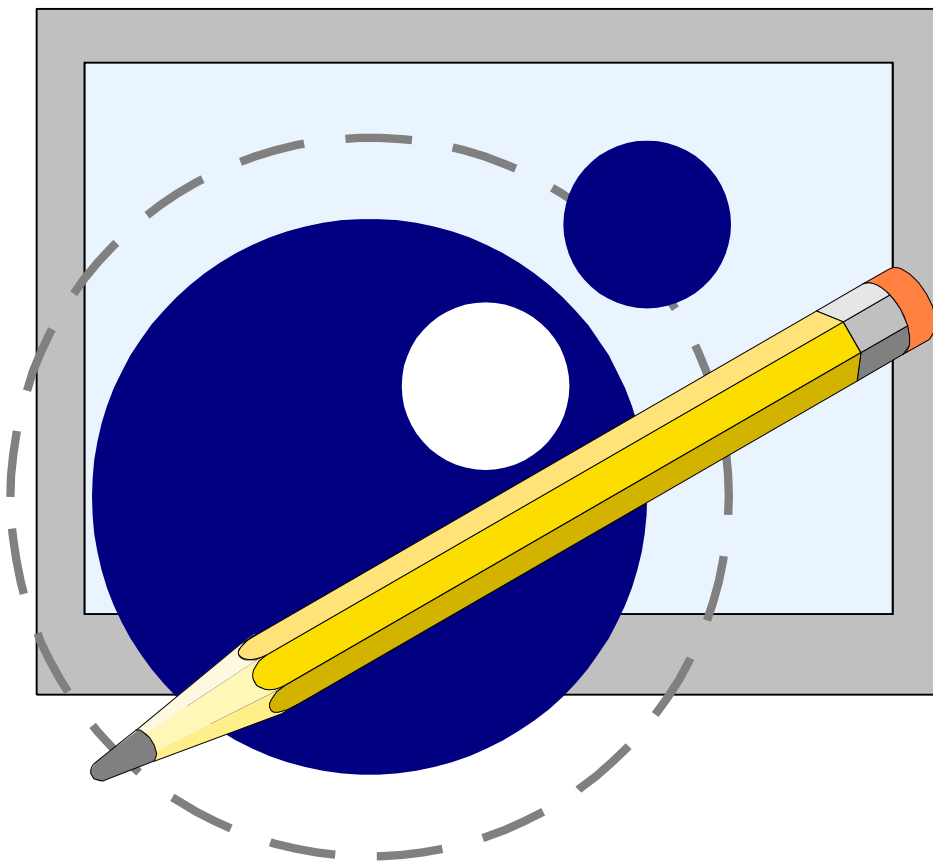


ezLCD-5x

Lua API Manual



ezLCD Documentation Overview

The ezLCD documentation consists of:

"ezLCD10x Manual"

Specific for each ezLCD device (ezLCD101, ezLCD102, .. etc.).

- *Provides "Quick Start" instructions.*
- *Describes the hardware of the particular device.*
- *Describes how to load a new firmware and how to customize your ezLCD device.*

"ezLCD External Commands Manual"

Common for all ezLCD products.

- *Describes the set of commands, which can be sent to the ezLCD through any of the implemented interfaces (USB, RS232, SPI, etc.). Those commands may be sent by an external host (PC or microcontroller).*
- *Describes the API of the ezLCD Windows USB driver.*

"ezLCD Lua API Manual"

Common for all ezLCD products.

All ezLCD products have an embedded Lua interpreter. The ezLCD Lua API has been developed to access all graphic and I/O capabilities of the ezLCD device using the Lua language.

*** Programming in Lua (second edition) By Roberto Ierusalimsky**

Common for all ezLCD products.

