`framebuf` — frame buffer manipulation ¶

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

class FrameBuffer

The FrameBuffer class provides a pixel buffer which can be drawn upon with pixels, lines, rectangles, ellipses, polygons, text and even other FrameBuffers. It is useful when generating output for displays.

For example:

```
import framebuf

# FrameBuffer needs 2 bytes for every RGB565 pixel
fbuf = framebuf.FrameBuffer(bytearray(100 * 10 * 2), 100, 10, framebuf.RGB565)

fbuf.fill(0)
fbuf.text('MicroPython!', 0, 0, 0xffff)
fbuf.hline(0, 9, 96, 0xffff)
```

Constructors

class **framebuf.FrameBuffer**(*buffer, width, height, format, stride=width, /*)

Construct a FrameBuffer object. The parameters are:

- *buffer* is an object with a buffer 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 FrameBuffer in pixels
- *height* is the height of the FrameBuffer in pixels
- *format* specifies the type of pixel used in the FrameBuffer; 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 FrameBuffer. This defaults to *width* but may need adjustments when implementing a FrameBuffer within another larger FrameBuffer or screen. The *buffer* size must accommodate an increased step size.

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

Drawing primitive shapes

The following methods draw shapes onto the FrameBuffer.

`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* [, *f*])

Draw a rectangle at the given location, size and color.

The optional *f* parameter can be set to `True` to fill the rectangle. Otherwise just a one pixel outline is drawn.

FrameBuffer.ellipse(*x, y, xr, yr, c* [, *f, m*])

Draw an ellipse at the given location. Radii *xr* and *yr* define the geometry; equal values cause a circle to be drawn. The *c* parameter defines the color.

The optional *f* parameter can be set to `True` to fill the ellipse. Otherwise just a one pixel outline is drawn.

The optional *m* parameter enables drawing to be restricted to certain quadrants of the ellipse. The LS four bits determine which quadrants are to be drawn, with bit 0 specifying Q1, b1 Q2, b2 Q3 and b3 Q4. Quadrants are numbered counterclockwise with Q1 being top right.

FrameBuffer.poly(*x, y, coords, c* [, *f*])

Given a list of coordinates, draw an arbitrary (convex or concave) closed polygon at the given x, y location using the given color.

The *coords* must be specified as a `array` of integers, e.g. `array('h', [x0, y0, x1, y1, ... xn, yn])`.

The optional *f* parameter can be set to `True` to fill the polygon. Otherwise just a one pixel outline is drawn.

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=-1, palette=None*)

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. (If the *palette* is specified then the *key* is compared to the value from *palette*, not to the value directly from *fbuf*.)

The *palette* argument enables blitting between FrameBuffers with differing formats. Typical usage is to render a monochrome or grayscale glyph/icon to a color display. The *palette* is a FrameBuffer instance whose format is that of the current FrameBuffer. The *palette* height is one pixel and its pixel width is the number of colors in the source FrameBuffer. The *palette* for an N-bit source needs 2**N pixels; the *palette* for a monochrome source would have 2 pixels representing background and foreground colors. The application assigns a color to each pixel in the *palette*. The color of the current pixel will be that of that *palette* pixel whose x position is the color of the corresponding source pixel.

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 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.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 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.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