Reference

This document is a work in progress. The libraries are fairly stable at this point, but the documentation still needs expansion and additional examples. Check back later.

A separate page covers GAMBY image formats, with special instructions on how to create icons, patterns, blocks and sprites. You may also get more information from the GAMBY Programming FAQ.

box()

Draw an empty rectangle.

In **GambyGraphicsMode**, the current **drawPattern** is used to draw the rectangle. In **GambyBlockMode**, the block to use is supplied as a parameter.

Syntax

- `gamby.box(x1, y1, x2, y2);         // GambyGraphicsMode`
- `gamby.box(x1, y1, x2, y2, block); // GambyBlockMode`

Parameters

- `x1`: The horizontal position of the box's first corner (`int` (graphics mode) or `byte` (block mode))
- `y1`: The vertical position of the box's first corner (`int` (graphics mode) or `byte` (block mode))
- `x2`: The horizontal position of the box's opposite corner (`int` (graphics mode) or `byte` (block mode))
- `y2`: The vertical position of the box's opposite corner (`int` (graphics mode) or `byte` (block mode))
- `block`: The block number to use when drawing the box in GambyBlockMode (`byte`)

circle()

Draw an empty circle centered at the given coordinates.

The current **drawPattern** is used to draw the circle.

Syntax

- `gamby.circle(cx, cy, radius);`

Parameters

- `cx`: The horizontal position of the circle's center (`int`)
- `cy`: The vertical position of the circle's center (`int`)
- `radius`: The circle's radius (`int`)

clearDisplay()

Erase the screen contents and place the cursor in the first column of the first page.

If used in `GambyBlockMode` or `GambyGraphicsMode`, only the screen is cleared; the contents of the buffer are preserved and will be redrawn when `update()` is called.

Generally, *you should use* *`clearScreen()`* *instead.* `clearDisplay()` is intended for use only in certain special cases. If you don't already know if your case is one of them, assume that it isn't.

Syntax

- `gamby.clearDisplay();`

`clearLine()`

Clears the current line from the current column to the right edge of the screen.

Syntax

- `gamby.clearLine();`

`clearScreen()`

Text-specific clear screen; resets scrolling offset.
Clears the display buffer.

The screen itself doesn't clear until `update()` is called.

Syntax

- `gamby.clearScreen();`

`disc()`

Draw a filled circle.

The current drawPattern is used to draw the rectangle.

Syntax

- `gamby.disc(cx, cy, radius);`

Parameters

- `cx`: The horizontal position of the disc's center (`int`)
- `cy`: The vertical position of the disc's center (`int`)
- `radius`: The disc's radius (`int`)

`drawBlock()`

Draw a block at a given location.

Syntax

- `gamby.drawBlock(x, y, block);`

Parameters

- `x`: The horizontal position, 0 to 23. (`byte`)
- `y`: The vertical position, 0 to 15. (`byte`)

- `block`: The index of the block to draw in the current `palette`, 0 to 15. (`byte`)

drawIcon()

Draw an 8px high icon at the current position on screen.

The icon itself is stored in `PROGMEM`. See the image format documentation for more information. This is frequently used after a call to `setPos()`.

Note: This function is available in all modes, but care needs to be taken in `GambyBlockMode` and `GambyGraphicsMode`. The icon is drawn directly to the screen and is not in the offscreen buffer, so any update of the screen in that area will erase it.

Icons can have more than one 'frame', which are additional images of the same dimension as the first. These could be used for animation, but have a variety of other uses.

Icons can also have a 'transform' applied to them; they can be mirrored horizontally (`HFLIP`), vertically (`VFLIP`) or both (either `HFLIP | VFLIP` or `HFLIP_VFLIP`). The form of `drawIcon()` that includes the transform requires a frame number; if your icon has only one frame, just use the number `0`. Note that transforming icons on the fly takes more computing than normal drawing; in most cases, it will be more efficient to create the mirrored versions as multiple icons in `PROGMEM`.

Syntax

- `gamby.drawIcon(icon);`
- `gamby.drawIcon(icon, frame);`
- `gamby.drawIcon(icon, frame, transform);`

Parameters

- `icon`: The icon's location in `PROGMEM` (e.g. the name of the `PROGMEM` constant). (`const prog_uchar *`)
- `frame`: The frame number to draw, if the icon has multiple frames (0 to (number of frames)-1) (`byte`)
- `transform`: A 'transformation' to apply, `HFLIP` and/or `VFLIP` (`byte`)

drawMode

The current drawing mode.

