Package ‘bittermelon’

September 9, 2021

Type    Package
Title   Monochrome Bitmap Font Tools
Version 0.1.3

Description Provides functions for creating and modifying bitmaps with special emphasis on bitmap fonts and their glyphs. Provides native read/write support for the 'hex' and 'yaff' bitmap font formats and if 'Python' is installed can also read/write several more bitmap font formats using an embedded version of 'monobit'.

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

BugReports https://github.com/trevorld/bittermelon/issues
License MIT + file LICENSE
Imports findpython, grDevices, grid, png, Unicode, utils
Suggests crayon, ragg, testthat, vdiffr, withr

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author Trevor L Davis [aut, cre],
    Rob Hagemans [cph] (Author of included 'monobit' library),
    Frederic Cambus [cph] (Author of included 'Spleen' font)

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

Repository CRAN

Date/Publication 2021-09-09 05:50:02 UTC

R topics documented:

as.matrix.bm_bitmap ........................................... 2
as_bm_bitmap .................................................. 3
as_bm_font ..................................................... 5
as_bm_list ..................................................... 6
bm_bitmap ....................................................... 7
bm_call ........................................................ 8
as.matrix.bm_bitmap

Coerce bitmap objects to matrix

Description

as.matrix.bm_bitmap() coerces bm_bitmap() objects to an integer matrix.
as_bm_bitmap

Usage

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

Arguments

x
A bm_bitmap() object

... Further arguments passed to or from other methods.

Value

An integer matrix

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 Coerce to bitmap glyph objects

Description

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

Usage

as_bm_bitmap(x, ...)

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

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

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

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

x  An object that can reasonably be coerced to a `bm_bitmap()` object.
...

direction  For horizontal binding either "left-to-right" (default) or its aliases "ltr" and "lr" OR "right-to-left" or its aliases "rtl" and "rl". For vertical binding either "top-to-bottom" (default) or its aliases "ttb" and "tb" OR "bottom-to-top" or its aliases "btt" and "bt". The direction argument is not case-sensitive.

font  A `bm_font()` object that contains all the characters within x.

width  Desired width of bitmap

height  Desired height of bitmap

png_device  A function taking arguments filename, width, and height that starts a graphics device that saves a png image with a transparent background. By default will use `ragg::agg_png()` if available else the “cairo” version of `grDevices::png()` if available else just `grDevices::png()`.

threshold  If any png channel weakly exceeds this threshold (on an interval from zero to one) then the pixel is determined to be “black”.

Value

A `bm_bitmap()` object.

See Also

`bm_bitmap()`

Examples

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

if (require("grid") && capabilities("png")) {
  circle <- as_bm_bitmap(circleGrob(r = 0.25), width = 16L, height = 16L)
polygon(circle, px = c(".", ";"))

  inverted_exclamation <- as_bm_bitmap(textGrob("!", rot = 180),
                          width = 8L, height = 16L)
polygon(inverted_exclamation, px = c(".", ";"))
}
```
Description

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

Usage

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

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

## S3 method for class 'list'
```r
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

```r
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**

```r
as_bm_list(x, ...) # Default S3 method:
as_bm_list(x, ...) # S3 method for class 'list'
as_bm_list(x, ...) # S3 method for class 'character'
```

**Arguments**

- `x` An object that can reasonably be coerced to a `bm_list()` object.
- `...` Further arguments passed to or from other methods.
- `font` A `bm_font()` object that contains all the characters within `x`.

**Value**

A `bm_list()` object.

**See Also**

`bm_list()`

**Examples**

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

`bm_bitmap()` creates an S3 object representing 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

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. Color bitmaps are supported (the integer can be any non-negative integer) but we are unlikely to ever support exporting color bitmap fonts. Color bitmaps can be cast to black-and-white bitmaps via `bm_clamp()`.

Value

An integer matrix with a “bm_bitmap” subclass.

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
- `which.bm_bitmap()` (with `which()` re-defined as a generic)

See Also

`as_bm_bitmap()`, `is_bm_bitmap()`

Examples

