **Boardgame Barrio's Small Board Game Jackets**.

Usage

```
grid_add_origami_bb()

grid_add_origami(
    ...,
    xc = unit(0.5, "npc"),
    yc = unit(0.5, "npc"),
    width = unit(JACKET_4x6_FRONT_WIDTH, "inches"),
    height = unit(JACKET_4x6_HEIGHT, "inches"),
    depth = unit(JACKET_4x6_SPINE_WIDTH, "inches")
)
```

Arguments

...	Ignored.
xc, yc	Center of the jacket as <code>grid::unit()</code> objects.
width	Front/back panel width as a <code>grid::unit()</code> object.
height	Jacket height as a <code>grid::unit()</code> object.
depth	Spine width as a <code>grid::unit()</code> object.

Value

NULL invisibly. As a side effect draws origami fold symbols on the active graphics device.

See Also

[pdf_add_origami\(\)](#)

Examples

```
grid::grid.newpage()
pnpmisc::grid_add_jacket_outer()
grid_add_origami()
```

grid_add_rects	<i>Draw (rounded) rectangles around components</i>
----------------	--

Description

grid_add_rects() draws (rounded) rectangles around components of a print-and-play layout.

Usage

```
grid_add_rects(
  ...,
  layout = "poker_3x3",
  r = unit(0, "in"),
  gp = gpar(col = "black", fill = NA, lwd = 1)
)
```

Arguments

...	Ignored for now.
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
r, gp	Passed to grid::grid.roundrect() .

Details

- This function draws in **inches** so make sure your graphics device is "big" enough.
- Sometimes if you use the same color as a solid background color this can be used to effectively "remove" unwanted card outlines.

Value

NULL invisibly. As a side effect draws rectangles to the active graphics device.

See Also

[pdf_add_rects\(\)](#), [grid::grid.roundrect\(\)](#)

Examples

```
grid::grid.newpage()
vp <- grid::viewport(width=8.5, height=11, default.units="in",
                     x=0.5, y=0.5, just=c("left", "bottom"))
grid::pushViewport(vp)
grid_add_rects(layout = "poker_3x3")
grid::popViewport()
```

layout_grid

Layout data frame for a grid of identically sized components

Description

layout_grid() calculates a layout data frame for a grid of identically sized print-and-play components.

Usage

```
layout_grid(
  nrow = 2L,
  ncol = 1L,
  width = 2.5,
  height = 3.5,
  bleed = 0,
  paper = getOption("papersize", "letter"),
  orientation = c("landscape", "portrait"),
  angle = 0,
  name = layout_name_fn(),
  direction = "left-to-right"
)
```

Arguments

nrow, ncol	Number of rows and columns of print-and-play components (e.g. cards)
width, height, bleed	Width, height, and bleed for each print-and-play component (in inches)
paper, orientation	Print-and-play paper size and orientation
angle	Vector of angles of the pieces in degrees (counter-clockwise) (assigned direction, top-to-bottom)
name	Either a vector of names of each piece or a function with arguments nrow and ncol to be used to generate a vector of names for each piece. These will be assigned direction, top-to-bottom. layout_name_fn() can be used to generate such a function.
direction	Should name and angle be assigned "left-to-right" or "right-to-left". Also supports aliases "ltr", "lr", "rtl", and "rl".

Value

A data frame with columns "row", "col", "x", "y", "angle", "width", "height", "bleed", "paper", "orientation", and "name".

Examples

```
# Button Shy card layout
layout_grid(nrow = 2L, ncol = 3L, height = 3.447, width = 2.469, bleed = 0.125)
```

layout_name_fn	<i>Function to generate layout names</i>
----------------	--

Description

layout_name_fn() returns a function that takes nrow and ncol arguments and generates a character vector of nrow * ncol names. Intended for use with [layout_grid\(\)](#).

Usage

```
layout_name_fn(prefix = "piece.", from = 1L, width = 1L, flag = "0")
```

Arguments

prefix	Prefix for the generated names.
from	Starting number for the generated name suffix.
width, flag	Passed to formatC() .

Value

A function with arguments nrow and ncol that would return a character vector of length nrow * ncol.

Examples

```
layout_name_fn()(nrow = 3, ncol = 3)
layout_name_fn("back_", from = 97, width = 3L)(nrow = 2, ncol = 3)
```

layout_octavo	<i>Layout data frame for an octavo booklet signature</i>
---------------	--

Description

layout_octavo() calculates a layout data frame for one side of a 16-page octavo booklet signature using a standard sheetwise imposition.

Usage

```
layout_octavo(
  width = 2.5,
  height = 3.5,
  page = 1,
  signature = 1,
  bolt_padding = 0,
  ...
)
```

Arguments

<code>width, height</code>	Width and height of each (booklet) page (in inches). Defaults to poker-card size (2.5" by 3.5").
<code>page</code>	Which side of the printed sheet: 1 (front) or 2 (back).
<code>signature</code>	Which 16-page signature of a larger book this represents. Each increment of signature adds 16 to the <code>page_#</code> labels.
<code>bolt_padding</code>	Extra space (in inches) inserted at the folds that will be trimmed open (the bolts): between the two rows, and between pages that aren't part of the same uncut two-page spread. Defaults to 0 (no extra space).
<code>...</code>	Passed to <code>layout_grid()</code>

Details

An octavo signature is created by printing both page sides of a sheet (duplex, flipped on the short edge) and then folding the sheet in half three times, giving 8 leaves (16 pages). `signature` refers to which 16-page signature of a larger book is being laid out; gathering multiple signatures together (each incrementing signature by one) builds up a full book. Once printed and folded, the head (top and bottom of every page) and the fore-edge (the right side of odd pages and left side of even pages) are folded shut and need to be trimmed open; adjacent pages that instead form an uncut two-page spread (e.g. an even page and the odd page to its right) are not cut. `bolt_padding` inserts extra space at these folds that will be trimmed open (and nowhere else) so there is deliberately blank material for the trim to remove, without disturbing the flush join between the pages of an uncut spread.

Value

A data frame with columns "row", "col", "x", "y", "angle", "width", "height", "bleed", "paper", "orientation", and "name".

Examples

```
layout_octavo()
layout_octavo(page = 2)
layout_octavo(signature = 2)
layout_octavo(bolt_padding = 0.25)
```

layout_preset	<i>Layout data frame for a named preset</i>
---------------	---

Description

layout_preset() calculates a layout data frame for a named preset. layout_names() returns the supported layout presets.

Usage

```
layout_preset(preset = "button_shy_cards", ...)
```

```
layout_names()
```

Arguments

preset	Preset name. Must be in layout_names().
...	Passed to layout_grid()

Details

Unless otherwise indicated assumes letter-sized paper. Supports the following presets:

button_shy_cards **Button Shy Games** PnP cards. **Caveat:** matches many but not **all** of their PnP files e.g. ROVE-PNP-V2.pdf better matches the poker_3x2_bleed layout.

button_shy_rules **Button Shy Games** PnP rule booklet pages (8-page booklet).

button_shy_rules_2x2 **Button Shy Games** PnP rule booklet pages (4-page booklet).

poker_3x2_bleed Poker-sized cards (2.5" by 3.5") in 3 columns of 2 cards (landscape) with an 1/8" bleed around each card. Examples of PnP games using this layout include **Galdor's Grip**.

poker_3x3 Poker-sized cards (2.5" by 3.5") in 3 columns of 3 cards (portrait) with an zero bleed around each card. Examples of PnP games using this layout include the original **Mini Rogue**.

poker_4x2 Poker-sized cards (2.5" by 3.5") in 4 columns of 2 cards (landscape) with zero bleed around each card. Examples of PnP games using this layout include the **Decktet**.

