Package 'bittermelon'

May 7, 2026

Type Package

Title Bitmap Tools

Version 2.3.1

Description Provides functions for creating, modifying, and displaying bitmaps including printing them in the terminal. There is a special emphasis on monochrome bitmap fonts and their glyphs as well as colored pixel art/sprites. Provides native read/write support for the 'hex' and 'yaff' bitmap font formats and if 'monobit' <https://github.com/robhagemans/monobit> is installed can also read/write several additional bitmap font formats.

URL https://trevorldavis.com/R/bittermelon/

BugReports https://github.com/trevorld/bittermelon/issues

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports cli, grDevices, grid, png, Unicode, utils

Suggests colorfast (>= 1.0.1), farver, gridpattern, hexfont (>= 0.5.1), knitr, magick, mazing, ragg, rmarkdown, testthat, vdiffr, withr

VignetteBuilder knitr, rmarkdown

SystemRequirements 'monobit' for reading/writing additional bitmap font formats.

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (ORCID: <https://orcid.org/0000-0001-6341-4639>), Frederic Cambus [tyg] (Developer of included 'Spleen' font), Markus Kuhn [tyg] (Maintainer of included 'Fixed' font), josehzz [art] (Artist of included 'Farming Crops 16x16' sprites)

Maintainer Trevor L. Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2026-03-27 17:40:08 UTC

Contents

`as.array.bm_bitmap` *Cast bitmap/pixmap objects to an array*

Description

`as.array.bm_bitmap()` / `as.array.bm_pixmap()` casts `bm_bitmap()` / `bm_pixmap()` objects to an array of numeric values representing the RGBA channels. These arrays can be used in functions such as `png::writePNG()`.

Usage

```
## S3 method for class 'bm_bitmap'
as.array(
  x,
  ...,
  first_row_is_top = TRUE,
  col = getOption("bittermelon.col", col_bitmap)
)

## S3 method for class 'bm_pixmap'
as.array(x, ..., first_row_is_top = TRUE)
```

Arguments

x	Either a bm_bitmap() or bm_pixmap() object.
...	Currently ignored.
first_row_is_top	If TRUE the first row of the matrix will represent the top of the bitmap (like grDevices::as.raster() objects). If FALSE the first row of the matrix will represent the bottom of the bitmap (like bm_bitmap() and bm_pixmap() objects).
col	Character vector of R color specifications. First color is used for values equal to 0, second color for values equal to 1, etc.

Examples

```
corn <- farming_crops_16x16()$corn$portrait
a <- as.array(corn)
f <- tempfile(fileext = ".png")
png::writePNG(a, f)
```

---

as.data.frame.bm_bitmap

Convert to data frame with pixel (x,y) coordinates

Description

as.matrix.bm_matrix() casts bm_bitmap() objects to a (normal) integer matrix and bm_pixmap() objects to a (normal) character matrix (of color strings).

Usage

```
## S3 method for class 'bm_bitmap'
as.data.frame(x, ..., filtrate = FALSE)

## S3 method for class 'bm_pixmap'
as.data.frame(x, ..., filtrate = FALSE)
```

Arguments

x	Either a bm_bitmap() or bm_pixmap() object.
...	Currently ignored.
filtrate	If FALSE (default) get coordinates for all values. If a single value only return the coordinates for pixels that equal that value.

Value

A data frame with "x", "y", and "value" columns.

Examples

```
font_file <- system.file("fonts/fixed/4x6.yaff.gz", package = "bittermelon")
font <- read_yaff(font_file)
bm <- as_bm_bitmap("RSTATS", font = font)
df <- as.data.frame(bm, filtrate = 1L)
if (require("grid")) {
  grid.newpage()
  grid.rect(df$x * 0.6, df$y * 0.6, width = 0.5, height = 0.5,
            gp = gpar(fill = 'black'), default.units = 'cm')
}

corn <- farming_crops_16x16()$corn$portrait
df <- as.data.frame(corn)
if (require("grid")) {
  grid.newpage()
  grid.circle(df$x * 0.6, df$y * 0.6, r = 0.25,
              gp = gpar(fill = df$value), default.units = 'cm')
}
```

---

as.matrix.bm_matrix *Cast bitmap/pixmap objects to a (normal) matrix*

Description

`as.matrix.bm_matrix()` casts `bm_bitmap()` objects to a (normal) integer matrix and `bm_pixmap()` objects to a (normal) character matrix (of color strings). Note unless `first_row_is_top = TRUE` the bottom left pixel will still be represented by the pixel in the first row and first column (i.e. these methods simply remove the class names).

Usage

```
## S3 method for class 'bm_matrix'
as.matrix(x, first_row_is_top = FALSE, ...)
```

Arguments

`x`	Either a `bm_bitmap()` or `bm_pixmap()` object.
`first_row_is_top`	If `TRUE` the first row of the matrix will represent the top of the bitmap (like `grDevices::as.raster()` objects). If `FALSE` the first row of the matrix will represent the bottom of the bitmap (like `bm_bitmap()` and `bm_pixmap()` objects).
`...`	Currently ignored.

Value

Either an integer matrix if `x` is a `bm_bitmap()` object or a character matrix if `x` is a `bm_pixmap()` object.

Examples

```
space_matrix <- matrix(0L, ncol = 8L, nrow = 8L)
space_glyph <- bm_bitmap(space_matrix)
print(space_glyph, px = ".")
print(as.matrix(space_glyph))
```

`as_bm_bitmap` *Cast to a bitmap matrix object*

Description

`as_bm_bitmap()` turns an existing object into a `bm_bitmap()` object.

Usage

```
as_bm_bitmap(x, ...)

## S3 method for class 'array'
as_bm_bitmap(
  x,
  ...,
  mode = c("alpha", "darkness", "brightness"),
  threshold = 0.5
)

## Default S3 method:
as_bm_bitmap(x, ...)

## S3 method for class 'bm_bitmap'
as_bm_bitmap(x, ...)

## S3 method for class 'bm_pixmap'
as_bm_bitmap(
  x,
  ...,
  mode = c("alpha", "darkness", "brightness"),
  threshold = 0.5
)

## S3 method for class 'character'
as_bm_bitmap(
  x,
  ...,
  direction = "left-to-right, top-to-bottom",
  font = bm_font(),
  hjust = "left",
  vjust = "top",
```

```
  compose = TRUE,
  pua_combining = character(0)
)

## S3 method for class 'glyph_bitmap'
as_bm_bitmap(x, ..., threshold = 0.5)

## S3 method for class 'grob'
as_bm_bitmap(
  x,
  ...,
  width = 8L,
  height = 16L,
  png_device = NULL,
  threshold = 0.25
)

## S3 method for class '`lofi-matrix`'
as_bm_bitmap(x, ...)

## S3 method for class '`magick-image`'
as_bm_bitmap(
  x,
  ...,
  mode = c("alpha", "darkness", "brightness"),
  threshold = 0.5
)

## S3 method for class 'matrix'
as_bm_bitmap(x, ...)

## S3 method for class 'maze'
as_bm_bitmap(
  x,
  ...,
  walls = FALSE,
  start = NULL,
  end = NULL,
  solve = !is.null(start) && !is.null(end)
)

## S3 method for class 'nativeRaster'
as_bm_bitmap(
  x,
  ...,
  mode = c("alpha", "darkness", "brightness"),
  threshold = 0.5
)
```

```
## S3 method for class 'pattern_square'
as_bm_bitmap(x, ...)

## S3 method for class 'pattern_weave'
as_bm_bitmap(x, ...)

## S3 method for class 'pattern_square'
as_bm_bitmap(x, ...)

## S3 method for class 'pixeltrix'
as_bm_bitmap(x, ...)

## S3 method for class 'pixmapGrey'
as_bm_bitmap(x, ..., mode = c("darkness", "brightness"), threshold = 0.5)

## S3 method for class 'pixmapIndexed'
as_bm_bitmap(x, ...)

## S3 method for class 'pixmapRGB'
as_bm_bitmap(x, ..., mode = c("darkness", "brightness"), threshold = 0.5)

## S3 method for class 'raster'
as_bm_bitmap(
  x,
  ...,
  mode = c("alpha", "darkness", "brightness"),
  threshold = 0.5
)
```

Arguments

x	An object that can reasonably be coerced to a `bm_bitmap()` object.
...	Further arguments passed to or from other methods.
mode	Method to determine the integer values of the `bm_bitmap()` object: **alpha** Higher alpha values get a `1L`. **darkness** Higher darkness values get a `1L`. `darkness = (1 - luma) * alpha`. **brightness** Higher brightness values get a `1L`. `brightness = luma * alpha`.
threshold	If the alpha/darkness/brightness value weakly exceeds this threshold (on an interval from zero to one) then the pixel is determined to be "black".
direction	For purely horizontal binding either "left-to-right" (default) or its aliases "ltr" and "lr" OR "right-to-left" or its aliases "rtl" and "rl". For purley vertical binding either "top-to-bottom" (default) or its aliases "ttb" and "tb" OR "bottom-to-top" or its aliases "btt" and "bt". For character vectors of length greater than one: for first horizontal binding within values in the vector and then vertical binding across values in the vector "left-to-right, top-to-bottom" (default) or its aliases "lrtb" and "lr-tb"; "left-to-right, bottom-to-top" or its aliases "lrbt" and "lr-bt";

	"right-to-left, top-to-bottom" or its aliases "rltb" and "rl-tb"; or "right-to-left, bottom-to-top" or its aliases "rlbt" and "rl-bt". For first vertical binding within values in the vector and then horizontal binding across values "top-to-bottom, left-to-right" or its aliases "tblr" and "tb-lr"; "top-to-bottom, right-to-left" or its aliases "tbrl" and "tb-rl"; "bottom-to-top, left-to-right" or its aliases "btlr" and "bt-lr"; or "bottom-to-top, right-to-left" or its aliases "btrl" and "bt-rl". The `direction` argument is not case-sensitive.
`font`	A `bm_font()` object that contains all the characters within `x`.
`hjust`	Used by `bm_extend()` if bitmap widths are different.
`vjust`	Used by `bm_extend()` if bitmap heights are different.
`compose`	Compose graphemes using `bm_compose()`.
`pua_combining`	Passed to `bm_compose()`.
`width`	Desired width of bitmap
`height`	Desired height of bitmap
`png_device`	A function taking arguments `filename`, `width`, and `height` that starts a graphics device that saves a png image with a transparent background. By default will use `ragg::agg_png()` if available else the "cairo" version of `grDevices::png()` if available else just `grDevices::png()`.
`walls`	If `TRUE` the values of 1L denote the walls and the values of 0L denote the paths.
`start`, `end`	If not `NULL` mark the `start` and `end` as value 2L. See `mazing::find_maze_refpoint()`.
`solve`	If `TRUE` then mark the solution path from `start` to `end` as value 3L. See `mazing::solve_maze()`.

Value

A `bm_bitmap()` object.

See Also

`bm_bitmap()`

Examples

```
space_matrix <- matrix(0L, nrow = 16L, ncol = 16L)
space_glyph <- as_bm_bitmap(space_matrix)
is_bm_bitmap(space_glyph)

font_file <- system.file("fonts/fixed/4x6.yaff.gz", package = "bittermelon")
font <- read_yaff(font_file)
bm <- as_bm_bitmap("RSTATS", font = font)
print(bm)

bm <- as_bm_bitmap("RSTATS", direction = "top-to-bottom", font = font)
print(bm)

if (require("grid") && capabilities("png")) {
  circle <- as_bm_bitmap(circleGrob(r = 0.25), width = 16L, height = 16L)
  print(circle)
```

```
}

if (require("grid") && capabilities("png")) {
  inverted_exclamation <- as_bm_bitmap(textGrob("!", rot = 180),
                                       width = 8L, height = 16L)
  print(inverted_exclamation)
}

if (requireNamespace("mazing", quietly = TRUE)) {
  m <- mazing::maze(16, 32)
  bm <- as_bm_bitmap(m, walls = TRUE)
  print(bm, compress = "vertical")
}

if (requireNamespace("gridpattern", quietly = TRUE)) {
  w <- gridpattern::pattern_weave("twill_herringbone", nrow=14L, ncol = 40L)
  bm <- as_bm_bitmap(w)
  print(bm, compress = "vertical")
}
```

---

as_bm_font *Coerce to bitmap font objects*

Description

as_bm_font() turns an existing object into a bm_font() object.

Usage

```
as_bm_font(x, ..., comments = NULL, properties = NULL)

## Default S3 method:
as_bm_font(x, ..., comments = NULL, properties = NULL)

## S3 method for class 'list'
as_bm_font(x, ..., comments = NULL, properties = NULL)
```

Arguments

x	An object that can reasonably be coerced to a bm_font() object.
...	Further arguments passed to or from other methods.
comments	An optional character vector of (global) font comments.
properties	An optional named list of font metadata.

Value

A bm_font() object.

See Also

bm_font()

Examples

```
plus_sign <- matrix(0L, nrow = 9L, ncol = 9L)
plus_sign[5L, 3:7] <- 1L
plus_sign[3:7, 5L] <- 1L
plus_sign_glyph <- bm_bitmap(plus_sign)

space_glyph <- bm_bitmap(matrix(0L, nrow = 9L, ncol = 9L))

l <- list()
l[[str2ucp("+")]] <- plus_sign_glyph
l[[str2ucp(" ")]] <- space_glyph
font <- as_bm_font(l)
is_bm_font(font)
```

---

as_bm_list *Coerce to bitmap list objects*

Description

as_bm_list() turns an existing object into a bm_list() object. In particular as_bm_list.character() turns a string into a bitmap list.

