arLCD Manual

Last Updated: March 15, 2014

Table of Contents

Before you begin

The EarthMake arLCD contains both an Arduino and an EarthLCD ezLCD-303 display on the same PCB. The arLCD also uses the ezLCD-303 to program the Atmel processor. No other boards are required to build many projects.

Therefore when connected to the USB and during configuration you will see references to the ezLCD-303. When you are upgrading firmware you will be upgrading the GPU on the ezLCD-303.

When downloading sketches it will be to the Atmel processor on the arLCD.

Getting Started

Downloading files & Installing the drivers

In order to write code and upload it to your arLCD, make sure you have the latest Arduino IDE installed and became familiar with it.

- **Download** the Arduino IDE from the Arduino website: http://arduino.cc/en/Main/Software

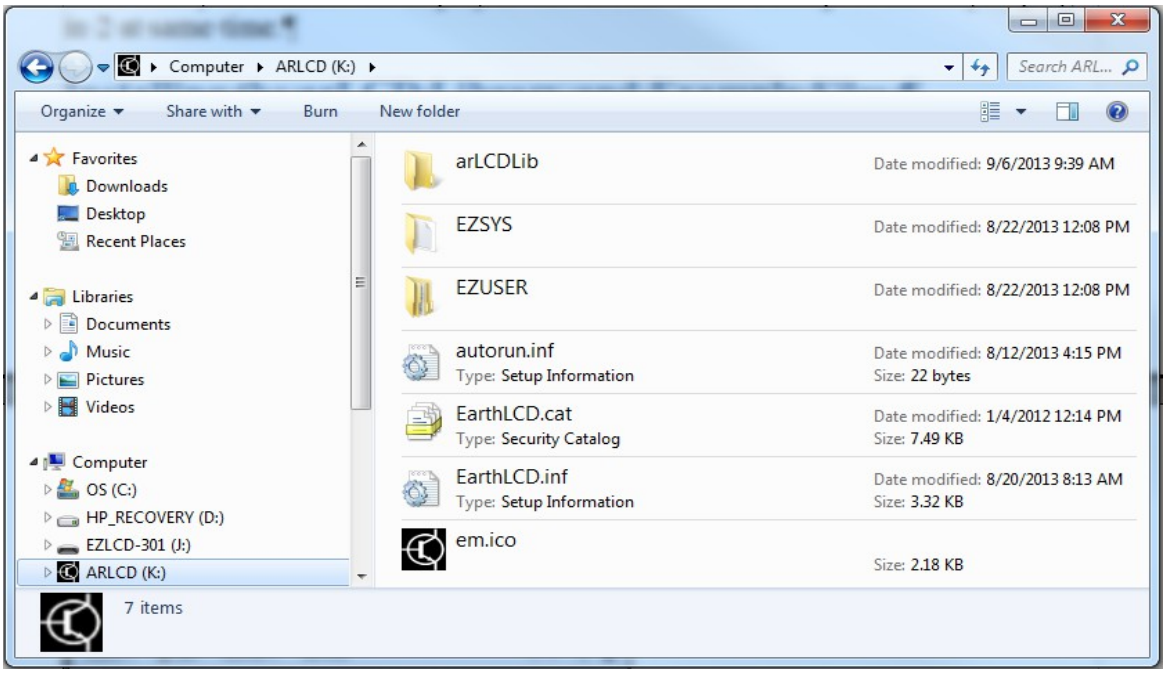
- Before connecting your arLCD **go to**: http://www.earthlcd.com/Downloads/arLCD_Software

And **download the latest zip files**:

arLCD_2pxx.zip
ezLCD-3xx-Firmware-Loader.zip

Now, **unzip the files**, and plug in the **arLCD** into a free USB port and wait a few seconds. Although the arLCD does not require any additional drivers, the system still needs to know what kind of hardware it is and how to configure it. The EarthLCD.inf included on the internal flash drive provides that information.

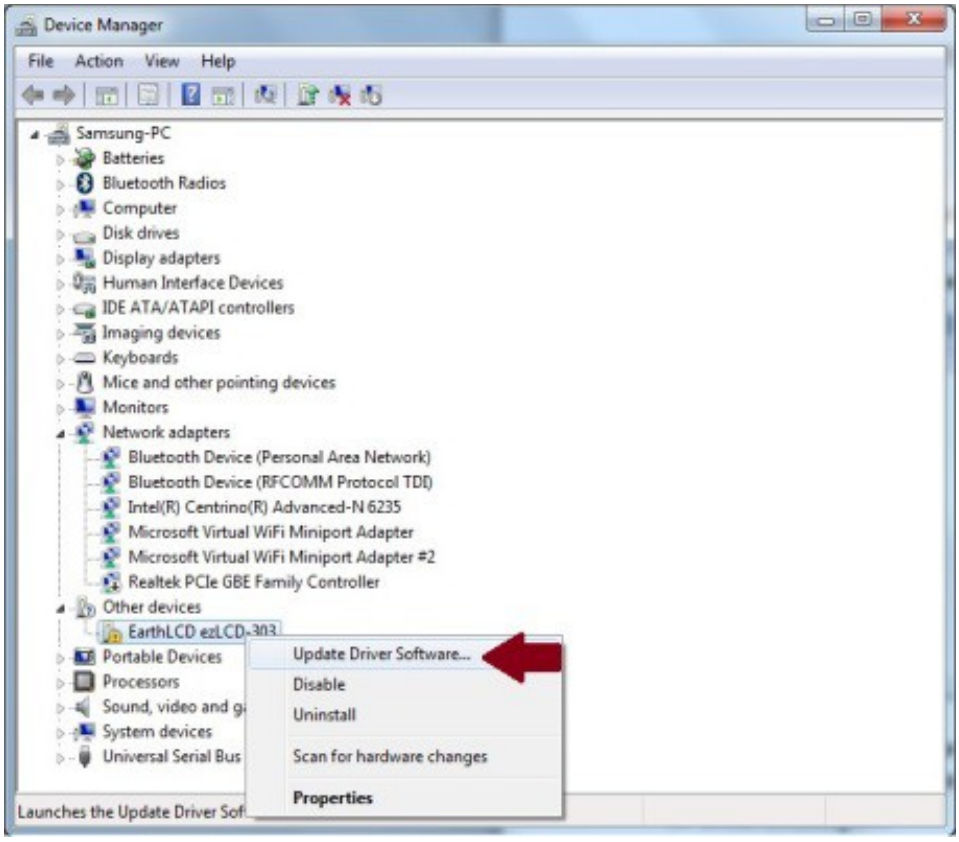
Windows will recognize the new hardware and show the “new hardware found” guide.



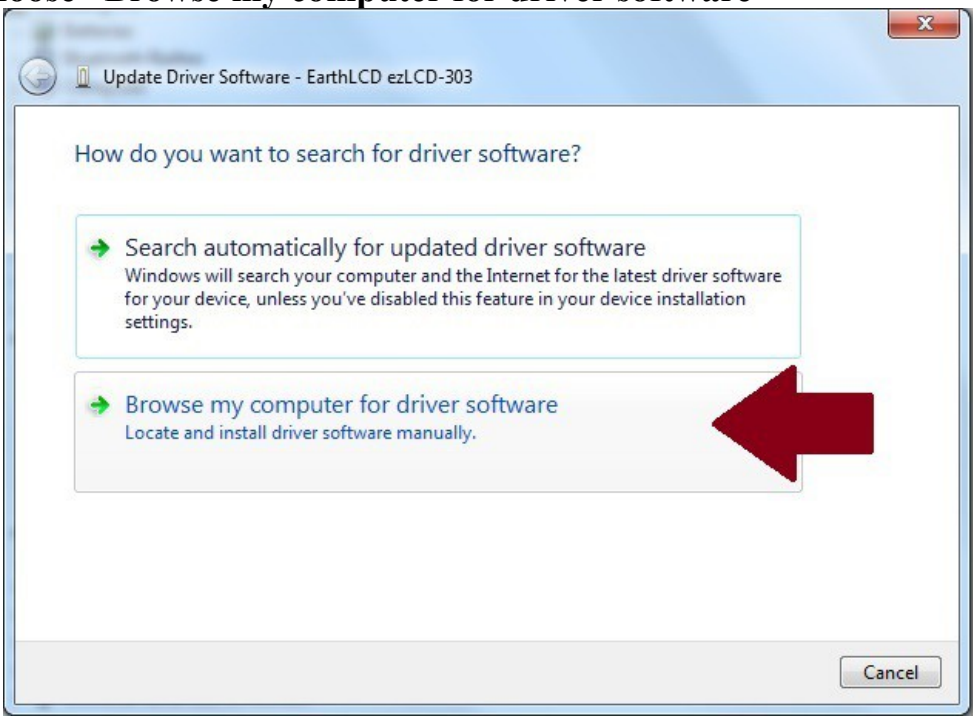
Any drivers should now be installed and ready to use. You can proceed to Installing the arLCD Library and Example Files. In most cases a reboot isn't necessary but it´s recommended.

Installing Device Driver

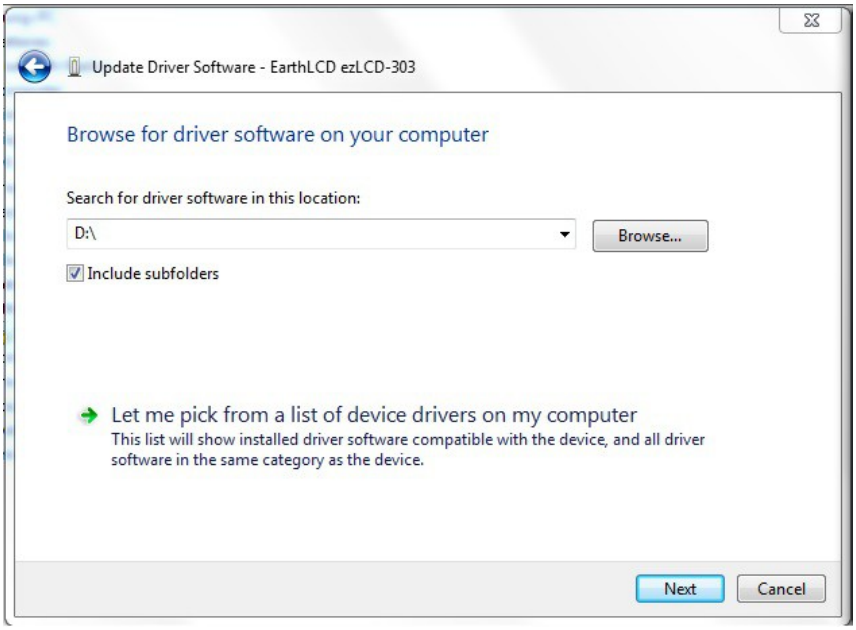
- Now **go to "Device Manager"** which you can find through the search bar in the "Start Menu".
- Then **right click on the EarthLCD device** and **select "Update Driver Software"** as shown in the picture below.

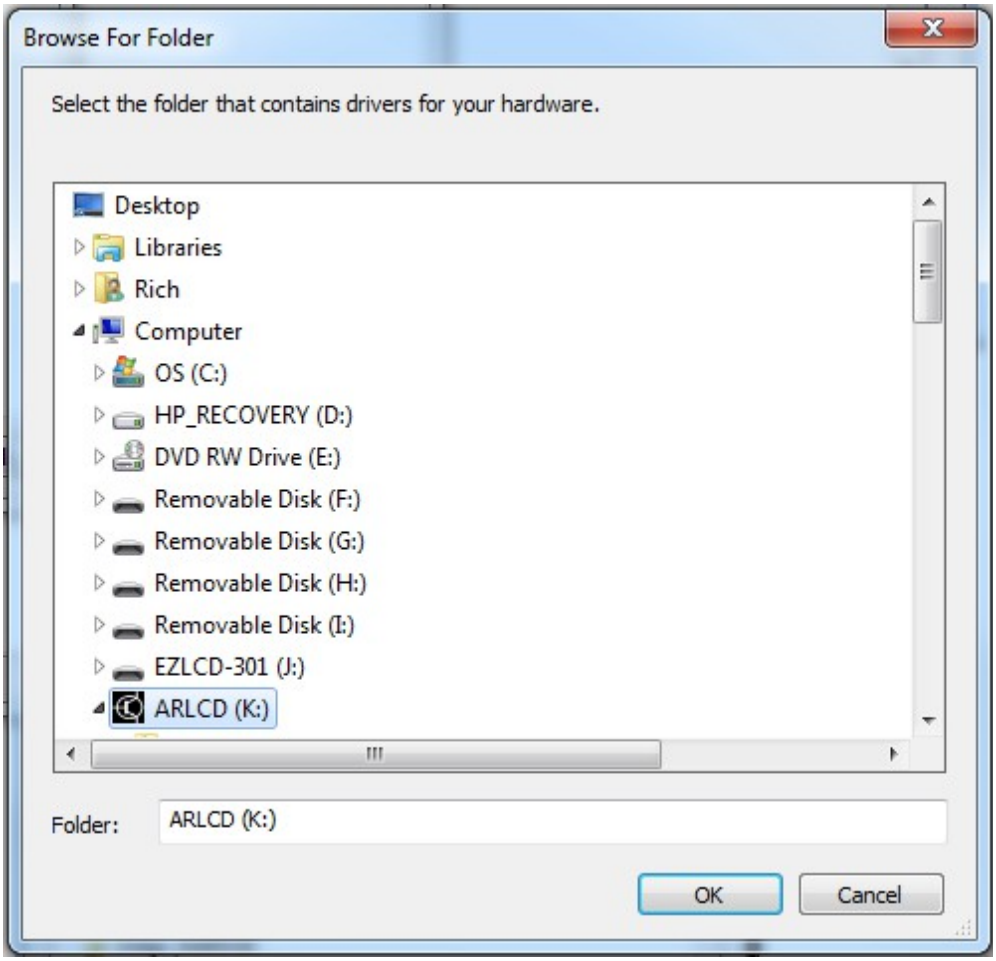


- **Choose “Browse my computer for driver software”**

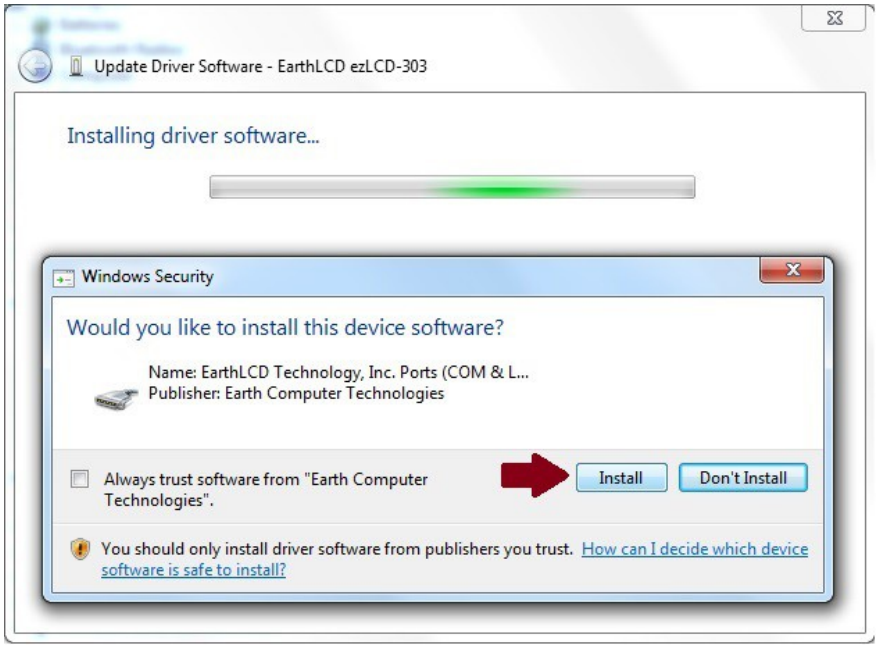


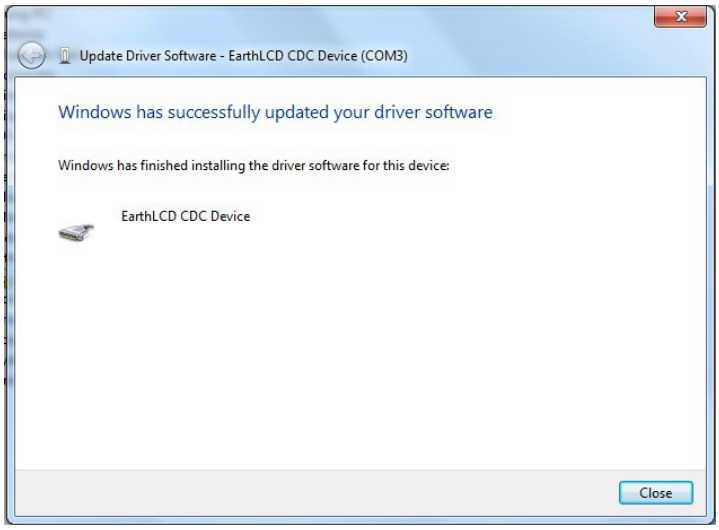
- **Click browse** and **select the ARLCD** and **click OK**.





- **Click "Next"** and the window below should appear. **Select "Always trust software from "Earth Computer Technologies""** and **click "Install"**.



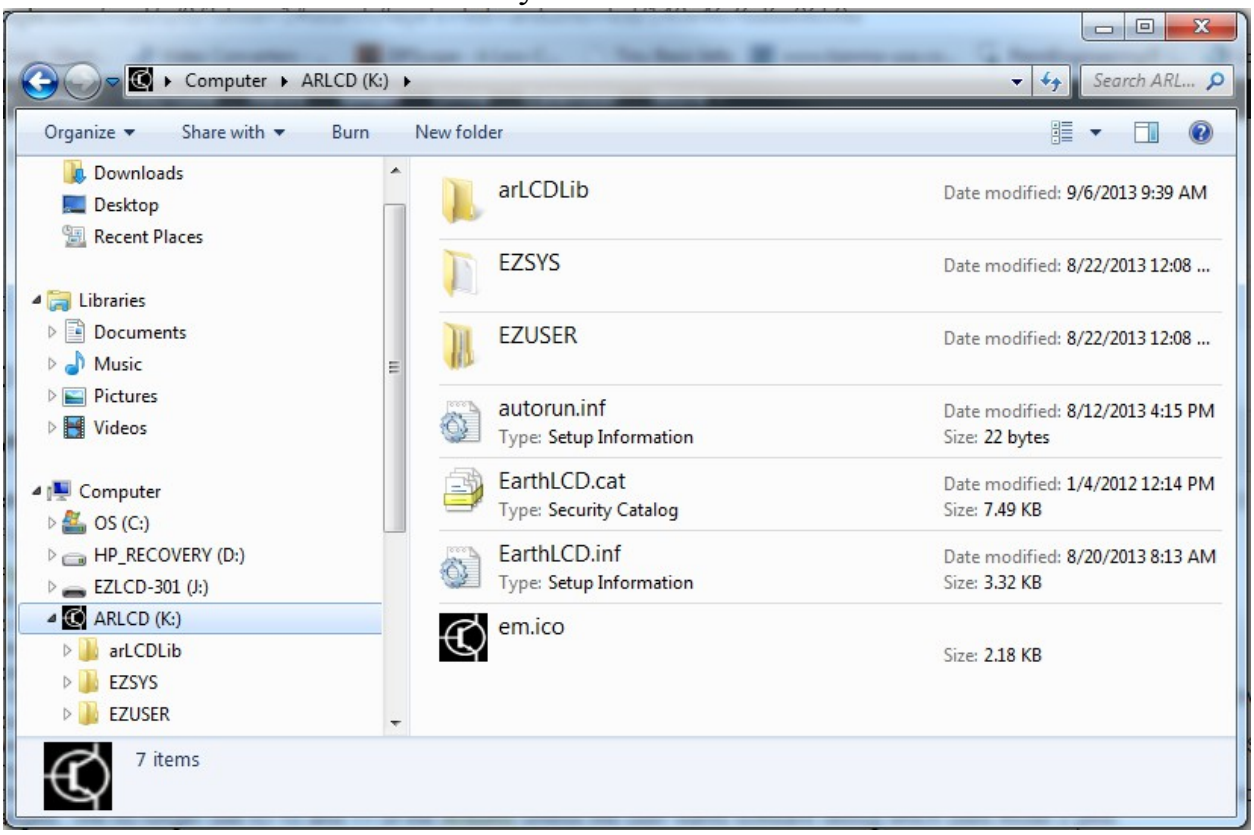


The driver is now installed. The window that appears will show you the COM port the device is using. **Remember the COM port** for later.