The official book about the Lua programming language. It is available at:

<http://www.amazon.com/exec/obidos/ASIN/8590379825/lua-docs-20>

More information about Lua can be found at:

<http://www.lua.org/>

** Not included. Must be downloaded or purchased separately.*

Table of Contents

Contents

Release History	8
Product Features	10
ezLCD Customization.....	13
1 Drawing on the ezLCD.....	14
2. Screen Coordinates	15
3. Vector Graphics	16
4. Raster Graphics (Bitmaps)	17
5. Drawing Parameters.....	18
5.1 General	18
5.2 Transparency	19
5.3 Pen	20
6 Fonts	22
6.1 Bitmap Fonts.....	23
6.2 True Type Fonts	24
Programming the ezLCD with Lua	25
7 Constants	26
8 Position Functions.....	27
8.1 ez.SetXY	28
8.2 ez.SetX	29
8.3 ez.SetY.....	30
8.4 ez.GetX.....	31
8.5 ez.GetY.....	32
9 Color Functions.....	33
9.1 ez.RGB	34
9.2 ez.GetRed.....	35
9.3 ez.GetGreen	36
9.4 ez.GetBlue.....	37
9.5 ez.SetColor.....	38
9.6 ez.SetBgColor.....	39
9.7 ez.ReplaceColor.....	40
9.8 ez.GetPixel	41
10 Transparency Functions	42
10.1 ez.SetAlpha.....	43
10.2 ez.TrColorNone.....	44
10.3 ez.SetTrColor	45
11 Pen Size Functions	46
11.1 ez.SetPenSize	47
12 Angle Functions	48
12.1 ez.Deg.....	49
12.2 ez.Rad.....	50
13 Button Functions	51

13.1	ez.Button	53
13.2	ez.Button	54
13.3	ez.DelButtons.....	55
13.4	ez.SetButtonEvent.....	56
14	Fill Area Functions	57
14.1	ez.Cls.....	59
14.2	ez.Fill	60
14.3	ez.FillBound	61
15	Line Drawing Functions	62
15.1	ez.HLine.....	63
15.2	ez.VLine.....	64
15.3	ez.Line	65
15.4	ez.LineAng	66
16	Curve Drawing Functions	67
16.1	ez.Circle.....	68
16.2	ez.CircleFill	69
16.3	ez.Ellipse	70
16.4	ez.EllipseFill	71
16.5	ez.Arc	72
16.6	ez.Pie	73
16.7	ez.EllipseArc	74
16.8	ez.EllipsePie.....	75
17	Polygon Drawing Functions	76
17.1	ez.Box.....	77
17.2	ez.BoxFill.....	78
17.3	ez.Polygon	79
18	Single Pixel Functions	80
18.1	ez.Plot.....	81
18.2	ez.GetPixel	82
19	Font Functions	83
19.1	ez.SetBmFont.....	84
19.2	ez.SetFtFont	85
19.3	ez.GetNoOfBmFonts.....	86
19.4	ez.GetNoOfFtFonts	87
19.6	ez.CacheFtChars.....	88
19.8	ez.SetFtUnibase.....	89
20	Text Orientation Functions	90
20.1	ez.TextNorth.....	91
20.2	ez.TextEast.....	92
20.3	ez.TextSouth	93
20.4	ez.TextWest.....	94
20.5	ez.SetFtAngle.....	95
21	Bitmap Functions.....	96
21.1	ez.PutPictNo.....	97
21.2	ez.PutPictFile	98
21.3	ez.GetPictHeight.....	99
21.4	ez.GetPictWidth	100
22	Backlight Functions.....	103
22.1	ez.LightOn.....	104
22.2	ez.LightOff.....	105
22.3	ez.LightBright	106
23	Screen Capture Functions.....	107