In `GambyGraphicsMode`, the `drawMode` byte is a used as a series of flags. There are currently three supported:

- `DRAW_NORMAL`: Regular drawing of the `drawPattern`, with both black and white pixels showing up.
- `DRAW_BLACK_TRANSPARENT`: The black pixels in the `drawPattern` are transparent, showing only the white ones.
- `DRAW_WHITE_TRANSPARENT`: The white pixels in the `drawPattern` are transparent, showing only the black ones.

Usage

```
// Just set the value like a normal variable:`
gamby.drawMode = DRAW_NORMAL;
```

drawPattern

The 4x4 pixel pattern to use when drawing.

The 4x4 pattern is stored as a 16 bit `int`. See the image format documentation for more information.

There are several predefined patterns in the GAMBY library you can also use:

- `PATTERN_WHITE`
- `PATTERN_BLACK`
- `PATTERN_GRAY`
- `PATTERN_DK_GRAY`
- `PATTERN_LT_GRAY`
- `PATTERN_DK_L_DIAGONAL`
- `PATTERN_LT_L_DIAGONAL`
- `PATTERN_DK_R_DIAGONAL`
- `PATTERN_LT_R_DIAGONAL`
- `PATTERN_DK_GRID_SOLID`
- `PATTERN_LT_GRID_SOLID`
- `PATTERN_DK_GRID_DOTS`
- `PATTERN_LT_GRID_DOTS`
- `PATTERN_CHECKER`
- `PATTERN_CHECKER_INV`

`drawSprite()`

Draw a bitmap graphic.

`drawSprite()` has two modes: the normal one, and a 'masked' mode. In the latter, a second sprite is used as a 'mask' to make portions of the first transparent; the foreground sprite will only be drawn where the mask has white pixels.

The mask must be the same dimensions as the foreground sprite.

Syntax

- `gamby.drawSprite(x, y, sprite);`
- `gamby.drawSprite(x, y, sprite, mask);`
- `gamby.drawSprite(x, y, sprite, frame);`
- `gamby.drawSprite(x, y, sprite, frame, mask, maskFrame);`

Parameters

- `x`: The sprite's horizontal position (`byte`)
- `y`: The sprite's vertical position (`byte`)
- `sprite`: The address of the foreground sprite in `PROGMEM` (e.g. the constant's name) (`const prog_uchar *`)
- `mask`: The address of the mask sprite in `PROGMEM` (e.g. the constant's name) (`const prog_uchar *`)
- `frame`: The frame number to draw, if the sprite has multiple frames (0 to (number of frames)-1) (`byte`)
- `maskFrame`: The frame number to draw, if the mask has multiple frames (0 to (number of frames)-1) (`byte`)

`drawText()`

Write a string to the display.

Syntax

- `gamby.drawText(x, y, s);`

Parameters

- `x`: The horizontal position at which to draw the text. (`int`)
- `y`: The vertical position at which to draw the text. (`int`)
- `s`: the string to draw. (`char *`)

`drawText_P()`

Write a `PROGMEM` string to the display.

Syntax

- `gamby.drawText_P(x, y, s);`

Parameters

- `x`: The horizontal position at which to draw the text. (`int`)
- `y`: The vertical position at which to draw the text. (`int`)
- `s`: the address of the string to draw (e.g. the name of the `PROGMEM` constant). (`const char *`)

`font`

The font to be used for drawing text, read from `PROGMEM`.

Before using text, you need to set this variable to reference the font data. The `GambyTextMode` template (`File` menu → `Examples` → `Gamby` → `_Templates` → `TextModeTemplate`) contains such data; you can copy the contents of the **font** tab (`font.ino`) into your own sketch for use in `GambyBlockMode` or `GambyGraphicsMode`.

Usage

Here's a (slightly modified) fragment of code taken from the `GambyTextMode` template.

```
// Bring in the font from the 'font' tab (font.ino)
extern prog_int32_t font[];

GambyTextMode gamby; // Could also be GambyBlockMode or GambyGraphicsMode

void setup () {
  // Set the font. You generally need to do this only once, usually just after
  // initializing Gamby. You could, however, do this elsewhere in your
  // sketch -- for example, if you wanted to change fonts on the fly.
  gamby.font = font;

  gamby.println("Hello, GAMBY!");
}
```

`getBlock()`

Retrieve the index (number) of the block at the given location.