In this case we will use **COM3**.
Each time you connect the display it will use the same COM port unless you plug in 2 at same time.

Installing the arLCD Library and Example Files

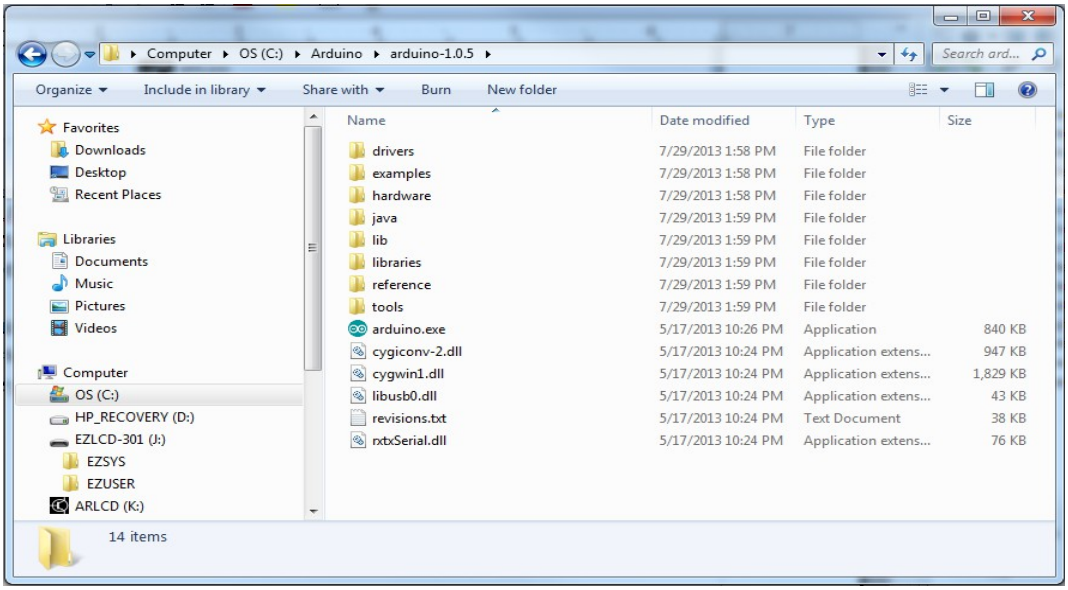
The arLCD comes with a library to be used with the Arduino IDE.



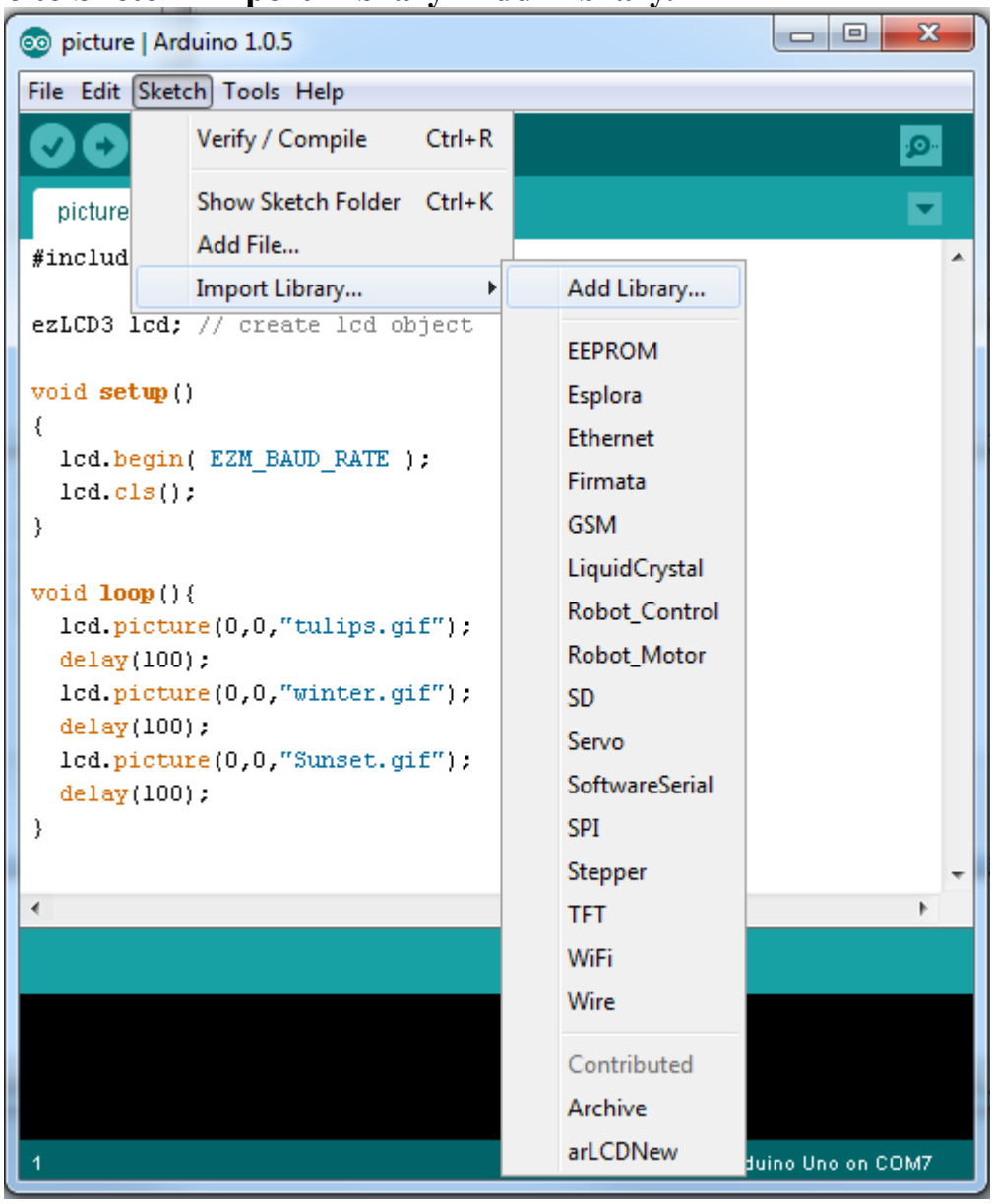
Select the arLCDLib as shown from your arLCD. This directory contains both the library and example files to be used with the arLCD.

To use the libraries and example files they must be installed into the Arduino IDE.

- **Open the Arduino IDE** on your computer

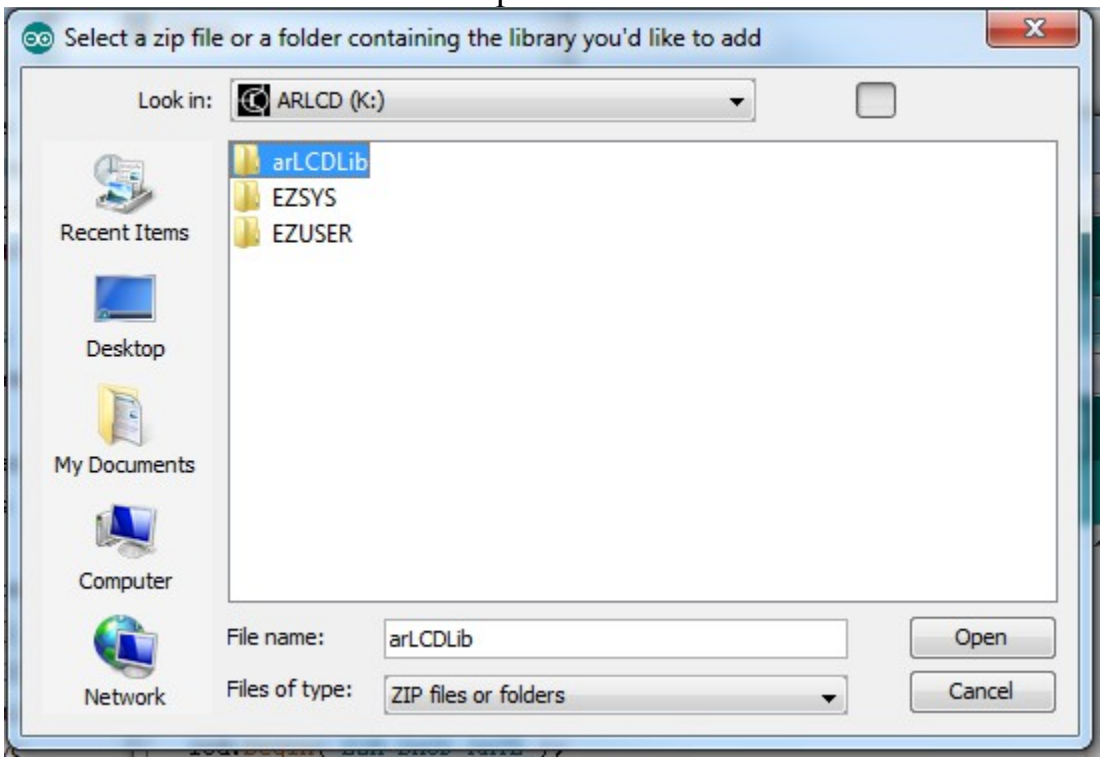


- **Go to Sketch>Import Library>Add Library.**



You must remove old library if present by deleting it from the Arduino\libraries directory in the Documents directory.

Now select the arLCDLib and click Open.

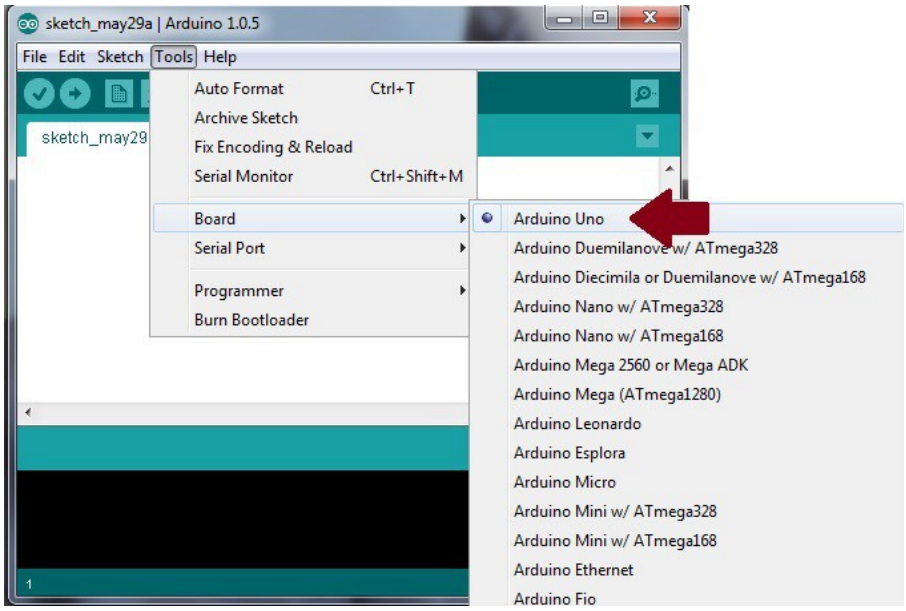


Now the arLCD Library is installed.

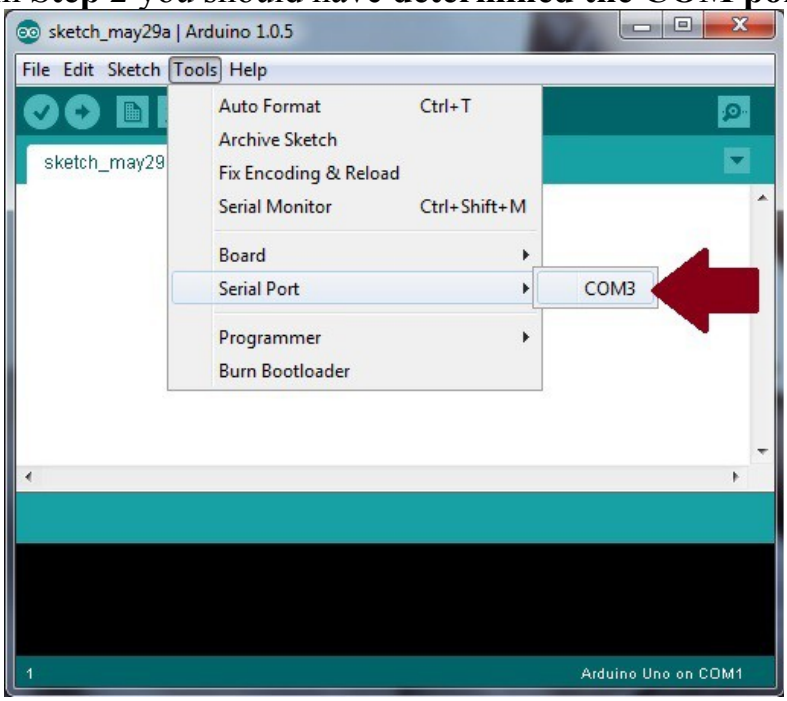
Loading an example sketch into your arLCD

You are almost ready to load the sketch onto your arLCD. You just need to set your Arduino IDE to the correct type of hardware and COM port.

- From the Arduino IDE toolbar select **Tools -> Board -> Arduino Uno**.

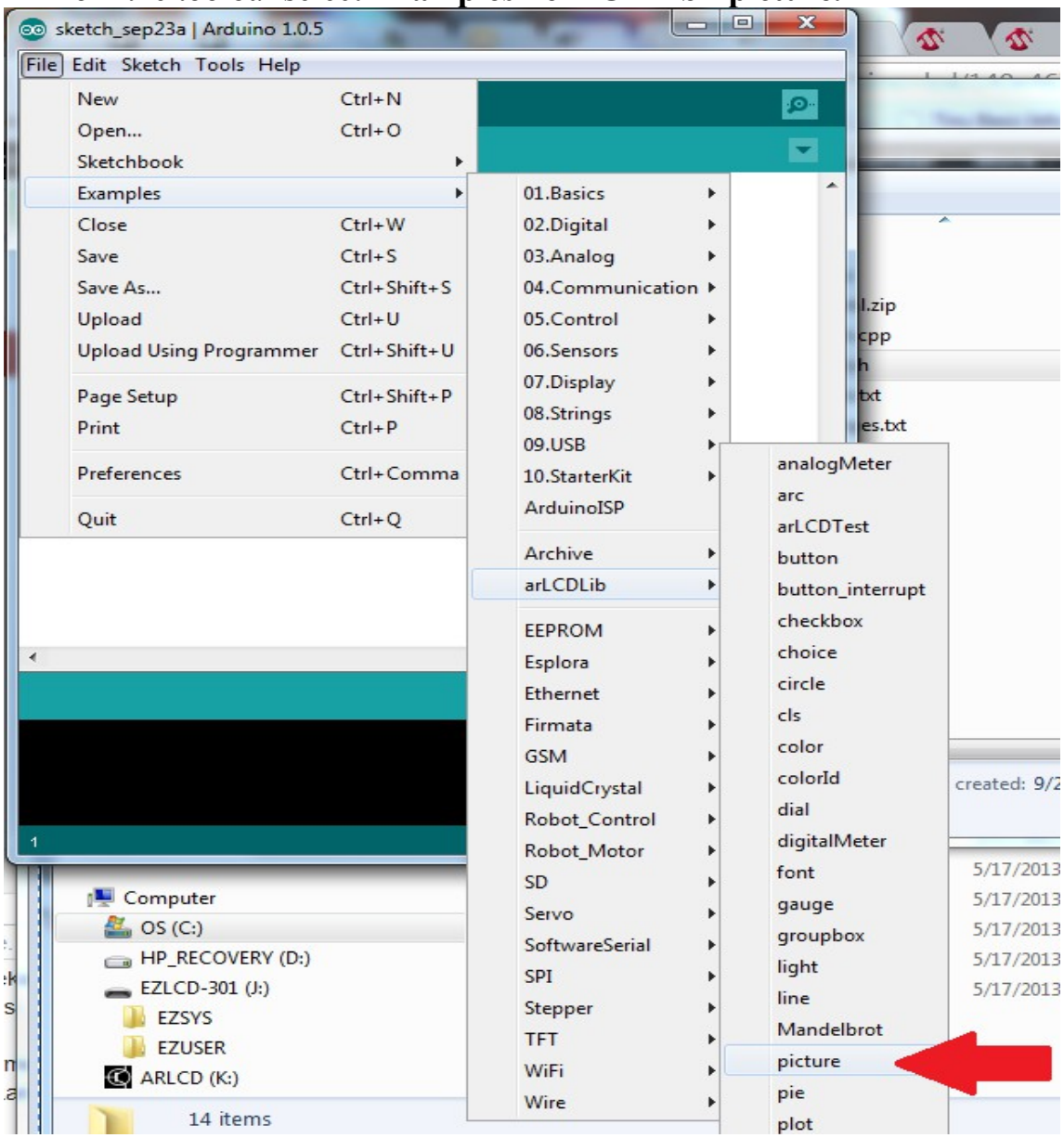


- Now from **Step 2** you should have **determined the COM port**.

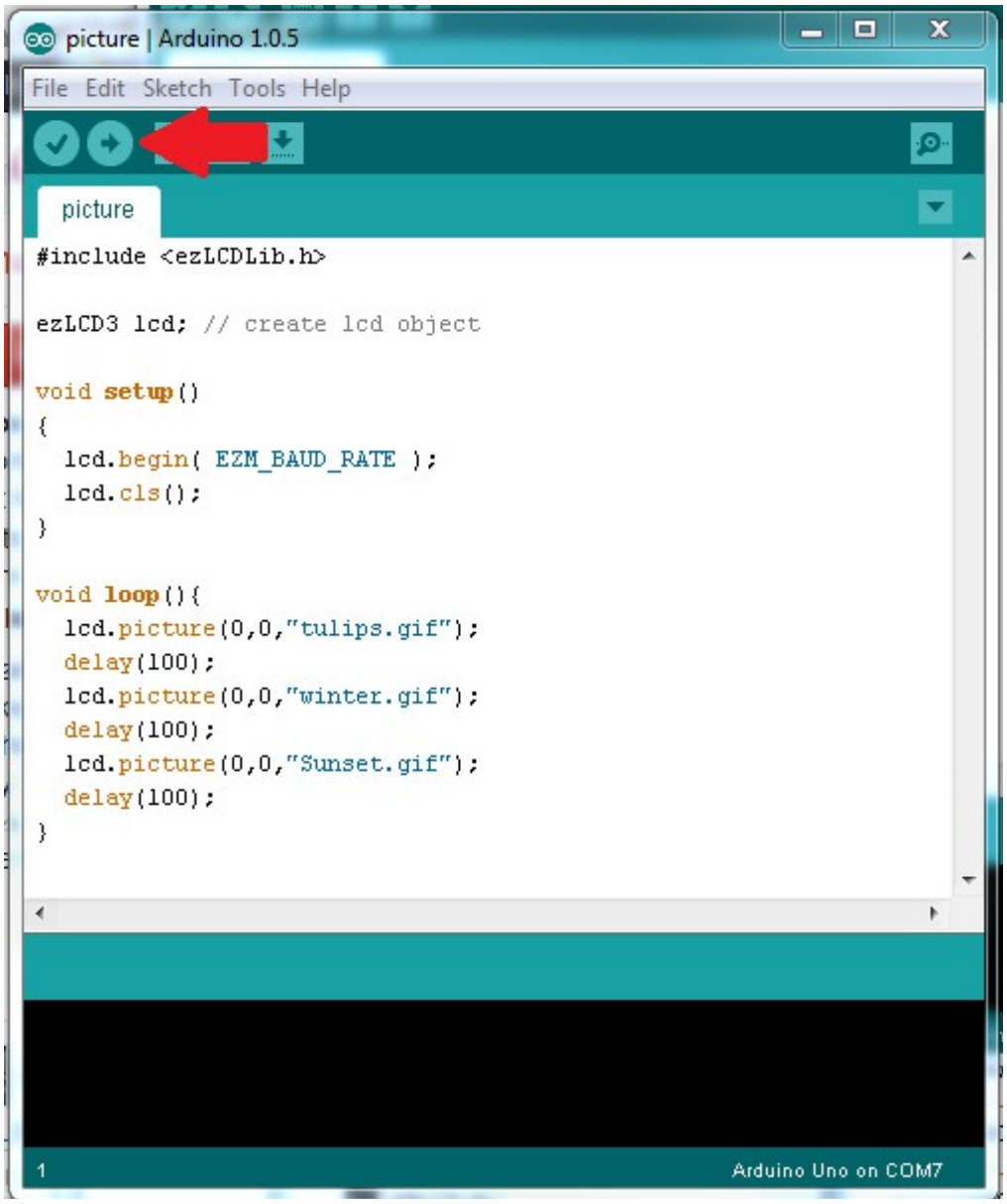


Go to **Tools -> Serial Port -> COM3** (This will vary on your computer)

- From the toolbar select **Examples> ezLCDLib> picture.**



- Now compile and upload the “picture” example to the arLCD by pressing the right arrow button.



