

# Package ‘bittermelon’

June 25, 2024

**Type** Package

**Title** Bitmap Tools

**Version** 2.0.2

**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

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

**Suggests** farver, gridpattern, hexfont, knitr, magick, mazing, ragg, rmarkdown, testthat, vdiff, withr

**VignetteBuilder** knitr, rmarkdown

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

**Encoding** UTF-8

**RoxygenNote** 7.3.1

**NeedsCompilation** no

**Author** Trevor L. Davis [aut, cre] (<<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** 2024-06-25 14:50:02 UTC

## Contents

as.array.bitmap	3
as.matrix.bitmap	4
as_bitmap	5
as_bm_font	9
as_bm_list	10
as_bm_pixmap	11
bm_bitmap	14
bm_call	15
bm_clamp	16
bm_compose	17
bm_compress	18
bm_distort	20
bm_edit	21
bm_expand	22
bm_extend	24
bm_flip	27
bm_font	29
bm_gray	30
bm_heights	31
bm_invert	32
bm_lapply	33
bm_list	34
bm_mask	35
bm_options	38
bm_outline	38
bm_overlay	40
bm_pad	42
bm_padding_lengths	45
bm_pixmap	46
bm_print	47
bm_replace	49
bm_resize	50
bm_rotate	53
bm_shadow	54
bm_shift	59
bm_trim	61
c.bitmap	64
cbind.bitmap	65
col2hex	66
col2int	67
farmland_crops_16x16	68
hex2ucp	69
is_bitmap	70
is_bm_font	71
is_bm_list	72
is_bm_pixmap	72

Ops.bm_bitmap . . . . .	73
plot.bm_matrix . . . . .	74
print.bm_bitmap . . . . .	76
print.bm_pixmap . . . . .	78
read_hex . . . . .	80
read_monobit . . . . .	81
read_yaff . . . . .	82
Summary.bm_list . . . . .	83
ucp2label . . . . .	84
[.bm_matrix . . . . .	85

## Index 86

---

as.array.bm_bitmap	<i>Cast bitmap/pixmap objects to an array</i>
--------------------	---

---

### 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 <a href="#">bm_bitmap()</a> or <a href="#">bm_pixmap()</a> object.
...	Currently ignored.
first_row_is_top	If TRUE the first row of the matrix will represent the top of the bitmap (like <a href="#">grDevices::as.raster()</a> objects). If FALSE the first row of the matrix will represent the bottom of the bitmap (like <a href="#">bm_bitmap()</a> and <a href="#">bm_pixmap()</a> 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.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	<i>Cast to a bitmap matrix object</i>
--------------	---------------------------------------

---

## 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 '`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 <code>bm_bitmap()</code> object.
...	Further arguments passed to or from other methods.
mode	Method to determine the integer values of the <code>bm_bitmap()</code> object: <b>alpha</b> Higher alpha values get a 1L. <b>darkness</b> Higher darkness values get a 1L. $\text{darkness} = (1 - \text{luma}) * \text{alpha}$ . <b>brightness</b> Higher brightness values get a 1L. $\text{brightness} = \text{luma} * \text{alpha}$ .
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 purely vertical binding either "top-to-bottom" (default) or its aliases "tb" 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 "tblr" 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 <code>bm_font()</code> object that contains all the characters within x.

hjust	Used by <code>bm_extend()</code> if bitmap widths are different.
vjust	Used by <code>bm_extend()</code> if bitmap heights are different.
compose	Compose graphemes using <code>bm_compose()</code> .
pua_combining	Passed to <code>bm_compose()</code> .
width	Desired width of bitmap
height	Desired height of bitmap
png_device	A function taking arguments <code>filename</code> , <code>width</code> , and <code>height</code> that starts a graphics device that saves a png image with a transparent background. By default will use <code>ragg::agg_png()</code> if available else the “cairo” version of <code>grDevices::png()</code> if available else just <code>grDevices::png()</code> .
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 <code>mazing::find_maze_refpoint()</code> .
solve	If TRUE then mark the solution path from <code>start</code> to <code>end</code> as value 3L. See <code>mazing::solve_maze()</code> .