Usage

```
as_bm_list(x, ...)

## Default S3 method:
as_bm_list(x, ...)

## S3 method for class 'bm_list'
as_bm_list(x, ...)

## S3 method for class 'list'
as_bm_list(x, ..., FUN = identity)

## S3 method for class 'character'
as_bm_list(x, ..., font = bm_font())
```

Arguments

x	An object that can reasonably be coerced to a bm_list() object.
...	Further arguments passed to or from other methods.
FUN	Function to apply to every element of a list such as as_bm_bitmap() or as_bm_pixmap().
font	A bm_font() object that contains all the characters within x.

Value

A bm_list() object.

See Also

bm_list()

Examples

```
# as_bm_list.character()
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
bml <- as_bm_list("RSTATS", font = font)
bml <- bm_extend(bml, sides = 1L, value = 0L)
bml <- bm_extend(bml, sides = c(2L, 1L), value = 2L)
bm <- do.call(cbind, bml)
print(bm, px = c(" ", "#", "X"))
```

---

as_bm_pixmap *Cast to a pixmap matrix object*

Description

as_bm_pixmap() casts an object to a [bm_pixmap()] object.

Usage

```
as_bm_pixmap(x, ...)

## S3 method for class 'character'
as_bm_pixmap(
  x,
  ...,
  direction = "left-to-right, top-to-bottom",
  font = bm_font(),
  hjust = "left",
  vjust = "top",
  compose = TRUE,
  pua_combining = character(0)
)

## Default S3 method:
as_bm_pixmap(x, ...)

## S3 method for class 'array'
as_bm_pixmap(x, ...)
```

```
## S3 method for class 'bm_bitmap'
as_bm_pixmap(x, ..., col = getOption("bittermelon.col", col_bitmap))

## S3 method for class 'bm_pixmap'
as_bm_pixmap(x, ...)

## S3 method for class 'glyph_bitmap'
as_bm_pixmap(x, ..., col = getOption("bittermelon.col", col_bitmap))

## S3 method for class 'grob'
as_bm_pixmap(x, ..., width = 16L, height = 16L, png_device = NULL)

## S3 method for class '`lofi-matrix`'
as_bm_pixmap(x, ..., col = getOption("bittermelon.col", col_bitmap))

## S3 method for class '`magick-image`'
as_bm_pixmap(x, ...)

## S3 method for class 'matrix'
as_bm_pixmap(x, ...)

## S3 method for class 'maze'
as_bm_pixmap(
  x,
  ...,
  walls = FALSE,
  start = NULL,
  end = NULL,
  solve = !is.null(start) && !is.null(end),
  col = getOption("bittermelon.col", col_bitmap)
)

## S3 method for class 'pattern_square'
as_bm_pixmap(x, ..., col = getOption("bittermelon.col", col_bitmap))

## S3 method for class 'pattern_weave'
as_bm_pixmap(x, ..., col = getOption("bittermelon.col", col_bitmap))

## S3 method for class 'pixmapGrey'
as_bm_pixmap(x, ...)

## S3 method for class 'pixmapIndexed'
as_bm_pixmap(x, ...)

## S3 method for class 'pixmapRGB'
as_bm_pixmap(x, ...)

## S3 method for class 'nativeRaster'
```

```
as_bm_pixmap(x, ...)

## S3 method for class 'pixeltrix'
as_bm_pixmap(x, ...)

## S3 method for class 'raster'
as_bm_pixmap(x, ...)
```

Arguments

x	an Object
...	Potentially passed to other methods e.g. as_bm_pixmap.default() passes ... to as.raster().
direction	For purely horizontal binding either "left-to-right" (default) or its aliases "ltr" and "lr" OR "right-to-left" or its aliases "rtl" and "rl". For purley vertical binding either "top-to-bottom" (default) or its aliases "ttb" and "tb" OR "bottom-to-top" or its aliases "btt" and "bt". For character vectors of length greater than one: for first horizontal binding within values in the vector and then vertical binding across values in the vector "left-to-right, top-to-bottom" (default) or its aliases "lrtb" and "lr-tb"; "left-to-right, bottom-to-top" or its aliases "lrbt" and "lr-bt"; "right-to-left, top-to-bottom" or its aliases "rltb" and "rl-tb"; or "right-to-left, bottom-to-top" or its aliases "rlbt" and "rl-bt". For first vertical binding within values in the vector and then horizontal binding across values "top-to-bottom, left-to-right" or its aliases "tblr" and "tb-lr"; "top-to-bottom, right-to-left" or its aliases "tbrl" and "tb-rl"; "bottom-to-top, left-to-right" or its aliases "btlr" and "bt-lr"; or "bottom-to-top, right-to-left" or its aliases "btrl" and "bt-rl". The direction argument is not case-sensitive.
font	A bm_font() object that contains all the characters within x.
hjust	Used by bm_extend() if bitmap widths are different.
vjust	Used by bm_extend() if bitmap heights are different.
compose	Compose graphemes using bm_compose().
pua_combining	Passed to bm_compose().
col	Character vector of R color specifications.
width	Desired width of bitmap
height	Desired height of bitmap
png_device	A function taking arguments filename, width, and height that starts a graphics device that saves a png image with a transparent background. By default will use ragg::agg_png() if available else the "cairo" version of grDevices::png() if available else just grDevices::png().
walls	If TRUE the values of 1L denote the walls and the values of 0L denote the paths.
start, end	If not NULL mark the start and end as value 2L. See mazing::find_maze_refpoint().
solve	If TRUE then mark the solution path from start to end as value 3L. See mazing::solve_maze().

Value

A bm_pixmap() object.

See Also

bm_pixmap(), is_bm_pixmap()

Examples

```
crops <- farming_crops_16x16()
corn <- crops$corn$portrait
is_bm_pixmap(corn)
all.equal(corn, as_bm_pixmap(as.array(corn)))
all.equal(corn, as_bm_pixmap(as.raster(corn)))
if (requireNamespace("farver", quietly = TRUE)) {
  all.equal(corn, as_bm_pixmap(as.raster(corn, native = TRUE)))
}
if (requireNamespace("magick", quietly = TRUE)) {
  all.equal(corn, as_bm_pixmap(magick::image_read(corn)))
}

if (requireNamespace("mazing", quietly = TRUE) &&
    cli::is_utf8_output() &&
    cli::num_ansi_colors() >= 8L) {
  pal <- grDevices::palette.colors()
  pm <- as_bm_pixmap(mazing::maze(24L, 32L),
                     start = "top", end = "bottom",
                     col = c(pal[6], "white", pal[7], pal[5]))
  pm <- bm_pad(pm, sides = 1L)
  print(pm, compress = "v", bg = "white")
}
if (requireNamespace("gridpattern", quietly = TRUE) &&
    cli::is_utf8_output() &&
    cli::num_ansi_colors() >= 256L) {
  s <- gridpattern::pattern_square(subtype = 8L, nrow = 8L, ncol = 50L)
  pm <- as_bm_pixmap(s, col = grDevices::rainbow(8L))
  print(pm, compress = "vertical")
}
```

---

bm_bitmap *Bittermelon bitmap matrix object*

Description

bm_bitmap() creates an S3 matrix subclass representing a bitmap.

Usage

```
bm_bitmap(x)
```

Arguments

x	Object to be converted to bm_bitmap(). If not already an integer matrix it will be cast to one by as_bm_bitmap().

Details

- Intended to represent binary bitmaps especially (but not limited to) bitmap font glyphs.
- Bitmaps are represented as integer matrices with special class methods.
- The bottom left pixel is represented by the first row and first column.
- The bottom right pixel is represented by the first row and last column.
- The top left pixel is represented by the last row and first column.
- The top right pixel is represented by the last row and last column.
- Non-binary bitmaps are supported (the integer can be any non-negative integer) but we are unlikely to ever support exporting color bitmap fonts.
- Non-binary bitmaps can be cast to binary bitmaps via `bm_clamp()`.
- See `bm_pixmap()` for an alternative S3 object backed by a color string matrix.

Value

An integer matrix with "bm_bitmap" and "bm_matrix" subclasses.

Supported S3 methods

- `[.bm_bitmap` and `[<-.bm_bitmap`
- `as.matrix.bm_bitmap()`
- `as.raster.bm_bitmap()` and `plot.bm_bitmap()`
- `cbind.bm_bitmap()` and `rbind.bm_bitmap()`
- `format.bm_bitmap()` and `print.bm_bitmap()`
- `Ops.bm_bitmap()` for all the S3 "Ops" Group generic functions

See Also

`as_bm_bitmap()`, `is_bm_bitmap()`

Examples

```
space <- bm_bitmap(matrix(0, nrow = 16, ncol = 16))
print(space)
```

bm_bytepad *Pad bitmap widths to the nearest byte*

Description

`bm_bytepad()` pads the width of bitmaps to the nearest multiple of 8 by padding with pixels on the right.

Usage

```
bm_bytepad(x, ...)
```

Arguments

`x`	Either a `bm_bitmap()`, `bm_font()`, `bm_list()`, "magick-image", "nativeRaster", `bm_pixmap()`, or "raster" object.
`...`	Passed to `bm_extend()` along with `hjust = "left"` and `width_multiples_of = 8L`.

Details

This is required by the BDF font format, which specifies that each row of bitmap data must be padded with zero bits on the right to the nearest byte.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

See Also

`bm_extend()`

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
ncol(capital_r) # already 8 pixels wide
capital_r_7 <- bm_trim(capital_r, right = 1L)
ncol(capital_r_7) # trimmed to 7 pixels wide
capital_r_8 <- bm_bytepad(capital_r_7)
ncol(capital_r_8) # padded back to 8 pixels wide
all.equal(capital_r, capital_r_8)
```

bm_call *Execute a function call on bitmap objects*

Description

`bm_call()` excutes a function call on bitmap objects. Since its first argument is the bitmap object it is more convenient to use with pipes then directly using `base::do.call()` plus it is easier to specify additional arguments.

Usage

```
bm_call(x, .f, ...)
```

Arguments

`x`	Either a `bm_bitmap()`, `bm_font()`, `bm_list()`, "magick-image", "nativeRaster", `bm_pixmap()`, or "raster" object.
`.f`	A function to execute.
`...`	Additional arguments to `.f`.

Value

The return value of `.f`.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
bml <- as_bm_list("RSTATS", font = font)
bml <- bm_flip(bml, "both")
bm <- bm_call(bml, cbind, direction = "RTL")
print(bm)
```

bm_clamp *Clamp bitmap values*

Description

`bm_clamp()` "clamps" `bm_bitmap()` integers that lie outside an interval. The default coerces a multiple-integer-valued bitmap into a binary bitmap (as expected by most bitmap font formats). For pixmap objects non-background pixels are all coerced to a single value.

Usage

```
bm_clamp(x, ...)

## S3 method for class 'bm_bitmap'
bm_clamp(x, lower = 0L, upper = 1L, value = upper, ...)

## S3 method for class 'bm_list'
bm_clamp(x, ...)

## S3 method for class 'bm_pixmap'
bm_clamp(x, value = col2hex("black"), bg = col2hex("transparent"), ...)

## S3 method for class '`magick-image`'
bm_clamp(x, value = "black", bg = "transparent", ...)

## S3 method for class 'nativeRaster'
bm_clamp(x, value = col2int("black"), bg = col2int("transparent"), ...)

## S3 method for class 'raster'
bm_clamp(x, value = "black", bg = "transparent", ...)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
...	Additional arguments to be passed to or from methods.
lower	Integer value. Any value below lower will be clamped.
upper	Integer value. Any value above upper will be clamped.
value	Integer vector of length one or two of replacement value(s). If value is length one any values above upper are replaced by value while those below lower are replaced by lower. If value is length two any values above upper are replaced by value[2] and any values below lower are replaced by value[1]. For pixmap objects indicate requested non-background color.
bg	Bitmap background value.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

Examples

```
plus_sign <- matrix(0L, nrow = 9L, ncol = 9L)
plus_sign[5L, 3:7] <- 2L
plus_sign[3:7, 5L] <- 2L
plus_sign_glyph <- bm_bitmap(plus_sign)
print(plus_sign_glyph)
```

```
plus_sign_clamped <- bm_clamp(plus_sign_glyph)
print(plus_sign_clamped)

tulip <- farming_crops_16x16()$tulip$portrait
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 8L) {
  print(bm_clamp(tulip, "magenta"), compress = "v")
}
```

---

bm_compose *Compose graphemes in a bitmap list by applying combining marks*

Description

`bm_compose()` simplifies `bm_list()` object by applying combining marks to preceding glpyhs (composing new graphemes).

Usage

```
bm_compose(bml, pua_combining = character(0), ...)
```

Arguments

`bml`	A `bm_list()` object. All combining marks need appropriate Unicode code point names to be recognized by `is_combining_character()`.
`pua_combining`	Additional Unicode code points to be considered as a "combining" character such as characters defined in the Private Use Area (PUA) of a font.
`...`	Passed to `bm_overlay()`.

Details

`bm_compose()` identifies combining marks by their name using `is_combining_character()`. It then combines such marks with their immediately preceding glyph using `bm_overlay()`.

Value

A `bm_list()` object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
grave <- font[[str2ucp("`")]]
a <- font[[str2ucp("a")]]
bml <- bm_list(`U+0061` = a, `U+0300` = grave)
print(bml)
print(bm_compose(bml))
```

bm_compress *Compress bitmaps by a factor of two*

Description

Compresses bm_bitmap() objects by a factor of two by re-mapping to a "block elements" scheme. For pixmap objects like bm_pixmap() we simply shrink the pixmap by a factor of two using bm_distort().