```r
space <- bm_bitmap(matrix(0, nrow = 16, ncol = 16))
print(space, px = ".")
```
bm_call

Execute a function call on bitmap objects

Description

bm_call() executes 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(bm_object, .f, ...)

Arguments

bm_object Either a bm_bitmap(), bm_list(), or bm_font() 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, px = px_ascii)

bm_clamp

Clamp bitmap integer values.

Description

bm_clamp() "clamps" 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).

Usage

bm_clamp(bm_object, lower = 0L, upper = 1L, value = upper)
Arguments

**bm_object**
Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

**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]`.

Value

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

Examples

```r
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, px = c(".", ",", ",")

plus_sign_clamped <- bm_clamp(plus_sign_glyph)
print(plus_sign_clamped, px = c(".", ",", ",")
```

---

**bm_compress**
*Compress bitmaps using a "block elements" scheme*

Description

Compress bitmaps by a factor of two by re-mapping to a “block elements” scheme.

Usage

```r
bm_compress(bm_object, direction = "vertical")
```

Arguments

**bm_object**
Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

**direction**
Either "vertical" or "v", "horizontal" or "h", OR "both" or "b".
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

Either a bm_bitmap(), bm_list(), or bm_font() object.

See Also

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

Examples

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

---

bm_distort

**Resize images via distortion.**

Description

bm_distort() resize images to arbitrary width and height via distortion.

Usage

```r
bm_distort(
  bm_object,
  width = NULL,
  height = NULL,
  interpolate = FALSE,
  vp = NULL,
  png_device = NULL,
  threshold = 0.25
)
```
bm_edit

Edit a bitmap via text editor

Description

Edit a binary bitmap in a text editor.

Usage

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

Arguments

bitmap 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 bitmap() object.

Examples

```r
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, px = px_ascii)

## 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

```r
bm_expand(bm_object, width = 1L, height = 1L)
```

Arguments

- **bm_object**: Either a bitmap(), bmlist(), or bffont() object.
- **width**: An integer of how many times to repeat each column.
- **height**: An integer of how many times to repeat each row.

Value

Either a bitmap(), bmlist(), or bffont() object.
See Also

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

Examples

```r
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, px = px_ascii)
print(bm_expand(capital_r, width = 2L),
     px = px_ascii)
print(bm_expand(capital_r, height = 2L),
     px = px_ascii)
print(bm_expand(capital_r, width = 2L, height = 2L),
     px = px_ascii)
```

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

```r
bm_extend(
  bm_object,
  value = 0L,
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)
```

Arguments

- `bm_object`: Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.
- `value`: Integer value for the new pixels.
sides

If not NULL then an integer vector indicating how many pixels to pad on all four sides. If the integer vector is of length one it indicates the number of pixels for all four sides. If of length two gives first the number for the vertical sides and then the horizontal sides. If of length three gives the number of pixels for top, the horizontal sides, and then bottom sides. If of length four gives the number of pixels for top, right, bottom, and then left sides. This is the same scheme as used by the CSS padding and margin properties.

top

How many pixels to pad the top.

right

How many pixels to pad the right.

bottom

How many pixels to pad the bottom.

left

How many pixels to pad the left.

width

How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.

height

How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.

hjust

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

vjust

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

Value

Either a bm_bitmap(), bm_list(), or bm_font() object.

See Also

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

Examples

```r
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, px = c(" ", "#", ".", "@"))
```
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(bm_object, direction = "vertical", in_place = FALSE)

Arguments

bm_object Either a bm_bitmap(), bm_list(), or bm_font() object.
direction Either "vertical" or "v", "horizontal" or "h", OR "both" or "b".
in_place If TRUE flip the glyphs in place (without changing any white space padding).

Value

Either a bm_bitmap(), bm_list(), or bm_font() 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, px = px_ascii)

# Can also modify glyphs "in place"
exclamation <- font[[str2ucp("!")]]
exclamation_flipped <- bm_flip(exclamation, in_place = TRUE)
print(exclamation_flipped, px = px_ascii)

bm_font

Bitmap font object

Description

bm_font() creates a bitmap font object.
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, ...)

Usage

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

Arguments

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

Details

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

Value

A named list with a “bm_font” subclass.

