GLCD Graphical LCD library

This library supports the use of Graphical LCDs (GLCD) for use on Arduino. This is an extensive modification of the ks0108 library that supports more arduino boards and is easier to integrate with different panels.
The graphical functions are backwards compatible (except where noted) so existing sketches should work with little or no modification to the code. The configuration mechanism has been changed to facilitate use with a broad range of GLCD chips and ATmega controllers. See the section on migrating your sketch for details on modifications for the new library.

Table of Contents:

Wiring and Configuration:

Panel characteristics (like pixel height and width) and the pins used to connect to the panel must be set in a configuration file. Configuration files are provided for the wiring shown below, you can change the file if you want to use different wiring or panels.

This release supports panels using the KS0108 and SED1520 chips.

KS0108 family
The KS0108 is one of the more common GLCD controller chips. The wiring for these panels is not standardized and it is important to check the datasheet for your panel to confirm how it should be wired. Incorrect connections of the signal lines are the most common cause of problems, and particular care should be taken with the power leads as wiring these incorrectly can destroy a panel.

Most GLCD panels require an external preset pot to set the LCD working voltage (contrast) and a fixed resistor to limit the current in the backlight. The datasheet for your panel should provide specific information on the wiring and choice of components for this.

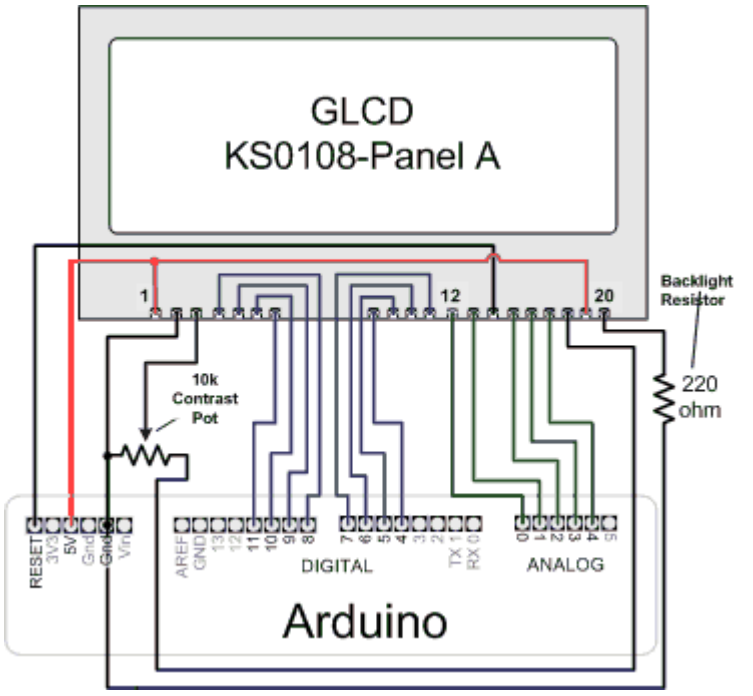
Here are three common pinouts for KS0108 panels.
The numbers under the Arduino column are the Arduino pins used in the configuration file provided in the download, if you alter the wiring to arduino pins then you must change the pin assignments in the configuration file.

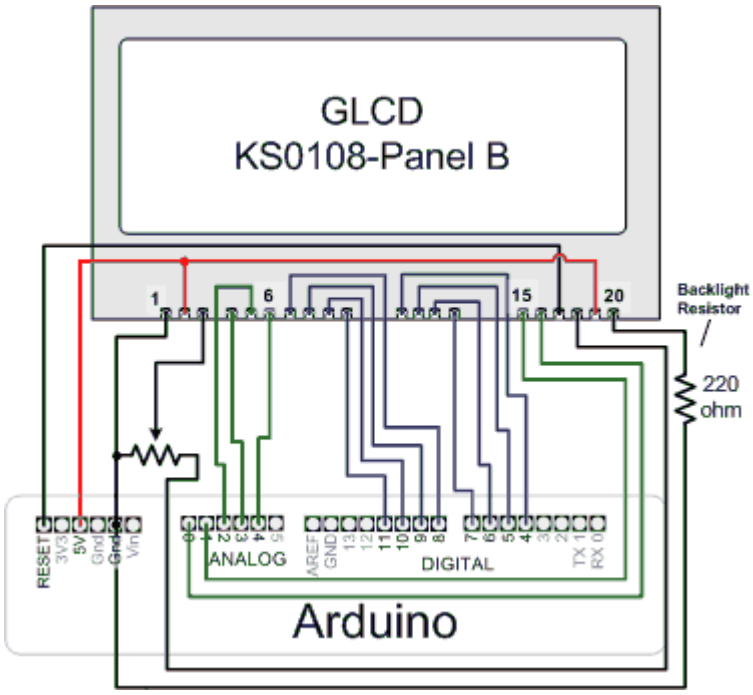
The numbers in the Panel A/B/C columns are the numbers of the GLCD display panel (see your datasheet).

