UTFT

Arduino and chipKit Universal TFT display library

Manual

PREFACE:

This library is the continuation of my ITDB02_Graph, ITDB02_Graph16 and RGB_GLCD libraries for Arduino and chipKit. As the number of supported display modules and controllers started to increase I felt it was time to make a single, universal library as it will be much easier to maintain in the future.

Basic functionality of this library was origianlly based on the demo-code provided by ITead studio (for the ITDB02 modules) and NKC Electronics (for the RGB GLCD module/shield).

This library supports a number of 8bit, 16bit and serial graphic displays, and will work with both Arduino and chipKit boards. For a full list of tested display modules and controllers, see the document **UTFT_Supported_display_modules_&_controllers.pdf**.

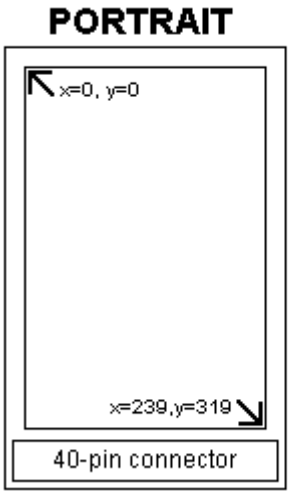
When using 8bit and 16bit display modules there are some requirements you must adhere to. These requirements can be found in the document **UTFT_Requirements.pdf**.
There are no special requirements when using serial displays.

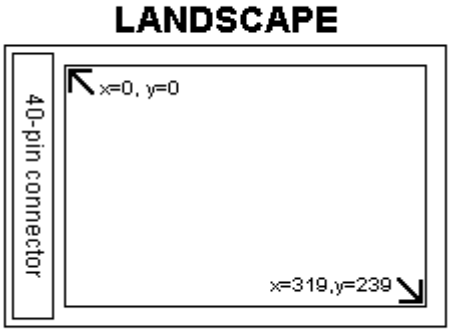
You can always find the latest version of the library at
http://electronics.henningkarlsen.com/

If you make any modifications or improvements to the code, I would appreciate that you share the code with me so that I might include it in the next release. I can be contacted through **http://electronics.henningkarlsen.com/contact.php**.

Version:	1.0	12 Feb 2012	• initial release

DISPLAY ORIENTATION:





DEFINED LITERALS:

Alignment

For use with print(), printNumI() and printNumF()

```
LEFT:   0
RIGHT:  9999
CENTER: 9998
```

Orientation

For use with InitLCD()

```
PORTRAIT:  0
LANDSCAPE: 1
```

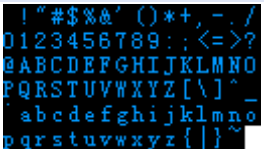
Display model

For use with ITDB02()

Please see **UTFT_Supported_display_modules_&_controllers.pdf**

INCLUDED FONTS:

SmallFont




```
Charactersize:  8x12 pixels
Number of characters:  95
```

BigFont


```
Charactersize:  16x16 pixels
Number of characters:  95
```

SevenSegNumFont


```
Charactersize:  32x50 pixels
Number of characters:  10
```

FUNCTIONS:

UTFT(Model, RS, WR, CS, RST);

The main class constructor when using 8bit or 16bit display modules.

Parameters:
```
Model:  See the separate document for the supported display modules
RS:     Pin for Register Select
WR:     Pin for Write
CS:     Pin for Chip Select
RST:    Pin for Reset
```

Usage:
```
UTFT myGLCD(ITDB32S,19,18,17,16); // Start an instance of the UTFT class
```

UTFT(Model, SDA, SCL, CS, RST[, RS]);

The main class constructor when using serial display modules.

Parameters:
```
Model:  See the separate document for the supported display modules
SDA:    Pin for Serial Data
SCL:    Pin for Serial Clock
CS:     Pin for Chip Select
RST:    Pin for Reset
RS:     <optional> Only used for 5pin serial modules
        Pin for Register Select
```

Usage:
```
UTFT myGLCD(ITDB18SP,11,10,9,12,8); // Start an instance of the UTFT class
```

InitLCD([orientation]);

Initialize the LCD and set display orientation.