See Also

is_bm_font(), as_bm_font(), hex2ucp()

Examples

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

# number of characters in font
length(font)

# print out "R"
R_glyph <- font[[str2ucp("R")]]
print(R_glyph, px = c(".", "+"))
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

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

# #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 zero.

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bm_object</code></td>
<td>Either a <code>bm_bitmap()</code>, <code>bm_list()</code>, or <code>bm_font()</code> object.</td>
</tr>
<tr>
<td><code>mask</code></td>
<td>A <code>bm_bitmap()</code> object to use as a “mask”. Only one of mask or base may be set.</td>
</tr>
<tr>
<td><code>base</code></td>
<td>A <code>bm_bitmap()</code> object which will be “masked” by mask. Only one of mask or base may be set.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>Either &quot;luminance&quot;</td>
</tr>
<tr>
<td><code>hjust</code></td>
<td>One of &quot;left&quot;, &quot;center-left&quot;, &quot;center-right&quot;, &quot;right&quot;. &quot;center-left&quot; and &quot;center-right&quot; will attempt to place in &quot;center&quot; if possible but if not possible will bias it one pixel left or right respectively. &quot;centre&quot;, &quot;center&quot;, and &quot;centre-left&quot; are aliases for &quot;center-left&quot;. &quot;centre-right&quot; is an alias for &quot;center-right&quot;.</td>
</tr>
<tr>
<td><code>vjust</code></td>
<td>One of &quot;bottom&quot;, &quot;center-bottom&quot;, &quot;center-top&quot;, &quot;top&quot;. &quot;center-bottom&quot; and &quot;center-top&quot; will attempt to place in &quot;center&quot; if possible but if not possible will bias it one pixel down or up respectively. &quot;centre&quot;, &quot;center&quot;, and &quot;centre-top&quot; are aliases for &quot;center-top&quot;. &quot;centre-bottom&quot; is an alias for &quot;center-bottom&quot;.</td>
</tr>
</tbody>
</table>
Details

If necessary bitmaps will be extended by \texttt{bm\_extend()} such that they are the same size. If necessary
the mask will be coerced into a “binary” mask. If mode is "luminance" then where the mask is \texttt{1L}
the corresponding pixel in base will be coerced to \texttt{0L}. If mode is "alpha" then where the mask is \texttt{0L}
the corresponding pixel in base will be coerced to \texttt{0L}.

Value

Either a \texttt{bm\_bitmap()}, \texttt{bm\_list()}, or \texttt{bm\_font()} object.

Examples

```r
if (require("grid") && 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, px = px_ascii)

  # U+2776 "Dingbat Negative Circled Digit One"
  circle_minus_one <- bm_mask(circle_large, one)
  print(circle_minus_one, px = px_ascii)

  # 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, px = px_ascii)

  circle_minus_lower_left <- bm_mask(circle_large, square_minus_lower_left, mode = "alpha")
  print(circle_minus_lower_left, px = px_ascii)
}
```

bm\_outline

\textit{Compute "outline" bitmap of a bitmap}

Description

\texttt{bm\_outline()} returns a bitmap that is just the “outline” of another bitmap.

Usage

\texttt{bm\_outline(bm\_object)}

Arguments

\texttt{bm\_object} \hspace{1cm} Either a \texttt{bm\_bitmap()}, \texttt{bm\_list()}, or \texttt{bm\_font()} object.
Value

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

Examples

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

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

```
---
bm_overlay  
Merge bitmaps by overlaying one over another

---

Description

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

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bm_object</code></td>
<td>Either a <code>bm_bitmap()</code>, <code>bm_list()</code>, or <code>bm_font()</code> object.</td>
</tr>
<tr>
<td><code>over</code></td>
<td>A <code>bm_bitmap()</code> object to overlay over the <code>bm_object</code> bitmap(s). Only one of over or under may be set.</td>
</tr>
<tr>
<td><code>under</code></td>
<td>A <code>bm_bitmap()</code> object which will be overlaid by the <code>bm_object</code> bitmap(s). Only one of over or under may be set.</td>
</tr>
<tr>
<td><code>hjust</code></td>
<td>One of &quot;left&quot;, &quot;center-left&quot;, &quot;center-right&quot;, &quot;right&quot;. &quot;center-left&quot; and &quot;center-right&quot; will attempt to place in &quot;center&quot; if possible but if not possible will bias it one pixel left or right respectively. &quot;centre&quot;, &quot;center&quot;, and &quot;centre-left&quot; are aliases for &quot;center-left&quot;. &quot;centre-right&quot; is an alias for &quot;center-right&quot;.</td>
</tr>
<tr>
<td><code>vjust</code></td>
<td>One of &quot;bottom&quot;, &quot;center-bottom&quot;, &quot;center-top&quot;, &quot;top&quot;. &quot;center-bottom&quot; and &quot;center-top&quot; will attempt to place in &quot;center&quot; if possible but if not possible will bias it one pixel down or up respectively. &quot;centre&quot;, &quot;center&quot;, and &quot;centre-top&quot; are aliases for &quot;center-top&quot;. &quot;centre-bottom&quot; is an alias for &quot;center-bottom&quot;.</td>
</tr>
</tbody>
</table>
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

Either a bm_bitmap(), bm_list(), or bm_font() object.

Examples