Syntax

- `gamby.getBlock(x, y);`

Parameters

- `x`: The horizontal position, 0 to 23. (`byte`)
- `y`: The vertical position, 0 to 15. (`byte`)

Return

- The index of the block (0 to 15) at the given coordinates. (`byte`)

`getPatternPixel()`

Get pixel from a 4x4 pixel pattern (stored as a 16b int) for the given screen coordinates.

The pattern is the `drawPattern` variable is used. The pixel returned will not be what's currently onscreen, but what you would get if you were to draw to the screen with the current pattern.

Syntax

- `gamby.getPatternPixel(x, y);`

Parameters

- `x`: Horizontal position on screen. (`byte`)
- `y`: Vertical position on screen. (`byte`)

Return

- The pixel for the given coordinates in the given pattern (`true` or `false`). (`boolean`)

`getTextWidth()`

Get the width (in pixels) of a string.

Syntax

- `gamby.getTextWidth(s);`

Parameters

- `s`: the string to measure. (`char *`)

`getTextWidth_P()`

Get the width (in pixels) of a string in `PROGMEM`.

Syntax

- `gamby.getTextWidth_P(s);`

Parameters

- `s`: the `PROGMEM` address (e.g. the constant's name) of the string to measure. (`const char *`)

`init()`

Initialize the GAMBY LCD.

You generally won't need to call this 'manually;' the initialization is automatically done when the GAMBY object is created.

Syntax

- `gamby.init();`

`inputs`

The D-Pad and button states.

Set by `readInputs()`. After calling `readInputs()`, you can then check to see if one or more buttons or D-pad directions are pressed by using the bitwise OR operator and one or more of the following GAMBY constants:

- `DPAD_UP`
- `DPAD_UP_LEFT`
- `DPAD_LEFT`
- `DPAD_DOWN_LEFT`
- `DPAD_DOWN`
- `DPAD_DOWN_RIGHT`
- `DPAD_RIGHT`
- `DPAD_UP_RIGHT`
- `BUTTON_1`
- `BUTTON_2`
- `BUTTON_3`
- `BUTTON_4`

```
// This would presumably be done somewhere inside a function like loop()

gamby.readInputs();

if (gamby.inputs & BUTTON_1) {
  // do something
}
else if (gamby.inputs & BUTTON_2) {
  // do something else
}

// You can check more than one input:
if (gamby.inputs & (BUTTON_1 | DPAD_UP)) {
  // do something else entirely
}
```

`line()`

Draw a single-pixel-wide line between two points.

Syntax

- `gamby.line(x0, y0, x1, y1);`

Parameters

- `x0`: Start horizontal position (`int`)
- `y0`: Start vertical position (`int`)

- `x1`: End horizontal position (`int`)
- `y1`: End vertical position (`int`)

`offscreen`

The offscreen buffer, where the screen is stored before being drawn.

In most cases, you won't access this directly; you'll use the various drawing methods. There are times, however, you may want to get at the buffer 'manually' -- if you needed especially fast access, or were doing some major changes to large portions the data.

For more information, see the GAMBY library source. Please note that direct access to the offscreen buffer

`palette`

The palette of 16 4x4 pixel blocks used in `GambyBlockMode`.

This is an attribute, which you set like a variable.

`print()`

Write a string to the display.

When printing numbers, GAMBY's `print()` and `println()` use the same arguments as the Arduino's standard `Serial.print()`.

When used in `GambyBlockMode` or `GambyGraphicsMode`, the various `print()` and `println()` functions have some limitations:

- Text is drawn directly to the LCD, bypassing the offscreen buffer. This means that any `update()` of the screen where text is drawn will erase the text.
- Text `print`ed to the display will overwrite anything underneath it.
- You should typically call `setPos()` before printing text. If you don't, the text may appear someplace unexpected.
- Vertical placement of text is limited to the same 8 lines as `GambyTextMode`.

Syntax

- `gamby.print(s);`
- `gamby.print(n);`
- `gamby.print(n, base);`
- `gamby.print(f);`
- `gamby.print(f, digits);`

Parameters

- `s`: the string to draw. (`char *`)
- `n`: a non-decimal number to draw (`int`, `long`, `unsigned int`, `unsigned long`, or `byte`)
- `base`: the number system to use when printing (constants `BIN`, `OCT`, `DEC`, or `HEX`). The default is `DEC`.
- `f`: a decimal number to draw (`float` or `double`)
- `digits`: the number of decimal digits to print (`byte`). The default is `2`.

`print_P()`

Write a `PROGMEM` string to the display.

See `print()` for more information.