Usage

```
bm_compress(x, direction = "vertical", ...)

## S3 method for class 'bm_bitmap'
bm_compress(x, direction = "vertical", ...)

## S3 method for class 'bm_pixmap'
bm_compress(x, direction = "vertical", ..., filter = "Point")

## S3 method for class '`magick-image`'
bm_compress(x, direction = "vertical", ..., filter = "Point")

## S3 method for class 'nativeRaster'
bm_compress(x, direction = "vertical", ..., filter = "Point")

## S3 method for class 'raster'
bm_compress(x, direction = "vertical", ..., filter = "Point")

## S3 method for class 'bm_list'
bm_compress(x, ...)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
direction	Either "vertical" or "v", "horizontal" or "h", OR "both" or "b".
...	Additional arguments to be passed to or from methods.
filter	Passed to magick::image_resize(). Use magick::filter_types() for list of supported filters. The default "Point" filter will maintain your sprite's color palette. NULL will give you the magick's default filter which may work better if you are not trying to maintain a sprite color palette.

Details

Depending on direction we shrink the bitmaps height and/or width by a factor of two and re-encode pairs/quartets of pixels to a "block elements" scheme. If necessary we pad the right/bottom

of the bitmap(s) by a pixel. For each pair/quartet we determine the most-common non-zero element and map them to a length twenty set of integers representing the "block elements" scheme. For integers greater than zero we map it to higher twenty character sets i.e. 1's get mapped to 0:19, 2's get mapped to 20:39, 3's get mapped to 40:59, etc. Using the default `px_unicode` will give you the exact matching "Block Elements" glyphs while `px_ascii` gives the closest ASCII approximation. Hence `print.bm_bitmap()` should produce reasonable results for compressed bitmaps if either of them are used as the `px` argument.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

See Also

See `https://en.wikipedia.org/wiki/Block_Elements` for more info on the Unicode Block Elements block.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
r <- font[[str2ucp("R")]]
print(r)
print(bm_compress(r, "vertical"))
print(bm_compress(r, "horizontal"))
print(bm_compress(r, "both"))

img <- png::readPNG(system.file("img", "Rlogo.png", package="png"))
logo <- as_bm_pixmap(img)
if (cli::is_utf8_output() &&
    cli::num_ansi_colors() > 256L &&
    requireNamespace("magick", quietly = TRUE)) {
  logo_c <- bm_compress(pm, "both", filter = NULL)
  print(logo_c, compress = "v")
}
```

`bm_distort` *Resize bitmaps via distortion.*

Description

`bm_distort()` resize bitmaps to arbitrary width and height value via `magick::image_resize()`. `bm_downscale()` is a wrapper to `bm_distort()` that downscales an image if (and only if) it is wider than a target width.

Usage

```
bm_distort(x, width = NULL, height = NULL, ...)

bm_downscale(x, width = getOption("width"), ...)

## S3 method for class 'bm_bitmap'
bm_distort(
  x,
  width = NULL,
  height = NULL,
  ...,
  filter = "Point",
  threshold = 0.5
)

## S3 method for class 'bm_list'
bm_distort(x, ...)

## S3 method for class 'bm_pixmap'
bm_distort(x, width = NULL, height = NULL, ..., filter = "Point")

## S3 method for class '`magick-image`'
bm_distort(x, width = NULL, height = NULL, ..., filter = "Point")

## S3 method for class 'nativeRaster'
bm_distort(x, width = NULL, height = NULL, ..., filter = "Point")

## S3 method for class 'raster'
bm_distort(x, width = NULL, height = NULL, ..., filter = "Point")
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
width	Desired width of bitmap
height	Desired height of bitmap
...	Additional arguments to be passed to or from methods.
filter	Passed to magick::image_resize(). Use magick::filter_types() for list of supported filters. The default "Point" filter will maintain your sprite's color palette. NULL will give you the magick's default filter which may work better if you are not trying to maintain a sprite color palette.
threshold	When the alpha channel weakly exceeds this threshold (on an interval from zero to one) then the pixel is determined to be "black".

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_expand() for expanding width/height by integer multiples. bm_resize() for resizing an image via trimming/extending an image.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
dim(capital_r) # 8 x 16
if (requireNamespace("magick", quietly = TRUE)) {
  capital_r_9x21 <- bm_distort(capital_r, width = 9L, height = 21L)
  print(capital_r_9x21)
}
crops <- farming_crops_16x16()
corn <- crops$corn$portrait
dim(corn) # 16 x 16
if (cli::is_utf8_output() &&
    cli::num_ansi_colors() >= 256L &&
    requireNamespace("magick", quietly = TRUE)) {
  corn_24x24 <- bm_distort(corn, width = 24L)
  print(corn_24x24, compress = "v")
}
```

---

bm_edit *Edit a bitmap via text editor*

Description

Edit a binary bitmap in a text editor.

Usage

```
bm_edit(bitmap, editor = getOption("editor"))
```

Arguments

bitmap	bm_bitmap() object. It will be coerced into a binary bitmap via bm_clamp().
editor	Text editor. See utils::file.edit() for more information.

Details

Represent zeroes with a . and ones with a @ (as in the yaff font format). You may also add/delete rows/columns but the bitmap must be rectangular.

Value

A bm_bitmap() object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
r <- font[[str2ucp("R")]]

# requires users to manually close file in text editor
## Not run:
  edited_r <- bm_edit(r)
  print(edited_r)

## End(Not run)
```

---

bm_expand *Expand bitmaps by repeating each row and/or column*

Description

bm_expand() expands bitmap(s) by repeating each row and/or column an indicated number of times.

Usage

```
bm_expand(x, width = 1L, height = width)

## S3 method for class 'bm_bitmap'
bm_expand(x, width = 1L, height = width)

## S3 method for class 'bm_list'
bm_expand(x, ...)

## S3 method for class 'bm_pixmap'
bm_expand(x, width = 1L, height = width)

## S3 method for class '`magick-image`'
bm_expand(x, width = 1L, height = width)

## S3 method for class 'nativeRaster'
bm_expand(x, width = 1L, height = width)

## S3 method for class 'raster'
bm_expand(x, width = 1L, height = width)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
width	An integer of how many times to repeat each column.

height	An integer of how many times to repeat each row.
...	Additional arguments to be passed to or from methods.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_extend() (and bm_resize() which makes larger bitmaps by adding pixels to their sides.

Examples

```
 font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
 font <- read_hex(font_file)
 capital_r <- font[[str2ucp("R")]]
 print(capital_r)
 print(bm_expand(capital_r, 2L),
       px = px_ascii)
 print(bm_expand(capital_r, width = 1L, height = 2L),
       px = px_ascii)
 print(bm_expand(capital_r, width = 2L, height = 1L),
       px = px_ascii)
crops <- farming_crops_16x16()
corn <- crops$corn$portrait
corn_2x <- bm_expand(corn, 2L)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_2x, compress = "v")
}
```

---

bm_extend *Extend bitmaps on the sides with extra pixels*

Description

bm_extend() extends bm_bitmap() objects with extra pixels. The directions and the integer value of the extra pixels are settable (defaulting to 0L).

Usage

```
bm_extend(
  x,
  value,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
```

```
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)

## S3 method for class 'bm_bitmap'
bm_extend(
  x,
  value = 0L,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)

## S3 method for class 'bm_pixmap'
bm_extend(
  x,
  value = col2hex("transparent"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)

## S3 method for class 'bm_list'
bm_extend(x, ...)
```

```
## S3 method for class '`magick-image`'
bm_extend(
  x,
  value = "transparent",
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)

## S3 method for class 'nativeRaster'
bm_extend(
  x,
  value = col2int("transparent"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)

## S3 method for class 'raster'
bm_extend(
  x,
  value = "transparent",
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
```

```
  vjust = "center-top",
  width_multiples_of = NULL,
  height_multiples_of = NULL,
  mode = c("constant", "edge")
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	Value for the new pixels.
sides	If not NULL then an integer vector indicating how many pixels to pad on all four sides. If the integer vector is of length one it indicates the number of pixels for all four sides. If of length two gives first the number for the vertical sides and then the horizontal sides. If of length three gives the number of pixels for top, the horizontal sides, and then bottom sides. If of length four gives the number of pixels for top, right, bottom, and then left sides. This is the same scheme as used by the CSS padding and margin properties.
top	How many pixels to pad the top.
right	How many pixels to pad the right.
bottom	How many pixels to pad the bottom.
left	How many pixels to pad the left.
width	How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.
height	How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.
hjust	One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right".
vjust	One of "bottom", "center-bottom", "center-top", "top". "center-bottom" and "center-top" will attempt to place in "center" if possible but if not possible will bias it one pixel down or up respectively. "centre", "center", and "centre-top" are aliases for "center-top". "centre-bottom" is an alias for "center-bottom".
width_multiples_of	If not NULL an integer: pad the bitmap's width to the smallest multiple of this value that is greater than or equal to the current width (e.g. width_multiples_of = 8L ensures the width is 8, 16, 24, ...). Use with the hjust argument to control which side receives the extra pixels. Cannot be combined with sides or width.
height_multiples_of	If not NULL an integer: pad the bitmap's height to the smallest multiple of this value that is greater than or equal to the current height. Use with the vjust argument to control which side receives the extra pixels. Cannot be combined with sides or height.

mode	Either "constant" (default) to fill new pixels with value, or "edge" to fill with the nearest edge pixel.
...	Additional arguments to be passed to or from methods.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_expand(), bm_pad(), bm_resize(), and bm_trim().

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
# add a border to an "R"
capital_r <- font[[str2ucp("R")]]
capital_r <- bm_extend(capital_r, value = 2L, sides = 1L)
capital_r <- bm_extend(capital_r, value = 3L, sides = 1L)
print(capital_r)

crops <- farming_crops_16x16()
corn <- crops$corn$portrait
corn_framed <- bm_extend(corn, value = "brown", sides = 1L)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_framed, compress = "v")
}
```

---

bm_extract *Extract part of a bitmap*

Description

bm_extract() can be used to extract part of a bitmap. For bm_bitmap() and bm_pixmap() objects it is a wrapper around [[()] with drop = FALSE for convenience in pipes.

Usage

```
bm_extract(x, ...)

## S3 method for class 'bm_matrix'
bm_extract(x, rows = seq_len(nrow(x)), cols = seq_len(ncol(x)), ...)

## S3 method for class 'bm_list'
bm_extract(x, ...)
```

```
## S3 method for class '`magick-image`'
bm_extract(x, rows = seq_len(bm_heights(x)), cols = seq_len(bm_widths(x)), ...)

## S3 method for class 'nativeRaster'
bm_extract(x, rows = seq_len(nrow(x)), cols = seq_len(ncol(x)), ...)

## S3 method for class 'raster'
bm_extract(x, rows = seq_len(nrow(x)), cols = seq_len(ncol(x)), ...)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
...	Additional arguments to be passed to or from methods.
rows, cols	Integer vectors of rows and columns to extract. Rows are indexed from the **bottom** of the image.

See Also

[[.bm_matrix()], bm_trim()

Examples

```
corn <- farming_crops_16x16()$corn$portrait
corn_top <- bm_extract(corn, rows = 9:16)
all.equal(corn_top, corn[9:16, ])
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_top, bg = "cyan", compress = "v")
}
```

---

bm_flip *Flip (reflect) bitmaps*

Description

bm_flip() flips (reflects) bitmaps horizontally, vertically, or both. It can flip the entire bitmap or just the glyph in place.

Usage

```
bm_flip(x, direction = "vertical", in_place = FALSE, value)

## S3 method for class 'bm_bitmap'
bm_flip(x, direction = "vertical", in_place = FALSE, value = 0L)

## S3 method for class 'bm_list'
bm_flip(x, ...)
```

```
## S3 method for class 'bm_pixmap'
bm_flip(
  x,
  direction = "vertical",
  in_place = FALSE,
  value = col2hex("transparent")
)

## S3 method for class '`magick-image`'
bm_flip(x, direction = "vertical", in_place = FALSE, value = "transparent")

## S3 method for class 'nativeRaster'
bm_flip(
  x,
  direction = "vertical",
  in_place = FALSE,
  value = col2int("transparent")
)

## S3 method for class 'raster'
bm_flip(x, direction = "vertical", in_place = FALSE, value = "transparent")
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
direction	Either "vertical" or "v", "horizontal" or "h", OR "both" or "b".
in_place	If TRUE flip the glyphs in place (without changing any background padding).
value	Background padding value (to use if in_place is TRUE)
...	Additional arguments to be passed to or from methods.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)

# Print upside down
bml <- as_bm_list("RSTATS", font = font)
bml <- bm_flip(bml, "both")
bm <- bm_call(bml, cbind, direction = "RTL")
print(bm)

# Can also modify glyphs "in place"
```

```
exclamation <- font[[str2ucp("!")]]
exclamation_flipped <- bm_flip(exclamation, in_place = TRUE)
print(exclamation_flipped)

crops <- farming_crops_16x16()
corn <- crops$corn$portrait
corn_fh <- bm_flip(corn, "h")
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_fh, compress = "v")
}
```

`bm_font` *Bitmap font object*

Description

`bm_font()` creates a bitmap font object.

Usage

```
bm_font(x = bm_list(), comments = NULL, properties = NULL)
```

Arguments

`x`	Named list of `bm_bitmap()` objects. Names must be coercible by `Unicode::as.u_char()`.
`comments`	An optional character vector of (global) font comments.
`properties`	An optional named list of font metadata.

Details

`bm_font()` is a named list. The names are of the form "U+HHHH" or "U+HHHHH". where the `H` are appropriate hexadecimal Unicode code points. It is a subclass of `bm_list()`.

Value

A named list with a "bm_font" subclass.

See Also

`is_bm_font()`, `as_bm_font()`, `hex2ucp()`

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
is_bm_font(font)

# number of characters in font
length(font)

# print out "R"
R_glyph <- font[[str2ucp("R")]]
print(R_glyph)
```

---

bm_gray *Gray a bitmap*

Description

`bm_gray()` grays a bitmap. `bm_grey()` is offered as an alias.