The picture example should take a few seconds to compile and be downloaded into the arLCD. You should see 3 GIF files sequentially displayed on the screen. If not make sure the three GIF images are in the EZSYS/IMAGE/ directory.

Recap and more information

Now that you are up and running you can try selecting some of the other 25+ examples from the library and explore how they work. Make some code changes and see how it affects the display. Don't be afraid to try things. If you don't understand lines in the sketch, make some changes and see what happens.

Remember to select each sketch from the File menu.

File > Examples > arLCDLib > selected *sketch*

and then upload

File > Upload

for each sketch you want to try. You can modify and re-upload it as many times as you want. If you want to save it, we recommend you save it with another name you can remember so you can go back and look at the original.

On RESET the ezLCD GPU will check the touchscreen to see if during reset the user wants to bypass the normal startup.ezm. The user can supply alternative startup files or none at all.

Note: Without one startup the COM ports are not setup and communication with the Arduino cannot happen. The startup.ezm files are changed by direct access to the flash drive on your PC.

Keep in mind, in order to communicate with your arLCD, the startup.ezm file must be setup correctly. If a startup.ezm is in the \EZUSER\MACROS\ then it will be used. If not it will use the startup.ezm in the \EZSYS\MACROS\ directory which is already in use.

The normal STARTUP.ezm is used without touching the screen on reset. You can use different startup files by holding a certain sections of the screen while resetting the arLCD (reset button on the back). The Upper left is STARTUP1.ezm, upper right is STARTUP2.ezm. Lower right is STARTUP3.ezm and lower left is STARTUP4.ezm. STARTUP5.ezm is executed if the screen is pressed but not inside the 50 x 50 pixel corners. If the startup.ezm file is not found, it is simply skipped. The other startup files can be found in our recent file systems online.

The latest filesystem includes;

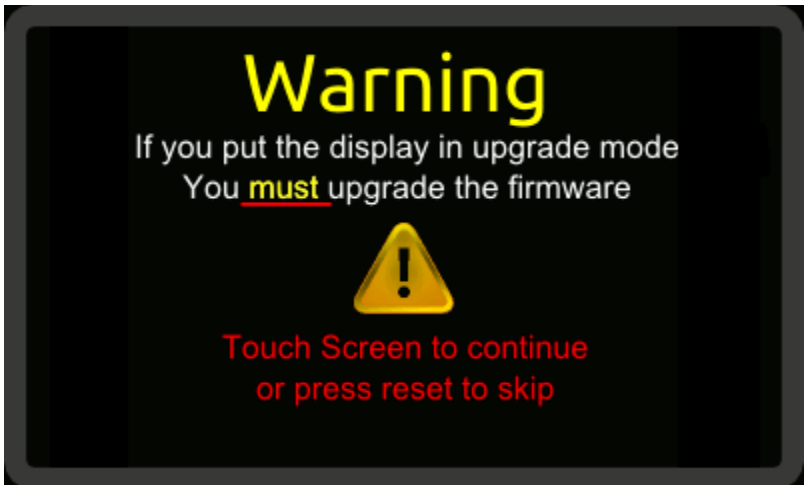
startup.ezm 'Configures the arLCD for normal use and programming.

Startup1.ezm 'Safemode. Disables the arLCD from receiving commands from the user. Use safemode if the user programs a sketch into the arLCD which disables or otherwise causes the GPU into an unwanted state and no longer can update new sketches.

Startup2.ezm 'configures the arLCD for firmware upgrade.

If the user wants to upgrade the firmware they simply touch the screen again and release. If the user did not want to upgrade the firmware, simply press the reset again before touching the screen.

If the user does put the firmware into upgrade mode, reset will not recover. The firmware must be upgraded. See the upgrading firmware section.



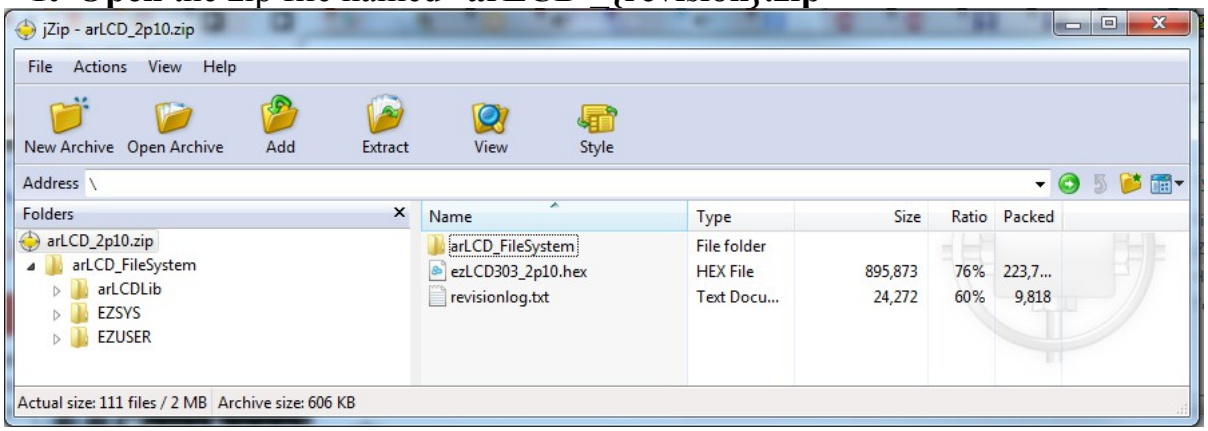
Visit our website or link below for more documentation and et al.

EarthLCD Downloads page:
http://www.earthlcd.com/Downloads/arLCD_Downloads

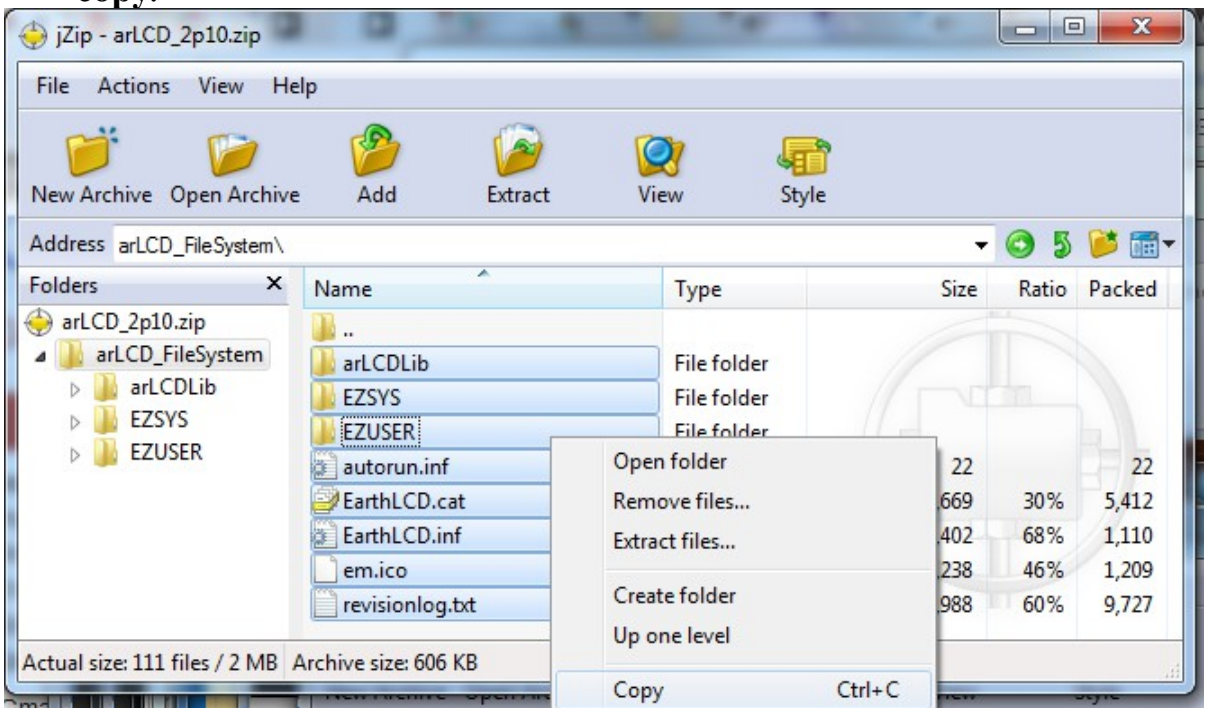
Updating the Filesystem

In order to get the most out of your arLCD, it is recommended to update the filesystem to the latest version, which you downloaded in Getting Started .

1. **Open** the zip file named "**arLCD _{revision}.zip**"

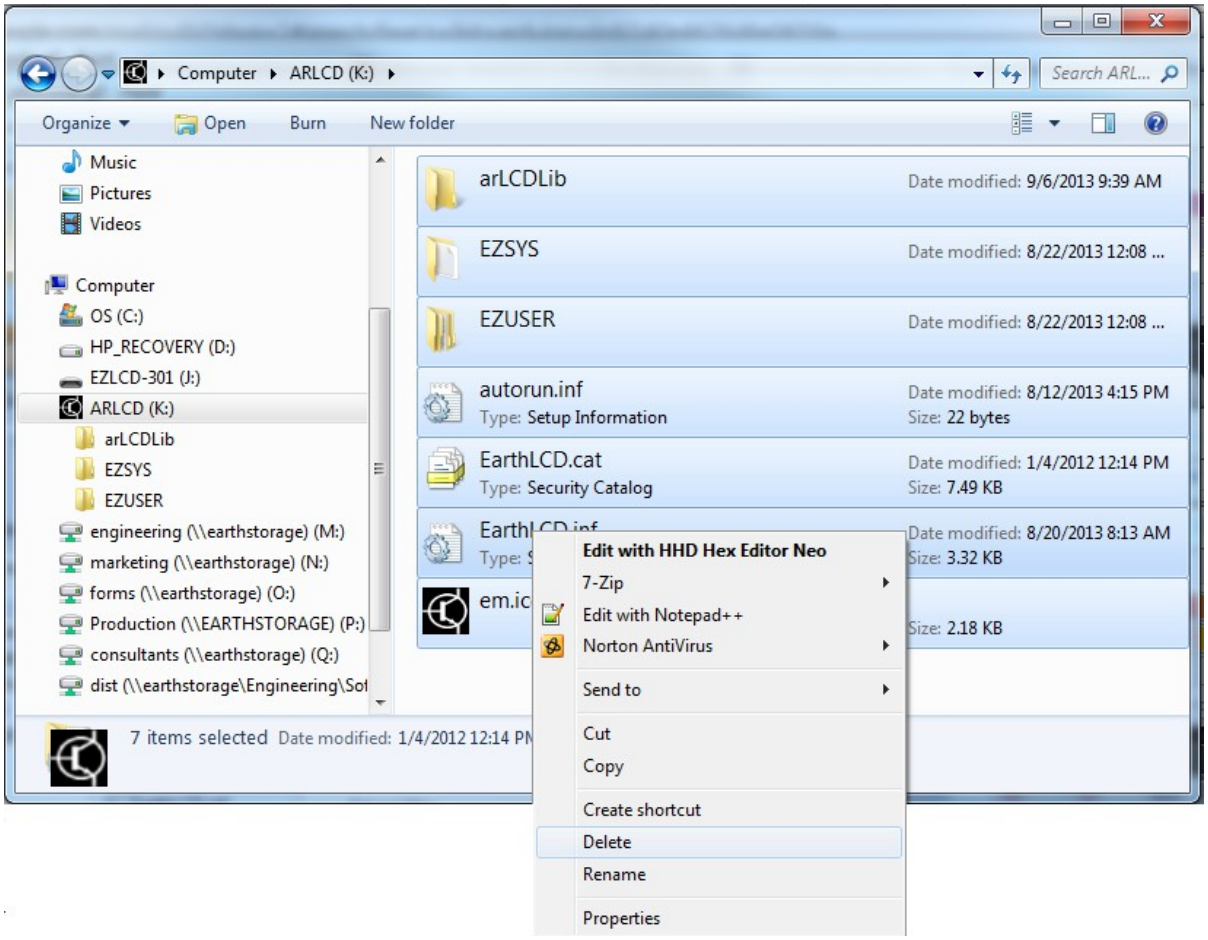


2. Like the pictures below, **select all files in the arLCD_FileSystem and copy**.



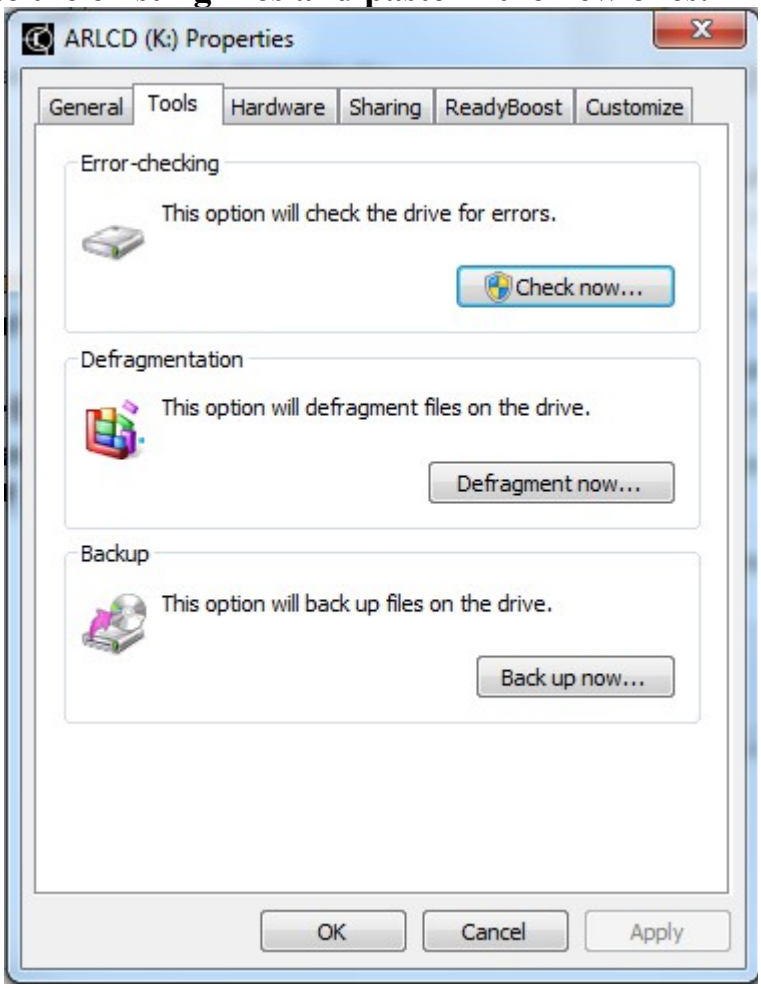
3. Go **to My Computer>ARLCD**

4. **Select all the files on the arLCD flash drive and delete them.**



5. Now **replace all the files** with the new ones with a simple paste command.
6. **Restart the arLCD** by pressing the reset button on the back of the display.

Note: Do not reformat the flash drive unless instructed so by a support engineer. Just delete the existing files and paste in the new ones.



You can run the Error checking and Defragmentation prior to upgrading the filesystem to clear out any problems that you created.

Upgrading the arLCD Firmware

A Windows PC is required to upgrade the firmware on an arLCD.

IMPORTANT: Never use any upgrade firmware that is not designed for the display you have.
Only arLCD firmware (arLCD_2pxx.hex or ezLCD-303_2pxx.hex) should be installed. Using the wrong firmware could make your unit inoperable and leave no way to install the correct firmware.

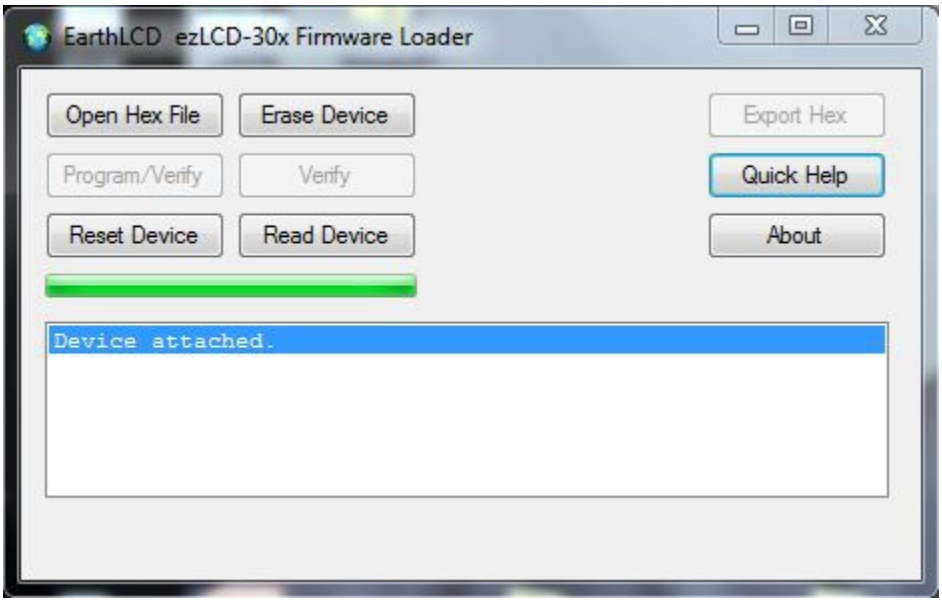
Before upgrading your arLCD firmware you should backup any macros or fonts you have created by copying them from the arLCD flash drive to your computer.

To put the arLCD in upgrade mode hold the top right corner of the screen and press the reset button. Then follow the onscreen instructions.

Once you put the arLCD in firmware upgrade mode, it cannot come out of this state until new firmware is programmed using the provided program even if you unplug it or reset!

1. Have the **Firmware Loader downloaded and installed and running**.
2. Download the most **recent firmware**, which you did in **Getting Started**.
3. **Plug in your arLCD to a USB port**.

4. **The arLCD should attach to the Firmware Loader as shown below**.