Syntax

- `gamby.print_P(s);`

Parameters

- `s`: the `PROGMEM` address (e.g. the name of the variable) of the string to draw. (`const char *`)

`println()`

Write a string to the display, followed by a newline.

When printing numbers, GAMBY's `print() and println()` uses the same arguments as the Arduino's standard `Serial.print()`.

To save a little memory, instead of using both `println()` and `print()`, consider using only `print()` and 'manually' end your strings with a 'newline' character ('`\n`').

See `print()` for more information.

Syntax

- `gamby.println(s);`
- `gamby.println(n);`
- `gamby.println(n, base);`
- `gamby.println(f);`
- `gamby.println(f, digits);`

Parameters

- `s`: the string to draw. (`char *`)
- `n`: a non-decimal number to draw (`int`, `long`, `unsigned int`, `unsigned long`, or `unsigned char`)
- `base`: the number system to use when printing (constants `BIN`, `OCT`, `DEC`, or `HEX`). The default is `DEC`.
- `f`: a decimal number to draw (`float` or `double`)
- `digits`: the number of decimal digits to print (`byte`). The default is `2`.

`println_P()`

Write a `PROGMEM` string to the display.

To save a little memory, instead of using both `println_P()` and `print_P()`, consider using only `print_P()` and 'manually' end your strings with a 'newline' character ('`\n`').

See `print()` for more information.

Syntax

- `gamby.println_P(s);`

Parameters

- `s`: the `PROGMEM` address of the string to draw. (`const char *`)

`readInputs()`

Read the state of the D-Pad and buttons, then set the GAMBY's `inputs` variable.

Syntax

- `gamby.readInputs();`

`rect()`

Draw a filled rectangle.

In `GambyGraphicsMode`, the drawing is done using the current `drawPattern`. In `GambyBlockMode`, the block to use is supplied as parameter.

Syntax

- `gamby.rect(x1, y1, x2, y2);        // GambyGraphicsMode`
- `gamby.rect(x1, y1, x2, y2, block); // GambyBlockMode`

Parameters

- `x1`: The horizontal position of the rectangle's first corner (`int` (graphics mode) or `byte` (block mode))
- `y1`: The vertical position of the rectangle's first corner (`int` (graphics mode) or `byte` (block mode))
- `x2`: The horizontal position of the rectangle's opposite corner (`int` (graphics mode) or `byte` (block mode))
- `y2`: The vertical position of the rectangle's opposite corner (`int` (graphics mode) or `byte` (block mode))
- `block`: The block number to use when drawing the rectangle in `GambyBlockMode` (`byte`)

`scroll()`

Scroll the screen up (or down) by one or more lines.

Syntax

- `gamby.scroll(s);`

Parameters

- `s`: The number of lines to scroll, positive or negative. (`int`)

`scrollMode`

How GAMBY should behave when text goes beyond the bottom of the screen.

This is an *attribute*, which you set like a variable, using one of the following constants:

- `SCROLL_NORMAL`: Standard scrolling: when text reaches the bottom of the screen, all the contents are shifted up, and the text continues on a fresh blank line.
- `SCROLL_WRAP`: When text reaches the bottom of the screen, it starts again on the top line. The rest of the screen stays the same.

`sendByte()`

Send a byte to the LCD.

This function sends data directly to the LCD and should be used with some caution. Use only if necessary.

The alternate version, `sendByteLSB()`, will send the byte 'backwards:' the lowest ('least significant') bit will be first. Use of this form should be done with even more caution, as it can cause GAMBY to do things you do not expect if used incorrectly.

Syntax

- `gamby.sendByte(data);`
- `gamby.sendByteLSB(data);`

Parameters

- `data`: The byte to send (`byte`)

`sendCommand()`

Send a command to the LCD.

This sends a one or two-byte command directly to the LCD; it is not recommended for casual users. The various commands are defined in `Gamby.h`, which can be found in your `Arduino/libraries/Gamby` folder.

Use cautiously; drawing handled by the various GAMBY modes will not be 'aware' of changes you make directly to the LCD, and everything could start getting weird.

Syntax

- `gamby.sendCommand(command);`
- `gamby.sendCommand(b1, b2);`

Parameters

- `command`: The single-byte command to send, typically one of the constants defined in `Gamby.h`. (`byte`)
- `b1`: The first byte of a two-byte command (`byte`)
- `b2`: The second byte of a two-byte command (`byte`)

`setBlock()`

Set a block at a given location without immediately updating the screen.

The change is made to the offscreen buffer; a separate call to `update()` is required for it to show up. Use this if you want to do a lot of drawing and only show the end results, or need the drawing to happen faster.