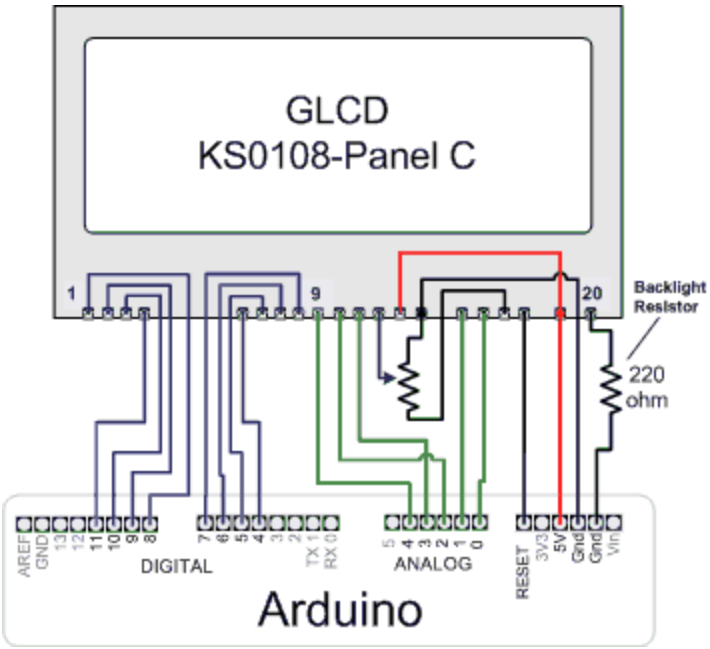
Arduino pins	Function	Panel A	Panel B	Panel C	Comments
5V	+5 volts	1	2	13	
Gnd	GND	2	1	14	
n/a	Contrast in	3	3	12	Wiper of contrast pot
8	D0	4	7	1	
9	D1	5	8	2	
10	D2	6	9	3	
11	D3	7	10	4	
4	D4	8	11	5	
5	D5	9	12	6	
6	D6	10	13	7	
7	D7	11	14	8	
14 (alog 0)	CSEL1	12	15	15	Chip 1 select
15 (alog 1)	CSEL2	13	16	16	Chip 2 select
Reset	Reset	14	17	18	Connect to reset pin
16 (alog 2)	R_W	15	5	10	Read/write
17 (alog 3)	D_I	16	4	11	Data/Instruction (RS)
18 (alog 4)	EN	17	6	9	Enable
n/a	Contrast out	18	18	17	10k or 20k preset
n/a	Backlight +5	19	19	19	See datasheet for
Gnd	Backlight Gnd	20	20	20	Backlight resistor

The following illustrations show the wiring for these panel types. Although the pin number on the GLCD panels are different, the corresponding functions are connected to the same Arduino pins in all three examples so the supplied ks0108_Config.h configuration file should work without change if you wire your panel following the appropriate diagram.