Parameters:
```
Orientation: <optional>
             PORTRAIT
             LANDSCAPE (default)
```

Usage:
```
myGLCD.initLCD(); // Initialize the display
```

Notes: This will reset color to white with black background. Font size will be reset to FONT_SMALL.

clrScr();

Clear the screen. The background-color will be set to black.

Parameters: None

Usage:
```
myGLCD.clrScr(); // Clear the screen
```

fillScr(r, g, b);

Fill the screen with a specified color.

Parameters:
```
r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
```

Usage:
```
myGLCD.fillScr(255,127,0); // Fill the screen with orange
```

setColor(r, g, b);

Set the color to use for all draw*, fill* and print commands.

Parameters:
```
r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
```

Usage:
```
myGLCD.setColor(0,255,255); // Set the color to cyan
```

setBackColor(r, g, b);

Set the background color to use for all print commands.

Parameters:
```
r: Red component of an RGB value (0-255)
g: Green component of an RGB value (0-255)
b: Blue component of an RGB value (0-255)
```

Usage:
```
myGLCD.setBackColor(255,255,255); // Set the background color to white
```

drawPixel(x, y);

Draw a single pixel.

Parameters:
```
x: x-coordinate of the pixel
y: y-coordinate of the pixel
```
Usage:
```
myGLCD.drawPixel(119,159); // Draw a single pixel
```

drawLine(x1, y1, x2, y2);

Draw a line between two points.

Parameters:
```
x1: x-coordinate of the start-point
y1: y-coordinate of the start-point
x2: x-coordinate of the end-point
y2: y-coordinate of the end-point
```
Usage:
```
myGLCD.drawLine(0,0,239,319); // Draw a diagonal line
```

drawRect(x1, y1, x2, y2);

Draw a rectangle between two points.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```
Usage:
```
myGLCD.drawRect(119,159,239,319); // Draw a rectangle
```

drawRoundRect(x1, y1, x2, y2);

Draw a rectangle with slightly rounded corners between two points. The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```
Usage:
```
myGLCD.drawRoundRect(0,0,119,159); // Draw a rounded rectangle
```

fillRect(x1, y1, x2, y2);

Draw a filled rectangle between two points.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```
Usage:
```
myGLCD.fillRect(119,0,239,159); // Draw a filled rectangle
```

fillRoundRect(x1, y1, x2, y2);

Draw a filled rectangle with slightly rounded corners between two points. The minimum size is 5 pixels in both directions. If a smaller size is requested the rectangle will not be drawn.

Parameters:
```
x1: x-coordinate of the start-corner
y1: y-coordinate of the start-corner
x2: x-coordinate of the end-corner
y2: y-coordinate of the end-corner
```
Usage:
```
myGLCD.fillRoundRect(0,159,119,319); // Draw a filled, rounded rectangle
```

drawCircle(x, y, radius);

Draw a circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```
Usage:
```
myGLCD.drawCircle(119,159,20); // Draw a circle with a radius of 20 pixels
```

fillCircle(x, y, radius);

Draw a filled circle with a specified radius.

Parameters:
```
x:      x-coordinate of the center of the circle
y:      y-coordinate of the center of the circle
radius: radius of the circle in pixels
```
Usage:
```
myGLCD.fillCircle(119,159,10); // Draw a filled circle with a radius of 10 pixels
```

print(st, x, y[, deg]);