Usage

```
bm_gray(x)

bm_grey(x)

## S3 method for class 'bm_bitmap'
bm_gray(x)

## S3 method for class 'bm_list'
bm_gray(x)

## S3 method for class 'bm_pixmap'
bm_gray(x)

## S3 method for class '`magick-image`'
bm_gray(x)

## S3 method for class 'nativeRaster'
bm_gray(x)

## S3 method for class 'raster'
bm_gray(x)
```

Arguments

x	Either a `bm_bitmap()`, `bm_font()`, `bm_list()`, "magick-image", "nativeRaster", `bm_pixmap()`, or "raster" object.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

Examples

```
corn <- farming_crops_16x16()$corn$portrait
corn_gray <- bm_gray(corn)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_gray, compress = "v")
}
```

---

bm_heights *Widths or heights of bitmaps*

Description

bm_widths() returns the widths of the bitmaps while bm_heights() returns the heights of the bitmaps. bm_widths() and bm_heights() are S3 generic functions.

Usage

```
bm_heights(x, ...)

## S3 method for class 'bm_matrix'
bm_heights(x, ...)

## S3 method for class 'bm_list'
bm_heights(x, unique = TRUE, ...)

## S3 method for class '`magick-image`'
bm_heights(x, ...)

## S3 method for class 'nativeRaster'
bm_heights(x, ...)

## S3 method for class 'raster'
bm_heights(x, ...)

bm_widths(x, ...)

## S3 method for class 'bm_matrix'
bm_widths(x, ...)

## S3 method for class 'bm_list'
bm_widths(x, unique = TRUE, ...)
```

```
## S3 method for class '`magick-image`'
bm_widths(x, ...)

## S3 method for class 'nativeRaster'
bm_widths(x, ...)

## S3 method for class 'raster'
bm_widths(x, ...)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
...	Ignored.
unique	Apply base::unique() to the returned integer vector.

Value

A integer vector of the relevant length of each of the bitmap objects in x. If unique is TRUE then any duplicates will have been removed.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
bm_widths(font) # every glyph in the font is 8 pixels wide
bm_heights(font) # every glyph in the font is 16 pixels high
corn <- farming_crops_16x16()$corn$portrait
bm_widths(corn)
bm_heights(corn)
```

---

bm_invert *Invert (negate) a bitmap*

Description

bm_invert() inverts (negates) a bitmap.

Usage

```
bm_invert(x)

## S3 method for class 'bm_bitmap'
bm_invert(x)

## S3 method for class 'bm_list'
bm_invert(x)
```

```
## S3 method for class 'bm_pixmap'
bm_invert(x)

## S3 method for class '`magick-image`'
bm_invert(x)

## S3 method for class 'nativeRaster'
bm_invert(x)

## S3 method for class 'raster'
bm_invert(x)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- as_bm_bitmap("R", font = font)
capital_r_inverted <- bm_invert(capital_r)
print(capital_r_inverted)

corn <- farming_crops_16x16()$corn$portrait
corn_inverted <- bm_invert(corn)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_inverted, compress = "v", bg = "black")
}
```

---

bm_lapply *Modify bitmap lists*

Description

bm_lapply() applies a function over a bitmap glyph list and returns a modified bitmap glyph list.

Usage

```
bm_lapply(X, FUN, ...)
```

Arguments

X	A bitmap glyph list object such as bm_list() or bm_font().
FUN	A function that takes a bitmap object supported by is_supported_bitmap() as its first argument and returns a bitmap object supported by is_supported_bitmap().
...	Additional arguments to pass to FUN.

Details

bm_lapply() is a wrapper around base::lapply() that preserves the classes and metadata of the original bitmap glyph list.

Value

A modified bitmap glyph list.

See Also

base::lapply(), bm_list(), bm_font(), bm_bitmap()

---

bm_list *Bitmap list object*

Description

bm_list() creates a bitmap list object.

Usage

```
bm_list(...)
```

Arguments

...	bm_bitmap() objects, possibly named.

Details

bm_list() is a list of bm_bitmap() objects with class "bm_list". It is superclass of bm_font().

Value

A named list with a "bm_list" subclass.

Supported S3 methods

- as.list.bm_list()
- Slicing with [] returns bm_list() objects.
- The min(), max(), and range() functions from the "Summary" group of generic methods.

See Also

is_bm_list(), as_bm_list()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)

gl <- font[c("U+0023", "U+0052", "U+0053", "U+0054", "U+0041", "U+0054", "U+0053")] # #RSTATS
gl <- as_bm_list(gl)
is_bm_list(gl)
```

---

bm_mask *Modify bitmaps via masking with a 'mask' bitmap*

Description

bm_mask() modifies bitmaps by using a binary bitmap "mask" to set certain elements to a background value.

Usage

```
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_bitmap'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_pixmap'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
```

```
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class '`magick-image`'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'nativeRaster'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'raster'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_list'
bm_mask(
  x,
  mask = NULL,
  base = NULL,
  mode = c("luminance", "alpha"),
  hjust = "center-left",
  vjust = "center-top"
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.

mask	An object to use as a binary bitmap "mask". Only one of mask or base may be set. Will be coerced to a bm_bitmap() object by as_bm_bitmap().
base	A bitmap/pixmap object which will be "masked" by mask. Only one of mask or base may be set.
mode	Either "luminance" (default) or "alpha".
hjust	One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right".
vjust	One of "bottom", "center-bottom", "center-top", "top". "center-bottom" and "center-top" will attempt to place in "center" if possible but if not possible will bias it one pixel down or up respectively. "centre", "center", and "centre-top" are aliases for "center-top". "centre-bottom" is an alias for "center-bottom".

Details

If necessary bitmaps will be extended by bm_extend() such that they are the same size. If necessary the mask will be coerced into a "binary" mask by bm_clamp(as_bm_bitmap(mask)). If mode is "luminance" then where the mask is 1L the corresponding pixel in base will be coerced to the background value. If mode is "alpha" then where the mask is 0L the corresponding pixel in base will be coerced to the background value.

Value

A bitmap/pixmap object that is the same type as x (if base is NULL) or base.

Examples

```
if (require("grid", quietly = TRUE) && capabilities("png")) {
  font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
  font <- read_hex(font_file)
  one <- font[[str2ucp("1")]]
  circle_large <- as_bm_bitmap(circleGrob(r = 0.50), width = 16L, height = 16L)
  circle_small <- as_bm_bitmap(circleGrob(r = 0.40), width = 16L, height = 16L)
  circle_outline <- bm_mask(circle_large, circle_small)
  print(circle_outline)
}
if (capabilities("png")) {
  # U+2776 "Dingbat Negative Circled Digit One"
  circle_minus_one <- bm_mask(circle_large, one)
  print(circle_minus_one)
}
# Can also do "alpha" mask
square_full <- bm_bitmap(matrix(1L, nrow = 16L, ncol = 16L))
square_minus_lower_left <- square_full
square_minus_lower_left[1:8, 1:8] <- 0L
print(square_minus_lower_left)
if (capabilities("png")) {
 circle_minus_lower_left <- bm_mask(circle_large, square_minus_lower_left, mode = "alpha")
  print(circle_minus_lower_left)
```

```
}

if (capabilities("png")) {
  m <- matrix(grDevices::rainbow(8L), byrow = TRUE, ncol = 8L, nrow = 8L)
  rainbow <- bm_expand(as_bm_pixmap(m), 2L)
  circle_rainbow <- bm_mask(rainbow, circle_large, mode = "alpha")
}
if (cli::is_utf8_output() &&
    cli::num_ansi_colors() >= 256L &&
    capabilities("png")) {
  print(circle_rainbow, compress = "v")
}
```

`bm_options` *Get bittermelon options*

Description

`bm_options()` returns the `bittermelon` package's global options.

Usage

```
bm_options(..., default = FALSE)
```

Arguments

`...`	`bittermelon` package options using `name = value`. The return list will use any of these instead of the current/default values.
`default`	If `TRUE` return the default values instead of current values.

Value

A list of option values. Note this function **does not** set option values itself but this list can be passed to `options()`, `withr::local_options()`, or `withr::with_options()`.

See Also

bittermelon for a high-level description of relevant global options.

Examples

```
bm_options()

bm_options(default = TRUE)

bm_options(bittermelon.compress = "vertical")
```

bm_outline *Compute "outline" bitmap of a bitmap*

Description

`bm_outline()` returns a bitmap that is just the "outline" of another bitmap.

Usage

```
bm_outline(x, value, bg)

## S3 method for class 'bm_bitmap'
bm_outline(x, value = 1L, bg = 0L)

## S3 method for class 'bm_list'
bm_outline(x, ...)

## S3 method for class 'bm_pixmap'
bm_outline(x, value = col2hex("black"), bg = col2hex("transparent"))

## S3 method for class '`magick-image`'
bm_outline(x, value = "black", bg = "transparent")

## S3 method for class 'nativeRaster'
bm_outline(x, value = col2int("black"), bg = col2int("transparent"))

## S3 method for class 'raster'
bm_outline(x, value = "black", bg = "transparent")
```

Arguments

x	Either a `bm_bitmap()`, `bm_font()`, `bm_list()`, "magick-image", "nativeRaster", `bm_pixmap()`, or "raster" object.
value	Bitmap "color" value for the outline.
bg	Bitmap "background" value.
...	Additional arguments to be passed to or from methods.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

Examples

```
square <- bm_bitmap(matrix(1L, nrow = 16L, ncol = 16L))
square_outline <- bm_outline(square)
print(square_outline)
```

```
if (require(grid) && capabilities("png")) {
  circle <- as_bm_bitmap(circleGrob(), width=16, height=16)
  circle_outline <- bm_outline(circle)
  print(circle_outline)
}

corn <- farming_crops_16x16()$corn$portrait
corn_outline <- bm_outline(corn, "magenta")
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_outline, bg = "white")
}
```

---

bm_overlay *Merge bitmaps by overlaying one over another*

Description

bm_overlay() merges bitmaps by overlaying a bitmap over another.

Usage

```
bm_overlay(
  x,
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  ...
)

## S3 method for class 'bm_bitmap'
bm_overlay(
  x,
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  bg = 0L,
  ...
)

## S3 method for class 'bm_list'
bm_overlay(x, ...)

## S3 method for class 'bm_pixmap'
bm_overlay(
  x,
```

```
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  bg = col2hex("transparent"),
  ...
)

## S3 method for class '`magick-image`'
bm_overlay(
  x,
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  bg = "transparent",
  ...
)

## S3 method for class 'nativeRaster'
bm_overlay(
  x,
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  bg = col2int("transparent"),
  ...
)

## S3 method for class 'raster'
bm_overlay(
  x,
  over = NULL,
  under = NULL,
  hjust = "center-left",
  vjust = "center-top",
  bg = "transparent",
  ...
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
over	A bitmap/pixmap object to overlay over the x bitmap(s). Only one of over or under may be set.
under	A bitmap/pixmap object which will be overlaid by the x bitmap(s). Only one of

over or under may be set.

hjust — One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right".

vjust — One of "bottom", "center-bottom", "center-top", "top". "center-bottom" and "center-top" will attempt to place in "center" if possible but if not possible will bias it one pixel down or up respectively. "centre", "center", and "centre-top" are aliases for "center-top". "centre-bottom" is an alias for "center-bottom".

... — Additional arguments to be passed to or from methods.

bg — Bitmap background value.

Details

If necessary bitmaps will be extended by `bm_extend()` such that they are the same size. Then the non-zero pixels of the "over" bitmap will be inserted into the "under" bitmap.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
grave <- font[[str2ucp("`")]]
a <- font[[str2ucp("a")]]
a_grave <- bm_overlay(a, over = grave)
print(a_grave)

# Can also instead specify the under glyph as a named argument
a_grave2 <- bm_overlay(grave, under = a)
print(a_grave2)

crops <- farming_crops_16x16()
corn <- bm_shift(crops$corn$portrait, right = 2L, top = 2L)
grapes <- bm_shift(crops$grapes$portrait, bottom = 1L)
grapes_and_corn <- bm_overlay(corn, grapes)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(grapes_and_corn, compress = "v")
}
```

bm_pad *Adjust bitmap padding lengths*

Description

bm_pad() adjusts bitmap padding lengths.