23.1	ez.SdScreenCapture	108
24	Time Functions	109
24.1	ez.Get_ms	110
24.2	ez.Wait_ms	111
24.3	ez.SetTime	112
25	Timer Management Functions	113
25.1	ez.Timer	114
25.2	ez.Timer	115
25.3	ez.TimerStart	116
25.4	ez.TimerStop	117
26	Touch Function	118
26.1	ez.GetTouchX	119
26.2	ez.GetTouchY	120
26.3	ez.TouchDn	121
26.4	ez.SetTouchEvent	122
27	SD Card Access	123
28	RS232 Functions	124
28.1	ez.SerialOpen	125
28.2	ez.SerialOpen	126
28.3	ez.SerialClose	127
28.4	ez.SerialTx	128
28.5	ez.SerialRxLen	129
28.6	ez.Serialgetc	130
29	I2C Functions	131
29.1	ez.I2CopenMaster	132
29.2	ez.I2CWrite	133
29.3	ez.I2Cread	134
30	GPIO Functions	135
30.1	ez.SetPinInp	136
30.2	ez.SetPinsInp	137
30.3	ez.SetPinOut	138
30.4	ez.SetPinsOut	139
30.5	ez.SetPinIntr	140
30.6	ez.RestorePin	141
30.7	ez.RestorePins	142
30.8	ez.Pin	143
30.9	ez.Pin	144
30.10	ez.Pins	145
30.11	ez.Pins	146
31	Quadrature Encoder Interface	147
31.1	ez.QuadOpen	148
31.2	ez.QuadClose	149
31.3	ez.QuadGetValue	150
32	CANBus Interface	151
32.1	Low-Level CANBus Interface	152
32.1.1	ez.CANBusOpen	153
32.1.2	ez.CANBusClose	154
32.1.3	ez.CANBusStart	155
32.1.4	ez.CANBusStop	156
32.1.5	ez.CANBusRegisterHandler	157
32.1.6	ez.CANBusConfigFilter	158
32.1.7	ez.CANBusGlobalFilter	159

32.1.8	ez.CANBusEnableISOMode	160
32.1.9	ez.CANBusTx	161
32.1.10	ez.CANBusTxAbort	162
32.1.11	ez.CANBusRx	163
32.1.12	ez.CANBusIsTxBuffFree	164
32.1.13	ez.CANBusIsRxMsgAvail	165
32.1.14	ez.CANBusGetFIFOLevel	166
32.1.15	ez.CANBusGetErrCnt	167
33	DMX512 Interface	168
33.1	ez.DMXOpen	169
33.2	ez.DMXClose	170
33.3	ez.DMXSetLevel	171
33.4	ez.DMXGetLevel	172
34	Advanced Topics	173
34.1	Frame Management Functions	174
34.1.1	ez.SetDispFrame	175
34.1.3	ez.GetDispFrame	176
34.1.4	ez.GetLayerVisibility	177
34.1.5	ez.SetLayerVisibility	178
34.1.6	ez.SetDrawFrame	179
34.1.7	ez.GetDrawFrame	180
34.1.8	ez.CopyFrame	181
34.1.9	ez.MergeFrame	182
34.1.10	ez.CopyRect	183
34.1.11	ez.MergeRect	184
GLOSSARY		185

Release History

Date	Description
06-JUL-2020	Initial Release

Introduction

Welcome to the ezLCD API (Application Programming Interface) Manual for Lua. This manual details how to programmatically manipulate the EarthLCD ezLCD series of programmable color LCD's using the Lua programming language. ezLCD displays are color touch screen displays that can be easily and quickly integrated into a wide variety of applications. The ezLCD with the Lua interpreter can operate as a stand-alone embedded system.

ezLCD displays are very similar to our original ezLCD Classic line of displays. ezLCD devices are programmable color LCD's and support the ezLCD command set as documented in the ezLCD External Commands Manual. ezLCD devices can be programmed using the Lua programming language.

You can find more information about our products from our web site at <http://store.earthlcd.com/LCD-Products/ezLCD>

For support on our products, contact us at 949-248-2333 Ext 235 or support@earthlcd.com