4x6_jacket A single 4x6 photo storage box jacket as created by [pdf_create_4x6_jacket\(\)](#). The width covers the full jacket (back + spine + front).

poker_jacket_1x1 A single poker deck storage box jacket as created by [pdf_create_poker_jacket\(\)](#). The width covers the full jacket (back + spine + front).

poker_jacket_1x2 Two poker deck storage box jackets as created by [pdf_create_poker_jacket\(\)](#) when passed a list of two items. Uses portrait paper. The bleed column is zero (the gap between jackets is encoded in the y positions).

Value

A data frame with columns "row", "col", "x", "y", "angle", "width", "height", "bleed", "paper", "orientation", and "name".

Examples

```
layout_preset("button_shy_cards")
layout_names()
```

ls_temp_pdfs	<i>List and remove temporary pdfs</i>
--------------	---------------------------------------

Description

Using the functions in this package can lead to several pdf files cluttering your `tempdir()`. `ls_temp_pdfs()` lists them while `rm_temp_pdfs()` removes them.

Usage

```
ls_temp_pdfs(exclude = character())

rm_temp_pdfs(exclude = character())
```

Arguments

`exclude` A character vector of filenames to exclude.

Value

`ls_temp_pdfs()` returns a character vector. `rm_temp_pdfs()` returns `invisible(NULL)`.

Examples

```
## Not run: # May delete user pdf files in `tempdir()`
f1 <- pdf_create_blank()
f2 <- pdf_compress(f1)
f3 <- pdf_subset(f2, pages = 1L)
ls_temp_pdfs()
rm_temp_pdfs()
ls_temp_pdfs()

## End(Not run)
```

pdf_add_cropmarks	<i>Add crop marks to a pdf</i>
-------------------	--------------------------------

Description

pdf_add_cropmarks() adds crop marks to the edges of components of a print-and-play layout.

Usage

```
pdf_add_cropmarks(
  input,
  output = NULL,
  ...,
  layout = "poker_3x3",
  gp = gpar(),
  cm_width = unit(0.25, "mm"),
  cm_length = unit(0.125, "in")
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_add_overlay() .
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
gp, cm_width, cm_length	Passed to piecepackr::grid.cropmark() .

Value

output pdf file name invisibly. As a side effect adds crop marks to a pdf.

See Also

[grid_add_cropmarks\(\)](#), [piecepackr::grid.cropmark\(\)](#)

Examples

```
if (pnpmisc:::donttest() &&
    requireNamespace("piecepackr", quietly = TRUE)) {
  input <- pdf_create_blank(length = 2L, width = 11, height = 8.5)
  output <- pdf_add_cropmarks(input, pages = "odd",
                             layout = "button_shy_cards")

  unlink(input)
  unlink(output)
}
```

pdf_add_crosshairs *Add crosshairs to a pdf*

Description

pdf_add_crosshairs() adds crosshairs to the corners of components of a print-and-play layout.

Usage

```
pdf_add_crosshairs(
  input,
  output = NULL,
  ...,
  layout = "button_shy_cards",
  gp = gpar(),
  ch_width = unit(1/6, "in"),
  ch_grob = NULL
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Passed to pdf_add_overlay() .
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
gp, ch_width, ch_grob	Passed to piecepackr::grid.crosshair() . ch_grob defaults to piecepackr::squaresCrosshairGro

Details

- The default layout supports Button Shy games.

Value

output pdf file name invisibly. As a side effect adds crosshairs to a pdf.

Examples

```
if (pnpmisc:::donttest() &&
    requireNamespace("piecepackr", quietly = TRUE) &&
    utils::packageVersion("piecepackr") >= "1.14.0-5") {
  input <- pdf_create_blank(length = 2L, width = 11, height = 8.5)
  output <- pdf_add_crosshairs(input, pages = "odd",
                              layout = "button_shy_cards")

  unlink(input)
  unlink(output)
}
```

pdf_add_lines	<i>Add lines to a pdf</i>
---------------	---------------------------

Description

pdf_add_lines() adds lines along the components of a print-and-play layout.

Usage

```
pdf_add_lines(input, output = NULL, ..., layout = "poker_3x3", gp = gpar())
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_add_overlay() .
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
gp	Passed to grid::grid.segments() .

Value

output pdf file name invisibly. As a side effect creates pdf file with added line segments.

See Also

[grid_add_lines\(\)](#), [pdf_add_rects\(\)](#)

Examples

```
f1 <- pdf_create_blank(length = 2L, paper = "letter")
f2 <- pdf_add_lines(f1, layout = "poker_3x3")
unlink(f1)
unlink(f2)
```

pdf_add_origami	<i>Add origami symbols to a pdf</i>
-----------------	-------------------------------------

Description

pdf_add_origami() adds origami symbols to the pdf. Currently only supports adding origami symbols to [Boardgame Barrio's Small Board Game Jackets](#).

Usage

```
pdf_add_origami(input, output = NULL, ...)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Passed to pdf_add_overlay() .

Value

output pdf file name invisibly. As a side effect creates pdf file with added origami symbols.

Examples

```
f1 <- pnpmisc::pdf_create_mock_sbgj()
f2 <- pdf_add_origami(f1)

unlink(f1)
unlink(f2)
```

pdf_add_overlay	<i>Overlay custom graphics on a pdf</i>
-----------------	---

Description

`pdf_add_overlay()` calls `grid_fn` on selected pages to draw custom graphics on top.

Usage

```
pdf_add_overlay(input, output = NULL, ..., grid_fn = grid::grid.null)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Passed to pdf_apply() .
grid_fn	A zero-argument function called to draw graphics on selected pages. Defaults to grid::grid.null() .

Value

output pdf file name invisibly. As a side effect creates a pdf with overlaid graphics.

See Also

[pdf_apply\(\)](#) which this function wraps and [pdf_add_cropmarks\(\)](#), [pdf_add_crosshairs\(\)](#), [pdf_add_lines\(\)](#), and [pdf_add_rects\(\)](#) for specific applications. [pdf_overlay_stamp\(\)](#) alternatively adds the contents of a stamp pdf file.

Examples

```
f1 <- pdf_create_blank(length = 2L, paper = "letter")
f2 <- pdf_add_overlay(f1, pages = "all", dpi = 75,
  grid_fn = function() {
    grid::grid.text("DRAFT", gp = grid::gpar(col = "red",
      fontsize = 72, alpha = 0.3))
  })
unlink(f1)
unlink(f2)
```

pdf_add_rects	<i>Add (rounded) rectangles to a pdf</i>
---------------	--

Description

pdf_add_rects() adds (rounded) rectangles around components of a print-and-play layout.

Usage

```
pdf_add_rects(
  input,
  output = NULL,
  ...,
  layout = "poker_3x3",
  r = unit(0, "in"),
  gp = gpar(col = "black", fill = NA, lwd = 1)
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_add_overlay() .
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).
r, gp	Passed to grid::grid.roundrect() .

Details

- Sometimes if you use the same color as a solid background color this can be used to effectively "remove" unwanted card outlines.

Value

output pdf file name invisibly. As a side effect creates pdf file with added rectangles.