```r
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, px = px_ascii)

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

bm_pad

Adjust bitmap padding lengths

Description

bm_pad() adjusts bitmap padding lengths.

Usage

```r
bm_pad(
  bm_object,
  value = 0L,
  type = c("exact","extend","trim"),
  sides = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)
```

Arguments

- `bm_object`: Either a bm_bitmap(), bm_list(), or bm_font() object.
- `value`: Integer 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.

Value

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

See Also

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

Examples

```r
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, px = c('.', '#'))
capital_r_padded <- bm_pad(capital_r, sides = 2L)
print(capital_r_padded, px = c('.', '#'))
```

---

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

```r
bm_padding_lengths(bm_object, value = 0L)
```
bm_resize

Arguments

bm_object
Either a bm_bitmap(), bm_list(), or bm_font() object.

value
The value of the “padding” integer to compute lengths for.

Value

If bm_object 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 bm_object 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, px = c(".", "@"))
print(bm_padding_lengths(capital_r))

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(
  bm_object,
  value = 0L,
  width = NULL,
  height = NULL,
  hjust = "center-left",
  vjust = "center-top"
)

Arguments

bm_object
Either a bm_bitmap(), bm_list(), or bm_font() object.

value
Integer value for the new pixels.

width
How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.

height
How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.
hjust
One of "left", "center-left", "center-right", "right". "center-left" and "center-right" will attempt to place in "center" if possible but if not possible will bias it one pixel left or right respectively. "centre", "center", and "centre-left" are aliases for "center-left". "centre-right" is an alias for "center-right".

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

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

Value
Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

See Also
`bm_extend()`, `bm_pad()`, and `bm_trim()`.

Examples
```r
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, px = c("-", ")")
capital_r <- bm_resize(capital_r, width = 12L, height = 12L, vjust = "top")
print(capital_r, px = c("-", "")]
```

```
bm_rotate

Rotate bitmaps 0, 90, 180, or 270 degrees

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

Usage
`bm_rotate(bm_object, angle = 0, clockwise = TRUE)`

Arguments
- `bm_object` Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` 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.
**bm_shadow**

**Value**

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` 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**

```r
# 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), px = px_ascii)
print(bm_rotate(capital_r, 180), px = px_ascii)
print(bm_rotate(capital_r, 270), px = px_ascii)
print(bm_rotate(capital_r, 90, clockwise = FALSE), px = px_ascii)
```

---

**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**

```r
bm_shadow(
  bm_object,
  value = 2L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
  extend = TRUE
)
```

```r
bm_bold(
  bm_object,
  value = 1L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL,
)```
bm_shift

```r
extend = TRUE

bm_glow(bm_object, value = 2L, extend = TRUE, corner = FALSE)
```

### Arguments

- **bm_object**: Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.
- **value**: The integer value for the shadow, bold, or glow effect.
- **top**: How many pixels above should the shadow go.
- **right**: How many pixels right should the shadow go. If `top`, `right`, `bottom`, and `left` are all `NULL` then defaults to `1L`.
- **bottom**: How many pixels below should the shadow go. If `top`, `right`, `bottom`, and `left` are all `NULL` then defaults to `1L` for `bm_shadow()` and `0L` for `bm_embolden()`.
- **left**: How many pixels left should the shadow go.
- **extend**: Make the bitmap larger to give the new glyph more "room".
- **corner**: Fill in the corners.