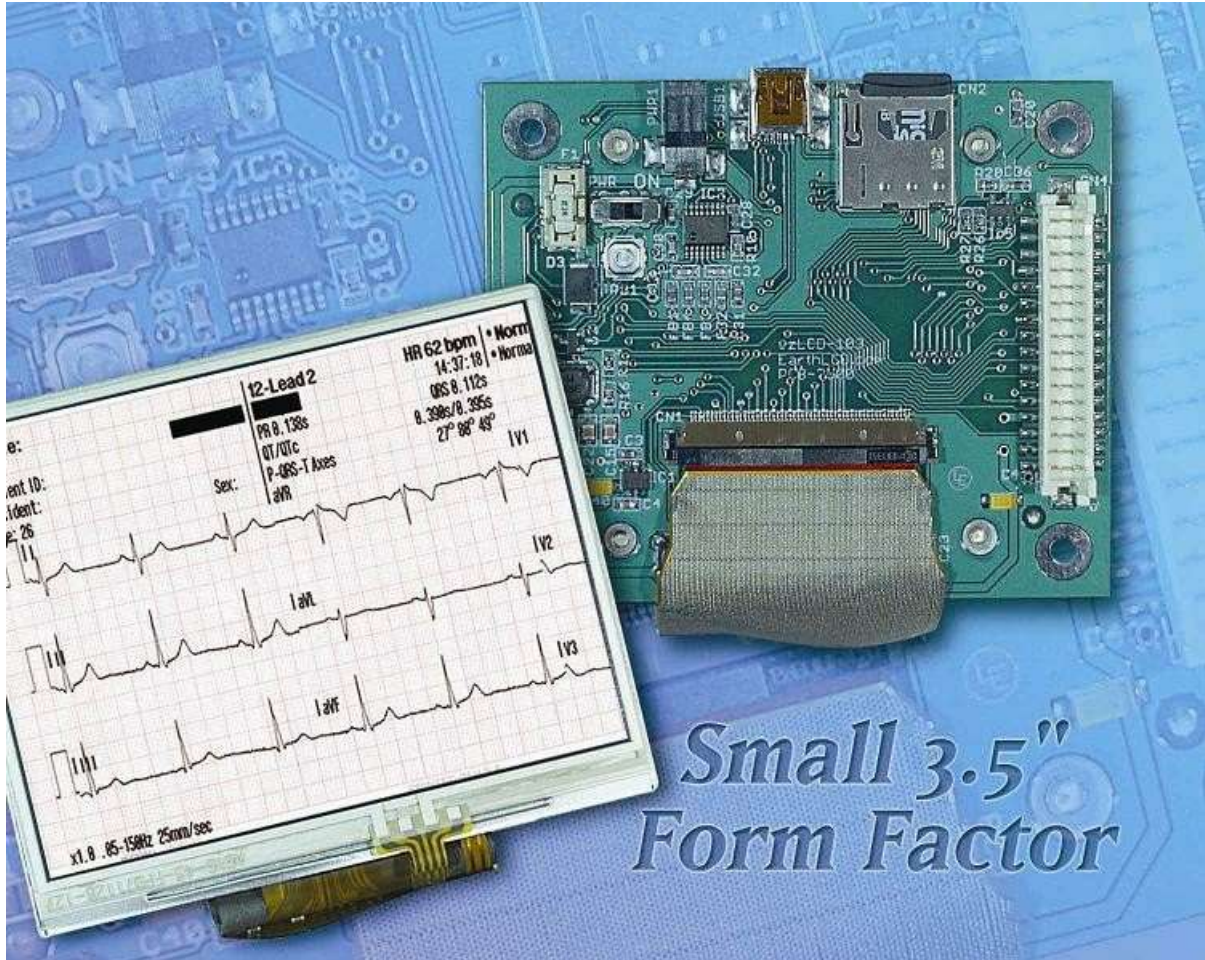
You can download the latest version of this manual at <http://www.ezlcd.com/support/>

An online version of the Lua programming manual can be found at <http://www.lua.org/docs.html>

We also offer consulting, design and implementation services to assist you in easily integrating our LCD's into your products. For details on these services, contact us at 949-248-2333 Ext 222 or sales@earthlcd.com

Product Features

The ezLCD series of programmable color LCD's consist of a color display, touch screen, USB Interface, RS-232 Interface, I2C Interface and is programmable using the Lua programming language. Other interfaces such as Ethernet and Audio will be available in future releases.