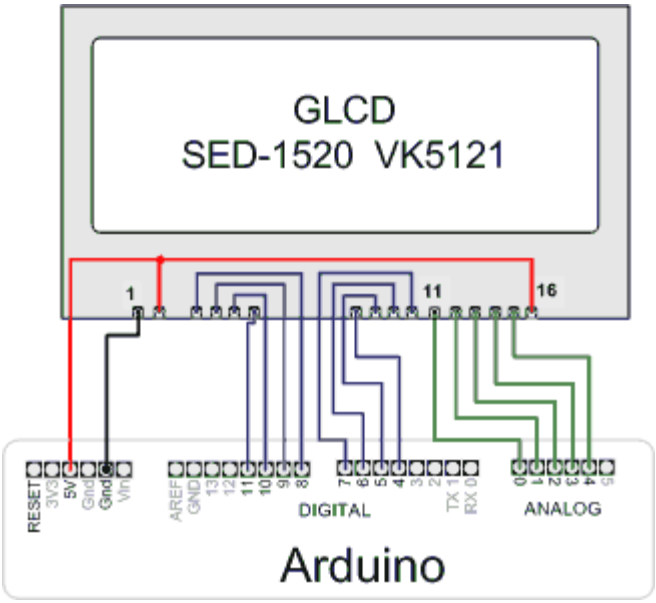
The ks0108 needs to have its reset pin taken low for a brief period after power is applied. The diagrams show the display's reset pin connected to the Arduino reset pin and this will automatically reset the display when the Arduino resets. You can also wire the display reset pin to a spare Arduino pin and control reset in software by adding a define to the panel configuration file (see configuration below), but this is only necessary if the display reset is not wired to Arduino reset.


GLCD
KS0108-Panel A
1
12
20
Backlight
Resistor
220
ohm
10k
Contrast
Pot
RESET
3V3
5V
Gnd
Gnd
Vin
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX 1
RX 0
DIGITAL
0
1
2
3
4
5
ANALOG
Arduino

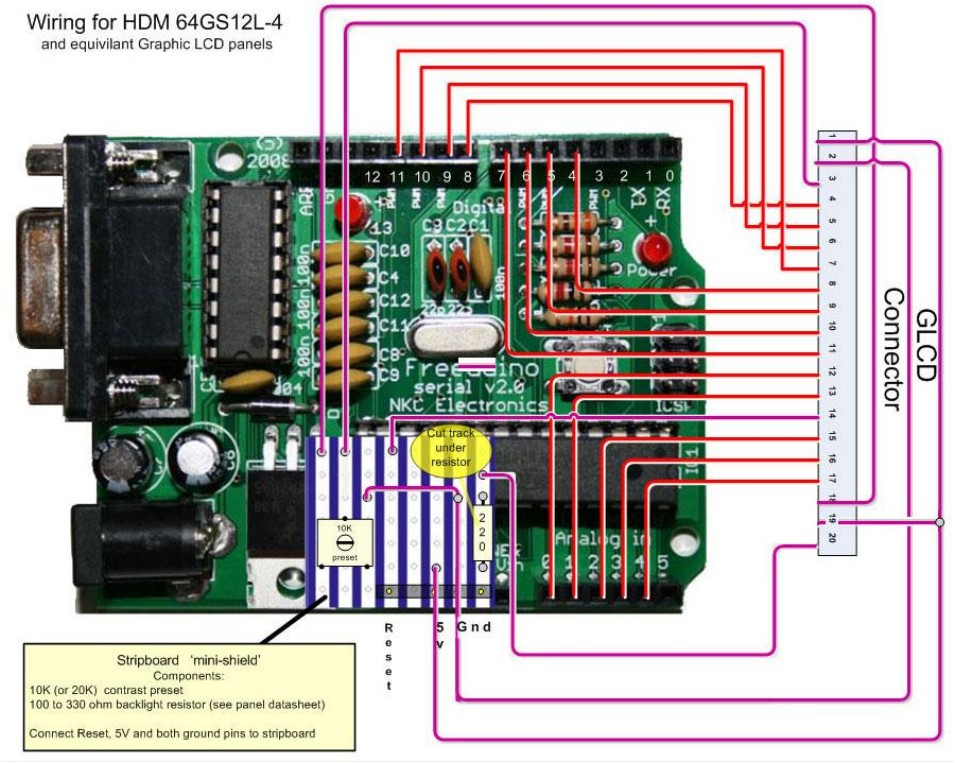

GLCD
KS0108-Panel B
1
6
15
20
Backlight
Resistor
220
ohm
RESET
3V3
5V
Gnd
Gnd
Vin
0
1
2
3
4
5
ANALOG
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
TX 1
RX 0
DIGITAL
Arduino