See Also

`grid_add_rects()`, `grid::grid.roundrect()`, `pdf_add_lines()`

Examples

```
f1 <- pdf_create_blank(length = 2L, paper = "letter")
f2 <- pdf_add_rects(f1, layout = "poker_3x3")
# "Remove" unwanted card border lines by covering up with white
f3 <- pdf_add_rects(f2, layout = "poker_3x3",
                    gp = grid::gpar(col = "white", fill = NA, lwd = 2))
unlink(f1)
unlink(f2)
unlink(f3)
```

pdf_append_blank	<i>Append blank pages to a pdf</i>
------------------	------------------------------------

Description

`pdf_append_blank()` appends blank pages to a pdf.

Usage

```
pdf_append_blank(input, output = NULL, ..., minimum = 1L, multiples_of = 1L)
```

Arguments

<code>input</code>	Input pdf filename.
<code>output</code>	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
<code>...</code>	Ignored.
<code>minimum</code>	Final number of pages should be at least this integer.
<code>multiples_of</code>	Final number of pages should be a multiple of this integer.

Examples

```
f1 <- pdf_create_blank(length = 1L)
f2 <- pdf_append_blank(f1, multiples_of = 4L)
qpdf::pdf_length(f2)

# Clean up
unlink(f1)
unlink(f2)
```

pdf_apply

*Modify selected pages on a pdf***Description**

pdf_apply() renders each page of a pdf as a raster image and on selected pages additionally calls bm_fn() to modify those raster images as well as grid_fn overlay custom graphics on top.

Usage

```
pdf_apply(
  input,
  output = NULL,
  ...,
  pages = "all",
  rasterize = rasterise,
  dpi = getOption("pnpmisc.dpi", 300),
  paper = NULL,
  bg = "white",
  scale = 1,
  bm_fn = identity,
  grid_fn = grid::grid.null,
  rasterise = NULL
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.
pages	A positive numeric vector of pages to include, a negative numeric vector of pages to exclude, or a string: all Include all the pages. even Include just the even pages. odd Include just the odd pages. The functions pdf_pages() and pdf_subset() also support the string: 2-up saddle stitch The order of the pages to create a saddle-stitch booklet if printing 2-up. interleave Interleave the first and second halves of the pages. Requires an even number of pages. interleave_first Repeat the first page $n - 1$ times and interleave with the remaining $n - 1$ pages. interleave_last Repeat the last page $n - 1$ times and interleave with the first $n - 1$ pages.

rasterize, rasterise

If TRUE rasterize the original content using `pdftools::pdf_render_page()`. If FALSE don't rasterize the original content and throw an error if a requested feature can not yet be implemented without rasterization. If NULL only rasterize if a requested feature requires it. Currently requires rasterization in the following cases:

1. (!is.null(paper) or scale != 1) and ghostscript is unavailable (see `pdf_gs()`)
2. any(pages != "all") and !identical(grid_fn, grid::grid.null)
3. !identical(bm_fn, identity)

rasterise is an alias for rasterize.

dpi

Dots per inch. Passed to `pdftools::pdf_render_page()` when we rasterize the original content. Defaults to `getOption("pngmisc.dpi", 300)`.

paper

If NULL (default) the output pdf will be the same size as the input pdf. Otherwise the output pdf will be created with this paper size (see `pdf_pad_paper()` for supported values but most commonly "letter" or "a4").

bg

output pdf background color.

scale

A numeric scalar to scale the content by. Values less than 1 shrink the content within the page; values greater than 1 enlarge it. By default the original content is **not** rescaled. Note this also adjusts the size of the `grid::viewport()` the `grid_fn` will draw in.

bm_fn

A one-argument function called to edit the selected pages rasterized. The input will be a "raster" object and the output must be an object coercible to a "raster" object by `grDevices::as.raster()`

grid_fn

A zero-argument function called to draw graphics on selected pages. Defaults to `grid::grid.null()`.

Details

- The original pdf document may be rasterized depending on the value of `bm_fn`, `pages`, `paper`, `scale`, and/or `rasterize` as well as whether ghostscript is available (see `pdf_gs()`).

Value

output pdf file name invisibly. As a side effect creates a pdf with overlaid graphics.

Examples

```
f1 <- pdf_create_blank(length = 2L, paper = "letter", bg = "magenta")
if (requireNamespace("bittermelon", quietly = TRUE)) {
  f2 <- pdf_apply(f1, pages = "even", dpi = 75,
    bm_fn = bittermelon::bm_invert,
    grid_fn = function() {
      grid::grid.text("DRAFT", gp = grid::gpar(col = "red",
        fontsize = 72, alpha = 0.3))
    })
  unlink(f2)
}
unlink(f1)
```

pdf_change_paper	<i>Change pdf paper size</i>
------------------	------------------------------

Description

pdf_change_paper() changes the paper size of a pdf, centering the original contents on the new paper size. pdf_pad_paper() is an alias semantically intended for padding an undersized pdf document to a normal paper size (e.g. padding a pdf smaller than letter to letter size).

Usage

```
pdf_change_paper(
  input,
  output = NULL,
  ...,
  paper = getOption("papersize", "letter")
)

pdf_pad_paper(
  input,
  output = NULL,
  ...,
  paper = getOption("papersize", "letter")
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_apply() .
paper	Paper size. Usually either "letter" or "a4" but in certain circumstances can be one of "letter", "a4", "poker", "bridge", "legal", "a5", "a3", "special".

Details

By default these function do **not** scale the original contents (but you can pass on a scale argument in the ... to [pdf_apply\(\)](#)).

To convert a document between A4 and letter paper without losing any of its original safe zone, scale by the ratio of the binding dimension (the dimension that shrinks) rather than leaving scale at its default of 1:

A4 to letter Height is the binding dimension (letter is shorter than A4), so use scale = 11 / (297 / 25.4) (about 0.94).

Letter to A4 Width is the binding dimension (A4 is narrower than letter), so use scale = (210 / 25.4) / 8.5 (about 0.97).

Value

output pdf file name invisibly. As a side effect creates a pdf with the new paper size.

Examples

```
# Some PnP files' size is the intersection of A4/letter page sizes
# i.e. shorter than A4 and narrower than letter.
# We usually want pad to full A4 or letter page size.
input <- tempfile(fileext = ".pdf")
grDevices::pdf(input, width = 8.3, height = 11, bg = "blue")
grid::grid.text("")
invisible(grDevices::dev.off())

pdf_width(input)
pdf_height(input)

output <- pdf_change_paper(input, paper = "letter")
pdf_width(output)
pdf_height(output)
unlink(output)

output_a4 <- pdf_pad_paper(input, paper = "a4")
pdf_width(output_a4)
pdf_height(output_a4)
unlink(output_a4)

unlink(input)

# Convert an A4 document to letter, preserving its safe zone
input_a4 <- pdf_create_blank(paper = "a4", bg = "blue")
output_letter <- pdf_change_paper(
  input_a4,
  paper = "letter",
  scale = 11 / (297 / 25.4)
)
pdf_width(output_letter)
pdf_height(output_letter)
unlink(input_a4)
unlink(output_letter)

# Convert a letter document to A4, preserving its safe zone
input_letter <- pdf_create_blank(paper = "letter", bg = "blue")
output_a4_safe <- pdf_change_paper(
  input_letter,
  paper = "a4",
  scale = (210 / 25.4) / 8.5
)
pdf_width(output_a4_safe)
pdf_height(output_a4_safe)
unlink(input_letter)
unlink(output_a4_safe)
```

pdf_clean

*Copies pdf file while removing temporary pdf files***Description**

pdf_clean() copies input pdf file to output and removes temporary pdf files with rm_temp_pdfs(exclude = output).