This manual applies to the following products

Product Name	ezLCD-4056	ezLCD-5035	ezLCD-105 Rev B
Display Size	5.6"	3.5"	8.0"
Screen Resolution	640x480	320x240	800x600
Flash	SDCard	SDCard	SDCard
DRAM (minimum)	8MB	32MB	64MB

Quick Start

Quick Start Requirements:

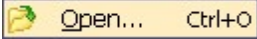
- PC Computer with at least 1 USB 2.0 port
- Microsoft Windows 10

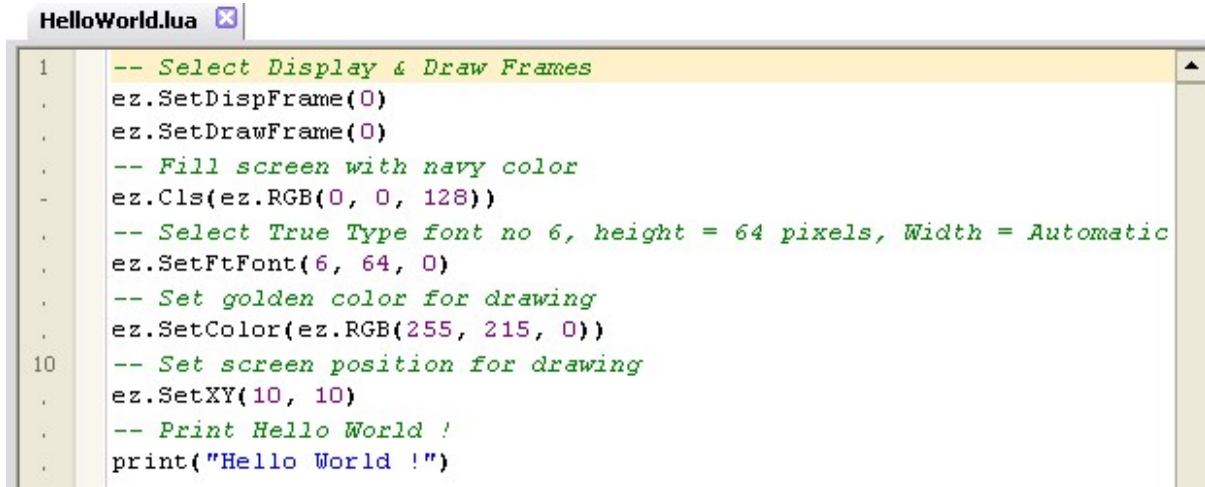
Note: *The ezLCD products do not need a PC computer to work. The above requirements are for the "Quick Start" only.*

Quick Start


1. Connect ezLCD USB to your computer and turn the ezLCD power on by sliding the PWR switch into "ON" position. "New Hardware Found" wizard should appear. Select automatic driver installation. Turn-off ezLCD after the driver have successfully been installed.
2. Go to chapter: "[Quick Start: Lua](#)".

1 Quick Start: Lua

1. Download the setup of "ezLuaIDE" from <http://www.ezlcd.com/support/>
2. Install "ezLuaIDE" by running the downloaded setup
3. Turn-on ezLCD and make sure that it is connected to your computer through USB.
4. Run "ezLuaIDE". From the Menu, select "File" - "Open" 
5. Select HelloWorld.lua file from the folder "Program Files\ezLuaIDE\Examples".



```
1  -- Select Display & Draw Frames
.  ez.SetDispFrame(0)
.  ez.SetDrawFrame(0)
.  -- Fill screen with navy color
.  ez.Cls(ez.RGB(0, 0, 128))
.  -- Select True Type font no 6, height = 64 pixels, Width = Automatic
.  ez.SetFtFont(6, 64, 0)
.  -- Set golden color for drawing
.  ez.SetColor(ez.RGB(255, 215, 0))
10 -- Set screen position for drawing
.  ez.SetXY(10, 10)
.  -- Print Hello World !
.  print("Hello World !")
```

6. Press  button. The ezLCD should display "Hello World!" in golden color over navy background:



For more information about Lua on ezLCD and ezLuaIDE, please refer to the "ezLCD Lua API Manual".

ezLCD Customization

To make the ezLCD easy to use we created a set of tools and features to configure, upgrade and enhance the functionality. The ezLCD customization features are documented in your "ezLCD-5x Manual" and updates are available at <http://www.ezLCD.com/support> .

1 Drawing on the ezLCD

Print

The Lua native print function is used to write strings to the ezLCD display.