Usage

```
bm_pad(
  x,
  value,
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)

## S3 method for class 'bm_bitmap'
bm_pad(
  x,
  value = 0L,
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)

## S3 method for class 'bm_list'
bm_pad(x, ...)

## S3 method for class 'bm_pixmap'
bm_pad(
  x,
  value = col2hex("transparent"),
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)
```

```
## S3 method for class '`magick-image`'
bm_pad(
  x,
  value = "transparent",
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)

## S3 method for class 'nativeRaster'
bm_pad(
  x,
  value = col2int("transparent"),
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)

## S3 method for class 'raster'
bm_pad(
  x,
  value = "transparent",
  type = c("exact", "extend", "trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	Value for the new pixels.
type	Either "exact", ‘"extend", or "trim". "exact" makes sure the padding is exactly the indicated amount, "extend" does not trim any padding if existing padding is more than the indicated amount, and "trim" does not extend any padding if existing padding is less than the indicated amount.
sides	If not NULL then an integer vector indicating the desired number of pixels of padding on all four sides. If the integer vector is of length one it indicates the number of pixels for all four sides. If of length two gives first the number for the

	vertical sides and then the horizontal sides. If of length three gives the number of pixels for top, the horizontal sides, and then bottom sides. If of length four gives the number of pixels for top, right, bottom, and then left sides. This is the same scheme as used by the CSS padding and margin properties.
`top`	Desired number of pixels of padding on the top.
`right`	Desired number of pixels of padding on the right.
`bottom`	Desired number of pixels of padding on the bottom.
`left`	Desired number of pixels of padding on the left.
`...`	Additional arguments to be passed to or from methods.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

See Also

`bm_extend()`, `bm_resize()`, and `bm_trim()`

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r)
capital_r_padded <- bm_pad(capital_r, sides = 2L)
print(capital_r_padded)
crops <- farming_crops_16x16()
corn <- crops$corn$portrait
corn_pad4 <- bm_pad(corn, sides = 4L)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_pad4, compress = "v", bg = "cyan")
}
```

`bm_padding_lengths` *Compute bitmap padding lengths*

Description

`bm_padding_lengths()` computes the padding lengths of a target value for the top, right, bottom, and left sides of the bitmap. If the entire bitmap is of the target value then the left/right and top/bottom will simply split the width/height in half.

Usage

```
bm_padding_lengths(x, value)

## S3 method for class 'bm_bitmap'
bm_padding_lengths(x, value = 0L)

## S3 method for class 'bm_list'
bm_padding_lengths(x, ...)

## S3 method for class 'bm_pixmap'
bm_padding_lengths(x, value = col2hex("transparent"))

## S3 method for class '`magick-image`'
bm_padding_lengths(x, value = "transparent")

## S3 method for class 'nativeRaster'
bm_padding_lengths(x, value = col2int("transparent"))

## S3 method for class 'raster'
bm_padding_lengths(x, value = "transparent")
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	The value of the "padding" element to compute lengths for.
...	Additional arguments to be passed to or from methods.

Value

If x is a bm_bitmap() object then a integer vector of length four representing the padding lengths for the top, right, bottom, and left sides respectively. If x is a bm_list() or bm_font() then a list of integer vectors of length four.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
# add a border to an "R"
capital_r <- font[[str2ucp("R")]]
print(capital_r)
print(bm_padding_lengths(capital_r))
corn <- farming_crops_16x16()$corn$portrait
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn, bg = "cyan", compress = "v")
}
print(bm_padding_lengths(corn))
```

`bm_pixel_picker` *Bitmap pixel picker*

Description

`bm_pixel_picker()` lets you use an interactive graphics device to click on a bitmap's pixels and learn the column/row coordinates for the clicked pixel and its integer/color value. To end the program click a non-left mouse button within the graphics device.

Usage

```
bm_pixel_picker(x, ...)

## S3 method for class 'bm_bitmap'
bm_pixel_picker(
  x,
  ...,
  col = getOption("bittermelon.col", col_bitmap),
  silent = FALSE
)

## S3 method for class 'bm_pixmap'
bm_pixel_picker(x, ..., silent = FALSE)

## S3 method for class 'raster'
bm_pixel_picker(x, ..., silent = FALSE)
```

Arguments

`x`	Either a `bm_bitmap()`, `bm_pixmap()`, or raster object
`...`	Currently ignored.
`col`	Character vector of R color specifications. First color is used for values equal to 0, second color for values equal to 1, etc.
`silent`	Don't generate messages about clicked pixels.

Value

A list with named components "row", "col", and "value" for the last clicked pixel returned invisibly.

See Also

This function wraps `grid::grid.locator()`.

Examples

```
if (interactive() && dev.interactive(orNone = TRUE)) {
  corn <- farming_crops_16x16()$corn$portrait
  bm_pixel_picker(corn)
}
```

`bm_pixmap` *Bittermelon pixmap matrix object*

Description

`bm_pixmap()` creates an S3 matrix subclass representing a pixmap.

Usage

```
bm_pixmap(x)
```

Arguments

`x`	Object to be converted to `bm_pixmap()`. If not already a color string matrix it will be cast to one by `as_bm_pixmap()`.

Details

- Intended to represent raster graphic pixmaps especially (but not limited to) pixel art/sprites.
- Pixmaps are represented as color string matrices with special class methods.
- The bottom left pixel is represented by the first row and first column.
- The bottom right pixel is represented by the first row and last column.
- The top left pixel is represented by the last row and first column.
- The top right pixel is represented by the last row and last column.
- Colors are converted to the `"#RRGGBBAA"` color string format.
- Fully transparent values like `"transparent"`, `NA`, `"#00000000"` are all standardized to `"#FFFFFF00"`.
- See `bm_bitmap()` for an alternative S3 object backed by a integer matrix.

Value

A character matrix of color strings with a "bm_pixmap" subclass.

Supported S3 methods

- `[.bm_bitmap` and `[<-.bm_bitmap`
- `as.matrix.bm_pixmap()`
- `as.raster.bm_bitmap()` and `plot.bm_bitmap()`
- `format.bm_pixmap()` and `print.bm_pixmap()`

See Also

as_bm_pixmap(), is_bm_pixmap()

Examples

```
# Bottom left pixel is **first** row and first column
pm <- bm_pixmap(matrix(c("red", "blue", "green", "black"),
                nrow = 2L, byrow = TRUE))
plot(pm)
```

---

bm_print *Print bitmap objects*

Description

bm_print() prints a representation of the bitmap object to the terminal while bm_format() returns just the character vector without printing it. They are wrappers around as_bm_bitmap() / as_bm_pixmap() and format.bm_bitmap() / format.bm_pixmap().

Usage

```
bm_print(x, ...)

bm_format(x, ...)
```

Arguments

x	A bitmap object that can be cast by as_bm_pixmap() to a bm_pixmap() object.
...	Passed to format.bm_pixmap() or format.bm_bitmap() depending on the class of x.

Value

A character vector of the string representation (bm_print() returns this invisibly). As a side effect bm_print() prints out the string representation to the terminal.

Fonts and terminal settings

Printing bitmaps/pixmaps may or may not look great in your terminal depending on a variety of factors:

- The terminal should support the Unicode - UTF-8 encoding. We use cli::is_utf8_output() to guess Unicode support which in turn looks at getOption("cli.unicode") and l10n_info().
- The terminal should support ANSI sequences and if it does it should support many colors.
  - We use cli::num_ansi_colors() to detect number of colors supported. num_ansi_colors() detection algorithm is complicated but it first looks at getOption("cli.num_colors").

  - If `cli::num_ansi_colors()` equals 16777216 then your terminal supports 24-bit ANSI colors.
  - If using the Windows Command Prompt window you may need to enable ANSI sequences support by doing `REG ADD HKCU\CONSOLE /f /v VirtualTerminalLevel /t REG_DWORD /d 1` from the command-line or running `regedit` (Registry Editor) and go to `Computer\HKEY_CURRENT_USER\Console` and set `VirtualTerminalLevel` to `1`.

- The font used by the terminal should be a monoscale font that supports the Block Elements Unicode block.
- The terminal text settings should have a cell spacing around 1.00 times width and 1.00 times height. For terminals configured by CSS styles this means a `line-height` of around `1.0`.

See Also

`.S3method()` to register this as the print method for a non-bittermelon bitmap class.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- as_bm_bitmap("R", font = font)
bm_print(capital_r)

corn_r <- as.raster(farming_crops_16x16()$corn$portrait)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  bm_print(corn_r, compress = "v")
}

if (requireNamespace("magick", quietly = TRUE) &&
    cli::is_utf8_output() &&
    cli::num_ansi_colors() > 256L) {
  rose_mi <- magick::image_read("rose:")
  bm_print(rose_mi, compress = "v")
}

## Not run: # Change other bitmap classes' `print()` to use `bm_print()` instead
  options(bittermelon.compress = "vertical",
          bittermelon.downscale = requireNamespace("magick", quietly = TRUE))
  for (cl in c("glyph_bitmap", "lofi-matrix", "magick-image",
               "nativeRaster", "pixeltrix",
               "pixmapGrey", "pixmapIndexed", "pixmapRGB", "raster")) {
    .S3method("print", cl, bittermelon::bm_print)
  }

## End(Not run)
```

`bm_replace` *Replace a color in a bitmap with another color*

Description

`bm_replace()` replaces a bitmap color with another color. In particular default arguments will try to replace the background color.

Usage

```
bm_replace(x, value, old)

## S3 method for class 'bm_bitmap'
bm_replace(x, value = 0L, old = x[1L, 1L])

## S3 method for class 'bm_list'
bm_replace(x, ...)

## S3 method for class 'bm_pixmap'
bm_replace(x, value = col2hex("transparent"), old = x[1L, 1L])

## S3 method for class '`magick-image`'
bm_replace(x, value = "transparent", old = as.raster(x)[1L, 1L])

## S3 method for class 'nativeRaster'
bm_replace(x, value = col2int("transparent"), old = x[1L, 1L])

## S3 method for class 'raster'
bm_replace(x, value = "transparent", old = x[1L, 1L])
```

Arguments

`x`	Either a `bm_bitmap()`, `bm_font()`, `bm_list()`, "magick-image", "nativeRaster", `bm_pixmap()`, or "raster" object.
`value`	New bitmap "color" value.
`old`	Old bitmap "color" value to replace.
`...`	Additional arguments to be passed to or from methods.

Value

Depending on `x` either a `bm_bitmap()`, `bm_font()`, `bm_list()`, magick-image, "nativeRaster", `bm_pixmap()`, or raster object.

Examples

```
corn <- farming_crops_16x16()$corn$portrait
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(bm_replace(corn, "cyan"), compress = "v")
}
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(bm_replace(capital_r, 2L))
```

`bm_resize` *Resize bitmaps by trimming and/or extending*

Description

Trim and/or extend bitmaps to a desired height and/or width.

Usage

```
bm_resize(
  x,
  value,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_bitmap'
bm_resize(
  x,
  value = 0L,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_list'
bm_resize(x, ...)

## S3 method for class 'bm_pixmap'
bm_resize(
  x,
  value = col2hex("transparent"),
  width = NULL,
  height = NULL,
```

```
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class '`magick-image`'
bm_resize(
  x,
  value = "transparent",
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'nativeRaster'
bm_resize(
  x,
  value = col2int("transparent"),
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'raster'
bm_resize(
  x,
  value = "transparent",
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	Value for the new pixels.
width	How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.
height	How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.
hjust	One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right".

vjust	One of "bottom", "center-bottom", "center-top", "top". "center-bottom" and "center-top" will attempt to place in "center" if possible but if not possible will bias it one pixel down or up respectively. "centre", "center", and "centre-top" are aliases for "center-top". "centre-bottom" is an alias for "center-bottom".
...	Additional arguments to be passed to or from methods.

Details

This function is a convenience wrapper around bm_trim() and bm_extend().

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_extend(), bm_pad(), and bm_trim().

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
# add a border to an "R"
capital_r <- font[[str2ucp("R")]]
print(capital_r)
capital_r <- bm_resize(capital_r, width = 12L, height = 12L, vjust = "top")
print(capital_r)
corn <- farming_crops_16x16()$corn$portrait
corn_rs <- bm_resize(corn, width = 20L, height = 20L, vjust = "top")
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_rs, bg = "cyan", compress = "v")
}
```

---

bm_rotate *Rotate bitmaps 0, 90, 180, or 270 degrees*

Description

bm_rotate() losslessly rotates bitmaps by 0, 90, 180, or 270 degrees. If 90 or 270 degrees are indicated the width and height of the bitmap will be flipped unless in_place is TRUE.

Usage

```
bm_rotate(x, angle = 0L, clockwise = TRUE, in_place = FALSE, value)

## S3 method for class 'bm_bitmap'
bm_rotate(x, angle = 0L, clockwise = TRUE, in_place = FALSE, value = 0L)
```

```
## S3 method for class 'bm_list'
bm_rotate(x, ...)