Usage

```
pdf_clean(input, output = NULL, ...)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.

Value

output pdf file name invisibly. As a side effect copies input to output and removes temporary pdf files.

Examples

```
## Not run: # May delete user files in `tempdir()`
f1 <- pdf_create_blank()
f2 <- pdf_compress(f1)
f3 <- pdf_subset(f2, pages = 1L)
ls_temp_pdfs()
pdf_clean(f3, "output.pdf")
ls_temp_pdfs()
unlink("output.pdf")

## End(Not run)
```

pdf_combine

*Wrappers around qpdf functions***Description**

These functions wrap around the utilities in the qpdf package but alters the arguments to match this package's conventions: the first two arguments must be input and output (which by default tempfile(fileext = ".pdf")), all other arguments must be named, and any pages argument is first processed by [pdf_pages\(\)](#) (so you can also use strings like "even" or "odd").

Usage

```
pdf_combine(input, output = NULL, ...)

pdf_compress(input, output = NULL, ...)

pdf_overlay_stamp(input, output = NULL, ...)

pdf_rotate_pages(input, output = NULL, ..., pages = "all")

pdf_subset(input, output = NULL, ..., pages = 1L)
```

Arguments

input	For pdf_combine(), a character vector of input pdf filenames. For all other functions, a single input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to the underlying qpdf functions.
pages	A positive numeric vector of pages to include, a negative numeric vector of pages to exclude, or a string: all Include all the pages. even Include just the even pages. odd Include just the odd pages. The functions <code>pdf_pages()</code> and <code>pdf_subset()</code> also support the string: 2-up saddle stitch The order of the pages to create a saddle-stitch booklet if printing 2-up. interleave Interleave the first and second halves of the pages. Requires an even number of pages. interleave_first Repeat the first page $n - 1$ times and interleave with the remaining $n - 1$ pages. interleave_last Repeat the last page $n - 1$ times and interleave with the first $n - 1$ pages.

Value

output filename of new pdf file invisibly.

See Also

`qpdf::pdf_combine()`, `qpdf::pdf_compress()`, `qpdf::pdf_overlay_stamp()`, `qpdf::pdf_rotate_pages()`, `qpdf::pdf_subset()`

Examples

```
f1 <- pdf_create_blank(width = 6, height = 4)
f2 <- pdf_create_blank(width = 4, height = 6)
f3 <- pdf_combine(c(f1, f2))
```

```

f4 <- pdf_overlay_stamp(f3, stamp = f1)
f5 <- pdf_compress(f4)
f6 <- pdf_subset(f5, pages = 1L)
f7 <- pdf_rotate_pages(f6, angle = 90)

unlink(f1)
unlink(f2)
unlink(f3)
unlink(f4)
unlink(f5)
unlink(f6)
unlink(f7)

```

pdf_create_blank *Create pdf of blank pages*

Description

pdf_create_blank() creates blank pdf pages.

Usage

```

pdf_create_blank(
  output = NULL,
  ...,
  length = 1L,
  paper = getOption("papersize", "letter"),
  orientation = c("portrait", "landscape"),
  width = 8.5,
  height = 11,
  bg = "white",
  grob = NULL
)

```

Arguments

output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.
length	Number of pages to create.
paper	Paper size. Usually either "letter" or "a4" but in certain circumstances can be one of "letter", "a4", "poker", "bridge", "legal", "a5", "a3", "special".
orientation	Either "portrait" or "landscape". Ignored if width and height are not missing.
width, height	Paper size in inches. Ignored if both are missing (in favor of paper and orientation).
bg	output pdf background color.
grob	A grid grob to draw on each page (e.g. grid::textGrob("This page intentionally left blank.")). Default NULL is to draw nothing.

Value

output pdf file name invisibly. As a side effect creates a blank pdf file.

Examples

```
f1 <- pdf_create_blank(paper = "a4", orientation = "landscape")
f2 <- pdf_create_blank(length = 4L)
unlink(f1)
unlink(f2)
```

pdf_create_jacket	<i>Create printable storage box jacket pdf</i>
-------------------	--

Description

pdf_create_jacket() creates a printable storage box jacket. pdf_create_4x6_jacket() is a wrapper targeting a 4x6 photo storage box. pdf_create_poker_jacket() is an alternative targeting one or two poker deck storage boxes. pdf_create_jacket_instructions() creates a printable sheet of instructions for making such a jacket (these instructions are repeated in the Details section below).

Usage

```
pdf_create_jacket(
  output = NULL,
  ...,
  front = NULL,
  back = NULL,
  spine = NULL,
  inner = NULL,
  paper = getOption("papersize", "letter"),
  orientation = "landscape",
  width = unit(JACKET_4x6_FRONT_WIDTH, "in"),
  height = unit(JACKET_4x6_HEIGHT, "in"),
  depth = unit(JACKET_4x6_SPINE_WIDTH, "in"),
  bg = "transparent"
)

pdf_create_4x6_jacket(
  output = NULL,
  ...,
  front = NULL,
  back = NULL,
  spine = NULL,
  inner = NULL,
  paper = getOption("papersize", "letter")
)
```

```

pdf_create_poker_jacket(
    output = NULL,
    ...,
    front = NULL,
    back = NULL,
    spine = NULL,
    inner = NULL,
    paper = getOption("papersize", "letter"),
    depth = unit(JACKET_POKER_SPINE_WIDTH, "in"),
    bg = "transparent"
)