5. Click "**Open Hex File**" and locate where you downloaded the firmware. (ezLCD-303_2pxx.hex)

6. Click "**Program/Verify**"

7. When the programming and verification is complete click "**Reset Device**"

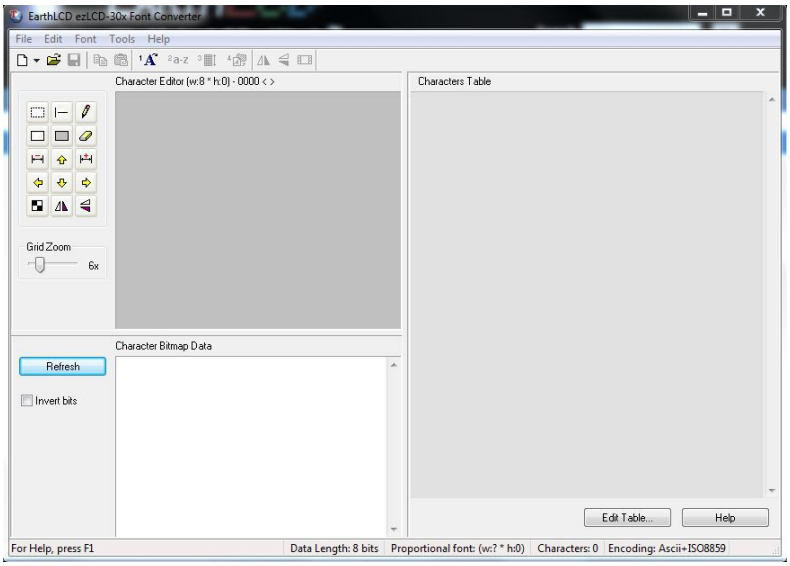
8. **You are done!**

Font Converter

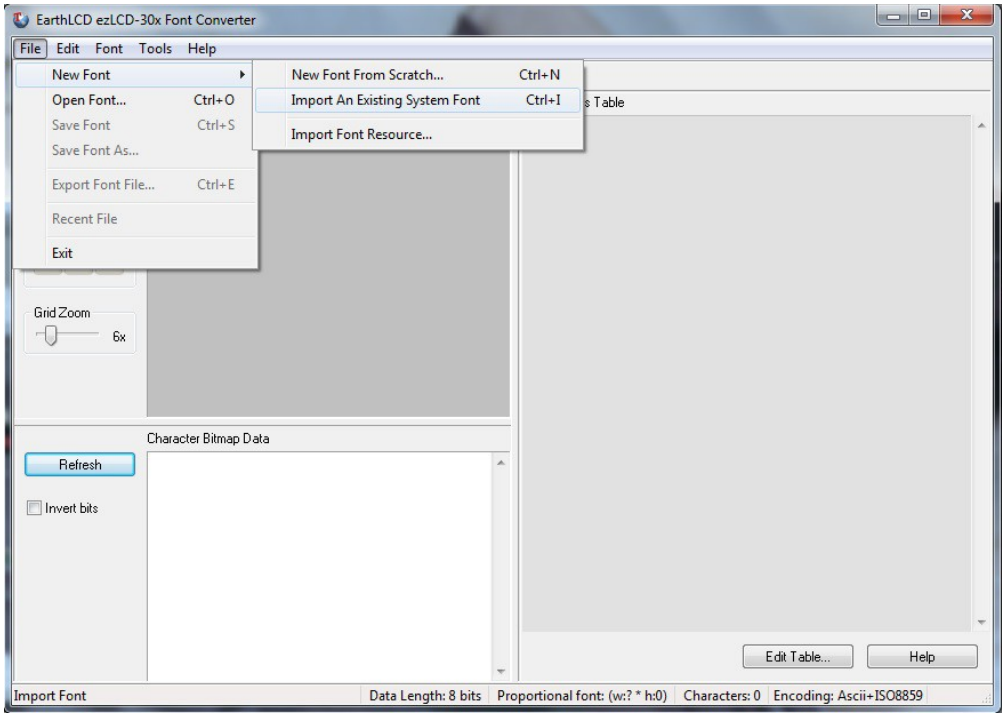
The ezLCD-3xx Font Converter is a great way to customize your own fonts, resize fonts or upload different fonts onto your arLCD. Make your own fonts, edit fonts, or upload pre-made ones to create a spectacular display!

1. **Download & Install** the ezLCD-3xx Font Converter.
2. **Download & Install** a font of your choice

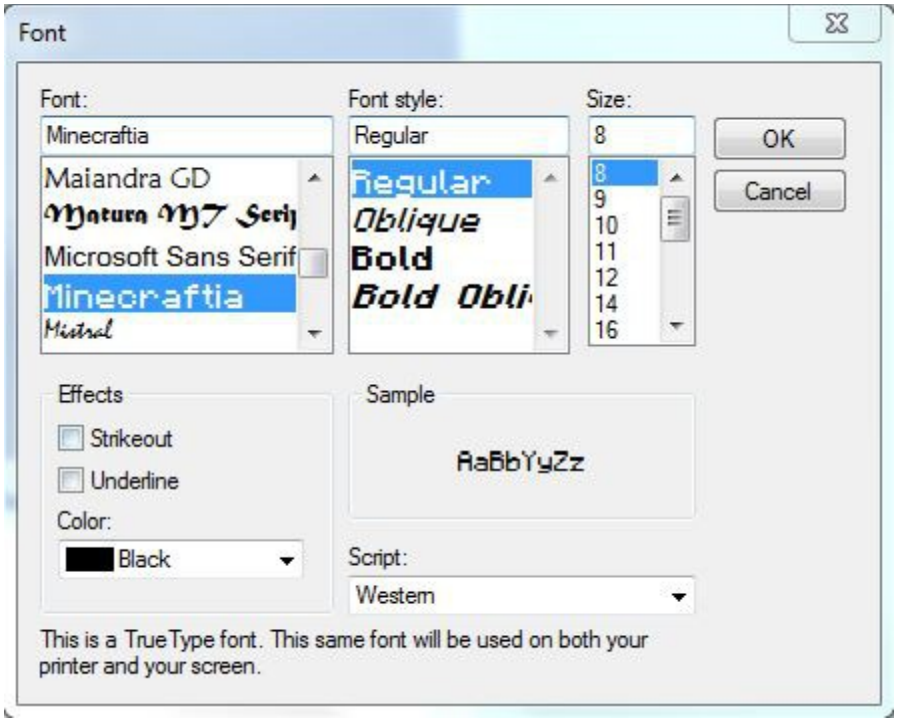
3. **Open** the Font Converter.



4. Go to **File > New Font > Import An Existing System Font**



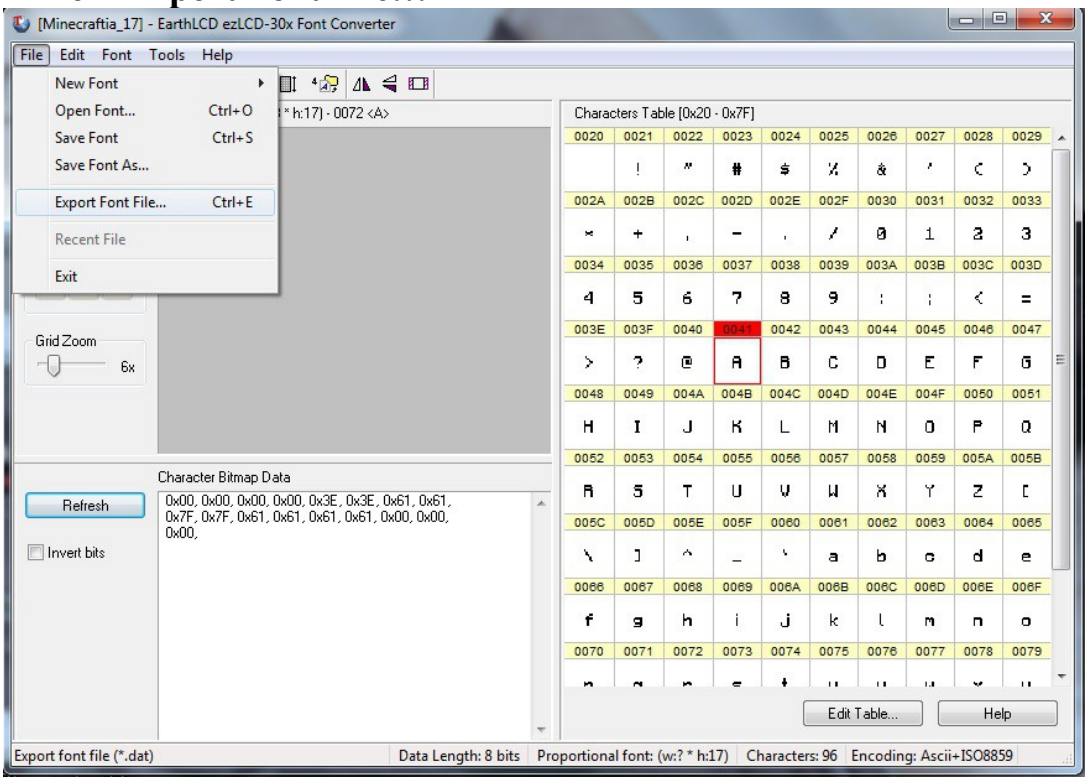
5. **Find** the font you installed & **Select** the settings you want. Click **ok.**

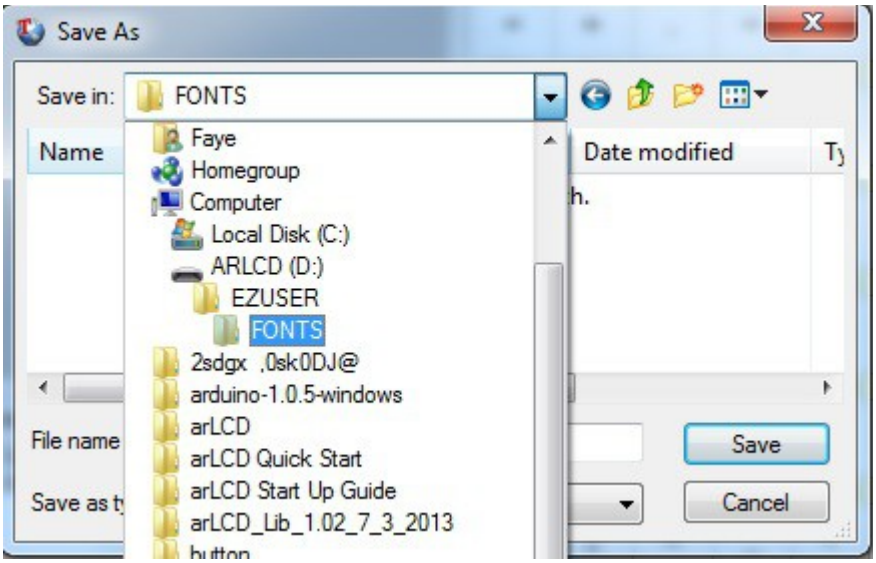


6. Edit it however you want. Play around with the options.
   NOTE: Below the options bar, you will see “Character Editor”. “w” means width and “h” means height in pixels of the font box.

Hint: The font will take up less space on the screen if you remove all the excess blank pixels above and below the selected font.

7. When you are ready to upload the font onto you arLCD, go to
   File > Export Font File…





8. **Save** your font in **ARLCD > EZUSER > FONTS**

Hint: The file name must be only 8 characters or less with a 3 character extension (myfont.ezf).

9. Your font is now ready to use! Create a sketch to test it out!
   NOTE: Make sure the font name has double quotations around it.
   lcd.font("myfont");

Basic Commands

To use the arLCD library some global storage is used.

```
volatile unsigned int timedOut;
volatile unsigned int error;
volatile unsigned long timeOutCount;
volatile unsigned long timeOut;
```

timedOut indicates the communication between the GPU and the Atmel has retried and timedout.

Error indicates an error has happened on the previous command. Like trying to open a picture that is not there.

TimeOutCount and timeOut are used to determine if a time exists.

CLS

The CLS command will clear the screen to black. It will also remove all widgets and clear any previous widget IDs being used.

The user has the option of specifying the clear screen background color (bcolor). The user also has the option of specifying the foreground color which will be used for text following the command.

Command:

lcd.cls( );

lcd.cls(bcolor);

lcd.cls(bcolor, fcolor);

Parameters:

bcolor is the background color used for clearing the screen
fcolor is the foreground color. Same as lcd.color fcolor.

COLOR

The COLOR command sets the foreground color which will be used for text. Line and other primitive commands.

Command:

lcd.color(fcolor);

Parameters:

fcolor is the foreground color used follow on commands.

COLORID

The COLORID command creates addition color numbers outside of the standard 168 colors defined. This allows the user to use colors outside of the color numbers defined.. The newly define colorID can be called by the color(), or any other command with a color parameter, to set the foreground color which will be used for text, line and other primitive commands.

Command:

colorID(ID, r, g, b);

Parameters:

ID is the ID, 168-199, used to reference the color using color()
r is the value, 0-255, of the red channel in the RGB color scheme
g is the value, 0-255, of the green channel in the RGB color scheme
b is the value, 0-255, of the blue channel in the RGB color scheme

XY

The XY command sets the position pointer (sometimes called cursor) for follow on print commands.

Command:xy x, y

lcd.xy(**x**, **y**);

Parameters:

x is the pixel position from left (0) to right (319).
y is the pixel position from top (0) to bottom (219).

The XY command can also be used to get the current position.

lcd.xy( );

These other commands do the same thing.

lcd.getX( );

lcd.getY( );

The X and Y are also returned to two global variables X and Y.

```
volatile unsigned int X;
volatile unsigned int Y;
```

LIGHT

The LIGHT command sets current backlight level of the display with optional timeout and timeout brightness. If timeout is not specified, no timeout will happen. Setting timeout to zero will terminate timeout.

After **timeout** in seconds the backlight brightness will be changed from the active brightness (**level**) to the inactive brightness (**timeoutbrightness**). Touching the screen will resume the brightness back to active brightness.

Command:light level

lcd.light(**level**);

lcd.light(**level, timeout, timeOutBrightness**);

Parameters:

level is the active brightness level of the backlight from 0 to 100.
timeout is active timeout in seconds before the backlight is changed.
timeoutbrightness is the inactive backlight level.

The light command can also be used to get the current level. A value form 0 to 100 will be returned.

lcd.light( );

GETXMAX and GETYMAX

The GETXMAX and GETYMAX commands return the max width and height of the display screen.

Command: getXmax or getYmax

lcd.getXmax( ); get the X max value of display (319)

lcd.getYmax( ); get the Y max value of display (239)

Parameters:

None

TOUCHX, TOUCHY and TOUCHS

The TOUCHX, TOUCHY and TOUCHS commands return the max width and height of the display screen.

Command: touchX, touchY or touchS

lcd.touchX( ); get the X value last location touched on the display (0-319).
lcd.touchY( ); get the Y value last location touched on the display (0-239).

lcd.touchS( ); get the status of the last touch event on the display.

This can return

0=No press
1=Moved
2=Pressed
3=Stillpressed
4=Released

STRING

The STRING command is used to set a string of a stringID. Most strings are set prior to use by follow on commands. Strings can be upto 64 characters long.

Command: string

lcd.string(ID, “string”); set the string for ID.

lcd.getStringID(ID, pointer); get the string for ID and put it in pointer.

Parameters:

ID is the ID to use to reference the string.

string can be any ascii string up to 64 characters

```
lcd.string( 1, "ALOG" ); // stringId 1

char temp[64];//setup string
lcd.getStringID( 1, temp ); // stringId 1
lcd.print( temp );
```

PRINT

The PRINT command is used to print a string to the display. The command will use the current color and font. Strings can be up to 64 characters long.

Command: print

lcd.print(“Hello World”); //prints the string “Hello World”.

lcd.printStringID(ID); //prints the string in ID.

```
lcd.print( “Hello World” );  //print the text to the display
lcd.string( 1, "Hello World” ); // stringId 1
lcd.printStringID( 1 );//prints “Hellow World” to the screen
```

PRINT Escape Sequences

When printing a string, various escape sequences can be used to modify the output without additional commands. Things that can be changed are color, font orientation and XY location.

COLOR: \[xxxm
\[dddm where ddd can be any of the colors in the colorID table.

FONT ORIENTATION:
\[dr where d can be **0** = 0º, 1 = 90º. 2 = 180º and 3 = 270º

XY LOCATION:
\n cause the position to go down one line (Y+1)
\r cause the position to go to the left side (X = 0)
\[dddx where ddd can be any valid X location (0-319)
\[dddy where ddd can be any valid Y location (0-239)
Note: One line is defined as the height of the current font.

```
//note the use of escape sequences below allow 1 line to replace 9 lines of code
//  lcd.xy(55,0);
//  lcd.color(RED);
//  lcd.print("L");
//  lcd.color(GREEN);
//  lcd.print("C");
//  lcd.color(BLUE);
//  lcd.print("D");
//  lcd.color(WHITE);
//  lcd.print(" Hardware Test V1.0");
  lcd.print("\\[55x\\[0yar\\[4mL\\[9mC\\[12mD\\[3m Hardware Test V1.0");
```

DEBUG

The DEBUG command is used to print information to the Serial Monitor in Arduino. Naturally it can be used for debugging code by printing the state of whatever variable or widget you would like.

Command: debug

```
lcd.Debug( “Hello World”);       //prints “Hello World” to the serial monitor.

lcd.Debug(variable, %format);  //prints variable to serial monitor in specified
                                          //format

unsigned long ultemp = 0xa5dd;
//print ultemp as an unsigned hexadecimal integer
lcd.Debug( ultemp, "%X" );       //prints “A5DD” to serial monitor
//print ultemp as unsigned long
lcd.Debug( ultemp, "%lu" );      //prints “42461” to serial monitor
//print ultemp as unsigned octal
lcd.Debug( ultemp, "%o" );       //prints “42461” to serial monitor

long ltemp = 0xa5dd;
// print ltemp as signed long integer
lcd.debug(ltemp, “%ld”);          //prints “42461” to serial monitor

int inttemp = 0xa5dd;
//print inttemp as a signed integer
lcd.debug(inttemp, “%i”);         //prints “-23075” to serial monitor

unsigned int uinttemp = 0xa5dd;
//print uinttemp as an unsiged integer
lcd.debug(uinttemp, “%u”);       //prints “42461” to serial monitor
```

CLIPPING

The CLIPPING command sets an area of the screen which is protected from change by primitive commands. First a clipping area is defined. Second clipping is enabled or disabled. When enabled primitives only work inside the clipping area.

is used to print a string to the display. The command will use the current color and font. Strings can be upto 64 characters long.

Command:

lcd.clipArea(L, T, R, B); //define the area with left top and right bottom
lcd.clipEnable(TRUE/FALSE); // turn it on or off

Parameters:

L is the left x position.
T is the top y position.
R is the right x position.
B is the bottom y position.

lcd.clipArea(50, 50, 200, 150); // define the area
lcd.clipEnable(TRUE); // turn clipping on
any attempt to write outside the window will be ignored

DRAWLED

The DRAWLED command draws a simple LED on the display to replace any physical LED that might otherwise be needed. This allows a larger number of possible LEDs and requires no pins to drive the LEDs. LEDs can be any color defined. The size of the LED be assigned as your needs dictate.

Command:

lcd.drawLed(x, y, dim, colorLed, colorHigh);

Parameters:

x is the x position of the LED.
y is the y position of the LED.
dim is the radius of the LED outer circle.
colorLed is the color of the LED. Used colorID or names (RED).
colorHigh is the color of the LED highlight. Typically WHITE.

lcd.drawLed(160, 50, 12, BLACK, WHITE);
draws an LED with radius of 12 pixels at X=160 and Y=50. Black is used for the color to show its OFF. White is used as the highlight colors.

lcd.drawLed(160, 50, 12, RED, WHITE);
draws an LED with radius of 12 pixels at X=160 and Y=50. RED is used for the color to show its ON. White is used as the highlight colors.

lcd.drawLed(160, 50, 12, LIME, WHITE);
draws an LED with radius of 12 pixels at X=160 and Y=50. LIME is used for the color to show its ON. White is used as the highlight colors.