Print a string at the specified coordinates. An optional background color can be specified. Default background is black.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
st:  the string to print
x:   x-coordinate of the upper, left corner of the first character
y:   y-coordinate of the upper, left corner of the first character
deg: <optional>
     Degrees to rotate text (0-359). Text will be rotated around the upper left corner.
```
Usage:
```
myGLCD.print("Hello, World!",CENTER,0); // Print "Hello, World!"
```
Notes:
```
CENTER and RIGHT will not calculate the coordinates correctly when rotating text.
```

printNumI(num, x, y);

Print an integer number at the specified coordinates. An optional background color can be specified. Default background is black.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.

Parameters:
```
num: the value to print (-2,147,483,648 to 2,147,483,647) INTEGERS ONLY
x:   x-coordinate of the upper, left corner of the first digit/sign
y:   y-coordinate of the upper, left corner of the first digit/sign
```
Usage:
```
myGLCD.print(num,CENTER,0); // Print the value of "num"
```

printNumF(num, dec, x, y[, divider]);

Print a floating-point number at the specified coordinates. An optional background color can be specified. Default background is black.
You can use the literals LEFT, CENTER and RIGHT as the x-coordinate to align the string on the screen.
WARNING: Floating point numbers are not exact, and may yield strange results when compared. Use at your own discretion.

Parameters:
```
num:      the value to print (See note)
dec:      digits in the fractional part (1-5) 0 is not supported. Use printNumI() instead.
x:        x-coordinate of the upper, left corner of the first digit/sign
y:        y-coordinate of the upper, left corner of the first digit/sign
divider:  <Optional>
          Single character to use as decimal point. Default is '.'
```
Usage:
```
myGLCD.print(num, 3, CENTER,0); // Print the value of "num" with 3 fractional digits
```
Notes:
```
Supported range depends on the number of fractional digits used.
Approx range is +/- 2*(10^(9-dec))
```

setFont(fontname);

Select font to use with print(), printNumI() and printNumF().

Parameters:
```
fontname: Name of the array containing the font you wish to use
```
Usage:
```
myGLCD.setFont(BigFont); // Select the font called BigFont
```
Notes:
```
You must declare the font-array as an external or include it in your sketch.
```

drawBitmap (x, y, sx, sy, data[, scale]);

Draw a bitmap on the screen.

Parameters:
```
x:     x-coordinate of the upper, left corner of the bitmap
y:     y-coordinate of the upper, left corner of the bitmap
sx:    width of the bitmap in pixels
sy:    height of the bitmap in pixels
data:  array containing the bitmap-data
scale: <optional>
       Scaling factor. Each pixel in the bitmap will be drawn as <scale>x<scale> pixels on screen.
```
Usage:
```
myGLCD.drawBitmap(0, 0, 32, 32, bitmap); // Draw a 32x32 pixel bitmap
```
Notes:
```
You can use the online-tool "ImageConverter 565" or "ImageConverter565.exe" in the Tools-folder to
convert pictures into compatible arrays. The online-tool can be found on my website.
Requires that you #include <avr/pgmspace.h> when using an Arduino.
```

drawBitmap (x, y, sx, sy, data, deg, rox, roy);

Draw a bitmap on the screen with rotation.

Parameters:
```
x:    x-coordinate of the upper, left corner of the bitmap
y:    y-coordinate of the upper, left corner of the bitmap
sx:   width of the bitmap in pixels
sy:   height of the bitmap in pixels
data: array containing the bitmap-data
deg:  Degrees to rotate bitmap (0-359)
rox:  x-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner
roy:  y-coordinate of the pixel to use as rotational center relative to bitmaps upper left corner
```
Usage:
```
myGLCD.drawBitmap(50, 50, 32, 32, bitmap, 45, 16, 16); // Draw a bitmap rotated 45 degrees around
its center
```
Notes:
```
You can use the online-tool "ImageConverter 565" or "ImageConverter565.exe" in the Tools-folder to
convert pictures into compatible arrays. The online-tool can be found on my website.
Requires that you #include <avr/pgmspace.h> when using an Arduino.
```

lcdOff();	
Turn off the LCD. No commands will be executed until a lcdOn(); is sent.	
Parameters:	None
Usage:	`myGLCD.lcdOff(); // Turn off the lcd`
Notes:	This function is currently only supported on PCF8833-based displays

lcdOn();	
Turn on the LCD after issuing a lcdOff()-command.	
Parameters:	None
Usage:	`myGLCD.lcdOn(); // Turn on the lcd`
Notes:	This function is currently only supported on PCF8833-based displays

setContrast(c);	
Set the contrast of the display.	
Parameters:	c: Contrast-level (0-64)
Usage:	`myGLCD.setContrast(64); // Set contrast to full (default)`
Notes:	This function is currently only supported on PCF8833-based displays