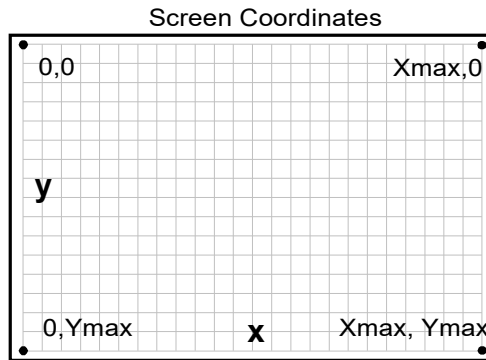
As the print function is part of the standard Lua language, make sure to ***not*** prepend "ez."

Example

Use `print("Hello World")`

Not `ez.print("Hello world")`

2 Screen Coordinates



For displaying both raster and vector graphics, the ezLCD uses the X-Y Cartesian coordinate system. The origin is located in the upper-left corner of the display. The X values increase to the right, while Y increase to the bottom of the display.

The ezLCD uses 16-bit numbers to specify X and Y coordinates. Negative numbers are represented using two's complement system. For example:

```

2 dec = 0000 0000 0000 0010 bin
1 dec = 0000 0000 0000 0001 bin
0 dec = 0000 0000 0000 0000 bin
-1 dec = 1111 1111 1111 1111 bin
-2 dec = 1111 1111 1111 1110 bin
etc.

```

This means that the numbers range

From: -32768 dec = 1000 0000 0000 0000 bin

To: 32767 dec = 0111 1111 1111 1111 bin

The above system is used to represent 16-bit signed integers by most of the CPUs and programming languages.

The ezLCD drawing position (Current Position) may be set outside the screen range. The portions of the image, which do not fit on the screen are just clipped-out. For example: if a circle is drawn with radius 100 and the center at $x = -20$, $y = -30$, the following figure will appear at the upper-left corner of the screen:



The Current Position is updated by some drawing commands. For example: if you set the Current Position to (10, 20) and then draw the line to (200, 100), the Current Position will change to (200, 100).

3 Vector Graphics

Vector Graphics is the use of geometrical primitives such as points, lines, curves, and polygons, which are all based upon mathematical equations to represent images in computer graphics. It is used in contrast to the term [Raster Graphics](#), which is the representation of images as a collection of pixels.

The ezLCD supports drawing of various geometrical shapes, like [lines](#), [polygons](#), [ellipses](#), [arcs](#), etc.

The rendering of Vector Graphics is affected by the following [Drawing Parameters](#):

- Current Position
- Current Color
- Transparency
- Pen
- Current Drawing Frame

Note: Since the ezLCD is physically a raster display, all Vector Graphics is converted to the Raster Graphics during rendering.

4 Raster Graphics (Bitmaps)

A Raster Graphics image, digital image, or bitmap, is the representation of images as a collection of pixels, or points of color. It is used in contrast to the term

[Vector Graphics](#) which is the use of geometrical primitives such as points, lines, curves, and polygons, all based upon mathematical equations to represent images.

Raster images are commonly stored in image files with varying formats. The ezLCD can display the following formats of raster images:

- 24-bit .bmp
- .jpg
- .ezp (16-bit color format used in other ezLCD products, added here for compatibility).

A bitmap corresponds bit-for-bit with an image displayed on a screen, in the same format used for storage in the display's video memory. Bitmap is technically characterized by the width and height of the image in pixels and by the number of bits per pixel (a color depth, which determines the number of colors it can represent).

The bitmaps (raster images), can be displayed from the User ROM or SD card using the [Bitmap Functions](#).

Additionally, ezLCD supports direct pixel drawing on the display using the [Single Pixel Functions](#)

The rendering of Raster Graphics is affected by the following [Drawing Parameters](#):

- Current Position
- Current Drawing Frame
- Transparency
- Transparent Color (direct pixel drawing is not affected)

5 Drawing Parameters

5.1 General

Graphics are drawn according to the following parameters:

Current Drawing Frame

- Set by [Frame Management Functions](#)

Current Position.

- Set by the [Position Functions](#)
- Updated by drawing commands

Current Color.

- Set by
- [ez.SetColor](#)
- [Raster Graphics \(Bitmaps\)](#) are not affected

Background Color.

- Set by
- [ez.SetBgColor](#)
- Only [Bitmap Fonts](#) are affected

Transparent Color.