## S3 method for class 'bm_pixmap'
bm_rotate(
  x,
  angle = 0L,
  clockwise = TRUE,
  in_place = FALSE,
  value = col2hex("transparent")
)

## S3 method for class '`magick-image`'
bm_rotate(
  x,
  angle = 0L,
  clockwise = TRUE,
  in_place = FALSE,
  value = "transparent"
)

## S3 method for class 'nativeRaster'
bm_rotate(
  x,
  angle = 0L,
  clockwise = TRUE,
  in_place = FALSE,
  value = col2int("transparent")
)

## S3 method for class 'raster'
bm_rotate(
  x,
  angle = 0L,
  clockwise = TRUE,
  in_place = FALSE,
  value = "transparent"
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
angle	Angle to rotate bitmap by.
clockwise	If TRUE rotate bitmaps clockwise. Note Unicode's convention is to rotate glyphs clockwise i.e. the top of the "BLACK CHESS PAWN ROTATED NINETY DEGREES" glyph points right.

in_place	If TRUE rotate the glyphs in place (without changing any background padding).
value	Background padding value (to use if in_place is TRUE)
...	Additional arguments to be passed to or from methods.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_distort() can do other (distorted) rotations by careful use of its vp grid::viewport() argument.

Examples

```
# as_bm_list.character()
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(bm_rotate(capital_r, 90))
print(bm_rotate(capital_r, 180))
print(bm_rotate(capital_r, 270))
print(bm_rotate(capital_r, 90, clockwise = FALSE))

corn <- farming_crops_16x16()$corn$portrait
corn_180 <- bm_rotate(corn, 180)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_180, compress = "v")
}
```

---

bm_shadow *Bitmap shadow, bold, and glow effects*

Description

bm_shadow() adds a basic "shadow" effect to the bitmap(s). bm_bold() is a variant with different defaults to create a basic "bold" effect. bm_glow() adds a basic "glow" effect to the bitmap(s).

Usage

```
bm_shadow(
  x,
  value,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
```

```
  extend = TRUE,
  bg
)

## S3 method for class 'bm_bitmap'
bm_shadow(
  x,
  value = 2L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE,
  bg = 0L
)

## S3 method for class 'bm_list'
bm_shadow(x, ...)

## S3 method for class 'bm_pixmap'
bm_shadow(
  x,
  value = col2hex("black"),
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE,
  bg = col2hex("transparent")
)

## S3 method for class '`magick-image`'
bm_shadow(
  x,
  value = "black",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE,
  bg = "transparent"
)

## S3 method for class 'nativeRaster'
bm_shadow(
  x,
  value = col2int("black"),
  top = NULL,
```

```
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE,
  bg = "transparent"
)

## S3 method for class 'raster'
bm_shadow(
  x,
  value = "black",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE,
  bg = "transparent"
)

bm_bold(
  x,
  value = 1L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

## S3 method for class 'bm_bitmap'
bm_bold(
  x,
  value = 1L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

## S3 method for class 'bm_list'
bm_bold(x, ...)

## S3 method for class 'bm_pixmap'
bm_bold(
  x,
  value = col2hex("black"),
  top = NULL,
```

```
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

## S3 method for class '`magick-image`'
bm_bold(
  x,
  value = "black",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

## S3 method for class 'nativeRaster'
bm_bold(
  x,
  value = col2int("black"),
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

## S3 method for class 'raster'
bm_bold(
  x,
  value = "black",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)

bm_glow(x, value, extend = TRUE, corner = FALSE, bg)

## S3 method for class 'bm_bitmap'
bm_glow(x, value = 2L, extend = TRUE, corner = FALSE, bg = 0L)

## S3 method for class 'bm_list'
bm_glow(x, ...)

## S3 method for class 'bm_pixmap'
```

```
bm_glow(
  x,
  value = col2hex("black"),
  extend = TRUE,
  corner = FALSE,
  bg = col2hex("transparent")
)

## S3 method for class '`magick-image`'
bm_glow(x, value = "black", extend = TRUE, corner = FALSE, bg = "transparent")

## S3 method for class 'nativeRaster'
bm_glow(x, value = "black", extend = TRUE, corner = FALSE, bg = "transparent")

## S3 method for class 'raster'
bm_glow(x, value = "black", extend = TRUE, corner = FALSE, bg = "transparent")
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	The integer value for the shadow, bold, or glow effect.
top	How many pixels above should the shadow go.
right	How many pixels right should the shadow go. if top, right, bottom, and left are all NULL then defaults to 1L.
bottom	How many pixels below should the shadow go. if top, right, bottom, and left are all NULL then defaults to 1L for bm_shadow() and 0L for bm_embolden().
left	How many pixels left should the shadow go.
extend	Make the bitmap larger to give the new glyph more "room".
bg	Bitmap background value.
...	Additional arguments to be passed to or from methods.
corner	Fill in the corners.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_extend() and bm_shift()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
```

```
print(capital_r)
print(bm_shadow(capital_r))
print(bm_bold(capital_r))
print(bm_glow(capital_r))
print(bm_glow(capital_r, corner = TRUE))

corn <- farming_crops_16x16()$corn$portrait
corn_shadow <- bm_shadow(corn, "red")
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_shadow, compress = "v")
}

corn_glow <- bm_glow(corn, "cyan", corner = TRUE)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_glow, compress = "v")
}
```

`bm_shift` *Shift elements within bitmaps*

Description

Shifts non-padding elements within bitmaps by trimming on a specified side and padding on the other while preserving the width and height of the original bitmap.

Usage

```
bm_shift(
  x,
  value,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)

## S3 method for class 'bm_bitmap'
bm_shift(
  x,
  value = 0L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)
```

```
## S3 method for class 'bm_list'
bm_shift(x, ...)

## S3 method for class 'bm_pixmap'
bm_shift(
  x,
  value = col2hex("transparent"),
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)

## S3 method for class '`magick-image`'
bm_shift(
  x,
  value = "transparent",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)

## S3 method for class 'nativeRaster'
bm_shift(
  x,
  value = col2int("transparent"),
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)

## S3 method for class 'raster'
bm_shift(
  x,
  value = "transparent",
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  overflow = "clip"
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
value	Value for the new pixels.
top	Number of pixels to shift towards the top side.
right	Number of pixels to shift towards the right side.
bottom	Number of pixels to shift towards the bottom side.
left	Number of pixels to shift towards the left side.
overflow	How to handle shifts that would push content off the bitmap. One of "clip", "error", or "wrap". If "clip" (the default) then content that is pushed off the bitmap is silently lost. If "error" then an error is thrown if the shift would clip any non-padding content. If "wrap" then content that is pushed off one side of the bitmap wraps around to the other side.
...	Additional arguments to be passed to or from methods.

Details

This function is a convenience wrapper around bm_trim() and bm_extend().

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_trim() and bm_extend()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r)
capital_r <- bm_shift(capital_r, bottom = 2L, right = 1L)
print(capital_r)
corn <- farming_crops_16x16()$corn$portrait
print(bm_padding_lengths(corn))
corn_shifted <- bm_shift(corn, left = 1L, top = 2L)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_shifted, bg = "cyan", compress = "v")
}
corn_wrapped <- bm_shift(corn, right = 4L, overflow = "wrap")
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_wrapped, bg = "cyan", compress = "v")
}
```

bm_trim *Trim bitmaps*

Description

bm_trim() trims bitmap objects reducing the number of pixels. The directions and amount of removed pixels are settable.

Usage

```
bm_trim(
  x,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_matrix'
bm_trim(
  x,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'bm_list'
bm_trim(x, ...)

## S3 method for class '`magick-image`'
bm_trim(
  x,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
```

```
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'nativeRaster'
bm_trim(
  x,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

## S3 method for class 'raster'
bm_trim(
  x,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)
```

Arguments

x	Either a bm_bitmap(), bm_font(), bm_list(), "magick-image", "nativeRaster", bm_pixmap(), or "raster" object.
sides	If not NULL then an integer vector indicating how many pixels to trim on all four sides. If the integer vector is of length one it indicates the number of pixels for all four sides. If of length two gives first the number for the vertical sides and then the horizontal sides. If of length three gives the number of pixels for top, the horizontal sides, and then bottom sides. If of length four gives the number of pixels for top, right, bottom, and then left sides. This is the same scheme as used by the CSS padding and margin properties.
top	How many pixels to trim the top.
right	How many pixels to trim the right.

bottom	How many pixels to trim the bottom.
left	How many pixels to trim the left.
width	How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.
height	How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.
hjust	One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right". Note if "left" we will trim on the right (and vice-versa).
vjust	One of "bottom", "center-bottom", "center-top", "top". "center-bottom" and "center-top" will attempt to place in "center" if possible but if not possible will bias it one pixel down or up respectively. "centre", "center", and "centre-top" are aliases for "center-top". "centre-bottom" is an alias for "center-bottom". Note if "top" we will trim on the bottom (and vice-versa).
...	Additional arguments to be passed to or from methods.

Value

Depending on x either a bm_bitmap(), bm_font(), bm_list(), magick-image, "nativeRaster", bm_pixmap(), or raster object.

See Also

bm_extend(), bm_pad(), and bm_resize().

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r)
capital_r_trimmed <- bm_trim(capital_r, c(1, 1, 3, 0))
print(capital_r_trimmed)
corn <- farming_crops_16x16()$corn$portrait
print(bm_padding_lengths(corn))
corn_trimmed <- bm_trim(corn, top = 1L, right = 1L, bottom = 1L)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn_trimmed, bg = "cyan", compress = "v")
}
```

`c.bm_bitmap` *Combine bitmap objects*

Description

`c()` combines bitmap objects into `bm_list()` or `bm_font()` objects. In particular when using it to combine fonts the later fonts "update" the glyphs in the earlier fonts.

Usage

```
## S3 method for class 'bm_bitmap'
c(...)

## S3 method for class 'bm_font'
c(...)

## S3 method for class 'bm_list'
c(...)

## S3 method for class 'bm_pixmap'
c(...)
```

Arguments

`...`	`bm_bitmap()`, `bm_list()`, and/or `bm_font()` objects to combine.

Details

The various bitmap objects are "reduced" in the following ways:

First	Second	Result
`bm_bitmap()`	`bm_bitmap()`	`bm_list()`
`bm_bitmap()`	`bm_font()`	`bm_font()`
`bm_bitmap()`	`bm_list()`	`bm_list()`
`bm_bitmap()`	`bm_pixmap()`	`bm_list()`
`bm_pixmap()`	`bm_bitmap()`	`bm_list()`
`bm_pixmap()`	`bm_font()`	ERROR
`bm_pixmap()`	`bm_list()`	`bm_list()`
`bm_pixmap()`	`bm_pixmap()`	`bm_list()`
`bm_font()`	`bm_bitmap()`	`bm_font()`
`bm_font()`	`bm_font()`	`bm_font()`
`bm_font()`	`bm_list()`	`bm_font()`
`bm_font()`	`bm_pixmap()`	ERROR
`bm_list()`	`bm_bitmap()`	`bm_list()`
`bm_list()`	`bm_font()`	`bm_font()`
`bm_list()`	`bm_list()`	`bm_list()`
`bm_list()`	`bm_pixmap()`	`bm_list()`

When combining with a bm_font() object if any bm_bitmap() objects share the same name we only keep the last one. Although names are preserved other attributes such as font comments and properties are not guaranteed to be preserved.

Value

Either a bm_list() or bm_font() object. See Details for more info.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
stats <- as_bm_list("STATS", font = font)
is_bm_list(c(capital_r, capital_r))
rstats <- c(capital_r, stats)
print(bm_call(rstats, cbind))
```

---

cbind.bm_bitmap *Combine bitmap/pixmap objects by rows or columns*

Description

cbind.bm_bitmap() / cbind.bm_pixmap() and rbind.bm_bitmap() / rbind.bm_pixmap() combine by columns or rows respectively.

Usage

```
## S3 method for class 'bm_bitmap'
cbind(..., direction = "left-to-right", vjust = "center-top")

## S3 method for class 'bm_bitmap'
rbind(..., direction = "top-to-bottom", hjust = "center-left")

## S3 method for class 'bm_pixmap'
cbind(..., direction = "left-to-right", vjust = "center-top")

## S3 method for class 'bm_pixmap'
rbind(..., direction = "top-to-bottom", hjust = "center-left")
```

Arguments

...	bm_bitmap() or bm_pixmap() objects.

direction	For cbind() either "left-to-right" (default) or its aliases "ltr" and "lr" OR "right-to-left" or its aliases "rtl" and "rl". For rbind() either "top-to-bottom" (default) or its aliases "ttb" and "tb" OR "bottom-to-top" or its aliases "btt" and "bt". The direction argument is not case-sensitive.
vjust	Used by bm_extend() if bitmap heights are different.
hjust	Used by bm_extend() if bitmap widths are different.