### Value

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

### See Also

`bm_extend()` and `bm_shift()`

### Examples

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

---

### 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

```r
bm_shift(
  bm_object,
  value = 0L,
  top = NULL,
  right = NULL,
  bottom = NULL,
  left = NULL
)
```

Arguments

- `bm_object`: Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.
- `value`: Integer 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.

Details

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

Value

Either a `bm_bitmap()`, `bm_list()`, or `bm_font()` object.

See Also

`bm_trim()` and `bm_extend()`

Examples

```r
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, px = c("-", "+"))
capital_r <- bm_shift(capital_r, bottom = 2L, right = 1L)
print(capital_r, px = c("-", "+"))
```
bm_trim

Trim bitmaps

Description

bm_trim() trims bm_bitmap() objects reducing the number of pixels. The directions and amount of removed pixels are settable (defaulting to 0L).

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bm_object</td>
<td>Either a bm_bitmap(), bm_list(), or bm_font() object.</td>
</tr>
<tr>
<td>sides</td>
<td>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.</td>
</tr>
<tr>
<td>top</td>
<td>How many pixels to trim the top.</td>
</tr>
<tr>
<td>right</td>
<td>How many pixels to trim the right.</td>
</tr>
<tr>
<td>bottom</td>
<td>How many pixels to trim the bottom.</td>
</tr>
<tr>
<td>left</td>
<td>How many pixels to trim the left.</td>
</tr>
<tr>
<td>width</td>
<td>How many pixels wide should the new bitmap be. Use with the hjust argument or just one of either the left or right arguments.</td>
</tr>
<tr>
<td>height</td>
<td>How many pixels tall should the new bitmap be. Use with the vjust argument or just one of either the top or bottom arguments.</td>
</tr>
<tr>
<td>hjust</td>
<td>One of &quot;left&quot;, &quot;center-left&quot;, &quot;center-right&quot;, &quot;right&quot;. &quot;center-left&quot; and &quot;center-right&quot; will attempt to place in &quot;center&quot; if possible but if not possible will bias it one pixel left or right respectively. &quot;centre&quot;, &quot;center&quot;, and &quot;centre-left&quot; are aliases for &quot;center-left&quot;. &quot;centre-right&quot; is an alias for &quot;center-right&quot;. Note if &quot;left&quot; we will trim on the right (and vice-versa).</td>
</tr>
</tbody>
</table>
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).

Value

Either a \texttt{bm_bitmap()}, \texttt{bm_list()}, or \texttt{bm_font()} object.

See Also

\texttt{bm_extend()}, \texttt{bm_pad()}, and \texttt{bm_resize()}.

Examples

\begin{verbatim}
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, px = c("-", "+"))
capital_r_trimmed <- bm_trim(capital_r, c(1, 1, 3, 0))
print(capital_r_trimmed, px = c("-", "+"))
\end{verbatim}

\begin{verbatim}

bm_widths

Widths or heights of bitmaps

Description

\texttt{bm_widths()} returns the widths of the bitmaps while \texttt{bm_heights()} returns the heights of the bitmaps.

Usage

\texttt{bm_widths(bm_object, unique = TRUE)}

\texttt{bm_heights(bm_object, unique = TRUE)}

Arguments

\begin{itemize}
  \item \texttt{bm_object} Either a \texttt{bm_bitmap()}, \texttt{bm_list()}, or \texttt{bm_font()} object.
  \item \texttt{unique} Apply \texttt{base::unique()} to the returned integer vector.
\end{itemize}

Value

A integer vector of the relevant length of each of the \texttt{bm_bitmap()} objects in \texttt{x}. If \texttt{unique} is \texttt{TRUE} then any duplicates will have been removed.