pdf_create_jacket_instructions(
    output = NULL,
    ...,
    paper = getOption("papersize", "letter"),
    orientation = "landscape",
    style = marquee::classic_style()
)

```

Arguments

output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Ignored.
front	Fill color/pattern/gradient or grob for front cover section of jacket Will be drawn in a viewport with a width of width and a height of height. If NULL (default) we draw a template. For <code>pdf_create_poker_jacket()</code> may also be a list with one or two of these values (in which case will generate two jackets).
back	Fill color/pattern/gradient or grob for back cover section of jacket Will be drawn in a viewport with a width of width and a height of height. If NULL (default) we draw a template. For <code>pdf_create_poker_jacket()</code> may also be a list with one or two of these values (in which case will generate two jackets).
spine	Fill color/pattern/gradient or grob for spine section of jacket Will be drawn in a rotated viewport with a width of height and a height of depth. If NULL (default) we draw a template. For <code>pdf_create_poker_jacket()</code> may also be a list with one or two of these values (in which case will generate two jackets).
inner	Fill color/pattern/gradient or grob for inner section of jacket. Will be drawn in a viewport with a width of $2 * \text{width} + \text{depth}$ inches and a height of height inches. If NULL (default) we do not create page for inner section. For <code>pdf_create_poker_jacket()</code> may also be a list with one or two of these values (in which case will generate two jackets).
paper	Paper size. Usually either "letter" or "a4" but in certain circumstances can be one of "letter", "a4", "poker", "bridge", "legal", "a5", "a3", "special".
orientation	Either "portrait" or "landscape". Ignored if width and height are not missing.
width	Width of the jacket face as a grid unit. For a 4x6 photo storage box a good value is 4.139 inches and for a poker deck storage box a good value is 2.6 inches.

height	Height of the jacket face as a grid unit. For a 4x6 photo storage box a good value is 6.141 inches and for a poker deck storage box a good value is 3.6 inches.
depth	Width of the spine as a grid unit. For a 4x6 photo storage box a good value is 1.052 inches and for a poker deck storage box a good value is 0.78 inches.
bg	Background fill for the storage jacket. Passed to <code>grid.full()</code> .
style	A style set such as <code>marquee::classic_style()</code> to be passed to <code>marquee::marquee_grob()</code> .

Details

To make the storage box jacket from the pdf file:

1. Print the jacket out (ideally on cardstock).
 - Print "actual size" (100%).
 - To include the jacket inside (if any) print double-sided.
2. Use the crop marks and origami symbols to cut and fold the jacket.
 - Exact procedure depends on your tools and preferred methods.
 - Example procedure #1:
 - (a) Use the origami mountain fold lines to score the fold lines (e.g. using a ruler and a penknife).
 - (b) Use the crop marks to cut the edges off.
 - (c) Make two mountain folds using the scored lines along the spine.
 - Example procedure #2:
 - (a) Use the crop marks to cut away the left and right edge.
 - (b) Make a *valley* fold by touching the origami dots symbols along the top edge together and then flatten the paper to make the fold and then unfold flat. Repeat this procedure with the origami dot symbols along the bottom edge.
 - (c) Cut away the top and bottom edges (you may use the ends of the origami mountain fold lines as well as the edges of the jacket as guide to where to make these cuts).
 - (d) Make two *mountain* folds by reversing the (valley) creases.
3. Insert the jacket into the storage box.

See Also

`pdf_add_origami()` to add origami symbols to pre-existing **Boardgame Barrio's Small Board Game Jackets**.

Examples

```
# Template `front`, `back`, and `spine`
if (requireNamespace("piecepackr", quietly = TRUE)) {
  f1 <- pdf_create_4x6_jacket()
  unlink(f1)
}

# Fill `front`, `back`, and `spine`
if (requireNamespace("piecepackr", quietly = TRUE)) {
```

```

f2 <- pdf_create_4x6_jacket(front = "red", back = "blue", spine = "purple")
unlink(f2)
}

# Grob `front`, `back`, and `spine`
if (requireNamespace("piecepackr", quietly = TRUE) &&
    require("gridpattern", quietly = TRUE)) {
  pal <- grDevices::palette()
  herringbone <- patternGrob("polygon_tiling", type = "herringbone",
    fill = pal[2], spacing = 0.1)
  rhombille <- patternGrob("polygon_tiling", type = "rhombille",
    fill = pal[3], spacing = 0.3)
  pythagorean <- patternGrob("polygon_tiling", type = "pythagorean",
    fill = pal[4], spacing = 0.1)
  f3 <- pdf_create_4x6_jacket(front = herringbone, back = pythagorean,
    spine = rhombille)
  unlink(f3)
}

```

pdf_create_wallet

*Create print-and-play card wallet pdf***Description**

pdf_create_wallet() creates print-and-play card origami wallets pdfs.

Usage

```

pdf_create_wallet(
  output = NULL,
  ...,
  front = NULL,
  back = NULL,
  spine = NULL,
  bleed = unit(0, "in"),
  paper = getOption("papersize", "letter")
)

```

Arguments

output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.
front	Fill color/pattern/gradient or grob for front cover section of wallet. Will be drawn in a rotated masked viewport with a width of $2 + 2 * \text{bleed}$ inches and a height of $8 + 2 * \text{bleed}$ inches. If NULL (default) we draw a template.
back	Fill color/pattern/gradient or grob for back cover section of wallet. Will be drawn in a rotated masked viewport with a width of $2 * 2 * \text{bleed}$ inches and a height of $8 + 2 * \text{bleed}$ inches. If NULL (default) we draw a template.

spine	Fill color/pattern/gradient or grob for spine section of wallet. Will be drawn in a masked viewport with a width of $8 + 2 * \text{bleed}$ inches and a height of $6 + 2 * \text{bleed}$ inches. If NULL (default) we draw a template.
bleed	Bleed zone size to assume: • If bleed is a <code>grid::unit()</code> simply use it • If bleed is numeric then convert via <code>grid::unit(bleed, "in")</code> • If bleed is TRUE assume 1/8 inch bleed zone size • If bleed is FALSE assume 0 inch bleed zone size
paper	Paper size. Usually either "letter" or "a4" but in certain circumstances can be one of "letter", "a4", "poker", "bridge", "legal", "a5", "a3", "special".

Details

To make the wallets from the pdf files:

1. Print out two-sided flipping on the **long** edge
2. Using the crop marks on the first page (the page showing the "spine") trim the edges with a cutting tool (the four dots should now be at the corner of each page).
3. Make the four corner valley folds each labeled "Folds 1"
4. Make the two valley folds each labeled "Folds 2"
5. Flip over the paper and make the two valley folds each labeled "Folds 3"
6. Fold in half (valley fold)

Value

output pdf file name invisibly. As a side effect creates a print-and-play card wallet pdf file.

Examples

```
# Template `front`, `back`, and `spine`
if (requireNamespace("piecepackr", quietly = TRUE) &&
    getRversion() >= "4.1") {
  f1 <- pdf_create_wallet()
  unlink(f1)
}

# Fill `front`, `back`, and `spine`
if (requireNamespace("piecepackr", quietly = TRUE) &&
    getRversion() >= "4.1") {
  f2 <- pdf_create_wallet(front = "red", back = "blue", spine = "purple")
  unlink(f2)
}

# Grob `front`, `back`, and `spine` with bleed
if (requireNamespace("piecepackr", quietly = TRUE) &&
    getRversion() >= "4.1" &&
    require("gridpattern", quietly = TRUE)) {
  pal <- grDevices::palette()
```

```

herringbone <- patternGrob("polygon_tiling", type = "herringbone",
                           fill = pal[2], spacing = 0.1)
rhombille <- patternGrob("polygon_tiling", type = "rhombille",
                         fill = pal[3], spacing = 0.2)
pythagorean <- patternGrob("polygon_tiling", type = "pythagorean",
                           fill = pal[4], spacing = 0.1)
f3 <- pdf_create_wallet(front = herringbone, back = rhombille,
                       spine = pythagorean, bleed = TRUE)
unlink(f3)
}

```

pdf_gs

*Process the pdf file with ghostscript***Description**

pdf_gs() processes the pdf file with ghostscript. This may prevent issues with other pdf processing functions like [pdftools::pdf_pagesize\(\)](#).

Usage

```
pdf_gs(input, output = NULL, ..., args = character(0L))
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.
args	Arguments to pass to ghostscript. We automatically add -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -sAutoRotatePages=None -sOutputFile={output}.

See Also

[tools::find_gs_cmd\(\)](#) and <https://ghostscript.readthedocs.io/en/latest/>

Examples

```

if (nzchar(tools::find_gs_cmd())) {
  f1 <- pdf_create_blank()
  f2 <- pdf_gs(f1)

  unlink(f1)
  unlink(f2)
}

```

pdf_orientation	<i>Tell whether pdf is in portrait or landscape mode</i>
-----------------	--

Description

pdf_orientation() tells whether a pdf is in portrait or landscape mode.

Usage

```
pdf_orientation(input, ...)
```

Arguments

input	Input pdf filename.
...	Ignored.

Value

A character vector with a length equal to the number of pages in input.