### 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 <code>bm_list()</code> object.
...	Further arguments passed to or from other methods.
FUN	Function to apply to every element of a list such as <code>as_bm_bitmap()</code> or <code>as_bm_pixmap()</code> .
font	A <code>bm_font()</code> 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)
bm1 <- as_bm_list("RSTATS", font = font)
bm1 <- bm_extend(bm1, sides = 1L, value = 0L)
bm1 <- bm_extend(bm1, sides = c(2L, 1L), value = 2L)
bm <- do.call(cbind, bm1)
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, ...)

## 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 '`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. <code>as_bm_pixmap.default()</code> passes ... to <code>as.raster()</code> .
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 <code>ragg::agg_png()</code> if available else the “cairo” version of <code>grDevices::png()</code> if available else just <code>grDevices::png()</code> .
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 <code>mazing::find_maze_refpoint()</code> .
solve	If TRUE then mark the solution path from start to end as value 3L. See <code>mazing::solve_maze()</code> .

**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\_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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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	<i>Clamp bitmap values</i>
----------	----------------------------

---

### 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 <a href="#">bm_bitmap()</a> , <a href="#">bm_font()</a> , <a href="#">bm_list()</a> , <a href="#">"magick-image"</a> , <a href="#">"nativeRaster"</a> , <a href="#">bm_pixmap()</a> , or <a href="#">"raster"</a> 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 glyphs (composing new graphemes).

**Usage**

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

**Arguments**

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

**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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , " <code>magick-image</code> ", " <code>nativeRaster</code> ", <code>bm_pixmap()</code> , or " <code>raster</code> " 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](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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , <code>"magick-image"</code> , <code>"nativeRaster"</code> , <code>bm_pixmap()</code> , or <code>"raster"</code> object.
width	Desired width of bitmap
height	Desired height of bitmap

...	Additional arguments to be passed to or from methods.
filter	Passed to <code>magick::image_resize()</code> . Use <code>magick::filter_types()</code> 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 <a href="#">bm_bitmap()</a> , <a href="#">bm_font()</a> , <a href="#">bm_list()</a> , <a href="#">"magick-image"</a> , <a href="#">"nativeRaster"</a> , <a href="#">bm_pixmap()</a> , or <a href="#">"raster"</a> 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"
)