RxUART

The RxUART command is used to receive data from a UART specified by the IO pin. This is useful to add additional hardware UARTs to Arduino with limited UARTs available. (Uses Arduino pin numbers) Pin 0 is used for USB. Make sure pin numbers don't conflict with other pin uses.

Command:

result = lcd.RxUART(pin);

Parameters:

Pin is the Arduino pin number the data will be received on.
D11, D12 and D2 are options. Pin 0 would use USB.
Result can be -1 when not configured. -2 when not valid UART. 0 when no data is available. Otherwise its received data.

```
result = lcd.RxUART( 11 );//see if was received
```

TxUART

The TxUART command is used to send data from a UART specified by the IO pin. This is useful to add additional hardware UARTs to Arduino with limited UARTs available. (Uses Arduino pin numbers) Pin 0 is used for USB. Make sure pin numbers don't conflict with other pin uses.

Command:

lcd.TxUART(pin, data);

Parameters:

Pin is the Arduino pin number the data will be received on.
D10, D13 and D2 are options. Pin 0 would use USB.
Result can be -1 when not configured. -2 when not valid UART. 0 OK

```
lcd.TxUART( 10, 77 );  //Single character

lcd.TxUART( 10, "Hello" );  //String

lcd.TxUART( 13, 0x88 );//Single character
```

FLUSH

The FLUSH command is used to clear the serial receive buffer of a UART previously configured on one of the valid IO pins.

Command:

lcd.flush(pin);

Parameters:

Pin is the Arduino pin number associated with a UART.
D10, D11, D12, D13 and D2 are options. Pin 0 would use USB.
Result can be -1 when not configured. 0 OK

```
lcd.flush( 10 );  //Flush the UART buffer
```

AVAILABLE

The Available command is used to sense when a UART receive contains a character. This is useful to add additional hardware UARTs to Arduino with limited UARTs available. (Uses Arduino pin numbers) Pin 0 is used for USB. Make sure pin numbers don't conflict with other pin uses.

Command:

lcd.Available(pin);

Parameters:

Pin is the Arduino pin number the data will be received on.
D10, D13 and D2 are options. Pin 0 would use USB.
Result can be -1 when not configured. -2 when not valid UART. 0 no data ready. 1 valid data ready

```
result = lcd.Available( 11 );//see if was received

if ( result == 1 )

    result = lcd.RxUART( 11 );//get character
```

WSTACK

The WSTACK command is used to see what widget touch commands have happened. It is a 32 deep stack to prevent missing any screen presses when the user is processing other things. It can be used with interrupts or not.

Three global variables are updated each time this command is called.

currentWidget gets the ID of the event
currentInfo gets the info about the event. 0=nothing to report.
currentData gets the Data related to the event.

By monitoring the currentInfo and currentWidget the user can track touch events without missing any.

Info and Data vary between widgets. See explanation below.

```
volatile unsigned int currentWidget;
volatile unsigned int currentInfo;
volatile unsigned int currentData;
```

Command:

lcd.wstack(cmd);

Parameters:

cmd is the command requested of the wstack.

0 = read data from FIFO	(FIFO)
1 = read data from LIFO	(LIFO)
2 = clear all data in the FIFO	(CLEAR)
3 = read data nondestuctive from FIFO	(PEEK)

Example:

```
cls black                  'clears the screen to black
string 1 "testing"         'the word will appear on the button
fontw 0 sans24             'widget font needs to be set before the theme
theme 2 5 20 2 2 2 4 4 0 0 0    'colors are to distinguish different parts of widget
button 1 165 86 150 100 1 0 0 2 1 'simple button
WSTACK 0
```

You can keep reading the stack to see all the presses you made to any of the current widgets. When you read 0 0 0 the FIFO is empty. The stack is currently 32 deep.

You can also look at the last FIFO entry with LIFO option.

FIFO and LIFO Methods

FIFO stands for *first-in, first-out*, meaning the data is pushed into the bottom of the stack and removed from the top of the stack. Therefore oldest data is returned first.

LIFO stands for *last-in, first-out*, meaning the data is still pushed into the bottom of the stack but removed rom the bottom of the stack. Therefore newest data is returned first.

Typically the user would turn ON WQUIET to suppress any touch events if using the widget stack. See WQUIET below.

Note: Default startup file has WQUIET enabled.

INFO and DATA vary from widget to widget

For the button, Touchzone:

4 = widget pressed
2 = widget canceled
1 = widget released
DATA = State

For the checkbox:

4 = widget checked
1 = widget unchecked
DATA = State

For the radio button:

4 = widget pressed
DATA = State

For the slider:

2 = widget decremented
1 = widget incremented
DATA = Value

For the dial:

2 = widget counter clockwise
1 = widget clockwise
DATA = Value

CALIBRATE

The CALIBRATE command is used to calibrate the touchscreen. It is typically not used but once or the screen goes out of calibration. Once executed the user presses three spots on the display as indicated. The display calculates and stores calibration data on the flash drive.

Do not put this command in a setup block of your sketch. Calibration blocks flash and USB access and makes upgrading sketches impossible.

Command:

lcd.calibrate( );

Parameters:

None

```
lcd.calibrate( );
```

SHADOW

The SHADOW command is used to display a bounding box where any touchzones are setup. This indicates where the active areas are. During normal operation the shadows would be turned off. The color, linetype and line width can be specified.

Command:

lcd.shadow(mode, Scolor, Slinetype, Slinethickness);

Parameters:

[mode] can enable or disable the shadow. Default is off. 0=OFF, 1=ON.

[Scolor] used to define the color of the shadow using existing colorID values. Default is 4 (red)

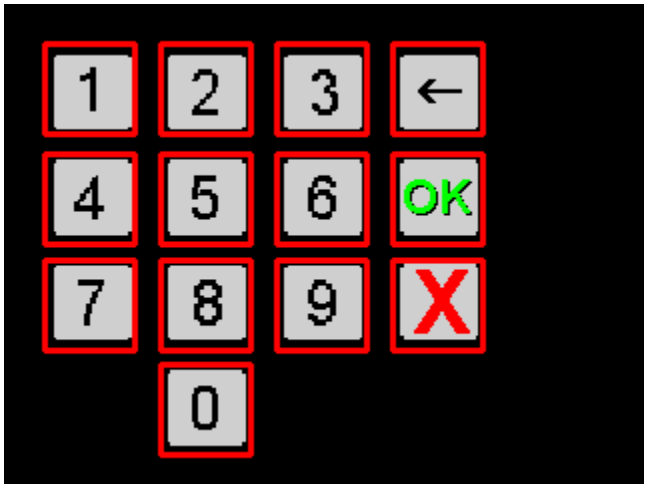
[Slinetype] used to define the line type used for the shadow. Default 0 (Continuous line)

[Slinethickness] uused to define the thickness of the shadow. Default 2 (Wide)

Values over 3 use the definitions in table below.

Example:

```
lcd.cls( WHITE ); //clears the screen to white
lcd.picture( 20, 20, "keypad.gif" );
lcd.shadow( 1, 4, 0, 2 );//shadow ON, RED, CONT, WIDE
//touchZone [ID][x][y][width][height][options]
lcd.touchZone( 1,  20, 20, 46, 46, 1 );
lcd.touchZone( 2,  78, 20, 46, 46, 1 );
lcd.touchZone( 3, 136, 20, 46, 46, 1 );
lcd.touchZone( 4, 194, 20, 46, 46, 1 );
```



WVALUE

The WVALUE command is used to get and set values from any active widget. When setting a value of a widget it will be redrawn. When getting a value from widgets, different widget types return different meaning

When getting value:

Button return 1 if pressed, 0 otherwise.
Slider returns Position.
Digital Meter returns Value
ProgressBar returns position.
Gauge returns position.
Dial returns Value.
StaticText cannot return string data.
Checkbox returns 1 if checked, 0 otherwise.
RadioButton returns 1 if checked, 0 otherwise.

Do not put this in a setup block of your sketch. Calibration blocks flash and USB access and makes upgrading sketches impossible. When used always make it conditional.

Command:

```
lcd.wvalue( ID, value );        //updates widget ID with value

lcd.wvalue( ID );               //gets the widget value
```

Parameters:

ID is the widget ID to reference by.
Value is the value to send to the widget.

Updating an Analog meter

```
int value = analogRead(0);      //read analog pin voltage
lcd.wvalue( 1, value );         //update widget with value

int value=lcd.wvalue(1);        //read widget 1 value
```

WSTATE

The WSTATE command is used to get and set state from any active widget. When getting state from widgets, different widget types return different meaning

Command:

lcd.wstate(ID, state); //updates widget ID with state

lcd.wstate(ID); //gets the widget state

Parameters:

[ID] must be the same as the **ID** of the widget you want to change.

[Options] are: **0 = delete, 1 = enable, 2 = disable, 3 = redraw**

0 = Delete the widget. This option redraws the widget to the common background color and then unlinks the widget ID from further processing. Once a widget is deleted its state can no longer be modified.

1 = Enable the widget. This option will enable a previously created widget that has been disabled. The Widget is redrawn with the enable colors in the Theme.

2 = Disable the widget. This option will disable a previously created widget that is enabled. The Widget is redrawn with the disable colors in the Theme.

3 = Redraw the widget. This option will redraw a previously created widget. This is useful if the widget has been overwritten by other text or if the string has been modified and needs to be redrawn on the widget.

Values over 3 use the definitions in table below.

Example:

```
lcd.cls( WHITE ); //clears the screen to white
lcd.string( 1, "testing"); //the word will appear at the bottom of the widget
lcd.fontw( 0, “sans24”); //widget font needs to be set before the theme
lcd.theme( 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 0 );//
lcd.staticText( 1, 10, 25, 220, 25, 5, 0, 1 );     'draws a static widget
delay(5000);//wait 5 seconds
```

You could use this

```
lcd.string( 1, “tested”);    //change the string to “tested”
lcd.wstate( 1, 3 );          //changes the string to “tested”
```

Note: Gauge and progress bar use the same wstate definitions.

Static Text	Hide	Draw	Update	~	~	~	~	~	~	~	Frame	Center	Right	Disabled	~	
Progress Bar	Hide	Draw	Redraw	~	~	~	~	~	~	~	~	~	Vertical	Disabled	~	
Slider	Hide	Draw	Draw Focus	Draw Thumb	~	~	~	~	~	~	~	Scroll Bar	Vertical	Disabled	~	
Rotary Dial	Hide	Draw	~	~	~	~	~	~	~	~	~	CCW	CW	Disabled	~	
Analog Meter	Hide	Draw	Redraw	Update	~	~	~	~	~	~	~	Accuracy	Ring	Disabled	~	
Digital Meter	Hide	Draw	Update Text	~	~	~	~	~	~	~	Frame	Center	Right	Disabled	~	
Radio Button	Hide	Draw	Draw Focus	Draw Check	~	~	~	~	~	~	~	First	Checked	Disabled	~	
Group Box	Hide	Draw	~	~	~	~	~	~	~	~	~	Center	Right	Disabled	~	
Check Box	Hide	Draw	Draw Focus	Draw Check	~	~	~	~	~	~	~	~	Checked	Disabled	~	
Touch Zone	Hide	Draw	Draw Focus	~	~	~	~	~	~	~	~	~	Pressed	Disabled	~	
Button	Hide	Draw	Draw Focus	~	~	~	No Panel	Two Tone	Top	Bot	Left	Right	Toggle	Pressed	Disabled	~
Data Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Defaults	**Hide**	**Draw**	**Draw Focus**	**Redraw**											**Disabled**	

Drawing Shapes

Arc (Pie)

This command allows you to draw an arc which is a segment of a circle. To better understand this command, imagine a circle that will not appear on your arLCD screen. Like the picture below, think of the light blue circle to be invisible and the arc, shown black, is what will appear on the screen. The arc can also be filled which gives a pie command.

Command:

lcd.arc(radius, start, end, fill);

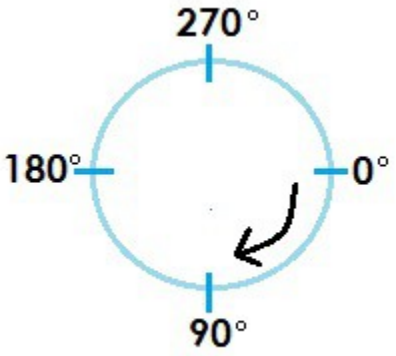
Parameters:

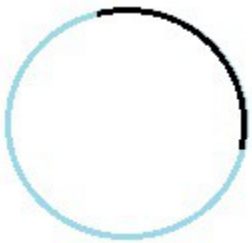
radius is the length in pixels from the center of your imaginary circle to its perimeter or edge

start is the angle in degrees the arc will begin drawing **clockwise**

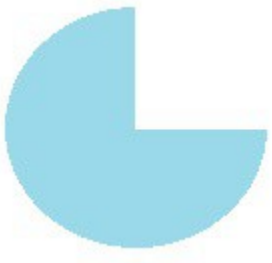
end is the angle in degrees the arc will stop (angles range from 0-359)

Fill by default is set to 0. **0 = no fill, 1 = fill**





ARC



PIE

Box (Rectangle)

This command draws a box starting at the current xy position.

Command:

lcd.box(width, height, fill);

lcd.rect(x, y, width, height, fill);

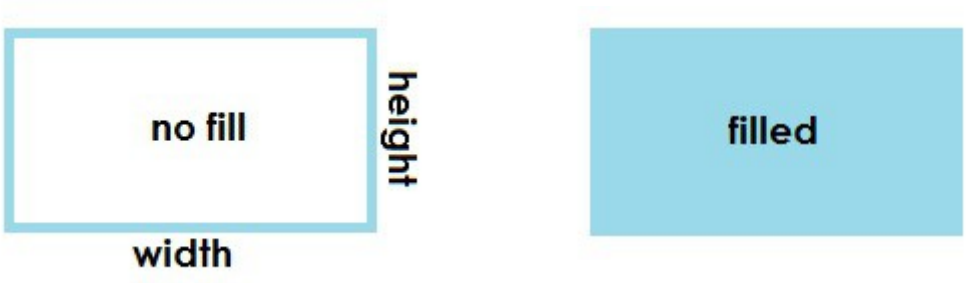
Parameters:

x and **y** are the points on the screen the box will be drawn from if used.

Width is the horizontal length of the box in pixels

Height is the vertical length of the box in pixels

Fill by default is set to 0. **0 = no fill, 1 = fill**



Circle

This command draws a circle at the current xy position or the specified position.

Command:

lcd.circle(radius, fill); 'circle from current xy with radius and fill options

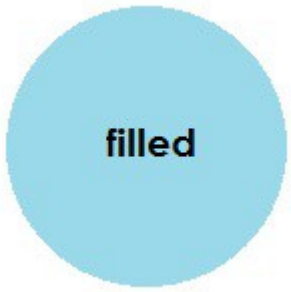
lcd.circle(x, y, radius, fill); 'circle from specified xy with radius and fill options

Parameters:

x and **y** are the points on the screen the box will be drawn from if used.

Radius is the length in pixels from the center of the circle to its perimeter or edge

Fill by default is set to 0. **0 = no fill, 1 = fill**



Arc (Pie)

This command allows you to draw an arc which is a segment of a circle. To better understand this command, imagine a circle that will not appear on your arLCD screen. Like the picture below, think of the light blue circle to be invisible and the arc, shown black, is what will appear on the screen. The arc can also be filled which gives a pie command.

Command:

lcd.arc(radius, start, end, fill);

Parameters:

radius is the length in pixels from the center of your imaginary circle to its perimeter or edge

start is the angle in degrees the arc will begin drawing **clockwise**

end is the angle in degrees the arc will stop (angles range from 0-359)

Fill by default is set to 0. **0 = no fill, 1 = fill**

Line

The line command draws a straight line on the screen.

Command:

lcd.line(x, y);

Parameters:

x and **y** are points on the screen the line will be drawn to from the current position.

Line Type

This command allows you to change the line style you want to draw in.

Command:

lcd.lineType(option);

Parameters:

options is the number value set to the different type of lines.

0=solid line, _____

1=dotted line, ...

2=dashed line, - - - - - - - - - - - -

Line Width

This command allows you to change the thickness of the line you are drawing.

Command:

lcd.lineWidth(width);

Parameters:

width is the thickness of the line. 1 = thin, 3 = thick

Plot (Point)

Plotting a point means to designate a location on the screen where you would like to start drawing. A plot coordinate is one pixel size. The arLCD has a resolution of 320x240. So think of the x coordinate as the horizontal axis of the screen (0-319) and the y coordinate as the vertical axis of the screen (0-239). The origin of the (0,0) coordinate is at the top left the screen.
The current color (fcolor) is used.

Command:

```
lcd.plot( );              'plot to current x, y

lcd.plot( x, y );         'plot to specified x, y

lcd.point( );             'plot to current x, y

lcd.point( x, y );        'plot to specified x, y
```

Parameters:

x and **y** are the coordinates to which a pixel is placed.

Picture

The picture command displays a picture on the screen a the specified location. The picture can be GIF, BMP or JPEG. GIF is the smallest and fastest format. The picture should be resized to fit the screen properly. Sending huge files that don't match the screen size can crash the GPU.

Command:

lcd.picture(x, y, picture); 'image.gif, image.jpg, image.bmp

lcd.picture(x, y, options, picture);

Parameters:

x and **y** is point on the screen the picture will start being drawn.
Options are 0=none, 1=centered, 2=downscaled, 3=both.

Filesystem

The filesystem on the GPU can be accessed by the ATMEL processor. This can be used for nonvolatile items and larger storage as needed by the system. Since the filesystem is shared with the GPU the overhead is reduced and performance is much better.

Note: Only one file can be open at a time. Any files written and read by the ATMEL processor may not be accessible by the USB flash drive until a reset.

Caution: Since this filesystem is shared with the GPU, removing directories, corrupting files or deleting files used by the GPU could make the display inoperable. Do NOT use directories used by the existing GPU.

```
FSopen      'used to open a file for read or write by the ATMEL processor.
FSread      'used to read bytes from an existing open file.
FSwrite     'used to write bytes to an open file.
FSclose     'used to close the open file.
FSrewind    'used to set the pointer back to the front of the open file.
Fsseek      'used to set the pointer to any place in an open file.
Fserror     'used to return error number from previous file commands.
Fstell      'used to get the location of the file pointer of an open file.
Fsattrib    'used to set the attributes of the open file.
GetFSattrib 'used to get the attributes of the open file.
Fseof       'used to get EOF (End Of File) status.
Fschdir     'used to change directories.
Fsmkdir     'used to make a directory.
Fscopy      'used to copy a file.
Fsrename    'used to rename a file.
Fsremove    'used to remove a file.
```

FSopen

This command is required before an other file access can be done. A file can be opened in one of 3 modes. READ, WRITE or APPEND. The interface to the GPU uses ASCII commands and content of the files read or written can not contain control characters. If you require file content with control characters, use hex mode. Hex mode sends 2 bytes per stored byte, therefore it is slightly slower.

Note: Control characters are characters 0-01Fh. Carriage return is 0Dh.

Command:

lcd.FSopen(**filename**, **mode**);

Parameters:

filename is an 8.3 compatible file name with extension to open
mode is the requested use of the file. READ, WRITE or APPEND.
Returns pointer to open file. 0=OK, -1 means open failed.

Read will open with existing file with pointer to first character. Write will open with new file with pointer to first character. If file exists it is opened with pointer to end of file. Append will open existing file with pointer at end of file.

Should your file content include control characters, add the hex option 'H' to the mode option. R, W, A, RH, WH, AH are the valid options.

```
int result;//setup the results

result = lcd.FSopen( "junkdata.bin", "w" );
if ( result == 0 )
        lcd.print( "File Opened OK" );
else
        lcd.print( "File Open Failed" );
```

FSread

This command will read data from an existing file into the specified buffer. The file must be opened before it can be read. The interface to the GPU uses ASCII commands and content of the files read or written can not contain control characters. If you require file content with control characters, use hex mode.