Value

A bm_bitmap() or bm_pixmap() object.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_b <- font[[str2ucp("B")]]
capital_m <- font[[str2ucp("M")]]
cbm <- cbind(capital_b, capital_m)
print(cbm)
cbm_rl <- cbind(capital_b, capital_m, direction = "right-to-left")
print(cbm_rl)
rbm <- rbind(capital_b, capital_m)
print(rbm)
rbm_bt <- rbind(capital_b, capital_m, direction = "bottom-to-top")
print(rbm_bt)
```

---

col2hex *Colors to standardized hex strings*

Description

col2hex() standardizes R color strings into a unique RGBA hex string. All fully transparent colors get standardized to "#FFFFFF00".

Usage

```
col2hex(x)
```

Arguments

x	Color value as supported by grDevices::col2rgb().

Value

A standardized RGBA hex string (as returned by grDevices::rgb()).

Examples

```
col2hex("red")
col2hex("green")
col2hex("blue")
col2hex("transparent")
col2hex(NA_character_)
col2hex("#00000000")
```

`col2int` *Color to (native) integer conversions*

Description

`col2int()` converts color strings to (native) color integers. `int2col()` converts (native) color integers to color strings.

Usage

```
col2int(x)

int2col(x)
```

Arguments

x	Color value to convert.

Details

- Colors are also standardized by `col2hex()`.
- Requires either the colorfast or the farver package.

Value

`col2int()` returns an integer. `int2col()` returns a (hex) color string.

Examples

```
if (requireNamespace("farver", quietly = TRUE)) {
  int2col(col2int("red"))
}
```

`farming_crops_16x16` *Sprites for twenty farming crops*

Description

`farming_crops_16x16()` returns a named list of `bm_list()` lists of twenty farming crops in five stages of growth plus a portrait as `bm_pixmap()` objects.

Usage

```
farming_crops_16x16()
```

Details

- Each sprite is sixteen by sixteen pixels large.
- Farming Crops 16x16 was made and dedicated to the public domain by josehzz.

Value

A named list of `bm_list()` lists of six `bm_pixmap()` objects (one through five stages of growth plus a portrait for each crop). The named list has the following twenty crop names:

- "avocado"
- "cassava"
- "coffee"
- "corn"
- "cucumber"
- "eggplant"
- "grapes"
- "lemon"
- "melon"
- "orange"
- "pineapple"
- "potato"
- "rice"
- "rose"
- "strawberry"
- "sunflower"
- "tomato"
- "tulip"
- "turnip"
- "wheat"

Examples

```
crops <- farming_crops_16x16()
names(crops)
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(crops$corn$portrait, compress = "v")
}

if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(crops$orange$stage5, compress = "v")
}
```

---

hex2ucp *Get Unicode code points*

Description

`hex2ucp()`, `int2ucp()`, `name2ucp()`, and `str2ucp()` return Unicode code points as character vectors. `is_ucp()` returns `TRUE` if a valid Unicode code point.

Usage

```
hex2ucp(x)

int2ucp(x)

str2ucp(x)

name2ucp(x, type = c("exact", "grep"), ...)

is_ucp(x)

block2ucp(x, omit_unnamed = TRUE)

range2ucp(x, omit_unnamed = TRUE)
```

Arguments

`x`	R objects coercible to the respective Unicode character data types. See `Unicode::as.u_char()` for `hex2ucp()` and `int2ucp()`, `base::utf8ToInt()` for `str2ucp()`, `Unicode::u_char_from_name()` for `name2ucp()`, `Unicode::as.u_char_range()` for `range2ucp()`, and `Unicode::u_blocks()` for `block2ucp()`.
`type`	one of `"exact"` or `"grep"`, or an abbreviation thereof.
`...`	arguments to be passed to `grepl` when using this for pattern matching.
`omit_unnamed`	Omit control codes or unassigned code points

Details

`hex2ucp(x)` is a wrapper for `as.character(Unicode::as.u_char(toupper(x)))`. `int2ucp` is a wrapper for `as.character(Unicode::as.u_char(as.integer(x)))`. `str2ucp(x)` is a wrapper for `as.character(Unicode::as.u_char(utf8ToInt(x)))`. `name2ucp(x)` is a wrapper for `as.character(Unicode::u_char_from_name(x))`. However missing values are coerced to `NA_character_` instead of `"<NA>"`. Note the names of `bm_font()` objects must be character vectors as returned by these functions and not `Unicode::u_char` objects.

Value

A character vector of Unicode code points.

See Also

`ucp2label()` and `is_combining_character()`.

Examples

```
# These are all different ways to get the same 'R' code point
hex2ucp("52")
hex2ucp(as.hexmode("52"))
hex2ucp("0052")
hex2ucp("U+0052")
hex2ucp("0x0052")
int2ucp(82) # 82 == as.hexmode("52")
int2ucp("82") # 82 == as.hexmode("52")
int2ucp(utf8ToInt("R"))
ucp2label("U+0052")
name2ucp("LATIN CAPITAL LETTER R")
str2ucp("R")

block2ucp("Basic Latin")
block2ucp("Basic Latin", omit_unnamed = FALSE)
range2ucp("U+0020..U+0030")
```

---

is_bm_bitmap *Test if the object is a bitmap object*

Description

`is_bm_bitmap()` returns `TRUE` for `bm_bitmap` objects (or subclasses) and `FALSE` for all other objects.

Usage

```
is_bm_bitmap(x)
```

Arguments

x	An object

Value

TRUE or FALSE

See Also

bm_bitmap()

Examples

```
space_matrix <- matrix(0L, nrow = 16L, ncol = 16L)
is_bm_bitmap(space_matrix)
space_glyph <- bm_bitmap(space_matrix)
is_bm_bitmap(space_glyph)
```

---

is_bm_font *Test if the object is a bitmap font object*

Description

is_bm_font() returns TRUE for bm_font objects (or subclasses) and FALSE for all other objects.

Usage

```
is_bm_font(x)
```

Arguments

x	An object

Value

TRUE or FALSE

See Also

bm_font()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
is_bm_font(font)
```

is_bm_list *Test if the object is a bitmap glyph list object*

Description

`is_bm_list()` returns `TRUE` for `bm_list()` objects (or subclasses) and `FALSE` for all other objects.

Usage

```
is_bm_list(x, class = NULL)
```

Arguments

`x`	An object
`class`	If `NULL` (the default) don't check the class of the elements. Otherwise a character vector: returns `TRUE` only if all elements (if any) of `x` inherit from at least one of the classes in `class`.

Value

`TRUE` or `FALSE`

See Also

`bm_list()`

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
is_bm_list(font)
is_bm_list(font, class = "bm_bitmap")
is_bm_list(font, class = "bm_pixmap")
```

is_bm_pixmap *Test if the object is a pixmap object*

Description

`is_bm_pixmap()` returns `TRUE` for `bm_pixmap` objects (or subclasses) and `FALSE` for all other objects.

Usage

```
is_bm_pixmap(x)
```

Arguments

x	An object

Value

TRUE or FALSE

See Also

bm_pixmap(), as_bm_pixmap()

Examples

```
pm <- bm_pixmap(matrix(c("red", "blue", "green", "black"),
                nrow = 2L, byrow = TRUE))
is_bm_pixmap(pm)
```

---

is_supported_bitmap *Test if the object is a bitmap object supported by the methods in this package*

Description

is_supported_bitmap() returns TRUE for bm_bitmap, bm_pixmap, magick-image, nativeRaster, and raster objects (or subclasses) and FALSE for all other objects.

Usage

```
is_supported_bitmap(x)
```

Arguments

x	An object

Value

TRUE or FALSE

See Also

is_bm_bitmap(), is_bm_pixmap(), grDevices::is.raster()

Examples

```
space_matrix <- matrix(0L, nrow = 16L, ncol = 16L)
space_glyph <- bm_bitmap(space_matrix)
is_supported_bitmap(space_glyph)
```

`Ops.bm_bitmap` *S3 Ops group generic methods for bitmap objects*

Description

The S3 Ops group generic methods for `bm_bitmap()` objects are simply the result of the generic integer matrix method cast back to a `bm_bitmap()` object (which is an integer matrix). The S3 Ops group generic methods for `bm_list()` and `bm_font()` objects simply returns another object with that operator applied to every bitmap in the original object.

Usage

```
## S3 method for class 'bm_bitmap'
Ops(e1, e2)

## S3 method for class 'bm_pixmap'
Ops(e1, e2)

## S3 method for class 'bm_list'
Ops(e1, e2)
```

Arguments

`e1, e2`	objects.

Value

The various `Ops.bm_bitmap` and `Ops.bm_pixmap` methods return a `bm_bitmap()` object. The various `Ops.bm_list` methods return a `bm_list()` object.

See Also

base::Ops

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)

# Examples applied to individual bitmaps
capital_r <- font[[str2ucp("R")]]
print(!capital_r)
capital_b <- font[[str2ucp("B")]]
print(capital_r & capital_b)
print(capital_r | capital_b)
print(capital_r + 1L)
print(capital_r + 1L > 1L)

# Examples applied to `bm_list()` objects
```

```
bml <- font[c("U+0023", "U+0052", "U+0053", "U+0054", "U+0041", "U+0054", "U+0053")] # #RSTATS
bml <- as_bm_list(bml)
bm <- do.call(cbind, bml)
print(bm)

bml <- !bml
bm <- do.call(cbind, bml)
print(bm)

bml <- 2 * (bml + 1L)
bm <- do.call(cbind, bml)
print(bm)

crops <- farming_crops_16x16()
corn <- crops$corn$portrait
print(corn == col2hex("transparent"))
```

`plot.bm_matrix` *Plot bitmap/pixmap objects*

Description

`plot.bm_bitmap()` plots a `bm_bitmap()` object to the graphics device while `plot.bm_pixmap()` plots a `bm_pixmap()` object to the graphics device. They are wrappers around `grid::grid.raster()` and `as.raster.bm_bitmap()` or `as.raster.bm_pixmap()`. which converts a bitmap glyph object to a raster object. `col_bitmap` is a builtin color string vectors intended for use with the `col` argument for casting `bm_bitmap()` objects to pixmap objects.

Usage

```
## S3 method for class 'bm_bitmap'
plot(
  x,
  ...,
  col = getOption("bittermelon.col", col_bitmap),
  interpolate = FALSE
)

## S3 method for class 'bm_pixmap'
plot(x, ..., interpolate = FALSE)

## S3 method for class 'bm_bitmap'
as.raster(
  x,
  native = FALSE,
  ...,
  col = getOption("bittermelon.col", col_bitmap)
)
```

```
## S3 method for class 'bm_pixmap'
as.raster(x, native = FALSE, ...)

col_bitmap
```

Arguments

x	A bm_bitmap() object
...	Passed to grid::grid.raster().
col	Character vector of R color specifications. First color is used for values equal to 0, second color for values equal to 1, etc.
interpolate	Passed to grid::grid.raster().
native	If TRUE return a "nativeRaster" object instead of a "raster" object. This will require that the suggested package farver is installed.

Format

An object of class character of length 4.

Value

plot.bm_bitmap() and plot.bm_pixmap() return a grid::rasterGrob() object silently. As a side effect will draw to graphics device. as.raster.bm_bitmap() and as.raster.bm_pixmap() return "raster" objects (see grDevices::as.raster()).

See Also

bm_bitmap(), bm_pixmap()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- bm_extend(font[[str2ucp("R")]], left = 1L)
capital_r <- bm_extend(capital_r, sides = 1L, value = 2L)  # add a border effect

plot(capital_r)

plot(capital_r, col = c("yellow", "blue", "red"))

crops <- farming_crops_16x16()
grapes <- crops$grapes$portrait
plot(grapes)
```

`print.bm_bitmap` *Print bitmap objects*

Description

`print.bm_bitmap()` prints a representation of bitmap objects to the terminal. It is a wrapper around `format.bm_bitmap()` which converts bitmap objects to a character vector. `px_unicode` and `px_ascii` are builtin character vectors intended for use with the `px` argument (the former contains Unicode "Block Elements" while the latter is purely ASCII). `px_auto()` chooses which character vector to use based on whether `cli::is_utf8_output()` is `TRUE` or not.

Usage

```
## S3 method for class 'bm_bitmap'
print(
  x,
  ...,
  px = getOption("bittermelon.px", px_auto()),
  fg = getOption("bittermelon.fg", FALSE),
  bg = getOption("bittermelon.bg", FALSE),
  compress = getOption("bittermelon.compress", "none"),
  downscale = getOption("bittermelon.downscale", FALSE)
)