## 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"
)

## 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"
)

## 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"
)

## 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"
)

## 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"
)

```

### Arguments

x	Either a <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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 <code>hjust</code> argument or just one of either the <code>left</code> or <code>right</code> arguments.
height	How many pixels tall should the new bitmap be. Use with the <code>vjust</code> argument or just one of either the <code>top</code> or <code>bottom</code> 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.

### 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\_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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , <code>"magick-image"</code> , <code>"nativeRaster"</code> , <code>bm_pixmap()</code> , or <code>"raster"</code> object.
direction	Either <code>"vertical"</code> or <code>"v"</code> , <code>"horizontal"</code> or <code>"h"</code> , OR <code>"both"</code> or <code>"b"</code> .
in_place	If TRUE flip the glyphs in place (without changing any background padding).
value	Background padding value (to use if <code>in_place</code> 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	<i>Bitmap font object</i>
---------	---------------------------

---

## Description

bm\_font() creates a bitmap font object.

## Usage

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

## Arguments

x	Named list of <code>bm_bitmap()</code> objects. Names must be coercible by <code>Unicode::as.u_char()</code> .
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	<i>Gray a bitmap</i>
---------	----------------------

---

### 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	<i>Widths or heights of bitmaps</i>
------------	-------------------------------------

---

### 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	<i>Invert (negate) a bitmap</i>
-----------	---------------------------------

---

### 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 `bm_bitmap()` object as its first argument and returns a `bm_bitmap()` object.

... 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 <a href="#">bm_bitmap()</a> , <a href="#">bm_font()</a> , <a href="#">bm_list()</a> , "magick-image", "nativeRaster", <a href="#">bm_pixmap()</a> , 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 <a href="#">bm_bitmap()</a> object by <a href="#">as_bm_bitmap()</a> .
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	<i>Get bittermelon options</i>
------------	--------------------------------

---

### Description

bm\_options() returns the bittermelon packages 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	<i>Compute "outline" bitmap of a bitmap</i>
------------	---

---

### 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , <code>"magick-image"</code> , <code>"nativeRaster"</code> , <code>bm_pixmap()</code> , or <code>"raster"</code> 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , <code>"magick-image"</code> , <code>"nativeRaster"</code> , <code>bm_pixmap()</code> , or <code>"raster"</code> 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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\_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 <code>bm_pixmap()</code> . If not already a color string matrix it will be cast to one by <code>as_bm_pixmap()</code> .
---	---

**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 monospace 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", "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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , or "raster" object.
value	Value for the new pixels.
width	How many pixels wide should the new bitmap be. Use with the <code>hjust</code> argument or just one of either the <code>left</code> or <code>right</code> arguments.
height	How many pixels tall should the new bitmap be. Use with the <code>vjust</code> argument or just one of either the <code>top</code> or <code>bottom</code> 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	<i>Rotate bitmaps 0, 90, 180, or 270 degrees</i>
-----------	--

---

**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.

**Usage**

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

## S3 method for class 'bm_matrix'
bm_rotate(x, angle = 0L, clockwise = TRUE)

## S3 method for class 'nativeRaster'
bm_rotate(x, angle = 0L, clockwise = TRUE)

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

## S3 method for class 'raster'
bm_rotate(x, angle = 0L, clockwise = TRUE)

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

**Arguments**

x	Either a <a href="#">bm_bitmap()</a> , <a href="#">bm_font()</a> , <a href="#">bm_list()</a> , <a href="#">"magick-image"</a> , <a href="#">"nativeRaster"</a> , <a href="#">bm_pixmap()</a> , or <a href="#">"raster"</a> 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.
...	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. [bm\\_flip\(\)](#) with direction "both" and in\_place TRUE can rotate glyphs 180 degrees in place.

**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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , "magick-image", "nativeRaster", <code>bm_pixmap()</code> , 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 <code>bm_shadow()</code> and 0L for <code>bm_embolden()</code> .
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	<i>Shift elements within bitmaps</i>
----------	--------------------------------------

---

**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)

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

## 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
)

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

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

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

### Arguments

x	Either a <a href="#">bm_bitmap()</a> , <a href="#">bm_font()</a> , <a href="#">bm_list()</a> , <a href="#">"magick-image"</a> , <a href="#">"nativeRaster"</a> , <a href="#">bm_pixmap()</a> , or <a href="#">"raster"</a> 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.
...	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")
}
```

---

bm_trim	<i>Trim bitmaps</i>
---------	---------------------

---

### 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 <code>bm_bitmap()</code> , <code>bm_font()</code> , <code>bm_list()</code> , <code>"magick-image"</code> , <code>"nativeRaster"</code> , <code>bm_pixmap()</code> , or <code>"raster"</code> 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 <code>hjust</code> argument or just one of either the <code>left</code> or <code>right</code> arguments.
height	How many pixels tall should the new bitmap be. Use with the <code>vjust</code> argument or just one of either the <code>top</code> or <code>bottom</code> 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.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
<code>bm_bitmap()</code>	<code>bm_bitmap()</code>	<code>bm_list()</code>
<code>bm_bitmap()</code>	<code>bm_font()</code>	<code>bm_font()</code>
<code>bm_bitmap()</code>	<code>bm_list()</code>	<code>bm_list()</code>
<code>bm_bitmap()</code>	<code>bm_pixmap()</code>	<code>bm_list()</code>
<code>bm_pixmap()</code>	<code>bm_bitmap()</code>	<code>bm_list()</code>
<code>bm_pixmap()</code>	<code>bm_font()</code>	ERROR
<code>bm_pixmap()</code>	<code>bm_list()</code>	<code>bm_list()</code>
<code>bm_pixmap()</code>	<code>bm_pixmap()</code>	<code>bm_list()</code>
<code>bm_font()</code>	<code>bm_bitmap()</code>	<code>bm_font()</code>
<code>bm_font()</code>	<code>bm_font()</code>	<code>bm_font()</code>
<code>bm_font()</code>	<code>bm_list()</code>	<code>bm_font()</code>
<code>bm_font()</code>	<code>bm_pixmap()</code>	ERROR
<code>bm_list()</code>	<code>bm_bitmap()</code>	<code>bm_list()</code>
<code>bm_list()</code>	<code>bm_font()</code>	<code>bm_font()</code>
<code>bm_list()</code>	<code>bm_list()</code>	<code>bm_list()</code>
<code>bm_list()</code>	<code>bm_pixmap()</code>	<code>bm_list()</code>