Here is the wiring for a typical SED1520 Panel:



A tip for making the physical connections is use a small piece of stripboard with header pins for 5V, ground and Reset providing connection to the Arduino. The picture has an example layout for a type 'A' panel



Configuration

Configuration has been significantly improved to give a wider choice of Arduino pins, support for more processor types and display panels.
Processor type is now determined from the board selected in the IDE, the following processor types are supported in the initial relase:
Standard Arduino (ATmega8,168,328), Mega(ATmega1280), Sanguino (ATmega644P), Teensy and Teensy++.

Panel information is now contained in the same configuration file used to set the pins. This version of the library has significantly improved the handling of display timing and most displays will work without requiring a configuration change for timing.

The active configuration used when the sketch is compiled is selected in a master config file named glcd_Config.h. This contains a list of all the available configuration files and one of these should be made active by removing the comments '//' preceding the desired file name.
Ensure that only one file is selected. You can add to the list of files by creating variations that provide different pin connections for different panels if you have more display variants.

There is configuration file for each display type. These files are in the configuration directory, the current release includes the following:
ks0108_Config.h - configuration file for ks0108 type panels using the standard Arduino pins illustrated above
ks0108_Mega_Config.h - config for Mega using same pins as ks0108_Mega.h from the old ks0108 library
ks0108_Sanguino_Config.h - config for Atmega644 using same pins as ks0108_Sanguino.h from ks0108 library

sed1520_Config.h - configuration file for sed1520 type panels.

You can use the standard ks0108_Config.h file for any board, including the Mega or Sanguino, but the performance is improved using the pin assignments given in the respective files for these boards. Bear in mind that the wiring diagrams shown above are for the pin assignments in ks0108_Config.h, if you use different pins then remember to wire your panel accordingly.

Pins can also be selected by AVR port and pin number, see the avrio.h file for more details.
Note that there is a an optional define named glcdRES that will reset the display when the library is initialized at startup. This is not needed in the usually configuration where the display reset pin is connected to the Arduino reset pin but can be useful if your boards reset pin is not accessible.

Troubleshooting

No pixels visible on the display
•Check +5v and Gnd connections between Arduino and GLCD panel
•Check that all data and command pins are wired correctly
•Check contrast voltage (typically between -3 and -4 volts) on contrast-in pin of LCD panel. While the sketch is operating, try gradually adjusting the pot through its range. Some
displays are very sensitive to this setting.
•Check sketch has compiled ok and downloaded to the arduino.

Left and right side of image reversed
•swap CSEL1 and CSEL2 wires (or swap pin assignments in header file)

Display garbled
•check all data and command pins are wired correctly and that these match the setting in the header file.
•if you change the pin assignments in the header file you must delete the ks0108.o object file in the library directory before re-compiling your sketch.

Create your own fonts and bitmaps

There is a free java application available that can convert PC fonts for use with this library. The software is called FontCreator2 and it can produce a header file that stores font definitions in program memory when included in your sketch.

Embedding images in your firmware
A Processing sketch is provided in the download that converts bmp images to files that can be used by the library to display the image on the LCD. See the documentation in the download for more information.

Migrating your sketch from ks0108 to the new GLCD library

The distribution includes the system font and arial14 font, but these are now in a subdirectory called 'fonts' so you will need to modify the include statement, for example:
change
#include "SystemFont5x7.h" // system font
To
#include "fonts/SystemFont5x7.h" // system font

Bitmaps are now in a bitmaps subdirectory so change :
#include "ArduinoIcon.h" // the bitmap distributed with the ks0108 lib
To
#include "bitmaps/ArduinoIcon64x64.h" // 64x64 bitmap

Note that the ArduinoIcon.h bitmap file is renamed to ArduinoIcon64x64.h to differentiate it from other size bitmaps supplied to work with panels of different pixel dimensions.