- Set by [ez.SetTransparentColor](#) and [ez.TrColorNone](#)
- Specifies the color, which is ignored during Bitmap drawing
- Only [Raster Graphics \(Bitmaps\)](#) are affected (direct pixel drawing is not affected).

[Transparency](#)

- Set by [ez.SetAlpha](#)

[Pen](#)

- Set by [ez.SetPenSize](#)
- Only the
- [Vector](#) Graphics is affected
- Pen Height affects only the drawing of curves (ellipse, circle, arc, etc)

5.2 Transparency

The ezLCD supports transparency by alpha-blending of the pixel being drawn with the background pixel at the particular position. Alpha blending is a technique for combining of two colors allowing for transparency effects in computer graphics. The alpha is a level of opaqueness of the pixel. The value of alpha ranges from 0 to 255, where 0 represents a fully transparent color, and 255 represents a fully opaque color. The drawing below shows a picture of electronic circuit drawn over another image using different values of alpha.



All of the vector graphics, bitmaps and fonts are drawn according to the alpha set by [SetAlpha](#). Upon power-up, alpha is set to 255 (fully opaque).

Drawing Performance Impact

Rendering is almost 3 time slower when alpha is set to any value other than 255 or 0.

5.3 Pen

Vector Graphics are drawn using the Pen. Calling [ez.SetPenSize](#) allows setting of the pen height and width.

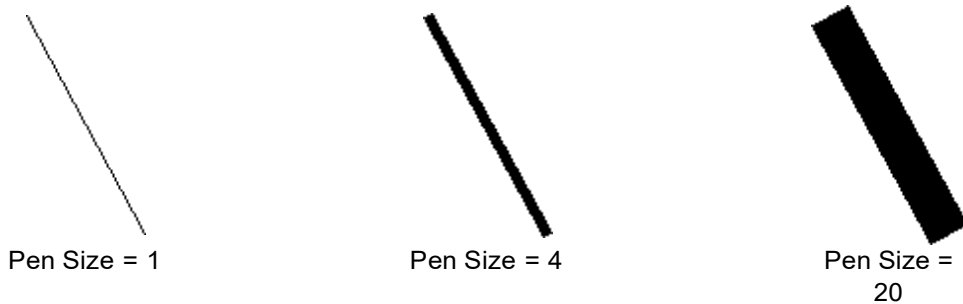
Pen Width specifies the horizontal dimension the drawing line (in pixels).

Pen Height specifies the vertical dimension of Pen (in pixels), when drawing curves. Note that the pen height is ignored when drawing straight lines.

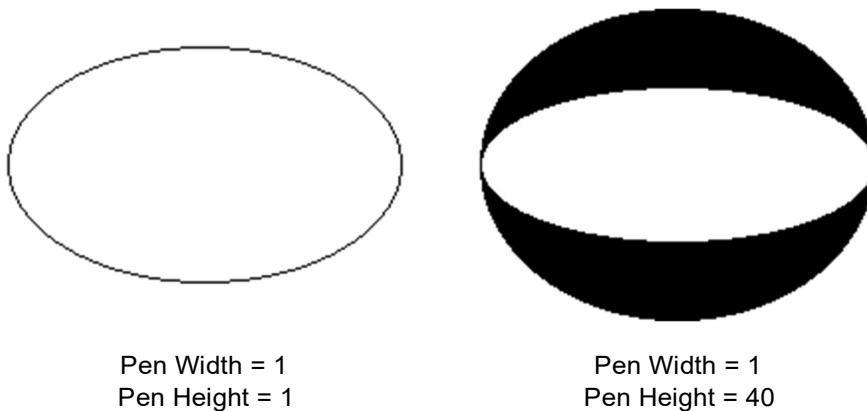
Notes:

1. Straight lines are not drawn when Pen Width is set to 0.
2. Curves are not drawn when either Pen Width or Height is set to 0.

The drawings below show a line drawn with different Pen Sizes



The drawings below show an ellipse drawn with different Pen Widths and Heights





Pen Width = 40
Pen Height = 4

6 Fonts

The ezLCD is capable of rendering 2 types of fonts:

1. [Bitmap Fonts](#)
2. [True Type Fonts](#) (Free Type Fonts)

The above font types have some advantages over one another. The table below describes some of them.