Examples

```r
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
```

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

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

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

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

Arguments

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

Details

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

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bm_bitmap()</code></td>
<td><code>bm_bitmap()</code></td>
<td><code>bm_list()</code></td>
</tr>
<tr>
<td><code>bm_bitmap()</code></td>
<td><code>bm_list()</code></td>
<td><code>bm_list()</code></td>
</tr>
<tr>
<td><code>bm_bitmap()</code></td>
<td><code>bm_font()</code></td>
<td><code>bm_font()</code></td>
</tr>
<tr>
<td><code>bm_list()</code></td>
<td><code>bm_bitmap()</code></td>
<td><code>bm_list()</code></td>
</tr>
<tr>
<td><code>bm_list()</code></td>
<td><code>bm_list()</code></td>
<td><code>bm_list()</code></td>
</tr>
<tr>
<td><code>bm_list()</code></td>
<td><code>bm_font()</code></td>
<td><code>bm_font()</code></td>
</tr>
<tr>
<td><code>bm_font()</code></td>
<td><code>fm_bitmap()</code></td>
<td><code>bm_font()</code></td>
</tr>
<tr>
<td><code>bm_font()</code></td>
<td><code>fm_list()</code></td>
<td><code>bm_font()</code></td>
</tr>
<tr>
<td><code>bm_font()</code></td>
<td><code>fm_font()</code></td>
<td><code>bm_font()</code></td>
</tr>
</tbody>
</table>

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

```r
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))
stats <- c(capital_r, stats)
print(bm_call(rstats, cbind), px = px_ascii)
```

---

**cbind.bm_bitmap**

Combine bitmaps by rows or columns

**Description**

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

**Usage**

```r
## 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")
```

**Arguments**

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

**Value**

A `bm_bitmap()` object.
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

```r
hex2ucp(x)
int2ucp(x)
str2ucp(x)
name2ucp(x, type = c("exact", "grep"), ...)
is_ucp(x)
block2ucp(x, omit_unnamed = TRUE)
range2ucp(x, omit_unnamed = TRUE)
```

### Arguments

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

hex2ucp(x) is a wrapper for as.character(Unicode::as.u_char(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")

# Potential gotcha as as.hexmode("52") == as.integer("82") == 52L
all.equal(hex2ucp(52L), int2ucp(52L)) # TRUE
all.equal(hex2ucp("52"), int2ucp("82")) # TRUE
all.equal(hex2ucp("82"), int2ucp("82")) # FALSE

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

---

is_bm_bitmap

Test if the object is a bitmap glyph object

Description

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

Usage

is_bm_bitmap(x)

Arguments

x

An object

Value

TRUE or FALSE

See Also

bm_bitmap()

Examples

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

is_bm_font

Test if the object is a bitmap font object

Description

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

Usage

is_bm_font(x)

Arguments

x

An object

Value

TRUE or FALSE

See Also

bm_font()

Examples

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

Test if the object is a bitmap glyph list object

Description

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

Usage

is_bm_list(x)

Arguments

x An object

Value

TRUE or FALSE

See Also

bm_list()

Examples

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

**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 binary bm_bitmap() object (which is an integer matrix of ones and zeros). 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. Since base::which() does not automatically cast its argument to a logical value we also redefine it as a generic and besides a default method which simply calls base::which() we offer a which.bm_bitmap() method that first casts the bitmap to logical before calling base::which().
Usage

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

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

which(x, arr.ind = FALSE, useNames = TRUE)

## Default S3 method:
which(x, arr.ind = FALSE, useNames = TRUE)

## S3 method for class 'bm_bitmap'
which(x, arr.ind = FALSE, useNames = TRUE)
```

Arguments

- `e1`: objects.
- `e2`: objects.
- `x`: objects.
- `arr.ind`: logical; should `array indices` be returned when `x` is an array?
- `useNames`: logical indicating if the value of `arrayInd()` should have (non-null) dimnames at all.

Value

`which.bm_bitmap()` returns a logical vector. The various `Ops.bm_bitmap` methods return a `bm_bitmap()` object. The various `Ops.bm_list` methods return a `bm_list()` object.

See Also

- `base::Ops`

Examples

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

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, px = px_ascii)
capital_b <- font[[str2ucp("B")]]
print(capital_r & capital_b, px = px_ascii)
print(capital_r | capital_b, px = px_ascii)
print(capital_r + 1L, px = px_ascii)
print(capital_r + 1L > 1L, px = px_ascii)
which(capital_r > 0L)

# Examples applied to 'bm_list()' objects
```
# RSTATS
bml <- as_bm_list(bml)
bm <- do.call(cbind, bml)
print(bm, px = px_ascii)

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

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

---

**plot.bm_bitmap**  
*Plot bitmap object*

**Description**

*plot.bm_bitmap()* plots a bitmap object to the graphics device. It is a wrapper around *grid::grid.raster()* and *as.raster.bm_bitmap()* which converts a bitmap glyph object to a raster object.