If you are using any of the following functions you should either change the code use the new function or include glcd_Deprecated.h file that will convert the old function name to the new function.
DrawVertLine(x, y, length, color) is now: DrawVLine(x, y, length, color)
DrawHoriLine(x, y, length, color) is now : DrawHLine(x, y, length, color)
ClearSysTextLine(row) is now EraseTextLine(row)

The character output functions behave differently in the new library .
The new library now sets or clears all pixels of a font (glyph), the old library did not consistently erase pixels in the whitespace below the glyph.
This means that if there are graphics very close below the character they may be overwritten with the new text ouput code.
The workaround is to either move the graphics objects so they are not cleared when the text is written or to draw the graphical objects after the text is displayed.

Another change is the way a string wraps on the newline character. The old library wrapped to the column where the string started, the new library now wraps to the

beginning of the text area. See the section on Text Areas to see how you can use this new capability to control where text will wrap on the display.

See the section on Configuration for details on the new configuration files. You can use the same wiring as the old library but the data pins are specified differently in the new library. The ks0108_Config,h file included in the distribution uses exactly the same pins as the ks0108_Arduino.h file supplied with the old library, but if you changed the Arduino pin numbers you will need to modify ks0108_Config to match your wiring.

GLCD Methods:

Here is a summary of the methods supported by this library.
Note that all coordinates start from 0 unless otherwise noted. 0,0 is the pixel on the upper left edge of the screen

Table of contents:

Puts_P()
PutChar()
Puts()
PrintNumber()

Init()

Description: This should be called in setup to initialize the library prior to calling any other function.
The display is cleared and ready for use after calling Init.

Syntax
GLCD.Init() ; // initialize the library to draw dark pixels on a light background
GLCD.Init(NON-INVERTED) ; // same as above
GLCD.Init(INVERTED) ; // initialize the library to draw light pixels on a dark background

Parameters
Init with no parameter is the standard initialization, this is identical to: Init(NON-INVERTED)
Init(INVERTED) will invert dark and light pixels when drawn

Graphic Functions

Coordinate system

0,0 is the upper left edge of the lcd.
Useful constants:
GLCD.Width equals the width of the display in pixels
GLCD.height equals the height of the display in pixels
GLCD.Right equals GLCD.Width-1, this is the right-most pixel
GLCD.Bottom equals GLCD.Height-1, this is the bottom pixel
GLCD.CenterX equals GLCD.Width/2, the horizontal center
GLCD.CenterY equals GLCD.Height/2, the vertical center

Colors

Two colors are supported in this version.
BLACK is a dark pixel, WHITE is a pixel that is not dark
BLACK is the default color

SetDot()

Description: sets the pixel at the given x,y coordinate to the given color

Syntax
SetDot(x,y, BLACK); // draws a BLACK pixel at x,y
SetDot(x,y, WHITE); // erases the pixel at x,y

Parameters
x – a value from 0 to GLCD.Width-1
y - a value fron 0 to GLCD.Height-1

DrawVLine()

Description: Draws a vertical line

Syntax
DrawVertLine(x, y, height); // draws a BLACK line from x,y to x, y + height
DrawVertLine(x, y, height, BLACK); // as above
DrawVertLine(x, y, height, WHITE); // as above but the pixels on the line are erased

Parameters
x – a value from 0 to GLCD.Width-1
y - a value from 0 to GLCD.Height-1
height – a value from 1 to GLCD.Height-y-1
BLACK or WHITE is an optional parameter indicating pixel color, default is BLACK

(Note this function was named DrawVertLine() in the previous library releases)

DrawHLine()

Description: Draws a horizontal line

Syntax
DrawHoriLine(x, y, width); // draws a BLACK line from x,y to x + width , y
DrawHoriLine(x, y, width, BLACK); // as above
DrawHoriLine(x, y, width, WHITE); // as above but the pixels on the line are erased

Parameters
x – a value from 0 to GLCD.Width-1
y - a value from 0 to GLCD.Height-1
width – a value from 1 to GLCD.Width-x-1
BLACK or WHITE is an optional parameter indicating pixel color, default is BLACK

(Note this function was named DrawHoriLine() in the previous library releases)

DrawLine()

Description: Draws a line between two coordinates.