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

---

## 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 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 <a href="#">Unicode::as.u_char()</a> for hex2ucp() and int2ucp(), <a href="#">base::utf8ToInt()</a> for str2ucp(), <a href="#">Unicode::u_char_from_name()</a> for name2ucp(), <a href="#">Unicode::as.u_char_range()</a> for range2ucp(), and <a href="#">Unicode::u_blocks()</a> for block2ucp().
type	one of "exact" or "grep", or an abbreviation thereof.
...	arguments to be passed to <a href="#">grep1</a> 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	<i>Test if the object is a bitmap glyph list object</i>
------------	---

---

**Description**

is\_bm\_list() returns TRUE for `bm_list()` objects (or subclasses) and FALSE for all other objects.

**Usage**

```
is_bm_list(x)
```

**Arguments**

x                   An object

**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_font(font)
```

---

is_bm_pixmap	<i>Test if the object is a pixmap object</i>
--------------	--

---

**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)
```

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_bml(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 <code>bm_bitmap()</code> object
...	Passed to <code>grid::grid.raster()</code> .
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 <code>grid::grid.raster()</code> .
native	If TRUE return a "nativeRaster" object instead of a "raster" object. This will require that the suggested package <code>farver</code> 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 <code>bm_bitmap()</code> 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 <code>cli_ansi_style</code> . FALSE (default) for no foreground colors. Will be recycled and passed to <code>cli::make_ansi_style()</code> .
bg	R color strings of background colors to use and/or cli ANSI style functions of class <code>cli_ansi_style</code> . FALSE (default) for no background colors. Will be recycled and passed to <code>cli::make_ansi_style()</code> with <code>bg = TRUE</code> .
compress	If "none" (default) or "n" don't compress first with <code>bm_compress()</code> . Otherwise compress first with <code>bm_compress()</code> passing the value of <code>compress</code> as its <code>direction</code> 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 <code>getOption("width")</code> then shrink the image to fit <code>getOption("width")</code> using <code>bm_downscale()</code> .
unicode	Character vector to use if <code>cli::is_utf8_output()</code> is TRUE.
ascii	Character vector to use if <code>cli::is_utf8_output()</code> 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 monospace 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 <code>bm_pixmap()</code> object
...	Currently ignored.
bg	R color string of background color to use and/or cli ANSI style function of class <code>cli_ansi_style</code> . 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 <code>getOption("width")</code> then shrink the image to fit <code>getOption("width")</code> using <code>bm_downscale()</code> .

### 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 monospace 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_bmpixmap(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 <code>base::readLines()</code> or <code>base::writeLines()</code> 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 <code>bm_font()</code> 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**

<code>file</code>	A character string of a filename.
<code>quietly</code>	If TRUE suppress any standard output/error from <code>monobit-convert</code> .
<code>monobit_path</code>	Path/name of <code>monobit-convert</code> to use. Passed to <code>base::Sys.which()</code> .
<code>font</code>	A <a href="#">bm_font()</a> 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 (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)
```

```
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.

`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\_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	<a href="#">bm_bitmap()</a> object
i, j	indices specifying elements to extract or replace. See [ <a href="#">base::[()]</a> ] for more information.
...	Passed to [ <a href="#">base::[()]</a> ].
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