**Usage**

```r
## S3 method for class 'bm_bitmap'
plot(x, ..., col = c("grey80", "black", "grey40"), interpolate = FALSE)

## S3 method for class 'bm_bitmap'
as.raster(x, ..., col = c("grey80", "black", "grey40"))
```

**Arguments**

- **x**  
  A *bm_bitmap()* object

- **...**  
  Passed to *grid::grid.raster()*.

- **col**  
  Character vector of R color specifications.

- **interpolate**  
  Passed to *grid::grid.raster()*.

**Value**

A grid rastergrob grob object silently. As a side effect will draw to graphics device.

**See Also**

*bm_bitmap()*.*as.raster.bm_bitmap()*
Examples

```r
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)

grid::grid.newpage()
plot(capital_r, col = c("yellow", "blue", "red"))
```

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).

Usage

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

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

px_unicode

px_ascii

Arguments

- **x** A bm_bitmap() object
- **...** Further arguments passed to or from other methods.
- **px** Character vector of the pixel to use for each integer value i.e. The first character for integer 0L, the second character for integer 1L, and so on. Will be recycled.
- **fg** R color strings of foreground colors to use. Requires suggested package crayon. FALSE (default) for no foreground colors. Will be recycled.
**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**

```r
read_hex(con)
write_hex(font, con = stdout())
```
**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**

```r
read_monobit(file, quietly = FALSE)
write_monobit(font, file, quietly = FALSE)
```

**Arguments**

- **file**
  A character string of a filename.

- **quietly**
  If TRUE suppress any standard output/error from monobit.

- **font**
  A `bm_font()` object.

**Examples**

```r
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, px = c(".", ","))

filename <- tempfile(fileext = ".hex.gz")
write_monobit(font, gzfile(filename))
```
read_yaff

Details

read_monobit() and write_monobit() require Python v3.6 or greater available on the system.
read_monobit() and write_monobit() uses monobit's convert.py script to convert to/from the
yaff font format which this package can natively read/write from/to. Note monobit is alpha level
software which may not always work (in particular font export may be buggy).

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. For more information about monobit
see https://github.com/robhagemans/monobit.

Examples

if (findpython::can_find_python_cmd(minimum_version = "3.6")) {
  font_file <- system.file("fonts/spleen/spleen-8x16.hex.gz", package = "bittermelon")
  tempfile <- tempfile(fileext = ".hex")
  writeLines(readLines(font_file), tempfile)

  try({
    # 'monobit' is alpha level software which may not always work
    font <- read_monobit(tempfile)
    capital_r <- font[[str2ucp("R")]]
    print(capital_r, px = c(".", ","))

    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](https://github.com/robhagemans/monobit#the-yaff-format).

Examples

```r
font_file <- system.file("fonts/fixed/4x6.yaff.gz", package = "bittermelon")
font <- read_yaff(font_file)
capital_r <- font[[str2ucp("R")]]
print(capital_r, px = c(".", ":"))

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

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

Arguments

... Passed to relevant functions.

na.rm Passed to `min()` and `max()`.
ucp2label

Value

An integer vector.

Examples

```r
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

```r
ucp2label(x)
ucp_sort(x, decreasing = FALSE)
is_combining_character(x)
```

Arguments

- `x`: A character vector of Unicode code points.
- `decreasing`: If TRUE do a decreasing sort.

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

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

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

`.bm_bitmap()` is defined so that it returns a `bm_bitmap()` object (if the value is a matrix) and `<-.bm_bitmap()` casts any replacement values as integers.

Usage

```r
## S3 method for class 'bm_bitmap'
x[i, j, ..., drop = TRUE]
## S3 replacement method for class 'bm_bitmap'
x[i, j, ...] <- value
```

Arguments

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

Value

`.bm_bitmap()` returns a `bm_bitmap()` object if the value is a matrix and/or `drop` is FALSE otherwise it returns an integer matrix.

