frame buf — Frame buf fer manipulation

This module provides a general frame buf fer which can be used to create bitmap images, which can then be sent to a display.

class Frame Buf fer

The Frame Buf fer class provides a pixel buffer which can be drawn upon with pixels, lines, rectangles, text and even other FrameBuffer's. It is useful when generating output for displays.

For example:

```
import frame buf

# Frame Buf fer needs 2 bytes for every RGB565 pixel
f buf  = Frame Buf fer(bytearray(10 * 100 * 2), 10, 100, frame buf .RGB565)

f buf .fill(0)
f buf .text('MicroPython!', 0, 0, 0xffff)
f buf .hline(0, 10, 96, 0xffff)
```

Constructors

class **frame buf .Frame Buf fer**(*buf fer, width, height, format, stride=width*)

Construct a Frame Buf fer object. The parameters are:

- *buf fer* is an object with a buf fer protocol which must be large enough to contain every pixel defined by the width, height and format of the FrameBuffer.
- *width* is the width of the Frame Buf fer in pixels
- *height* is the height of the Frame Buf fer in pixels
- *format* specifies the type of pixel used in the Frame Buf fer; permissible values are listed under Constants below. These set the number of bits used to encode a color value and the layout of these bits in *buffer*. Where a color value c is passed to a method, c is a small integer with an encoding that is dependent on the format of the FrameBuffer.
- *stride* is the number of pixels between each horizontal line of pixels in the Frame Buf fer. This defaults to *width* but may need adjustments when implementing a Frame Buf fer within another larger FrameBuffer or screen. The *buffer* size must accommodate an increased step size.

One must specify valid *buf fer*, *width*, *height*, *format* and optionally *stride*. Invalid *buf fer* size or dimensions may lead to unexpected errors.

Drawing primitive shapes

FrameBuffer.fill(*c*)

Fill the entire FrameBuffer with the specified color.

FrameBuffer.pixel(*x*, *y*[, *c*])

If *c* is not given, get the color value of the specified pixel. If *c* is given, set the specified pixel to the given color.

FrameBuffer.hline(*x*, *y*, *w*, *c*)

FrameBuffer.vline(*x*, *y*, *h*, *c*)

FrameBuffer.line(*x1*, *y1*, *x2*, *y2*, *c*)

Draw a line from a set of coordinates using the given color and a thickness of 1 pixel. The `line` method draws the line up to a second set of coordinates whereas the `hline` and `vline` methods draw horizontal and vertical lines respectively up to a given length.

FrameBuffer.rect(*x*, *y*, *w*, *h*, *c*)

FrameBuffer.fill_rect(*x*, *y*, *w*, *h*, *c*)

Draw a rectangle at the given location, size and color. The `rect` method draws only a 1 pixel outline whereas the `fill_rect` method draws both the outline and interior.

Drawing text

FrameBuffer.text(*s*, *x*, *y*[, *c*])

Write text to the FrameBuffer using the the coordinates as the upper-left corner of the text. The color of the text can be defined by the optional argument but is otherwise a default value of 1. All characters have dimensions of 8x8 pixels and there is currently no way to change the font.

Other methods

FrameBuffer.scroll(*xstep*, *ystep*)

Shift the contents of the FrameBuffer by the given vector. This may leave a footprint of the previous colors in the FrameBuffer.

FrameBuffer.blit(*fbuf*, *x*, *y*[, *key*])

Draw another FrameBuffer on top of the current one at the given coordinates. If *key* is specified then it should be a color integer and the corresponding color will be considered transparent: all pixels with that color value will not be drawn.

This method works between Frame **Buf** fer instances utilising different formats, but the resulting colors may be unexpected due to the mismatch in color formats.

Constants

`framebuf.MONO_VLSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are vertically mapped with bit 0 being nearest the top of the screen. Consequently each byte occupies 8 vertical pixels. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered at locations starting at the leftmost edge, 8 pixels lower.

`framebuf.MONO_HLSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 0 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower.

`framebuf.MONO_HMSB`

Monochrome (1-bit) color format This defines a mapping where the bits in a byte are horizontally mapped. Each byte occupies 8 horizontal pixels with bit 7 being the leftmost. Subsequent bytes appear at successive horizontal locations until the rightmost edge is reached. Further bytes are rendered on the next row, one pixel lower.

`framebuf.RGB565`

Red Green Blue (16-bit, 5+6+5) color format

`framebuf.GS2_HMSB`

Grayscale (2-bit) color format

`framebuf.GS4_HMSB`

Grayscale (4-bit) color format

`framebuf.GS8`

Grayscale (8-bit) color format