Syntax

- `gamby.setBlock(x, y, block);`

Parameters

- `x`: The horizontal position, 0 to 23. (`byte`)
- `y`: The vertical position, 0 to 15. (`byte`)
- `block`: The index of the block to draw (0 to 15). (`byte`)

`setColumn()`

Set the horizontal position of the cursor.

Syntax

- `gamby.setColumn(column);`

Parameters

- `column`: The horizontal position on screen (0-95). (`byte`)

`setPixel()`

Set a pixel, either explicitly on or off, or using the current `drawPattern`.

Syntax

- `gamby.setPixel(x, y);`
- `gamby.setPixel(x, y, p);`

Parameters

- `x`: The pixel's horizontal position (0-95) (`byte`)
- `y`: The pixel's vertical position (0-63) (`byte`)
- `p`: The pixel, either off (`0`/`false`) or on (1/`true`) (`boolean`)

`setPos()`

Set the column and 'page' (text line) location at which the next data will be displayed.

This has no real effect when using drawing methods specific to `GambyBlockMode` or `GambyGraphicsMode` (`setBlock()`, `box()`, `line()`, etc.), and calling any of those after `setPos()` will change the position from the one you specified. In these modes, use `setPos()` immediately before `GambyTextMode` methods like `drawIcon()`.

Syntax

- `gamby.setPos(col, line);`

Parameters

- `col`: The column (0 to 95) (`byte`)
- `line`: The vertical 8 pixel 'page' (0 to 7) (`byte`)

`textDraw`

The current drawing mode used for text.

`textDraw` is a byte that controls the 'styling' of the text. It is typically used with the following predefined constants:

- `TEXT_NORMAL`: Regular text, dark on light.
- `TEXT_INVERSE`: Light text on a dark background.
- `TEXT_UNDERLINE`: Dark on light with an underline. Pixels in the letters will invert where they touch the line.
- `TEXT_STRIKETHRU`: Dark on light with a line through the center. Pixels in the letters will invert where they touch the line.

Usage

```
// GambyTextMode example. It's assumed that 'font' is defined in another tab
// (as in the Gamby/examples/_Templates/TextModeTemplate sketch).
#include <Gamby.h>
```

```
GambyTextMode gamby;
extern const long font[];

void setup() {
  gamby.font = font;
  gamby.println("This is normal.");
  gamby.textDraw = TEXT_INVERSE;
  gamby.println("This is inverse.");
  gamby.textDraw = TEXT_UNDERLINE;
  gamby.println("This is underlined.");

  // The textDraw is actually bits XORed over the text, so you could even
  // do something like this:
  gamby.textDraw = B01010101;
  gamby.println("This has lines through it and is probably unreadable.");
}

void loop() {
}
```

`update()`

Redraws changes to the LCD.

In `GambyGraphicsMode`, only the changed portions (i.e. every draw made since the last call to `update()`) are updated. In `GambyBlockMode`, everything is redrawn, but you can optionally specify a particular region to update.

Due to a peculiarity in the way `GambyBlockMode`'s offscreen buffer is implemented, vertical blocks are always updated two at a time: even and odd. If you supply an odd `y1` (top), it will be rounded down by 1; if you supply an even `y2` (bottom coordinate), it will be rounded up by 1. Columns don't have this limitation; you can use odd or even numbers for either `x1` or `x2`. `x2` and `y2` must be equal to or greater than `x1` and `y1`, respectively.

Syntax

- `gamby.update();`
- `gamby.update(x1, y1, x2, y2);` // (`GambyBlockMode` only!)

Parameters

- `x1`: The left edge of the portion of the screen to redraw, 0 to 23 (`byte`)
- `y1`: The top edge of the portion of the screen to redraw, 0 to 15 (`byte`)
- `x2`: The right edge of the portion of the screen to redraw, `x1` to 23 (`byte`)
- `y2`: The bottom edge of the portion of the screen to redraw, `y1` to 15 (`byte`)

`wrapMode`

How GAMBY should behave when text goes beyond the right margin.

This is an *attribute*, which you set like a variable, using one of these pre-defined constants:

- `WRAP_CHAR`: (default) the first character that won't fit gets wrapped to the next line.
- `WRAP_NONE`: text wider than the screen disappears off the edge of the display.

Usage

```
// This would presumably appear inside a function (like setup() or loop()) and assumes
// you've already created a GambyTextMode object called gamby.
gamby.wrapMode = WRAP_CHAR;
```