Examples

```
f1 <- pdf_create_blank(width = 8.5, height = 11)
pdf_orientation(f1)

f2 <- pdf_create_blank(width = 11, height = 8.5, length = 2L)
pdf_orientation(f2)

unlink(f1)
unlink(f2)
```

pdf_pages	<i>Get integer vector of subset of pdf pages</i>
-----------	--

Description

pdf_pages() calculates an integer vector of subset of pdf pages.

Usage

```
pdf_pages(
  input,
  ...,
  pages = c("all", "even", "odd", "2-up saddle stitch", "interleave", "interleave_first",
    "interleave_last")
)
```

Arguments

input	Input pdf filename.
...	Ignored.
pages	A positive numeric vector of pages to include, a negative numeric vector of pages to exclude, or a string: all Include all the pages. even Include just the even pages. odd Include just the odd pages. The functions <code>pdf_pages()</code> and <code>pdf_subset()</code> also support the string: 2-up saddle stitch The order of the pages to create a saddle-stitch booklet if printing 2-up. interleave Interleave the first and second halves of the pages. Requires an even number of pages. interleave_first Repeat the first page $n - 1$ times and interleave with the remaining $n - 1$ pages. interleave_last Repeat the last page $n - 1$ times and interleave with the first $n - 1$ pages.

Value

An integer vector.

Examples

```
f <- pdf_create_blank(length = 8L)
pdf_pages(f, pages = 1:4)
pdf_pages(f, pages = -(1:4))
pdf_pages(f, pages = "all")
pdf_pages(f, pages = "even")
pdf_pages(f, pages = "odd")
pdf_pages(f, pages = "interleave")
pdf_pages(f, pages = "interleave_first")
pdf_pages(f, pages = "interleave_last")

unlink(f) # Clean up
```

pdf_paper

Infer the paper size of a PDF document

Description

`pdf_paper()` infers the paper size of a PDF document

Usage

```
pdf_paper(input)
```

Arguments

input Input pdf filename.

Value

A character vector.

Examples

```
f <- pdf_create_blank(width = 8.5, height = 11)
pdf_paper(f)
unlink(f)
```

pdf_pdfxup	<i>N-up pages of a document with pdfxup</i>
------------	---

Description

pdf_pdfxup() creates an n-up pdf using the pdfxup command-line program. pdf_pdfxup_booklet() is a convenience wrapper around pdf_pdfxup() with opinionated defaults better suited for printing a physical booklet on a home printer. pdfxup computes the bounding box of the input to remove margins, keeping text readable.

Usage

```
pdf_pdfxup(
  input,
  output = NULL,
  ...,
  ncol = 2L,
  nrow = 1L,
  byrow = TRUE,
  booklet = FALSE,
  orientation = "landscape",
  paper = getOption("papersize", "letter"),
  margins = unit(5, "pt"),
  intspaces = unit(1, "pt"),
  innermargins = unit(5, "pt"),
  framewidth = ifelse(isFALSE(booklet), unit(0.4, "pt"), unit(0, "pt")),
  default.units = "pt",
  clip = TRUE,
  pages = "all",
  bb_pages = "all"
)

pdf_pdfxup_booklet(
  input,
```

```

    output = NULL,
    ...,
    booklet = "se",
    margins = unit(0.375, "in"),
    intspaces = unit(0.1875, "in"),
    innermargins = unit(0, "pt")
)

```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Ignored.
ncol, nrow	Number of columns/rows per page.
byrow	If TRUE fill pages row by row. If FALSE fill pages column by column.
booklet	If TRUE configure for printing as a 2-up booklet. Can also be "se" (short-edge mode) or "le" (long-edge mode) for explicit control over two-sided printing; booklet = TRUE defaults to "le" mode.
orientation	Either "landscape" or "portrait".
paper	Paper size (e.g. "letter" or "a4"). The name must be known by the LaTeX geometry package.
margins	Margin of pages of the new document as a <code>grid::unit()</code> object.
intspaces	Space between pages as a <code>grid::unit()</code> object.
innermargins	Inner margin between frame and page as a <code>grid::unit()</code> object.
framewidth	Width of the frame around each page as a <code>grid::unit()</code> object.
default.units	Default units when a numeric value is passed for margins, intspaces, innermargins, or framewidth.
clip	If TRUE clip pages to the computed bounding box.
pages	A positive numeric vector of pages to include, a negative numeric vector of pages to exclude, or a string: all Include all the pages. even Include just the even pages. odd Include just the odd pages. The functions <code>pdf_pages()</code> and <code>pdf_subset()</code> also support the string: 2-up saddle stitch The order of the pages to create a saddle-stitch booklet if printing 2-up. interleave Interleave the first and second halves of the pages. Requires an even number of pages. interleave_first Repeat the first page $n - 1$ times and interleave with the remaining $n - 1$ pages. interleave_last Repeat the last page $n - 1$ times and interleave with the first $n - 1$ pages.
bb_pages	Pages to use when computing the bounding box (see pages argument for supported values). Defaults to "all". Note negative numbers indicate pages to omit from computation of bounding box.

Details

pdf_pdfxup() tries to closely follow the pdfxup defaults. In contrast pdf_pdfxup_booklet() makes these alternative defaults:

- booklet = "se" which I find easier to reason about than "le"
- margins = unit(3/8, "in") to provide adequate margins for home printers
- intspaces = unit(3/16, "in") to provide adequate gutter space for folding
- innermargins = unit(0, "pt")

Value

output pdf file name invisibly. As a side effect creates an n-up pdf.

See Also

<https://ctan.org/tex-archive/support/pdfxup> for more information about pdfxup

Examples

```
f1 <- system.file("doc", "Sweave.pdf", package = "utils")
if (nzchar(Sys.which("pdfxup")) && file.exists(f1)) {
  f2 <- pdf_pdfxup_booklet(f1)
  unlink(f2)
}
```

pdf_rasterize

Rasterize a pdf

Description

pdf_rasterize() re-renders a pdf by rasterizing each page.

Usage

```
pdf_rasterize(input, output = NULL, ..., dpi = getOption("pnpmisc.dpi", 300))
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_apply() .
dpi	Dots per inch. Passed to pdftools::pdf_render_page() when we rasterize the original content. Defaults to getOption("pnpmisc.dpi", 300).

Value

output pdf file name invisibly. As a side effect creates a rasterized pdf.

See Also

[pdf_apply\(\)](#) which this function wraps.

Examples

```
f1 <- pdf_create_blank(length = 2L, paper = "letter")
f2 <- pdf_rasterize(f1, dpi = 75)
unlink(f1)
unlink(f2)
```

pdf_render_bm_list	<i>Render all pages of a pdf into bittermelon pixmap objects</i>
--------------------	--

Description

`pdf_render_bm_list()` renders all pages of a pdf into bittermelon pixmap objects.

Usage

```
pdf_render_bm_list(input, ..., dpi = getOption("pnpmisc.dpi", 300))
```

Arguments

input	Input pdf filename.
...	Ignored.
dpi	Dots per inch. Passed to pdftools::pdf_render_page() when we rasterize the original content. Defaults to <code>getOption("pnpmisc.dpi", 300)</code> .

Value

A [bittermelon::bm_list\(\)](#) of [bittermelon::bm_pixmap\(\)](#) objects.

See Also

[pdf_render_bm_pixmap\(\)](#)

Examples