- \* **datasets**
  - plot.bm\_matrix, 74
  - print.bm\_bitmap, 76
  - .S3method(), 48
  - [.bm\_bitmap, 14, 47
  - [.bm\_bitmap ([.bm\_matrix), 85
  - [.bm\_matrix, 85
  - [.bm\_pixmap ([.bm\_matrix), 85
  - [<-.bm\_bitmap ([.bm\_matrix), 85
  - [<-.bm\_pixmap ([.bm\_matrix), 85
- as.array.bm\_bitmap, 3
- as.array.bm\_pixmap
  - (as.array.bm\_bitmap), 3
- as.matrix.bm\_bitmap
  - (as.matrix.bm\_matrix), 4
- as.matrix.bm\_bitmap(), 14
- as.matrix.bm\_matrix, 4
- as.matrix.bm\_pixmap
  - (as.matrix.bm\_matrix), 4
- as.matrix.bm\_pixmap(), 47
- as.raster(), 12
- as.raster.bm\_bitmap (plot.bm\_matrix), 74
- as.raster.bm\_bitmap(), 14, 47
- as.raster.bm\_pixmap (plot.bm\_matrix), 74
- as\_bm\_bitmap, 5
- as\_bm\_bitmap(), 10, 14, 36, 47
- as\_bm\_font, 9
- as\_bm\_font(), 29
- as\_bm\_list, 10
- as\_bm\_list(), 34
- as\_bm\_pixmap, 11
- as\_bm\_pixmap(), 10, 46–48, 73

  

- base::do.call(), 15
- base::lapply(), 34
- base::Ops, 73
- base::readLines(), 80, 83
- base::Summary, 83
- base::Sys.which(), 81
- base::unique(), 32
- base::utf8ToInt(), 69
- base::writeLines(), 80, 83
- bittermelon, 38
- block2ucp (hex2ucp), 69
- block2ucp(), 84
- bm\_bitmap, 14
- bm\_bitmap(), 3, 4, 8, 15–24, 26, 28–31, 33, 34, 36, 39, 41, 42, 44, 46, 47, 50, 52, 53, 58, 60, 62–66, 71, 73–75, 78, 85
- bm\_bold (bm\_shadow), 54
- bm\_call, 15
- bm\_clamp, 16
- bm\_clamp(), 14, 22
- bm\_compose, 17
- bm\_compose(), 8
- bm\_compress, 18
- bm\_compress(), 77
- bm\_distort, 20
- bm\_distort(), 18, 53
- bm\_downscale (bm\_distort), 20
- bm\_downscale(), 77, 79
- bm\_edit, 21
- bm\_expand, 22
- bm\_expand(), 21, 26
- bm\_extend, 24
- bm\_extend(), 8, 23, 45, 52, 58, 60, 63, 66
- bm\_flip, 27
- bm\_flip(), 53
- bm\_font, 29
- bm\_font(), 7, 9, 10, 15–21, 23, 26, 28, 30, 31, 33, 34, 36, 39, 41, 42, 44, 46, 50, 52, 53, 58, 60, 62–65, 71, 80–83
- bm\_format (bm\_print), 47
- bm\_glow (bm\_shadow), 54
- bm\_gray, 30
- bm\_grey (bm\_gray), 30
- bm\_heights, 31
- bm\_invert, 32