Note: Control characters are characters 0-01Fh. Carriage return is 0Dh.

Command:

lcd.FSread(**buffer**, **count**);

Parameters:

buffer is the pointer to the buffer where can will be placed.

count is the number of bytes to be read.

Returns number of bytes read. (this may not be the same as requested bytes)

Note: The number of bytes read is limited to 62 bytes due to system restrictions and timeout concerns. Any count over 62 bytes is broken up into 62 byte blocks and assembled into the buffer when complete.

```
//file read
char buffer[ 1024 ];//make room for the buffer
int result, bcount;//setup the results
lcd.xy( 20, 0);
lcd.print( "Open File: " );
result = lcd.FSopen( "junkdata.bin", "r" );
if ( result == 0 )
        lcd.print( "OK" );
else
        lcd.print( "Failed" );
lcd.xy( 200, 0);
bcount = 75;//set the count
result = lcd.FSread( buffer, bcount );
lcd.print( buffer );//question mark means non ascii string
result = lcd.FSclose();
if ( result == 0 )
        lcd.print( "OK" );
```

FSwrite

This command will write data to a file from the specified buffer. The file must be opened in write or append mode before it can be written. The interface to the GPU uses ASCII commands and content of the files read or written can not contain control characters. If you require file content with control characters, use hex mode.

Note: Control characters are characters 0-01Fh. Carriage return is 0Dh.

Command:

lcd.FSwrite(**buffer**, **count**);

Parameters:

buffer is the pointer to the buffer where data is.

count is the number of bytes to be written.

Returns number of bytes written.

Note: The number of bytes written is limited to 62 bytes due to system restrictions and timeout concerns. Any count over 62 bytes is broken up into 62 byte blocks.

```
//file write
char buffer[ 1024 ];//make room for the buffer
int result, bcount, i;//setup the results
lcd.xy( 20, 0);
lcd.print( "Open File: " );
result = lcd.FSopen( "junkdata.bin", "w" );
bcount = 75;//set the count
//fill the buffer with data for writes
for ( i = 0; i < bcount; i++ ) {
        buffer[ i ] = i+1;//fill with valid data “Never use data of 0”
        buffer[ i ] = buffer[ i ] & 0xFF;//filter
        if ( buffer[ i ] == 0x22 ) buffer[ i ] = 0x3F;//make it a ?
        buffer[ i+1 ] = 0;//fill with end
}
lcd.xy( 20, 80 );
result = lcd.FSwrite( buffer, bcount );
lcd.print( buffer );//question mark means non ascii string
```

FSclose

This command will close an open file. A file must be closed for the writes to the directory to take place and directories updated.

Note: To access any file changes from the USB the device must be reset to get the PC to rescan the flash drive.

Command:

lcd.FSclose( );

Parameters:

Returns result of close. 0=OK, -1=failed.

```
result = lcd.FSopen( "junkdata.bin", "w" );
result = lcd.FSwrite( buffer, 123 );
result = lcd.FSclose();
```

FSseek

This command will move the position pointer in a file to any location. The input number is defined as long. The position can relative to 3 options.

0=SEEK_SET From beginning of file.

1=SEEK_CUR From current location.

2=SEEK_END From the end of the file.

Command:

lcd.FSseek(**offset, whence**);

Parameters:

offset is the offset for the pointer.

whence is the mode to use to offset the position of the pointer.

Returns result of close. 0=OK, -1=failed.

```
result = lcd.FSopen( "junkdata.bin", "w" );
result = lcd.FSwrite( buffer, 123 );
result = lcd.FSrewind( );
result = lcd.FSseek( 15, SEEK_SET);          'move to 15 bytes from begin
result = lcd.FSseek( 2, SEEK_CUR);           'move forward 2 bytes
result = lcd.FSclose();
result = lcd.FSwrite( buffer, 123 );
```

FSerror

This command will get the error number from the file system after any operation that failed.

Command:

lcd.FSerror( );

Parameters:

Returns

```
result = lcd.FSopen( "junkdata.bin", "w" );
result = lcd.FSwrite( buffer, 123 );
result = lcd.FSerror();
result = lcd.FSclose();
result = lcd.FSwrite( buffer, 123 );
```

List of all possible error codes.

```
0   CE_GOOD                     // No error
1   CE_ERASE_FAIL               // An erase failed
2   CE_NOT_PRESENT              // No device was present
3   CE_NOT_FORMATTED            // The disk is of an unsupported format
4   CE_BAD_PARTITION            // The boot record is bad
5   CE_UNSUPPORTED_FS           // The file system type is unsupported
6   CE_INIT_ERROR               // An initialization error has occured
7   CE_NOT_INIT                 // An operation was performed on an
uninitialized device
8   CE_BAD_SECTOR_READ          // A bad read of a sector occured
9   CE_WRITE_ERROR              // Could not write to a sector
10  CE_INVALID_CLUSTER          // Invalid cluster value > maxcls
11  CE_FILE_NOT_FOUND           // Could not find the file on the device
12  CE_DIR_NOT_FOUND            // Could not find the directory
13  CE_BAD_FILE                 // File is corrupted
14  CE_DONE                     // No more files in this directory
15  CE_COULD_NOT_GET_CLUSTER    // Could not load/allocate next cluster in
file
16  CE_FILENAME_2_LONG          // A specified file name is too long to use
17  CE_FILENAME_EXISTS          // A specified filename already exists on the device
18  CE_INVALID_FILENAME         // Invalid file name
19  CE_DELETE_DIR               // The user tried to delete a directory with FSremove
```

```
20      CE_DIR_FULL                     // All root dir entry are taken
21      CE_DISK_FULL                    // All clusters in partition are taken
22      CE_DIR_NOT_EMPTY                // This directory is not empty yet, remove
files before deleting
23      CE_NONSUPPORTED_SIZE            // The disk is too big to format as FAT16
24      CE_WRITE_PROTECTED              // Card is write protected
25      CE_FILENOTOPENED                // File not opened for the write
26      CE_SEEK_ERROR           // File location could not be changed successfully
27      CE_BADCACHEREAD                 // Bad cache read
28      CE_CARDFAT32                    // FAT 32 - card not supported
29      CE_READONLY                     // The file is read-only
30      CE_WRITEONLY                    // The file is write-only
31      CE_INVALID_ARGUMENT             // Invalid argument
32      CE_TOO_MANY_FILES_OPEN      // Too many files are already open
33      CE_UNSUPPORTED_SECTOR_SIZE      // Unsupported sector size
```

FStell

This command will return the location of the current position pointer. The return value is a long.

Command:

lcd.FStell( );

Parameters:

Returns number of location of the current position pointer

FSattrib

This command is used to set the attributes for the currently open file.
1=Read Only, 2=Hidden, 4=System, 0x20=Archive. Bits may be combined. The attribute byte is updated on the flash drive when the file is closed.

Note: A read only file can still be deleted. The user software should be written such that if the file is marked read only, it should not attempt to delete or modify it.

Command:

lcd.FSattrib( );

Parameters:

attributes new attributes to apply to curently open file
Returns result. 0=OK, -1=failed.

FS ATTRIBUTES

Description: The read-only attribute. A file with this attribute should not be written to. Note that this attribute will not actually prevent a write to the file; that functionality is operating-system dependant. The user should take care not to write to or delete a read-only file.

ATTR_READ_ONLY 0x01

Description: The hidden attribute. A file with this attribute may be hidden from the user, depending on the implementation of the operating system.

ATTR_HIDDEN 0x02

Description: The system attribute. A file with this attribute is used by the operating system, and should not be modified.

ATTR_SYSTEM 0x04

Description: The volume attribute. If the first directory entry in the root directory has the volume attribute set, the device will use the name in that directory entry as the volume name.

ATTR_VOLUME 0x08

Description: The long-name attributes. If a directory entry is used in a long-file name implementation, it will have all four lower bits set. This indicates that any software that does not support long file names should ignore that entry.

ATTR_LONG_NAME 0x0F

Description: The directory attribute. If a directory entry has this attribute set, the file it points to is a directory-type file, and will contain directory entries that point to additional directories or files.

ATTR_DIRECTORY 0x10

Description: The archive attribute. This attribute will indicate to some archiving programs that the file with this attribute needs to be backed up. Most operating systems create files with the archive attribute set.

ATTR_ARCHIVE 0x20

GetFSattrib

This command will get the current attributes of an open file.

Command:

lcd.GetFSattrib( );

Parameters:

Returns existing attributes.

```
//get attribute          * 1=ReadOnly, 2=Hidden, 4=System, 0x20=Archive
  lcd.xy( 20, 180 );
  result = lcd.GetFSattrib( );
  lcd.print( "Get attribute:" );
  lcd.print( result );//should be 33
```

FSeof

This command will return the status of the End Of File flag.

0=Not at end of file, else EOF

Command:

lcd.FSeof( );

Parameters:

Returns EOF status. 0=Not at EOF.

```
///eof command
  lcd.xy( 200, 100 );
  result = lcd.FSeof( );
  lcd.print( "EOF:" );
  if ( result == 0 )
    lcd.print( "NO" );
  else
    lcd.print( "YES" );
```

FSchdir

This command is for changing the current directory.

Note: The “\” character is an escape character and therefore discarded when typed in alone. To change to root for instance is done with “\\\\” instead of “\\”.

Command:

lcd.FSchdir(**directory**);

Parameters:

directory is the requested directory to change to.
Returns 0=OK, -1 error

```
result = lcd.FSchdir( "\\\\" );//change to root
lcd.xy( 20, 20 );
result = lcd.FSchdir( "Files" );
if ( result == 0 ) {
  lcd.print( "Directory change OK" );
  result = lcd.FSremove( "junkdata.bin" );//remove it if its there
}else{
  lcd.print( "Directory change Failed" );
  result = lcd.FSmkdir( "Files" );
  result = lcd.FSchdir( "Files" );
}
```

FSmkdir

This command is for making a new directory.

Note: The "\" character is an escape character and therefore discarded when typed in alone. To change to root for instance is done with "\\\\" instead of "\\".

Command:

lcd.FSmkdir(**directory**);

Parameters:

directory is the requested directory to make.
Returns 0=OK, -1 error

```
result = lcd.FSchdir( "\\\\" );//change to root
lcd.xy( 20, 20 );
result = lcd.FSchdir( "Files" );
if ( result == 0 ) {
  lcd.print( "Directory change OK" );
  result = lcd.FSremove( "junkdata.bin" );//remove it if its there
}else{
  lcd.print( "Directory change Failed" );
  result = lcd.FSmkdir( "Files" );
  result = lcd.FSchdir( "Files" );
}
```

FScopy

This command will copy one existing file to a new file with different name.

Command:

lcd.FScopy(**source**, **destination**);

Parameters:

source is the name of the existing file to copy from.
destination is the name of the copy you want to generate.
Returns 0=copy OK, -1=copy failed.

```
//file copy
lcd.xy( 20, 0);
lcd.print( "Copy File: " );
result=lcd.FScopy( “source.bin”, “destinat.bin” );
if ( result == 0 )
        lcd.print( "OK" );
else
        lcd.print( "Failed" );
```

FSrename

This command will rename an existing file.

Command:

lcd.FSrename(**oldname, newname**);

Parameters:

oldname is the name of the original file.
newname is the name of the file after it is renamed.
Returns 0=OK, -1=Failed

```
//file rename
lcd.xy( 20, 0);
lcd.print( "Rename File: " );
result=lcd.FSrename( “source.bin”, “destinat.bin” );
if ( result == 0 )
        lcd.print( "OK" );
else
        lcd.print( "Failed" );
```

FSremove

This command will remove (delete) an existing file. The file must be in the current directory.

Command:

lcd.FSremove(**filename**);

Parameters:

filename is the name of the file to be removed.
Returns 0=OK, -1=Failed

```
//file remove
lcd.xy( 20, 0);
lcd.print( "Remove File: " );
result=lcd.FSremove( “source.bin” );
if ( result == 0 )
        lcd.print( "OK" );
else{
        lcd.print( "Failed" );
        result = lcd.FSerror();//get the reason for fail
        lcd.print( result );//print it
}
```

Fonts

To be able to print text on to the display screen a reference must be made to a FONT. There are internal fonts and external fonts. Internal are very fast but only come in 2 sizes (1=Small and 2=Medium)

Fonts are set with the Font command.
Internal Font:
lcd.font(1); //internal small
lcd.font(2); //internal medium (also default which is 0)

External Font: (in directory \EZSYS\FONTS)
lcd.font(“Serif24”); //external font Serif24 point font

Widgets that have text require a font to be defined prior to being created. Since there can be a lot of widget there can be 16 widget fonts defined. They have an ID included in the definition.
Widget Fonts use the FONTW command.
Internal Font:
lcd.fontw(1, 1); //widget font 1 is set to internal small font
lcd.fontw(7, 2); //widget font 7 is set to internal medium font

External Font: (in directory \EZSYS\FONTS)
lcd.fontw(2, “Serif24”); //widget font 2 is set to external font Serif24 point font

Font orientation is set with fonto. Setting it to 1 would rotate the text.
lcd.fonto(0);
We recommend to leave it set to 0.

Theme

Themes are color schemes of your widget. Without a good theme, your widgets will not appear as what they are supposed to be on the screen. Manipulating the colors in the theme command will give your button more dimension to make it seem more realistic. To better understand this, download the Theme Generator BETA and play around with the different options.

Note: Not all of the parameters in the theme command effect every widget the same way. Some widgets use only a few theme parameters while others utilize all of them.

Command:

lcd.theme(ID, DE, LE, TC0 , TC1, TCD, C0, C1, CD, BKC, FontW);

Parameters:

ID this is the theme ID number. This needs to be a unique number from all other Theme IDs.

DE – Dark Emboss Color This is the part of you widget where you would imagine the shadow will fall if there was a light hitting it.

LE – Light Emboss Color This is where you can imagine where the light would hit your widget

TC0 – Text Color 0 Color of the text

TC1 – Text Color 1 Color of the text

TCD – Text Color Disabled Color of the text when the widget is disabled

C0 – Color 0 The color for a general part of the widget

C1 – Color1 The color for a general part of the widget

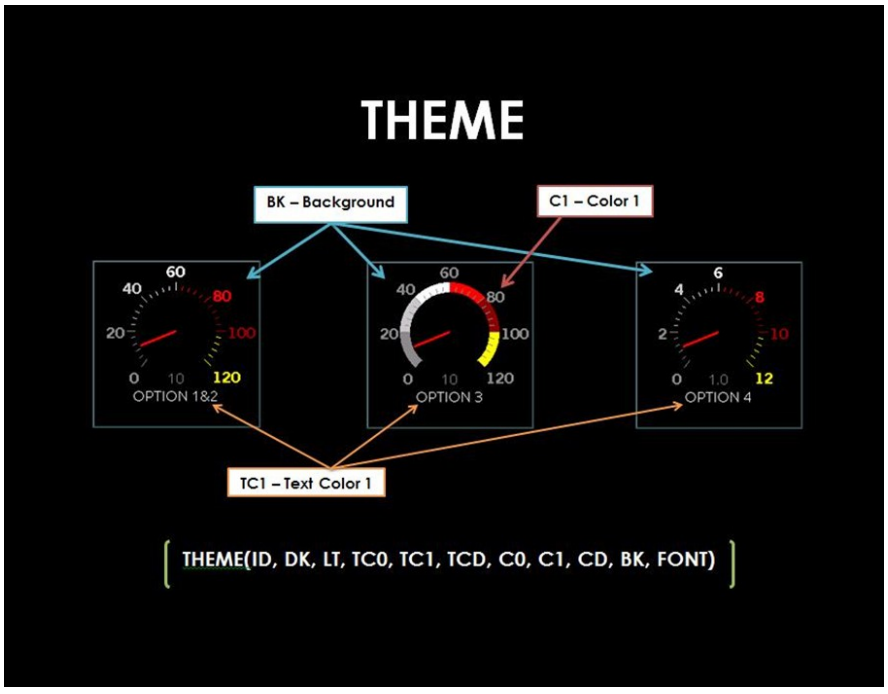
CD – Color Disabled The color of the widget when it is disabled

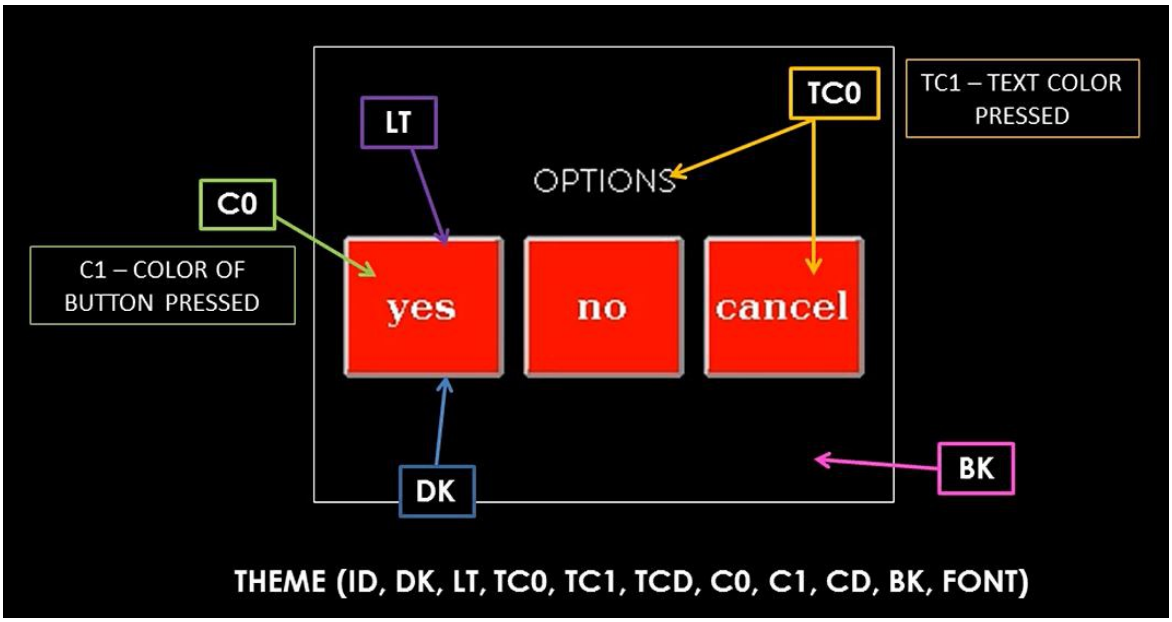
BKC – Background Color

FontW – Font ID. The font you want to use for the widget. Font IDs must be established before the theme command.