Examples

```r
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], px = px_ascii)
print(capital_r, px = px_ascii)
```
Index

* datasets
- print.bm_bitmap, 38
- [.bm_bitmap, 7, 44
- [<-bm_bitmap([.bm_bitmap), 44
- as.matrix.bm_bitmap, 2
- as.matrix.bm_bitmap(), 7
- as.raster.bm_bitmap(plot.bm_bitmap), 37
- as.raster.bm_bitmap(), 7, 11, 37
- as_bm_bitmap, 3
- as_bm_bitmap(), 7
- as_bm_bitmap.grob(), 11
- as_bm_font, 5
- as_bm_list, 6
- as_bm_list(), 18
- base::do.call(), 8
- base::lapply(), 17
- base::Ops, 36
- base::readLines(), 40, 42
- base::Summary, 42
- base::unique(), 29
- base::utf8ToInt(), 32
- base::which(), 35
- base::writeLines(), 40, 42
- block2ucp(hex2ucp), 32
- block2ucp(), 43
- bm_bitmap, 7
- bm_bitmap(), 4, 8–31, 34, 36, 37, 39, 44
- bm_bold(bm_shadow), 25
- bm_call, 8
- bm_clamp, 8
- bm_clamp(), 7, 12
- bm_compress, 9
- bm_compress(), 39
- bm_distort, 10
- bm_distort(), 25
- bm_edit, 11
- bm_expand, 12
- bm_expand(), 11, 14
- bm_extend, 13
- bm_extend(), 13, 22, 24, 26, 27, 29, 31
- bm_flip, 15
- bm_flip(), 25
- bm_font, 15
- bm_font(), 4–6, 8–15, 17–31, 34, 39–42
- bm_glow(bm_shadow), 25
- bm_heights(bm_widths), 29
- bm_lapply, 16
- bm_list, 17
- bm_list(), 6, 8–31, 35, 36
- bm_mask, 18
- bm_outline, 19
- bm_overlay, 20
- bm_pad, 21
- bm_pad(), 14, 24, 29
- bm_padding_lengths, 22
- bm_resize, 23
- bm_resize(), 11, 13, 14, 22, 29
- bm_rotate, 24
- bm_shadow, 25
- bm_shift, 26
- bm_shift(), 26
- bm_trim, 28
- bm_trim(), 14, 22, 24, 27
- bm_widths, 29
- c.bm_bitmap, 30
- c.bm_font(b.c.bm_bitmap), 30
- c.bm_list(c.bm_bitmap), 30
- cbind.bm_bitmap, 31
- cbind.bm_bitmap(), 7
- format.bm_bitmap(print.bm_bitmap), 38
- format.bm_bitmap(), 7
- grDevices::png(), 4, 11
- grepl, 32
- grid::grid.raster(), 11, 37
- grid::rasterGrob(), 11

45
INDEX

grid::viewport(), 11, 25
hex2ucp, 32
hex2ucp(), 16, 43
int2ucp (hex2ucp), 32
int2ucp(), 43
is_bm_bitmap, 33
is_bm_bitmap(), 7
is_bm_font, 34
is_bm_font(), 16
is_bm_list, 35
is_bm_list(), 18
is_combining_character (ucp2label), 43
is_combining_character(), 33
is_ucp (hex2ucp), 32
name2ucp (hex2ucp), 32
name2ucp(), 43
Ops.bm_bitmap, 35
Ops.bm_bitmap(), 7
Ops.bm_list (Ops.bm_bitmap), 35
plot.bm_bitmap, 37
plot.bm_bitmap(), 7
print.bm_bitmap, 38
print.bm_bitmap(), 7
px_ascii (print.bm_bitmap), 38
px_unicode (print.bm_bitmap), 38
ragg::agg_png(), 4, 11
range2ucp (hex2ucp), 32
range2ucp(), 43
rbind.bm_bitmap (cbind.bm_bitmap), 31
rbind.bm_bitmap(), 7
read_hex, 39
read_hex(), 41
read_monobit, 40
read_yaff, 41
read_yaff(), 41
str2ucp (hex2ucp), 32
str2ucp(), 43
Summary.bm_list, 42
ucp2label, 43
ucp2label(), 33
ucp_sort (ucp2label), 43
Unicode::as.u_char, 16, 32
Unicode::as.u_char_range, 32
Unicode::u_blocks, 32
Unicode::u_char_from_name, 32
utils::file.edit, 12
which (Ops.bm_bitmap), 35
which.bm_bitmap(), 7
write_hex (read_hex), 39
write_hex(), 41
write_monobit (read_monobit), 40
write_yaff (read_yaff), 41
write_yaff(), 41