Syntax
DrawLine(x1, y1, x2, y2; // draws a BLACK line from x1,y1 to x2,y2
DrawLine(x1, y1, x2, y2, BLACK); // as above
DrawLine(x1, y1, x2, y2, WHITE); // as above but the pixels on the line are erased

Parameters
x1 – a value from 0 to GLCD.Width-1 indicating start x coordinate
y1 - a value fron 0 to GLCD.Height-1 indicating start y coordinate
x2 – a value from 0 to GLCD.Width-1 indicating end x coordinate
y2 - a value fron 0 to GLCD.Height-1 indicating end y coordinate
BLACK or WHITE are optional parameters specifying pixel color, default is BLACK

DrawRect()

Description: Draws a rectangle of given width and height
x,y is the upper left edge of the rectangle
The lower right edge is at x+width, y+height
Note that the length of the horizontal sides will be width+1 pixels, the vertical sides will be height+1 pixels

Syntax
DrawRect(x, y, width, height); // draws a BLACK rectangle of given width and height starting at x,y
DrawRect(x, y, width, height, BLACK); // as above
DrawRect(x, y, width, height, WHITE); // as above but the rectangle pixels are erased

Parameters
x, y – the x,y coordinates of the rectangle to be drawn
width, height – the width and height of the rectangle

FillRect()

Description: Fills the interior of a rectangle specified by a pair of coordinates, a width, and a height.
The left and right edges of the rectangle are at `x` and `x + width - 1`.
The top and bottom edges are at `y` and `y + height - 1`.

The resulting rectangle covers an area `width` pixels wide by `height` pixels tall starting from the pixel at x,y.
The rectangle is filled using the given color (BLACK if none given)
(Note that FillRect behavior has changed from the previous versions of the library. The filled rectangle will be one pixel smaller in width and height than the old version. This change was to make the functionality consistent with the way Java and C# create filled rectangles)

InvertRect()

Description: Sets BLACK pixels WHITE and WHITE pixels BLACK within the given rectangular area.
The left and right edges of the inverted area are at `x` and `x + width - 1`.
The top and bottom edges are at `y` and `y + height - 1`

Syntax
InvertRect(x, y, width, height); // inverts pixels in the given rectangular area

Parameters
As FillRect but without the color parameter

DrawRoundRect()

Description: Draws a rectangle with rounded corners

Syntax
DrawRoundRect(x, y, width, height, radius); // draws a BLACK rectangle similar to DrawRect but with corners of the given radius
DrawRoundRect(x, y, width, height, radius, BLACK); // as above
DrawRoundRect(x, y, width, height, radius, WHITE); // as above but the rectangle pixels are erased

Parameters
x,y,width,height as DrawRectangle
radius- a value from 1 to half the height or width of the rectangle

DrawCircle

Description: Draws a circle centered at x,y with the given radius
The circle will fit inside a rectangular area bounded by x-radius,y-radius and x+radius,y+radius
Note that because the circle is drawn from the center pixel out, the diameter will be 2 * radius +1 pixels.

Syntax
DrawCircle(x, y, r); // draws a BLACK circle centered at x,y with radius r
DrawCircle(x, y, r, BLACK); // as above
DrawCircle(x, y, r, WHITE); // draws a WHITE circle centered at x,y with radius r

Parameters
x – a value from 0 to GLCD.Right (GLCD.Width-1)
y - a value fron 0 to GLCD.Bottom (GLCD.Height-1)
radius- a value from 1 to half the height or width of the rectangle

FillCircle()

Description: Draws a filled in circle centered at x,y with the given radius

Syntax and Parameters
see DrawCircle

SetDisplayMode()

Description: sets the graphical state to normal (BLACK colored pixels are dark), or inverted (WHITE colored pixels are dark)

Syntax
SetDisplayMode(NON_INVERTED); // sets the state to normal
SetDisplayMode(INVERTED); // sets the state to inverted