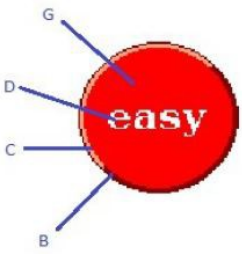
WIDGET	ID A	Emboss Dark Color B	Emboss Light Color C	Text Color 0 D	Text Color 1 E	Text Color Disabled F	Color 0 G	Color 1 H	Color Disabled I	Common Background Color J	K=Font
AMETER		x	x	x	Label	x	Back	Top Text Option 3	x	x	
BUTTON		bot	top	up	down	disabled	up	down	disabled	x	
CHECKBOX		top	bot	check/text	x	disabled	box	x	box	back	
RADIO		top	bot	check/text	x	disabled	circle	x	circle	back	
DIAL		bot/button	top	x	x	x	dial	x	dial	x	
CHOICE		bot	top	x	text	x	button	x	x		
SLIDER		bot	top	x	x	x	slider	handle	slider	x	
PROGRESS		top	bot	text	x	x	Back	foreground	x	x	
STATIC		x	x	x	text	text	frame	x	x	back	
DMETER		x	x	text	x	x	x	frame	x	back	
GBOX		x	frame	text	x	disabled	x	x	x	back/text	

Table 1: Widget Parameters

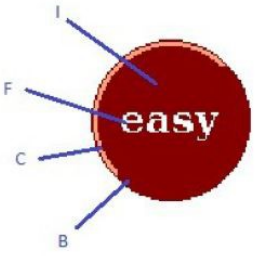




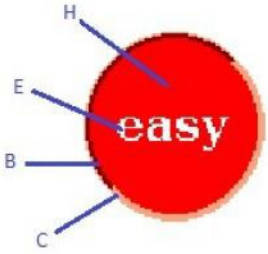
Choice



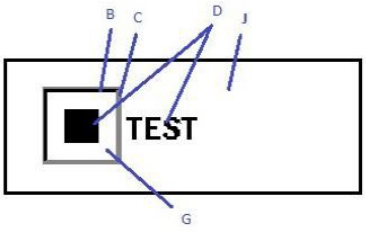
Unpressed



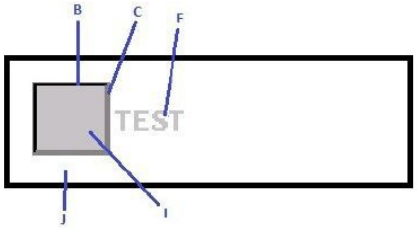
Disabled



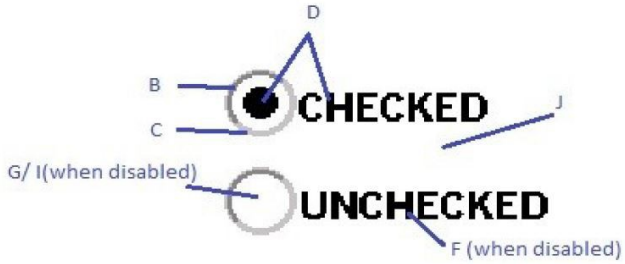
Pressed



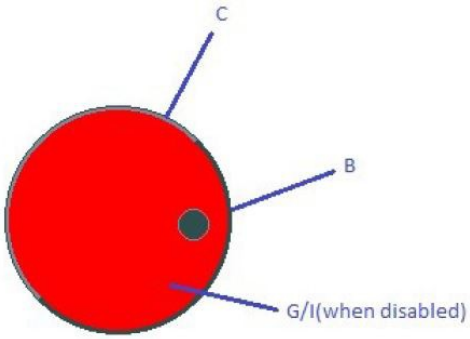
Checked



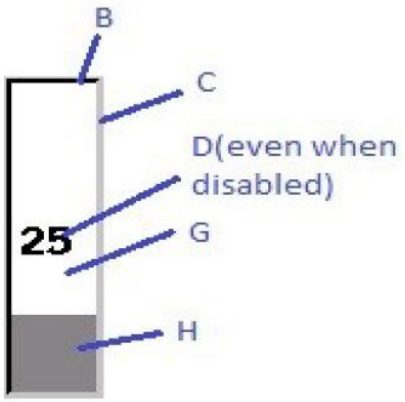
Unchecked



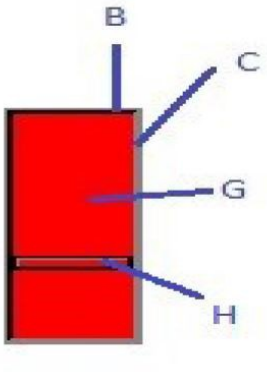
Radio Buttons



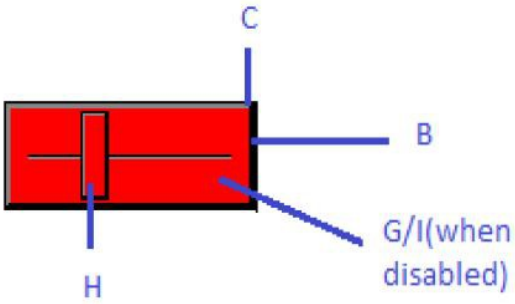
Dial



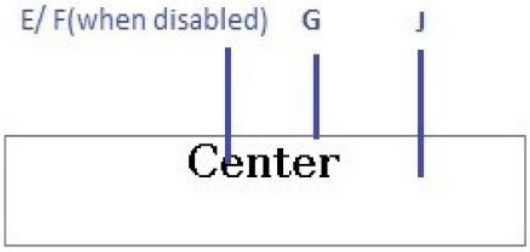
Progress Bar and Gauge



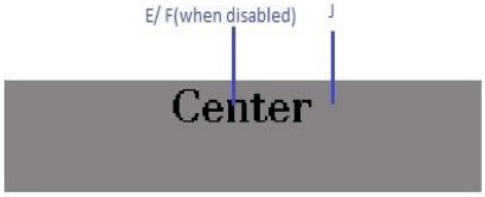
Scrollbar



Slider



Static Text (Active)



Static Text Disabled

Digital Meter

Widgets

Widgets are small components of an interface. On your computer, widgets are the calendar, clock and buttons on your display screen. Widgets have an ID to differentiate each of them. This is called the widget ID and can be from 1-99.

Only one widget can use an ID at a time. Clear screen will remove all widgets and clear all widget usage.

You can create many different kind of widgets as long as each ID is **unique** regardless of the widget type.
The ID of the widget is used to get responses about the widget or to change the widget after it is created.

Analog Meter

Analog meters are gauges, which you often see on weight scales or speedometers.

Command:

lcd.analogMeter(ID, x, y, width, height, options, initial, min, max, theme, stringID, type)

Parameters:

ID is an ID number, **1-99,** specified to a particular widget.

x & y values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width & height values designate the width and height of the widget in pixels.

options designates the DRAW options of the analog meter.

> **Option choices**: 1=draw, 2=disabled, 3=ring, 4=accuracy
> **Draw** prints the widget to screen.
> **Disabled** draws a widget that cannot be affected or changed by touch.
> **Ring** draws the widget with an arc'ed bar around numbers.
> **Accuracy** allows you to display numbers with a decimal point for more exact numbers. Uses 10 x times the actual value by the user because of integer math.

value designates the initial value setting of the needle on the meter.

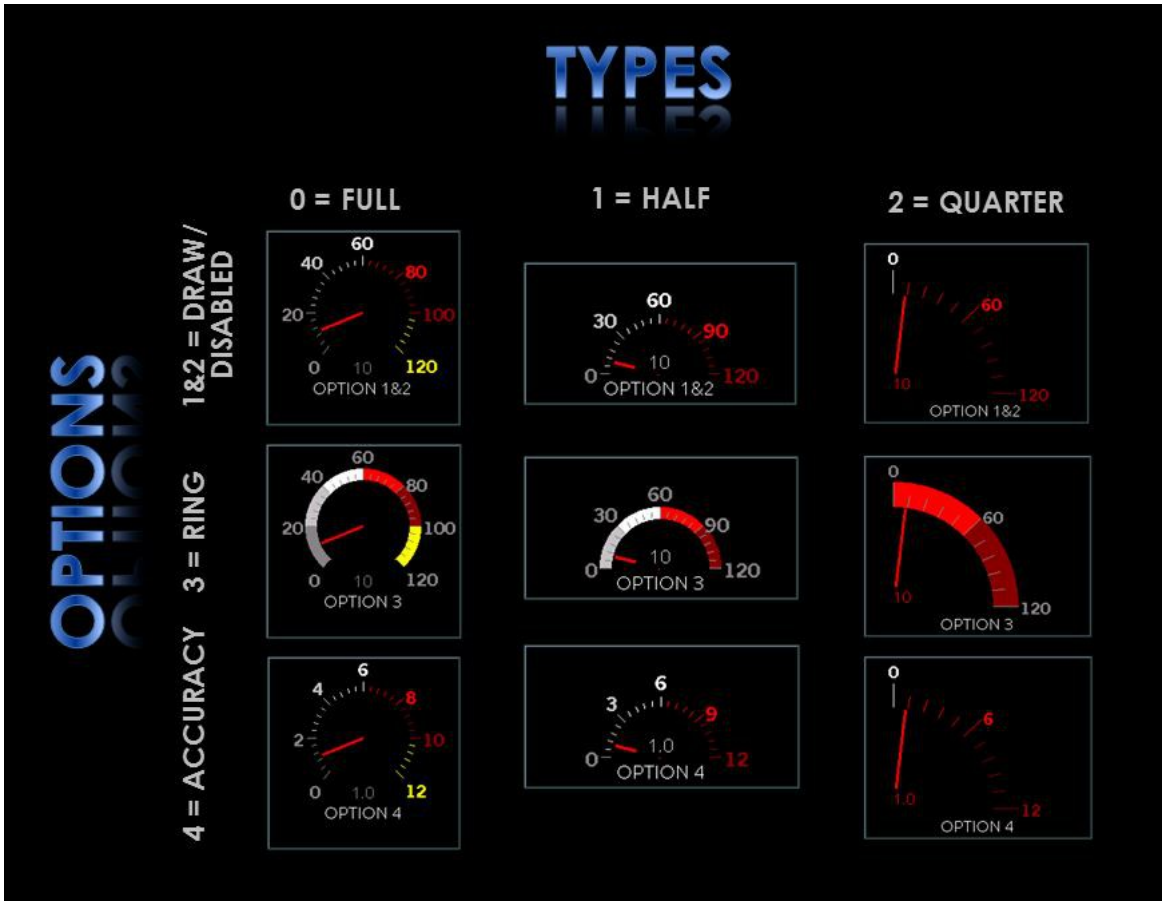
min designates the minimum value on the meter scale.

max designates the maximum value on the meter scale.

theme is the ID of the theme you want to use.

stringID designates the ID number of the text string that you'd like displayed below the meter.

type is the meter type/style you want to use. By default, the meter type is set to full. For the half size, you will need to adjust **width** to make the meter proportional. **Type choices**: **0=full, 1=half, 2=quarter.**


TYPES
OPTIONS
0 = FULL
1 = HALF
2 = QUARTER
1&2 = DRAW / DISABLED
3 = RING
4 = ACCURACY
OPTION 1&2
OPTION 3
OPTION 4

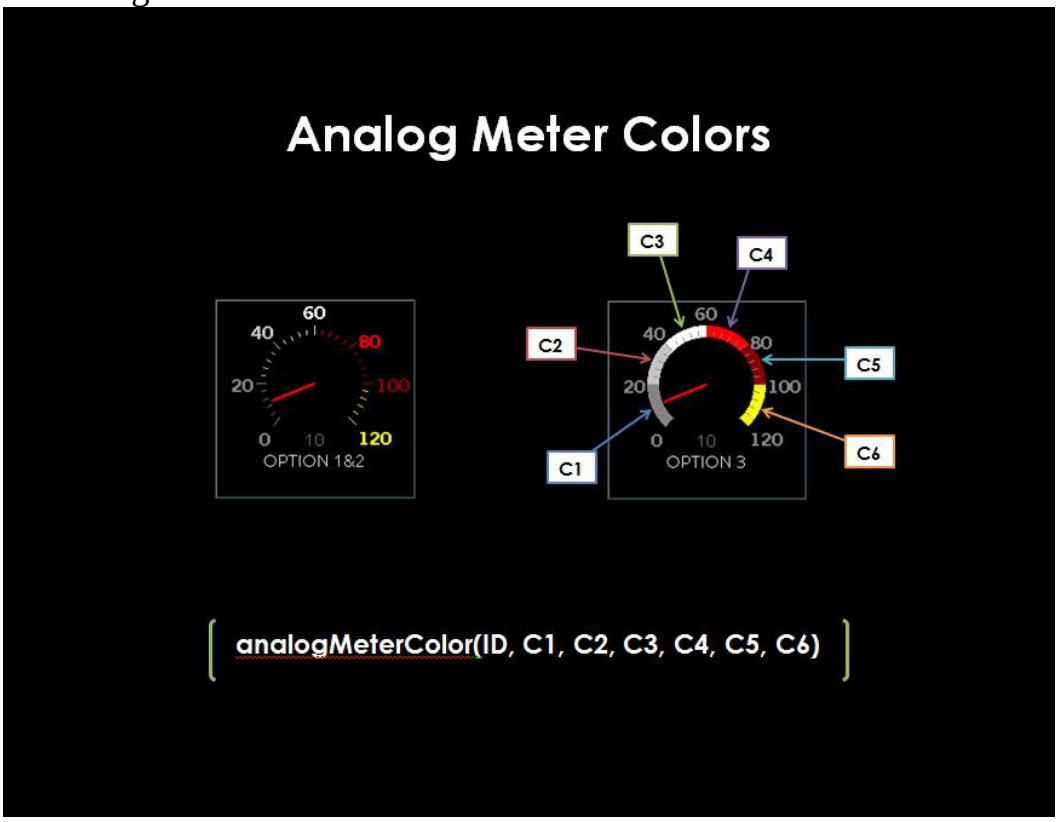
Analog Meter Color

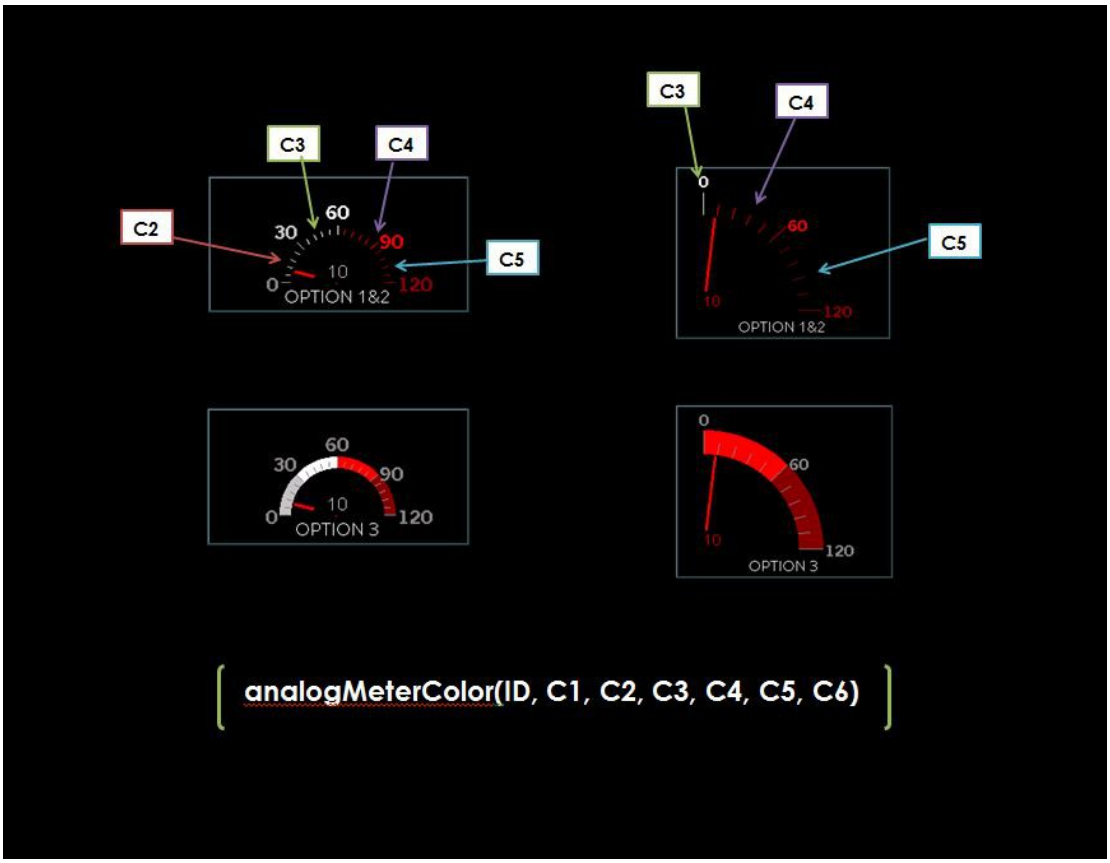
Analog Meter Color is a command that allows you to change the colors of the interval marks.

Command:
lcd.analogMeterColor(ID,color1,color2,color3,color4,color5,color6)

Parameters:
ID should be the ID of the analog meter you want to change.
color 1-6 changes the colors of the 6 "zones" of indicating lines and numbers of the meter arc. The zones start at **1** at the left end of the arc and **6** being the right end of the arc. For half meter type only *colors 3-5* are used. For quarter meter type only *colors 3-5* are used.

Hint: To change the color of an Analog Meter before its created use ID 101. Then when the analog meter is created it will have the desired colors.




C3
C4
C2
C5
0
30
60
90
10
120
OPTION 1&2
C3
C4
0
60
C5
10
120
OPTION 1&2
60
30
90
0
10
120
OPTION 3
0
60
10
120
OPTION 3
analogMeterColor(ID, C1, C2, C3, C4, C5, C6)

Button

Button widget provides the user a simple way to trigger an event. You can use the button command to draw or make different size and shapes of buttons.

Command:
lcd.button(ID, x, y, width, height, options, align, radius, theme, string);

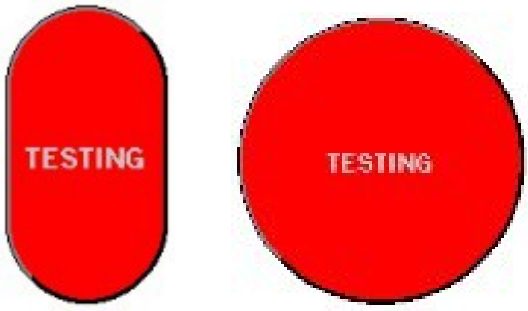
Parameters:
ID is an ID number, **1-99,** specified to a particular widget.
x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.
width and **height** values designate the width and height of the button in pixels.
options designates the state of the button, whether it is pressed, disabled and etc. when its created
Option choices: 1=draw, 2=disabled, 3=toggle pressed, 4=toggle not pressed, 5=toggle pressed disabled, 6=toggle not pressed disabled.

Unpressed *Pressed* *Disabled*

align value, designates the alignment of the text as it appears on the button.
Alignment choices: 0=centered, 1=right, 2=left, 3=bottom, 4=top.

Left *Center* *Right* *Top* *Bottom*

radius designates the radius of the button's corners in pixels. A value of 0 creates a square corner, while a value that is half the length of one side will give a round button. To see some different shapes for buttons, run demo buttons.ezm.



Tall Button *Round*

theme value designates the widget theme
stringID designates the ID number of the text string that you'd like displayed on the button.
Note: To create multi-line text on buttons, use \n in the string contents. Example: string 5 "Wrap\nText" will appear on 2 lines.

Checkbox

This widget allows you to display a check box with a string next to it.

Command:
lcd.checkbox(ID, x, y, width, height, option, theme, string);

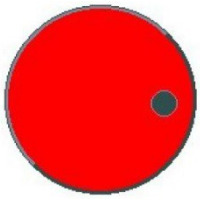
Parameters:
ID is the widget ID number.
x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.
width and **height** values designate the width and height of the widget in pixels.
option designates the initial state of the checkboxes.
Option choices: 1=draw unchecked, 2=draw disabled, 3=draw checked, 4=redraw
theme the theme ID you want to use in order to change the colors on the widget.
stringID designates the ID number of the text string that you'd like displayed next to the box that indicates checked or not.

Dial

The DIAL widget allows you to display a dial that looks like an analog volume control found on stereo equipment.

Command:
lcd.dial(ID, x, y, radius, option, resolution, initial, max, theme);

Parameters:
ID is an ID number.
x and **y** values designate the location of the widget on the screen as the **XY** coordinate of the center of the dial.
radius values means that the radius of the dial is 75, which the diameter will be 150.
option designates the state of the dial. Option choices: **1=draw, 2=disabled.**
resolution designates the increments in the range. A resolution of 1 will be a value of every number, such as, 10,11,12,13,14,15.... has a resolution of **1**. A resolution of 3 for instance will show a value of 10,13,16,19.
value designates the initial dial value.
max value designates the largest value of the dial's input.
theme is the ID of the theme you want to use.



Digital Meter

The Digital Meter widget DMETER allows you to display a digital meter as in a panel meter.