- bm\_lapply, 33
- bm\_list, 34
- bm\_list(), 10, 11, 15–21, 23, 26, 28–31, 33, 34, 36, 39, 41, 42, 44, 46, 50, 52, 53, 58, 60, 62–65, 68, 72, 73
- bm\_mask, 35
- bm\_options, 38
- bm\_outline, 38
- bm\_overlay, 40
- bm\_overlay(), 17
- bm\_pad, 42
- bm\_pad(), 26, 52, 63
- bm\_padding\_lengths, 45
- bm\_pixmap, 46
- bm\_pixmap(), 3, 4, 13–21, 23, 26, 28, 30, 31, 33, 36, 39, 41, 42, 44, 46, 48, 50, 52, 53, 58, 60, 62–66, 68, 73–75, 79, 85
- bm\_print, 47
- bm\_replace, 49
- bm\_resize, 50
- bm\_resize(), 21, 23, 26, 45, 63
- bm\_rotate, 53
- bm\_shadow, 54
- bm\_shift, 59
- bm\_shift(), 58
- bm\_trim, 61
- bm\_trim(), 26, 45, 52, 60
- bm\_widths (bm\_heights), 31
- c.bm\_bitmap, 64
- c.bm\_font (c.bm\_bitmap), 64
- c.bm\_list (c.bm\_bitmap), 64
- c.bm\_pixmap (c.bm\_bitmap), 64
- cbind.bm\_bitmap, 65
- cbind.bm\_bitmap(), 14
- cbind.bm\_pixmap (cbind.bm\_bitmap), 65
- cli::is\_utf8\_output(), 48, 76, 77, 79
- cli::make\_ansi\_style(), 77
- cli::num\_ansi\_colors(), 48, 77, 79
- col2hex, 66
- col2hex(), 67
- col2int, 67
- col\_bitmap (plot.bm\_matrix), 74
- farming\_crops\_16x16, 68
- farver, 67, 75
- format.bm\_bitmap (print.bm\_bitmap), 76
- format.bm\_bitmap(), 14, 47, 48
- format.bm\_pixmap (print.bm\_pixmap), 78
- format.bm\_pixmap(), 47, 48
- grDevices::as.raster(), 3, 4, 75
- grDevices::col2rgb(), 66
- grDevices::png(), 8, 13
- grDevices::rgb(), 66
- grepl, 69
- grid::grid.raster(), 74, 75
- grid::rasterGrob(), 75
- grid::viewport(), 53
- hex2ucp, 69
- hex2ucp(), 29, 84
- int2col (col2int), 67
- int2ucp (hex2ucp), 69
- int2ucp(), 84
- is\_bmp\_bitmap, 70
- is\_bmp\_bitmap(), 14
- is\_bmp\_font, 71
- is\_bmp\_font(), 29
- is\_bmp\_list, 72
- is\_bmp\_list(), 34
- is\_bmp\_pixmap, 72
- is\_bmp\_pixmap(), 13, 47
- is\_combining\_character (ucp2label), 84
- is\_combining\_character(), 17, 70
- is\_ucp (hex2ucp), 69
- l10n\_info(), 48, 77, 79
- magick-image, 17, 19, 21, 23, 26, 28, 30, 33, 39, 42, 44, 50, 52, 53, 58, 60, 63
- magick::filter\_types(), 19, 21
- magick::image\_resize(), 19–21
- mazing::find\_maze\_refpoint(), 8, 13
- mazing::solve\_maze(), 8, 13
- name2ucp (hex2ucp), 69
- name2ucp(), 84
- Ops.bm\_bitmap, 73
- Ops.bm\_bitmap(), 14
- Ops.bm\_list (Ops.bm\_bitmap), 73
- Ops.bm\_pixmap (Ops.bm\_bitmap), 73
- options(), 38
- plot.bm\_bitmap (plot.bm\_matrix), 74
- plot.bm\_bitmap(), 14, 47
- plot.bm\_matrix, 74

`plot.bm_pixmap(plot.bm_matrix)`, 74  
`png::writePNG()`, 3  
`print.bm_bitmap`, 76  
`print.bm_bitmap()`, 14  
`print.bm_pixmap`, 78  
`print.bm_pixmap()`, 47  
`px_ascii(print.bm_bitmap)`, 76  
`px_auto(print.bm_bitmap)`, 76  
`px_unicode(print.bm_bitmap)`, 76

`ragg::agg_png()`, 8, 13  
`range2ucp(hex2ucp)`, 69  
`range2ucp()`, 84  
`raster`, 17, 19, 21, 23, 26, 28, 30, 33, 39, 42, 44, 50, 52, 53, 58, 60, 63  
`rbind.bm_bitmap(cbind.bm_bitmap)`, 65  
`rbind.bm_bitmap()`, 14  
`rbind.bm_pixmap(cbind.bm_bitmap)`, 65  
`read_hex`, 80  
`read_hex()`, 82  
`read_monobit`, 81  
`read_yaff`, 82  
`read_yaff()`, 82

`str2ucp(hex2ucp)`, 69  
`str2ucp()`, 84  
`Summary.bm_list`, 83

`ucp2label`, 84  
`ucp2label()`, 70  
`ucp_sort(ucp2label)`, 84  
`Unicode::as.u_char()`, 29, 69  
`Unicode::as.u_char_range()`, 69  
`Unicode::u_blocks()`, 69  
`Unicode::u_char_from_name()`, 69  
`utils::file.edit()`, 22

`withr::local_options()`, 38  
`withr::with_options()`, 38  
`write_hex(read_hex)`, 80  
`write_hex()`, 82  
`write_monobit(read_monobit)`, 81  
`write_yaff(read_yaff)`, 82  
`write_yaff()`, 82