(Note this function was named SetInverted() in the previous library releases)

DrawBitmap()

Description: Draws a bitmap image with the upper left edge at the x,y coordinates. Bitmap data is in program memory (Flash)
A utility for creating bitmap header files, glcdMakeBimtap, is supplied with the download

Syntax
DrawBitmap(*bitmap, x, y);
DrawBitmap(*bitmap, x, y, BLACK); // as above
DrawBitmap(*bitmap, x, y, WHITE); // inverts pixels

ClearScreen()

Description: Erases all screen pixels (pixels from 0,0 to GLCD.Width-1,GLCD.Height-1)

Syntax
ClearScreen(); // sets all pixels to WHITE (if NORMAL mode or BLACK if INVERTED)
ClearScreen(WHITE); // same as above
ClearScreen(BLACK); // clears screen writing BLACK pixels

Note: If the display is in INVERTED mode, then the color WHITE will paint the screen BLACK and the color BLACK will paint the screen WHITE.

Font Functions

SelectFont()

Description: Selects the font definition as the current font. Subsequent printing functions will use this font. Font definitions are stored in program memory.
You can have as many fonts defines as will fit in program memory and can switch between them with this function.

Syntax
SelectFont(font) ; // font is a font pointer defined in a font definition file. Output is rendered using dark pixels.
SelectFont(font, BLACK) // as above
SelectFont(font, WHITE) // printed output rendered as WHITE pixels

SetFontColor()

Description: Sets the color of the currently selected font.

Syntax
SetFontColor(BLACK) // printed output rendered as BLACK pixels
SetFontColor(WHITE) // printed output rendered as WHITE pixels

SetTextMode()
Description: // Sets the given text mode (currently only scroll direction is supported)

Syntax
SetTextMode(SCROLL_UP) // normal scroll direction, old lines scroll up
SetTextMode(SCROLL_DOWN) // reverse scroll direction, old lines scroll down

Arduino print functions

All of the Arduino print functions can be used in this library, see:
http://www.arduino.cc/en/Serial/Print
The functions work with any selected font although the ability to scroll to a new line is only supported with the fixed width system font.
All of these functions print from the current cursor position (see GotoXY)

GLCD.print(character); // prints the character at the current cursor position
GLCD.print(integer); // prints the decimal value of the integer
GLCD.print(integer,DEC); // as above
GLCD.print(integer, HEX); // prints the hexadecimal value of the integer
GLCD.print(integer, OCT) ; // prints the octal value of the integer
GLCD.print(integer, BIN) ; // prints the binary value of the integer
GLCD.print(integer, BYTE); // prints the ASCII character represented by the integer
GLCD.print(float); // prints a floating point number using two decimal places
GLCD.print(float, digits); // prints a floating point number using the given number of digits after the decimal point
GLCD.print(string) ; // prints the string

The println variants of these functions are also supported. GLCD.println(variable); will wrap to the next
line at the end of the print.

Printing strings can consume a lot of RAM. Printing strings using the flashStr prefix results in the compiler using flash rather than RAM to store the string
 GLCD.print("string") ; // string stored in RAM: the compiler reserves 7 bytes of RAM (string length + 1) to store the string
 GLCD.print(flashStr "string") ; // stores the string in Flash memory (Progmem) , no RAM is used to store the string

CharWidth()

Description: returns the width in pixels of the given character including any inter-character gap pixels following the character when rendered on the display.

Syntax
byte width = CharWidth(c);

StringWidth()

Description: returns the width in pixels of the given string

Syntax
byte width = StringWidth(string);

StringWidth_P()

Description: returns the width in pixels of the given string stored in program memory

Syntax
byte width = StringWidth_P(PgmStrring);

Text Area Functions

A text area acts like a virtual serial monitor and text output is displayed within the confines of a rectangle given in the DefineArea command.
All of the following text area function operates on a user defined text area. For example:
gText textTop = gText(textAreaTOP); // create a text area covering the top half of the display
gText myTextArea = gText(16,16,GLCD.Right-16, GLCD.Bottom -16); // create a text area covering the center of the display
Any Arduino print function can be used:
textTop.println("a line of text"); // print a line of text to the text area.
Any of these areas can be redefined using the text.DefineArea command
myTextArea.DefineArea(textAreaTopLeft); // change this text area to cover the top left quadrant of the display

See the download sketches for example usage.