```
if (requireNamespace("bittermelon", quietly = TRUE)) {
  f <- pdf_create_wallet()
  bml <- pdf_render_bm_list(f, dpi = 75)
  grid::grid.raster(bml[[1L]])
  unlink(f)
}
```

pdf_render_bm_pixmap *Render a pdf page into a bittermelon pixmap object*

Description

pdf_render_bm_pixmap() renders a pdf page into a bittermelon pixmap object.

Usage

```
pdf_render_bm_pixmap(  
  input,  
  ...,  
  page = 1L,  
  dpi = getOption("pnpmisc.dpi", 300)  
)
```

Arguments

input	Input pdf filename.
...	Ignored.
page	Integer of page to render.
dpi	Dots per inch. Passed to pdftools::pdf_render_page() when we rasterize the original content. Defaults to <code>getOption("pnpmisc.dpi", 300)</code> .

Value

A [bittermelon::bm_pixmap\(\)](#) object.

See Also

[pdftools::pdf_render_page\(\)](#), [pdf_render_raster\(\)](#), [pdf_render_bm_list\(\)](#)

Examples

```
if (requireNamespace("bittermelon", quietly = TRUE)) {  
  f <- pdf_create_wallet()  
  bm <- pdf_render_bm_pixmap(f, page = 1L, dpi = 75)  
  grid::grid.raster(bm)  
  unlink(f)  
}
```

pdf_render_raster	<i>Render a pdf page into a raster object</i>
-------------------	---

Description

`pdf_render_raster()` renders a pdf page into a raster object.

Usage

```
pdf_render_raster(input, ..., page = 1L, dpi = 300, native = FALSE)
```

Arguments

<code>input</code>	Input pdf filename.
<code>...</code>	Ignored.
<code>page</code>	Integer of page to render.
<code>dpi</code>	Dots per inch. Passed to <code>pdftools::pdf_render_page()</code> when we rasterize the original content. Defaults to <code>getOption("pnmisc.dpi", 300)</code> .
<code>native</code>	If TRUE return a <code>nativeRaster</code> object else a raster object.

Value

If `native = TRUE` returns a `nativeRaster` object else a raster object.

See Also

[pdftools::pdf_render_page\(\)](#), [pdf_render_bm_pixmap\(\)](#)

Examples

```
if (requireNamespace("bittermelon", quietly = TRUE)) {  
  f <- pdf_create_wallet()  
  r <- pdf_render_raster(f, page = 1L, dpi = 75)  
  grid::grid.raster(r)  
  unlink(f)  
}
```

pdf_rep	<i>Repeat pages of a pdf</i>
---------	------------------------------

Description

pdf_rep() repeats pages of a pdf similar to [base::rep\(\)](#).

Usage

```
pdf_rep(
  input,
  output = NULL,
  ...,
  pages = "all",
  times = 1L,
  length.out = NA_integer_,
  each = 1L
)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Ignored.
pages	A positive numeric vector of pages to include, a negative numeric vector of pages to exclude, or a string: all Include all the pages. even Include just the even pages. odd Include just the odd pages. The functions pdf_pages() and pdf_subset() also support the string: 2-up saddle stitch The order of the pages to create a saddle-stitch booklet if printing 2-up. interleave Interleave the first and second halves of the pages. Requires an even number of pages. interleave_first Repeat the first page $n - 1$ times and interleave with the remaining $n - 1$ pages. interleave_last Repeat the last page $n - 1$ times and interleave with the first $n - 1$ pages.
times, length.out, each	Passed to base::rep() .

Value

output pdf file name invisibly. As a side effect creates a pdf with repeated pages.

See Also[pdf_subset\(\)](#)**Examples**

```
f1 <- pdf_create_blank(length = 3L, paper = "letter")
f2 <- pdf_rep(f1, times = 2L)
f3 <- pdf_rep(f1, pages = 1:2, each = 2L)
unlink(f1)
unlink(f2)
unlink(f3)
```

pdf_rm_crosshairs	<i>Remove crosshairs</i>
-------------------	--------------------------

Description

pdf_rm_crosshairs() removes unwanted crosshairs.

Usage

```
pdf_rm_crosshairs(input, output = NULL, ..., layout = "poker_3x2_bleed")
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to <code>tempfile(fileext = ".pdf")</code> .
...	Passed to pdf_apply() .
layout	Either a layout preset name in layout_names() or a data frame with layout data (as returned by layout_grid()).

Details

- In order to work the PnP layout needs a solid color bleed.
- The default layout supports **Galdor's Grip** (PnP letter size v1).

Value

output pdf file name invisibly. As a side effect removes from crosshairs from a pdf.

Examples

```
if (requireNamespace("bittermelon", quietly = TRUE) &&
    requireNamespace("piecepackr", quietly = TRUE) &&
    utils::packageVersion("piecepackr") >= "1.14.0-5") {
  f1 <- pdf_create_blank(length = 2L, width = 11, height = 8.5)
  f2 <- pdf_add_crosshairs(f1, pages = "all",
                          layout = "poker_3x2_bleed")
  f3 <- pdf_rm_crosshairs(f2, pages = "odd",
                          layout = "poker_3x2_bleed", dpi = 75)

  unlink(f1)
  unlink(f2)
  unlink(f3)
}
```

pdf_scale

Scale pdf contents centered within the page

Description

pdf_scale() scales the contents of a pdf centered within the page.

Usage

```
pdf_scale(input, output = NULL, ..., scale)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Passed to pdf_apply() .
scale	A numeric scalar to scale the content by. Values less than 1 shrink the content within the page; values greater than 1 enlarge it. By default the original content is not rescaled. Note this also adjusts the size of the grid::viewport() the grid_fn will draw in.

Value

output pdf file name invisibly. As a side effect creates a scaled pdf file.

Examples

```
input <- pdf_create_blank(paper = "letter", bg = "blue")
output <- pdf_scale(input, scale = 0.9, dpi = 75)
pdf_width(output)
pdf_height(output)
unlink(input)
unlink(output)
```

pdf_set_bookmarks *Wrappers around xmpdf functions*

Description

These functions wrap around the utilities in the xmpdf package but alters the arguments to match this package's conventions: the first two arguments must be input and output, by default output = tempfile(fileext = ".pdf"), and all other arguments must be named.

Usage

```
pdf_set_bookmarks(input, output = NULL, ..., bookmarks)
```

```
pdf_set_docinfo(input, output = NULL, ..., docinfo)
```

```
pdf_set_xmp(input, output = NULL, ..., xmp)
```

Arguments

input	Input pdf filename.
output	Output pdf filename. NULL defaults to tempfile(fileext = ".pdf").
...	Currently ignored.
bookmarks	See xmpdf::set_bookmarks() .
docinfo	See xmpdf::set_docinfo() .
xmp	See xmpdf::set_xmp() .

Value

output filename of new pdf file invisibly.