Command:

lcd.digitalMeter(ID, x, y, width, height, option, initial, digits, dotpos, theme);

Parameters:

ID is an ID number

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option determines the alignment of the digits and whether the box is framed. Option choices: 1=left, 2=disabled, 3=right, 4=center, 11=left framed, 12=disable framed, 13=right framed, 14=center framed, 6=redraw.

value designates and displays the initial setting of the readout as it appears on the meter.

digits value designates the number of digits displayed on the meter.

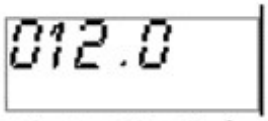
dp value designates the **dot position** of the decimal point from the 'right' most number.

theme is the theme ID you want to use (theme must be designated before the widget)

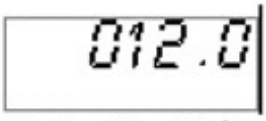
Option 1 = Left

Option 3 = Right

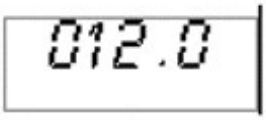
Option 4 = Center



Option 11 = Left Framed



Option 13 = Right Framed



Option 14 = Center Framed

Group Box

The groupbox widget generates a border/box and by changing the options positions the header text at different alignments. Group boxes help visually distinguish related items by framing them. The groupbox consists of a frame, title and a title background.

Command:

lcd.groupBox(ID, x, y, width, height, option, theme, string);

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (320,240) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option determines the header alignments. (The options do not affect the contents' alignment) *Option choices:* 1=left,2=disabled,3=right,4=center

Option 1 = Left *Option 3 = Right* *Option 4 = Center*

theme is the theme ID you want to use (theme must be designated before the widget)

string is the ID number of the string you want as a header of the box.

Progress Bar

The PROGRESS widget allows you to display a progression bar at an initial state.

Command:

lcd.progressBar(ID, x, y, width,height, option, initial, range, theme, suffix);

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (320,240) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option designates the option of the progress bar.

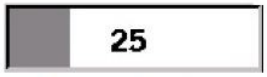
Option choices: 1=draw horizontal, 2=horizontal disabled, 3=vertical, 4=vertical disabled, 5=redraw horizontal, 6=redraw horizontal disabled, 7=redraw vertical, 8=redraw vertical disabled

value designates the initial value. By using the WVALUE command changes the initial value to a different one.

max value, 100, designates the maximum value that can be reached.

theme is the theme ID you want to use (theme must be designated before the widget)

string is the ID of the string you want to use.



Option 1 = Horizontal

Option 3 = Vertical

Gauge

The Gauge widget allows you to display a gauge at an initial state. It is similar to the progress bar but also allows negative numbers.

Command:

lcd.gauge(ID, x, y, width, height, option, value, min, max, theme, suffix);

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option designates the option of the progress bar.

Option choices: 1=draw horiz, 2=horiz disabled, 3=vert, 4=vert disabled, 5=redraw horiz, 6=redraw horiz disabled, 7=redraw vert, 8=redraw vert disabled

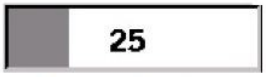
value designates the initial value.

min value, designates the minimum value that can be reached.

max value, designates the maximum value that can be reached.

theme is the theme ID you want to use (theme must be designated before the widget)

string is the ID of the string you want to use.



Option 1 = Horizontal

Option 3 = Vertical

Radio Button

The radio button widget allows you to display buttons for making a selection. Radio buttons differ from checkboxes in that only one button can be filled in at a time, while checkboxes can have many filled in at once. This makes radio buttons interconnected. If one button is checked then the others will go to or remain as an 'unchecked' state.

Command:

lcd.radioButton(ID, x, y, width, height, option, theme, string);

Parameters:

ID is an ID number.
Although radio buttons are connected as a group, each button still needs its own ID number.

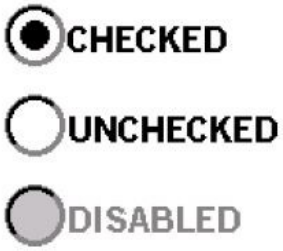
x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the entire widget in pixels which are the area that encompass the radio button and the string. The size of the radio button itself is defined by **height**.

option allow you to draw radio buttons checked, unchecked, or disabled. By disabling a button, the user will not be able to change its state. Options 4 (first checked) and Options 5(first unchecked) specify that it is the first button in a group. This allows you to have more than one group of buttons occupying the screen at the same time. When Options 4 or 5 are specified in a radio button command, the following buttons created will be in a group until another "first" button is defined. The buttons created after will be in the second group. If you make a button, "first unchecked", remember to draw one button in the group as "checked".
Option choices*:* 1=unchecked, 2=disabled, 3=checked, 4=FIRST unchecked, 5=FIRST checked

theme is the theme ID you want to use.

string designates the ID number of the text string that you want displayed on the button.


CHECKED
UNCHECKED
DISABLED

Slider

The slider widget allows you to display a vertical or horizontal slider or scroll bar that looks like a light dimmer. The slider widget components are the slider and a handle, also known as the thumb or indicator.

Command:

lcd.slider(ID, x, y, width, height, option, max, resolution, initial, theme);

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option designates the options of the slider.

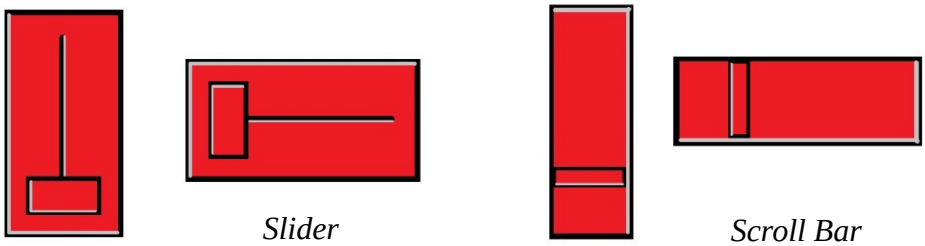
Option choices: 1=draw horizontal, 2=horizontal disabled, 3=vertical, 4=vertical disabled, 5=horizontal slider scrollbar, 6= disabled horizontal scrollbar, 7=vertical slider scrollbar, 8=disabled vertical scrollbar

max designates what the maximum value the slider can have

resolution is the increments or steps the indicator will move along the bar. ex. max = 10 & resolution = 5 then the only possible values are 0,5,10,

value designates the initial value of the indicator.

theme is the ID of the theme you want to use.



Slider *Scroll Bar*

Static Text Box

The static text box widget generates a framed text box with a string (or text) at different alignments. This command changes text within a box without having to overwrite its background.

Command:

lcd.staticText(ID, x, y, width, height, option, theme, string);

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

option designates the alignments of the static widget. Redraw clears the background of the assigned area then rewrites the text.

> *Option choices*: 1=left, 2=disabled, 3=right, 4=center, 5=left framed, 6=disabled framed, 7=right framed, 8=center framed, 9=redraw

theme is the ID of the theme you want to use.

string designates the ID number of the text string that you'd like displayed.

Touch Zone

Touchzone is similar to a button in action but without the button visualization. The user would typically place a graphic or text on the screen and create zones under specific parts of the graphic depending on what the function is. When the user touches these areas, the touchzone will indicate to the user that a touch has occurred and where.

Command:

lcd.touchZone(ID, x, y, width, height, option);

Note: There is no need for a theme or string as that would be user created without regard to the Touchzone widget.

Parameters:

ID is an ID number.

x and **y** values designate the location of the widget on the screen as the **XY** coordinate. Origin, (0,0), being the top left corner and (319,239) the bottom right.

width and **height** values designate the width and height of the widget in pixels.

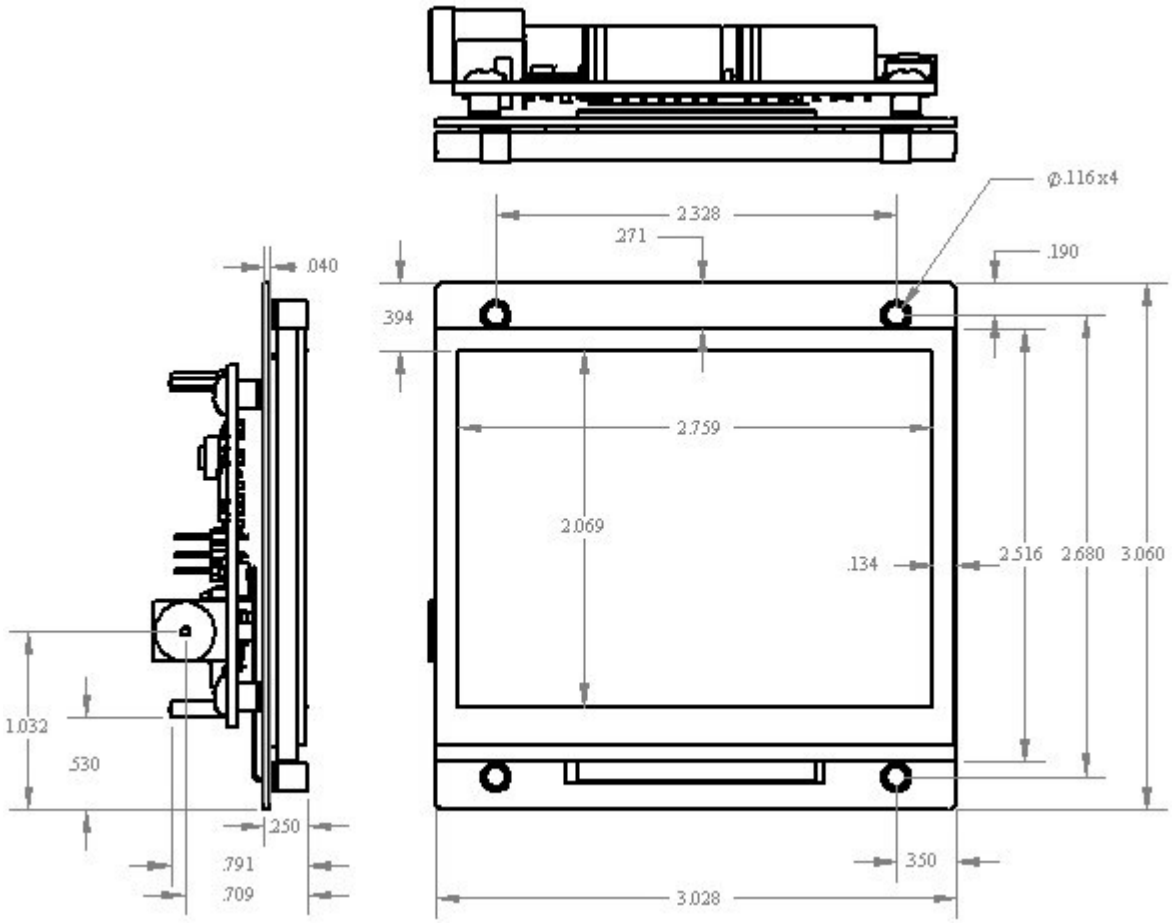
option designates the "mode" of the touch zone being either enabled or disabled. When disabled, "touches" in the designated area on the screen, will do nothing.

Option choices: 1 = enable, 2 = disable

Color Index


0 Black 0, 0, 0
1 Gray 128, 128, 128
2 Silver 192, 192, 192
3 White 255, 255, 255
4 Red 255, 0, 0
5 Maroon 128, 0, 0
6 Yellow 255, 255, 0
7 Olive 128, 128, 0
8 Lime 0, 255, 0
9 Green 0, 128, 0
10 Aqua 0, 255, 255
11 Teal 0, 128, 128
12 Blue 0, 0, 255
13 Navy 0, 0, 128
14 Fuchsia 255, 0, 255
14 Magenta 255, 0, 255
15 Purple 128, 0, 128
16 IndianRed 205, 92, 92
17 LightCoral 240, 128, 128
18 Salmon 250, 128, 114
19 DarkSalmon 233, 150, 122
20 LightSalmon 255, 160, 122
4 21 Red 255, 0, 0
22 Crimson 220, 20, 60
23 FireBrick 178, 34, 34
24 DarkRed 139, 0, 0
25 Pink 255, 192, 203
26 LightPink 255, 182, 193
27 HotPink 255, 105, 180
28 DeepPink 255, 20, 147
29 MediumVioletRed 199, 21, 133
30 PaleVioletRed 219, 112, 147
31 LightSalmon 255, 160, 122
32 Coral 255, 127, 80
33 Tomato 255, 99, 71
34 OrangeRed 255, 69, 0
35 DarkOrange 255, 140, 0
36 Orange 255, 165, 0
37 Gold 255, 215, 0
6 38 Yellow 255, 255, 0
39 LightYellow 255, 255, 224
40 LemonChiffon 255, 250, 205
41 LightGoldenrodYellow 250, 250, 210
42 PapayaWhip 255, 239, 213
43 Moccasin 255, 228, 181
44 PeachPuff 255, 218, 185
45 PaleGoldenrod 238, 232, 170
46 Khaki 240, 230, 140
47 DarkKhaki 189, 183, 107
48 Lavender 230, 230, 250
49 Thistle 216, 191, 216
50 Plum 221, 160, 221
51 Violet 238, 130, 238
52 Orchid 218, 112, 214
14 53 Fuchsia 255, 0, 255
14 53 Magenta 255, 0, 255
54 MediumOrchid 186, 85, 211
55 MediumPurple 147, 112, 219
56 BlueViolet 138, 43, 226
57 DarkViolet 148, 0, 211
58 DarkOrchid 153, 50, 204
59 DarkMagenta 139, 0, 139
60 Purple 128, 0, 128
61 Indigo 75, 0, 130
63 SlateBlue 106, 90, 205
62 DarkSlateBlue 72, 61, 139
65 GreenYellow 173, 255, 47
66 Chartreuse 127, 255, 0
67 LawnGreen 124, 252, 0
8 68 Lime 0, 255, 0
69 LimeGreen 50, 205, 50
70 PaleGreen 152, 251, 152
71 LightGreen 144, 238, 144
72 MediumSpringGreen 0, 250, 154
73 SpringGreen 0, 255, 127
74 MediumSeaGreen 60, 179, 113
75 SeaGreen 46, 139, 87
76 ForestGreen 34, 139, 34
9 77 Green 0, 128, 0
78 DarkGreen 0, 100, 0
79 YellowGreen 154, 205, 50
80 OliveDrab 107, 142, 35
7 81 Olive 128, 128, 0
82 DarkOliveGreen 85, 107, 47
83 MediumAquamarine 102, 205, 170
84 DarkSeaGreen 143, 188, 143
85 LightSeaGreen 32, 178, 170
86 DarkCyan 0, 139, 139
11 87 Teal 0, 128, 128
10 88 Aqua 0, 255, 255
10 89 Cyan 0, 255, 255
90 LightCyan 224, 255, 255
91 PaleTurquoise 175, 238, 238
92 Aquamarine 127, 255, 212
93 Turquoise 64, 224, 208
94 154 MediumTurquoise 72, 209, 204
95 DarkTurquoise 0, 206, 209
96 CadetBlue 95, 158, 160
97 SteelBlue 70, 130, 180
98 LightSteelBlue 176, 196, 222
99 PowderBlue 176, 224, 230
100 LightBlue 173, 216, 230
101 SkyBlue 135, 206, 235
102 LightSkyBlue 135, 206, 250
103 DeepSkyBlue 0, 191, 255
104 DodgerBlue 30, 144, 255
105 CornflowerBlue 100, 149, 237
105 MediumSlateBlue 123, 104, 238
106 RoyalBlue 65, 105, 225
107 Blue 0, 0, 255
108 MediumBlue 0, 0, 205
109 DarkBlue 0, 0, 139
13 110 Navy 0, 0, 128
111 MidnightBlue 25, 25, 112
112 Cornsilk 255, 248, 220
113 BlanchedAlmond 255, 235, 205
114 Bisque 255, 228, 196
115 NavajoWhite 255, 222, 173
116 Wheat 245, 222, 179
117 BurlyWood 222, 184, 135
118 Tan 210, 180, 140
119 RosyBrown 188, 143, 143
120 SandyBrown 244, 164, 96
121 Goldenrod 218, 165, 32
122 DarkGoldenrod 184, 134, 11
123 Peru 205, 133, 63
124 Chocolate 210, 105, 30
125 SaddleBrown 139, 69, 19
126 Sienna 160, 82, 45
127 Brown 165, 42, 42
5 128 Maroon 128, 0, 0
3 129 White 255, 255, 255
130 Snow 255, 250, 250
131 Honeydew 240, 255, 240
132 MintCream 245, 255, 250
133 Azure 240, 255, 255
134 AliceBlue 240, 248, 255
135 GhostWhite 248, 248, 255
136 WhiteSmoke 245, 245, 245
137 Seashell 255, 245, 238
138 Beige 245, 245, 220
139 OldLace 253, 245, 230
140 FloralWhite 255, 250, 240
141 Ivory 255, 255, 240
142 AntiqueWhite 250, 235, 215
143 Linen 250, 240, 230
144 LavenderBlush 255, 240, 245
145 MistyRose 255, 228, 225
146 Gainsboro 220, 220, 220
147 LightGrey 211, 211, 211
2 148 Silver 192, 192, 192
149 DarkGray 169, 169, 169
1 150 Gray 128, 128, 128
151 DimGray 105, 105, 105
152 LightSlateGray 119, 136, 153
153 SlateGray 112, 128, 144
155 DarkSlateGray 47, 79, 79
0 156 Black 0, 0, 0

Mechanical Dimensions



Starting an arLCD Project

Starting your first, or any, project with your arLCD can seem daunting. Our goal was to make this easier, but here are a couple of pointers that will help you get set up quicker. Be sure to check out the Arduino website for tips and help on writing a sketch.

1) Search the web for examples of similar projects to yours. The Arduino community is vast and there is bound to be someone who had a similar idea to yours. Check out their code and tweak it to see how it works.
2) We have provided 25+ sample sketches. These should be a pretty good starting point for your project. Mix and match is how you learn new things.
3) Most devices require "power" and "ground" in order to function. The component's "power" and "ground" wires should correspondingly go into the arLCD's ground and power pins.
4) Writing a sketch may take a bit more time if you are not familiar with coding. However, don't fret, check out Examples and play around with them to see how it works. Also, Arduino has a page that helps with the basics on writing a sketch. Provided below, is a sketch template that has important code that you need to run a sketch on the arLCD.
5) Template

```
#include <ezLCDLib.h>

ezLCD3 lcd; // create lcd object

void setup(){//use the setup function to initialize the display
  lcd.begin( EZM_BAUD_RATE );
  lcd.cls();
  lcd.print("\\[50x\\[50y\\[4mTemplate");
  //your setup code goes here
}

void loop(){//loop function is the main part of your program
//your project code goes here
}
```

How pins on Arduino are mapped to ezLCD303

All nine of the ezLCD IO pins are connected to the Atmel CPU (Atmega328). The startup file in the ezLCD303 sets up the IO pins to control the communication between the GPU and the Atmel CPU.

Some of the uses have changed with different firmware releases.

Version 2.08 used D10 and D11 for serial access to ezLCD command port. This used software serial UART. 38K command baud rate (Rev A)

Version 2.10 stopped using software serial and used the hardware serial for programming and command port. 115K command baud rate. Serial Monitor would use software serial.

Version 2.11 uses the hardware serial for command port and uses the USB port of the GPU to program the Atmega328 through SPI. 230K command baud rate. Serial Monitor uses GPU USB port.

The connections of the GPU to the Atmel CPU are;

Signal	Arduino Pin	ezLCD303 IO Pin	UART Sharing
RESETn	RESET	1	-
TX	D0	2	COM TX
MOSI	D11	3	UART2 RX
D10/SS	D10	4	UART2 TX
MISO	D12	5	UART3 RX
RX	D1	6	COM RX
DTR	DTR	7	-
SCK	D13 (LED)	8	UART3 TX
INT	D2	9	Spare

* Used by GPU for programming Atmega328 CPU (SPI) in 2.11. Must be enabled at beginning of sketch as programming will disable it. If INT is not used it is a spare pin and can be used for any UART function or IO. DTR is no longer used with SPI programming.