## S3 method for class 'bm_bitmap'
format(
  x,
  ...,
  px = getOption("bittermelon.px", px_auto()),
  fg = getOption("bittermelon.fg", FALSE),
  bg = getOption("bittermelon.bg", FALSE),
  compress = getOption("bittermelon.compress", "none"),
  downscale = getOption("bittermelon.downscale", FALSE)
)

px_unicode

px_ascii

px_auto(unicode = px_unicode, ascii = px_ascii)
```

Arguments

`x`	A `bm_bitmap()` object
`...`	Further arguments passed to or from other methods.
`px`	Character vector of the pixel to use for each integer value i.e. The first character for integer `0L`, the second character for integer `1L`, and so on. Will be recycled.

fg	R color strings of foreground colors to use and/or cli ANSI style functions of class cli_ansi_style. FALSE (default) for no foreground colors. Will be recycled and passed to cli::make_ansi_style().
bg	R color strings of background colors to use and/or cli ANSI style functions of class cli_ansi_style. FALSE (default) for no background colors. Will be recycled and passed to cli::make_ansi_style() with bg = TRUE.
compress	If "none" (default) or "n" don't compress first with bm_compress(). Otherwise compress first with bm_compress() passing the value of compress as its direction argument (i.e. either "vertical" or "v", "horizontal" or "h", OR "both" or "b").
downscale	If TRUE and the printed bitmap will be wider than getOption("width") then shrink the image to fit getOption("width") using bm_downscale().
unicode	Character vector to use if cli::is_utf8_output() is TRUE.
ascii	Character vector to use if cli::is_utf8_output() is FALSE.

Format

An object of class character of length 20.

An object of class character of length 20.

Value

A character vector of the string representation (print.bm_bitmap() does this invisibly). As a side effect print.bm_bitmap() prints out the string representation to the terminal.

Fonts and terminal settings

Printing bitmaps/pixmaps may or may not look great in your terminal depending on a variety of factors:

- The terminal should support the Unicode - UTF-8 encoding. We use cli::is_utf8_output() to guess Unicode support which in turn looks at getOption("cli.unicode") and l10n_info().
- The terminal should support ANSI sequences and if it does it should support many colors.
  - We use cli::num_ansi_colors() to detect number of colors supported. num_ansi_colors() detection algorithm is complicated but it first looks at getOption("cli.num_colors").
  - If cli::num_ansi_colors() equals 16777216 then your terminal supports 24-bit ANSI colors.
  - If using the Windows Command Prompt window you may need to enable ANSI sequences support by doing REG ADD HKCU\CONSOLE /f /v VirtualTerminalLevel /t REG_DWORD /d 1 from the command-line or running regedit (Registry Editor) and go to Computer\HKEY_CURRENT_USER\Console and set VirtualTerminalLevel to 1.
- The font used by the terminal should be a monoscale font that supports the Block Elements Unicode block.
- The terminal text settings should have a cell spacing around 1.00 times width and 1.00 times height. For terminals configured by CSS styles this means a line-height of around 1.0.

See Also

bm_bitmap()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
bm_R <- font[[str2ucp("R")]]
print(bm_R)

if (cli::is_utf8_output())
  print(bm_R, px = px_unicode, compress = "vertical")

bm_8 <- font[[str2ucp("8")]]
bm_8_with_border <- bm_extend(bm_extend(bm_8, left = 1L),
                              sides = 1L, value = 2L)
print(bm_8_with_border, px = c(".", "@", "X"))

if (cli::num_ansi_colors() >= 16L) {
  print(bm_8_with_border, px = " ",
        bg = c(cli::bg_br_white, cli::bg_blue, cli::bg_red))
}
```

---

print.bm_pixmap *Print pixmap objects*

Description

print.bm_pixmap() prints bittermelon pixmap objects to the terminal It is a wrapper around format.bm_pixmap().

Usage

```
## S3 method for class 'bm_pixmap'
print(
  x,
  ...,
  bg = getOption("bittermelon.bg", FALSE),
  compress = getOption("bittermelon.compress", "none"),
  downscale = getOption("bittermelon.downscale", FALSE)
)

## S3 method for class 'bm_pixmap'
format(
  x,
  ...,
  bg = getOption("bittermelon.bg", FALSE),
  compress = getOption("bittermelon.compress", "none"),
```

```
  downscale = getOption("bittermelon.downscale", FALSE)
)
```

Arguments

x	A bm_pixmap() object
...	Currently ignored.
bg	R color string of background color to use and/or cli ANSI style function of class cli_ansi_style. FALSE (default) for no background color (i.e. use default terminal background).
compress	How to print the image: * "none" (default) or "n" use one character per pixel. * "vertical" or "v" use one character per two vertical pixels (makes pixels look closest to square in typical terminal). * "horizontal" or "h" use one character per two horizontal pixels. * "both" or "b" use one character per four pixels (this will be a lossy conversion whenever there are more than two colors per four pixels).
downscale	If TRUE and the printed pixmap will be wider than getOption("width") then shrink the image to fit getOption("width") using bm_downscale().

Value

A character vector of the string representation (print.bm_pixmap() does this invisibly). As a side effect print.bm_pixmap() prints out the string representation to the terminal.

Fonts and terminal settings

Printing bitmaps/pixmaps may or may not look great in your terminal depending on a variety of factors:

- The terminal should support the Unicode - UTF-8 encoding. We use cli::is_utf8_output() to guess Unicode support which in turn looks at getOption("cli.unicode") and l10n_info().
- The terminal should support ANSI sequences and if it does it should support many colors.
  - We use cli::num_ansi_colors() to detect number of colors supported. num_ansi_colors() detection algorithm is complicated but it first looks at getOption("cli.num_colors").
  - If cli::num_ansi_colors() equals 16777216 then your terminal supports 24-bit ANSI colors.
  - If using the Windows Command Prompt window you may need to enable ANSI sequences support by doing REG ADD HKCU\CONSOLE /f /v VirtualTerminalLevel /t REG_DWORD /d 1 from the command-line or running regedit (Registry Editor) and go to Computer\HKEY_CURRENT_USER\Console and set VirtualTerminalLevel to 1.
- The font used by the terminal should be a monoscale font that supports the Block Elements Unicode block.
- The terminal text settings should have a cell spacing around 1.00 times width and 1.00 times height. For terminals configured by CSS styles this means a line-height of around 1.0.

Examples

```
crops <- farming_crops_16x16()
corn <- crops$corn$portrait
if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn)
}

if (cli::is_utf8_output() && cli::num_ansi_colors() >= 256L) {
  print(corn, compress = "v", bg = cli::bg_br_white)
}

if (cli::is_utf8_output() &&
    cli::num_ansi_colors() > 256L &&
    getOption("width") >= 100L) {
  img <- png::readPNG(system.file("img", "Rlogo.png", package="png"))
  pm <- as_bm_pixmap(img)
  print(pm, compress = "v")
}
```

---

read_hex *Read and write hex bitmap font files*

Description

`read_hex()` reads in hex format bitmap font files as a `bm_font()` object while `write_hex()` writes a `bm_font()` object as a hex format bitmap font file.

Usage

```
read_hex(con, ucp = NULL)

write_hex(font, con = stdout())
```

Arguments

con	A connection object or a character string of a filename. See `base::readLines()` or `base::writeLines()` for more info. If it is a connection it will be explicitly closed.
ucp	Character vector of Unicode Code Points: glyphs not in this vector won't be read in. If `NULL` (default) read every glyph in the font.
font	A `bm_font()` object.

Value

`read_hex()` returns a `bm_font()` object. `write_hex()` returns invisibly a character vector of the contents of the hex font file it wrote to `con` as a side effect.

See Also

bm_font()

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r)

filename <- tempfile(fileext = ".hex.gz")
write_hex(font, gzfile(filename))

font <- read_hex(font_file, ucp = block2ucp("Basic Latin"))
capital_r <- font[[str2ucp("R")]]
print(capital_r)
```

---

read_monobit *Read and write bitmap font files using monobit*

Description

read_monobit() reads in bitmap font file as a bm_font() object while write_monobit() writes a bm_font() object as a bitmap font file. It uses the file extension to determine the appropriate bitmap font format to use.

Usage

```
read_monobit(
  file,
  ...,
  quietly = FALSE,
  monobit_path = getOption("bittermelon.monobit_path", "monobit-convert")
)

write_monobit(
  font,
  file,
  quietly = FALSE,
  monobit_path = getOption("bittermelon.monobit_path", "monobit-convert")
)
```

Arguments

file	A character string of a filename.
...	Further arguments passed to read_yaff().
quietly	If TRUE suppress any standard output/error from monobit-convert.

monobit_path	Path/name of monobit-convert to use. Passed to base::Sys.which().
font	A bm_font() object.

Details

- read_monobit() and write_monobit() require that the monobit-convert command is available on the system.
- read_monobit() and write_monobit() uses monobit-convert to convert to/from the yaff font format which this package can natively read/write from/to.
- One may install monobit-convert using pip3 install monobit.
- For more information about monobit see https://github.com/robhagemans/monobit.

Value

read_monobit() returns a bm_font() object. write_monobit() returns NULL invisibly and as a side effect writes file.

See Also

bm_font() for more information about bitmap font objects. read_hex(), write_hex(), read_yaff(), write_yaff() for pure R bitmap font readers and writers.

Examples

```
# May take more than 5 seconds on CRAN servers
if (nzchar(Sys.which("monobit-convert"))) {
  try({
    font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
    font <- read_monobit(font_file)
    capital_r <- font[[str2ucp("R")]]
    print(capital_r)

    filename <- tempfile(fileext = ".yaff")
    write_monobit(font, filename)
  })
}
```

---

read_yaff *Read and write yaff bitmap font files*

Description

read_yaff() reads in yaff format bitmap font files as a bm_font() object while write_yaff() writes a bm_font() object as a yaff format bitmap font file.

Usage

```
read_yaff(con, ..., fg = "#000000FF")

write_yaff(font, con = stdout())
```

Arguments

con	A connection object or a character string of a filename. See base::readLines() or base::writeLines() for more info. If it is a connection it will be explicitly closed.
...	Currently ignored.
fg	Foreground color used when reading greyscale fonts (i.e. when the yaff levels property is greater than 2). An R color string in any format accepted by col2hex(). Its RGB values are used for the pixel color and its alpha is multiplied by the level fraction to determine pixel alpha. Default is "#000000FF" (opaque black).
font	A bm_font() object.

Value

read_yaff() returns a bm_font() object. write_yaff() returns invisibly a character vector of the contents of the yaff font file it wrote to con as a side effect.

See Also

bm_font() for information about bitmap font objects. For more information about yaff font format see https://github.com/robhagemans/monobit#the-yaff-format.

Examples

```
# May take more than 5 seconds on CRAN servers
font_file <- system.file("fonts/fixed/4x6.yaff.gz", package = "bittermelon")
font <- read_yaff(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r)

filename <- tempfile(fileext = ".yaff")
write_yaff(font, filename)
```

---

summary.bm_font *Summarize a bitmap font*

Description

summary.bm_font() summarizes a bm_font() object, showing the font name (if available), the number of characters, and coverage of Unicode blocks.

Usage

```
## S3 method for class 'bm_font'
summary(object, ...)

## S3 method for class 'summary_bm_font'
print(x, ...)
```

Arguments

object	A bm_font() object.
...	Currently ignored.
x	A summary_bm_font object.

Value

A summary_bm_font object (a list) with the following components:

name Font name string, or NULL if not available.

bitmap_class The bitmap class of the glyphs (e.g. "bm_bitmap" or "bm_pixmap"), or NULL if the font is empty.

n_chars Number of characters in the font.

coverage A data frame with columns block (Unicode block name), n (number of font characters in that block), and total (total number of named code points in that block). Rows are sorted in Unicode code point order and include only blocks with at least one character in the font.

See Also

bm_font()

Examples

```
font <- read_yaff(system.file("fonts/fixed/4x6.yaff.gz", package = "bittermelon"))
summary(font)
```

Summary.bm_list *max, min, and range for bitmap objects*

Description

max(), min(), and range() will provide the maximum and minimum integer values found in the bm_bitmap(), bm_list(), or bm_list() objects. The other four S3 base::Summary methods - all(), any(), sum, and prod - are only supported for bm_bitmap() objects (which are subclasses of integer matrices).

Usage

```
## S3 method for class 'bm_list'
Summary(..., na.rm = FALSE)
```

Arguments

`...`	Passed to relevant functions.
`na.rm`	Passed to `min()` and `max()`.

Value

An integer vector.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
min(font)
max(font)
range(font)
```

---

ucp2label *Other Unicode utilities*

Description

`ucp2label()` returns Unicode code point "labels" as a character vector. `ucp_sort()` sorts Unicode code points. `is_combining_character()` returns `TRUE` if the character is a "combining" character.

Usage

```
ucp2label(x)

ucp_sort(x, decreasing = FALSE)

is_combining_character(x, pua_combining = character(0))
```

Arguments

`x`	A character vector of Unicode code points.
`decreasing`	If `TRUE` do a decreasing sort.
`pua_combining`	Additional Unicode code points to be considered as a "combining" character such as characters defined in the Private Use Area (PUA) of a font.

Value

`ucp2label()` returns a character vector of Unicode labels. `ucp_sort()` returns a character vector of Unicode code points. `is_combining_character()` returns a logical vector.

See Also

block2ucp(), hex2ucp(), int2ucp(), name2ucp(), range2ucp(), and str2ucp() all return Unicode code points.

Examples

```
# Get the Unicode Code Point "label" for "R"
ucp2label(str2ucp("R"))

is_combining_character(str2ucp("a"))
is_combining_character("U+0300") # COMBINING GRAVE ACCENT
```

[.bm_matrix *Extract or replace parts of a bitmap/pixmap matrix*

Description

[.bm_matrix() is defined so that it returns a bm_bitmap() or bm_pixmap() object (if the value is a matrix). [<-.bm_bitmap() casts any replacement values as integers while [<-.bm_pixmap() casts any replacement values as standardized color strings.

Usage

```
## S3 method for class 'bm_matrix'
x[i, j, ..., drop = TRUE]

## S3 replacement method for class 'bm_bitmap'
x[i, j, ...] <- value

## S3 replacement method for class 'bm_pixmap'
x[i, j, ...] <- value
```

Arguments

x	bm_bitmap() object
i, j	indices specifying elements to extract or replace. See [base::[()] for more information.
...	Passed to [base::[()].
drop	If TRUE the result is coerced to a integer vector.
value	Replacement value

Value

[.bm_matrix() returns a bm_bitmap() or bm_pixmap() object if the value is a matrix and/or drop is FALSE otherwise it returns a vector of integers or color strings.

Examples

```
font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
font <- read_hex(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r[4:14,2:8])
capital_r[11:13,3:5] <- 2L
print(capital_r)
```

Index