See Also

[xmpdf::set_bookmarks\(\)](#), [xmpdf::set_docinfo\(\)](#), [xmpdf::set_xmp\(\)](#)

Examples

```
f1 <- pdf_create_blank(length = 2L)

if (requireNamespace("xmpdf", quietly = TRUE) &&
    xmpdf::supports_set_bookmarks()) {
  bm <- data.frame(title = c("Page 1", "Page 2"), page = c(1, 2))
  f2 <- pdf_set_bookmarks(f1, bookmarks = bm)
  unlink(f2)
}

if (requireNamespace("xmpdf", quietly = TRUE) &&
    xmpdf::supports_set_docinfo()) {
```

```

di <- xmpdf::docinfo(title = "A Title", author = "The Author")
f3 <- pdf_set_docinfo(f1, docinfo = di)
unlink(f3)
}

if (requireNamespace("xmpdf", quietly = TRUE) &&
    xmpdf::supports_set_xmp()) {
  xmp <- xmpdf::xmp(title = "A Title", creator = "The Author")
  f4 <- pdf_set_xmp(f1, xmp = xmp)
  unlink(f4)
}
unlink(f1)

```

pdf_width

*Get dimensions of pdf pages***Description**

`pdf_width()` and `pdf_height()` get the dimensions of the pdf file pages.

Usage

```

pdf_width(input, ..., units = "inches", numeric = FALSE)

pdf_height(input, ..., units = "inches", numeric = FALSE)

```

Arguments

input	Input pdf filename.
...	Ignored.
units	Units to use. See <code>grid::unit()</code> .
numeric	If TRUE return numeric else a <code>grid::unit()</code> object.

Value

If numeric = TRUE a numeric vector else a `grid::unit()` object.

Examples

```

f <- pdf_create_blank(width = 8.5, height = 11)
pdf_width(f)
pdf_height(f, units = "cm")
pdf_height(f, units = "mm", numeric = TRUE)
unlink(f)

```

zip_extract_bm_pixmap *Extract/render an image from a zipfile into a bittermelon pixmap object*

Description

zip_extract_bm_pixmap() extracts/renderes an image from a zipfile into a `bittermelon::bm_pixmap()` object.

Usage

```
zip_extract_bm_pixmap(zipfile, filename, ...)
```

Arguments

zipfile	Path to a zip file
filename	Filename within zip file
...	If filename is a pdf file then passed to <code>pdf_render_bm_pixmap()</code>

Value

A `bittermelon::bm_pixmap()` object.

Examples

```
## Not run:
# "https://boardgamegeek.com/filepage/279177/packvelopes-storage-boxes"
zipfile <- "Everdeck_Packvelopes.zip"
bm <- zip_extract_bm_pixmap(zipfile, "Red and Black.pdf")

# https://boardgamegeek.com/filepage/62704/cards-in-png-format
zipfile <- "decktet_cards_PNG.zip"
bm <- zip_extract_bm_pixmap(zipfile, "PNG/a_card_cover3.png")

## End(Not run)
```

Index

as_fill(fullGrob), 7

base::rep(), 46

bittermelon::bm_list(), 6, 43

bittermelon::bm_pixmap(), 4–7, 43, 44, 51

bm_create_pdf, 3

bm_crop_layout, 4

bm_crop_layout(), 6

bm_rm_crosshairs, 5

bm_split_layout, 6

bm_split_layout(), 4

formatC(), 15

fullGrob, 7

grDevices::as.raster(), 26

grid.full(fullGrob), 7

grid.full(), 34

grid::grid.null(), 22, 26

grid::grid.roundrect(), 13, 23, 24

grid::grid.segments(), 11, 21

grid::grob(), 7

grid::pattern(), 7

grid::rasterGrob(), 7

grid::rectGrob(), 7, 8

grid::unit(), 12, 36, 41, 50

grid::viewport(), 26, 48

grid_add_cropmarks, 8

grid_add_cropmarks(), 19

grid_add_crosshairs, 9

grid_add_layout, 10

grid_add_lines, 11

grid_add_lines(), 21

grid_add_origami(grid_add_origami_bb), 12

grid_add_origami_bb, 12

grid_add_rects, 13

grid_add_rects(), 24

layout_grid, 14

layout_grid(), 4–6, 8–11, 13, 15–17, 19–21, 23, 47

layout_name_fn, 15

layout_name_fn(), 14

layout_names(layout_preset), 17

layout_names(), 4–6, 8–11, 13, 19–21, 23, 47

layout_octavo, 15

layout_preset, 17

ls_temp_pdfs, 18

marquee::classic_style(), 34

marquee::marquee_grob(), 34

pdf_add_cropmarks, 19

pdf_add_cropmarks(), 9, 22

pdf_add_crosshairs, 20

pdf_add_crosshairs(), 22

pdf_add_lines, 21

pdf_add_lines(), 22, 24

pdf_add_origami, 21

pdf_add_origami(), 12, 34

pdf_add_overlay, 22

pdf_add_overlay(), 19–23

pdf_add_rects, 23

pdf_add_rects(), 13, 21, 22

pdf_append_blank, 24

pdf_apply, 25

pdf_apply(), 22, 27, 42, 43, 47, 48

pdf_change_paper, 27

pdf_clean, 29

pdf_combine, 29

pdf_compress(pdf_combine), 29

pdf_create_4x6_jacket
(pdf_create_jacket), 32

pdf_create_4x6_jacket(), 17

pdf_create_blank, 31

pdf_create_jacket, 32

pdf_create_jacket_instructions
(pdf_create_jacket), 32

pdf_create_poker_jacket
 (pdf_create_jacket), 32
 pdf_create_poker_jacket(), 17
 pdf_create_wallet, 35
 pdf_gs, 37
 pdf_gs(), 26
 pdf_height(pdf_width), 50
 pdf_height(), 50
 pdf_orientation, 38
 pdf_overlay_stamp(pdf_combine), 29
 pdf_overlay_stamp(), 22
 pdf_pad_paper(pdf_change_paper), 27
 pdf_pad_paper(), 26
 pdf_pages, 38
 pdf_pages(), 25, 29, 30, 39, 41, 46
 pdf_paper, 39
 pdf_pdfxup, 40
 pdf_pdfxup_booklet(pdf_pdfxup), 40
 pdf_rasterize, 42
 pdf_render_bm_list, 43
 pdf_render_bm_list(), 44
 pdf_render_bm_pixmap, 44
 pdf_render_bm_pixmap(), 4, 6, 43, 45, 51
 pdf_render_raster, 45
 pdf_render_raster(), 44
 pdf_rep, 46
 pdf_rm_crosshairs, 47
 pdf_rm_crosshairs(), 5
 pdf_rotate_pages(pdf_combine), 29
 pdf_scale, 48
 pdf_set_bookmarks, 49
 pdf_set_docinfo(pdf_set_bookmarks), 49
 pdf_set_xmp(pdf_set_bookmarks), 49
 pdf_subset(pdf_combine), 29
 pdf_subset(), 25, 30, 39, 41, 46, 47
 pdf_width, 50
 pdf_width(), 50
 pdftools::pdf_pagesize(), 37
 pdftools::pdf_render_page(), 26, 42–45
 piecepackr::grid.cropmark(), 8, 9, 19
 piecepackr::grid.crosshair(), 9, 20
 piecepackr::pmap_piece(), 10
 piecepackr::pp_cfg(), 10
 piecepackr::squaresCrosshairGrob(), 9, 20

 qpdf::pdf_combine(), 30
 qpdf::pdf_compress(), 30
 qpdf::pdf_overlay_stamp(), 30

 qpdf::pdf_rotate_pages(), 30
 qpdf::pdf_subset(), 30

 rm_temp_pdfs(ls_temp_pdfs), 18

 tempdir(), 18
 tools::find_gs_cmd(), 37

 xmpdf::set_bookmarks(), 49
 xmpdf::set_docinfo(), 49
 xmpdf::set_xmp(), 49

 zip_extract_bm_pixmap, 51