User defined areas can be created using one of the three Define Area methods:

DefineArea()

Description: defines the rectangular area for text output. TextArea
The rectangular area can be specified using either: a predefined area, a rectangular

Syntax
DefineArea(preDefinedArea, scrollDirection); // create a text area using one of the predefined values
preDefinedArea is one of: textAreaFULL, textAreaTOP, textAreaBOTTOM, textAreaLEFT, textAreaRIGHT, textAreaTOPLEFT,textAreaTOPRIGHT,textAreaBOTTOMLEFT,textAreaBOTTOMRIGHT.
scrollDirection can be SCROLL_UP or SCROLL_DOWN, if scrollDir is omitted the direction will be SCROLL_UP.

DefineArea(area, x1, y1, columns, rows, font, scrolldirection);
As above but the height and width of the area is determined by the number of columns and rows for the given font.

DefineArea(area, x1, y1, x2, y2, scrolldir); (or width,height as you prefer
As above but x1, y1, x2, y2, determine the rectangular area of the text window

ClearArea()

Description: clears the current text area using the current font background color. The cursor is set to the upper left corner.

Syntax

ClearArea(); // clears the text area and sets the cursor to the upper left corner of the text area

EraseTextLine(row)

Description: clears all text on the given row within the text area.

EraseTextLine()

Description: clears text on the current line from the cursor to the end of the text area

CursorTo()

Description: move the cursor to the given row and column. When variable width fonts are used, the column calculation uses the width of the widest character.

Syntax
CursorTo(column, row);   // 0 based coordinates for character columns and rows

CursorToXY()

Description: moves the text cursor for the currently selected area to the coordinates given by x,y relative to the upper left corner of the text area.

Syntax
CursorToXY(x,y);

DrawString(str, x, y);
Description: prints the given string of characters starting from the given x and y coordinates. The coordinates are relative to the text area- use a text area covering the full display if you want the x,y parameters to be the same as the coordinates for Graphical functions like GotoXY.

Text.DrawString_P(ProgMemString, x, y);
Description: as above bu the string is defined in Program memory (flash)

All text area methods except DefineArea and ClearArea can be used in a default text area that covers the entire display, for example:

```
GLCD.CursorToXY(12,24); // position the text cursor
GLCD.print("hello world");
```

Puts_P()

Description: prints a string stored in program memory starting form the current cursor position.

Syntax
Puts_P(progMemString);

Legacy text output functions

The following functions are supported for backward compatibility with previous library versions, the GLCD.print functions have more functionality and are compatible with Serial.print routines.

PutChar()

Description: prints the given character at the current cursor position. It is suggested that the Arduino print character function,
GLCD.print(character) is used in new applications – it has identical functionality with the benefit of similar syntax to other arduino print methods.
Note that there is a suble difference in the way this function handles the newline character compared to the ks0108 library. The old PutChar() would treat the newline character just like any other character and since many fonts don't don't have a glyph defined for the newline its would be thrown away.

The new PutChar() does newline processing and will wrap and potentially scroll the text window.

Syntax
PutChar(c); // print the character c at the current cursor position (same as GLCD.print(c);

Puts()

Description: prints the given string of characters starting from the current cursor position. Note that the old library would process newlines in Puts(). It would wrap to the text line below the current line but back to the X position where string printing started.

The new code lets PutChar() process the newlines and so a new line will wrap to the line below but will wrap to the start X position of the text window rather than the X position when the Puts()started.

Also, the old Puts() assumed zero padding below a font while the new Puts() handles padding consistently across all the functions.
So the new Puts()/Putchar() will wrap 1 pixel lower than the old Puts() routine.

Syntax
Puts(string); // // print the string at the current cursor position (same as GLCD.print(string);

PrintNumber()

See GLCD.print(number);