	Bitmap Font	True Type Font
Scalable	No	Yes
Anti-aliased Rendering	No	Yes
Full Unicode Support	No	Yes
Rotation Angle	0°, 90°, 180°, 270°	any angle
Rendering Speed	fast	medium to very slow
Small Font Rendering Quality	good	poor
Medium and Big Font Rendering Quality	acceptable	excellent
Max. No. of characters Per Font	256	65,536

While the ezLCD True Type fonts have a lot advantages, their rendering is much slower with the comparison to the speed in which the Bitmap Fonts are rendered. Also, the rendering quality is usually poor for the True Type Fonts with the height smaller than 16 pixels.

Note: Throughout this manual the term "True Type Fonts" is used interchangeably with "Free Type Fonts", "Open Type Fonts" and "Scalable Fonts". They all mean the same.

The drawing below shows rendered Bitmap Font (left) and True Type Font (right).

aa

The drawing below shows the same drawing as above, however magnified 8 times.



6.1 Bitmap Fonts

The ezLCD bitmap fonts reside in the User ROM, which is described in the "*ezLCD10x Manual*". They are created using ezLCDrom or ezLCDconfig utility and saved as .ezf files.

Note: Both ezLCDrom and ezLCDconfig utilities have been written for the other ezLCD products, however the .ezf files generated by them are compatible with the ezLCD. They can be downloaded from the support section of the <http://www.ezlcd.com>. In the nearest future, a special bitmap font utility will be developed for ezLCD.

Bitmap font files (.ezf) can be copied from the SD card to the User ROM by the ezLCD Executable: User.eze. The whole procedure is described in the "*ezLCD10x Manual*".

The rendering of Bitmap Fonts is affected by the following [Drawing Parameters](#):

- Current Position.
- Current Color.
- Background Color.
- Transparency

6.2 True Type Fonts

The ezLCD True Type Fonts can reside in the User ROM, which is described in the *"ezLCD10x Manual"*. Also, they can be dynamically loaded from the SD card. The True Type fonts are generally available as files with the extensions: .ttf and .otf.

The True Type Fonts can be copied from the SD card to the User ROM by the ezLCD Executable: User.eze. The whole procedure is described in the *"ezLCD10x Manual"*.

Acknowledgement: The True Type Fonts rendering software is based in part on the work of the FreeType Team (<http://www.freetype.org>). The ezLCD uses the FreeType 2 engine.

Note: Throughout this manual the term "True Type Fonts" is used interchangeably with "Free Type Fonts", "Open Type Fonts" and "Scalable Fonts". They all mean the same.

Rendering of the True Type Fonts is much slower than in the case of [Bitmap Fonts](#). There are significant differences in the speed in which the different True Type Fonts are rendered. Some of them are rendered quite fast, other: very slow. This means that the users should choose their fonts wisely. The ezLCD has a font cache mechanism, which significantly reduces rendering time of already used characters.

Quite often, the True Type Font contains a lot of regional characters, which may be of no use for the particular application. The font file size may be significantly reduced when such characters are removed from the .ttf file by using font editing software like, for example, FontCreator by High-Logic.

The rendering of True Type Fonts is affected by the following [Drawing Parameters](#):

- Current Position
- Current Color
- Transparency

7 Programming the ezLCD with Lua

This programming manual details the functions to manipulate the ezLCD series of intelligent touch screen displays using the Lua language. All ezLCD series of smart displays contain a Lua interpreter.

There is a pre-defined table (or library) named "ez" which must be prepended to all ezLCD functions so that the Lua interpreter knows to access these function from the ezLCD library.

An online version of the Lua programming manual can be found at <http://www.lua.org>

Numbers

While there is only 1 native numeric data type in Lua (double precision floating point), the ezLCD functions have "sanitized" numeric values. Where documented, the "Integer" data type, while not part of the Lua language are 32 bit whole numbers.

Images

Images are numbered from 0 to 255 and are cached in RAM. The cache is initially loaded via config.txt on the SDCard. The Advanced Cache Management commands may be used to load and/or unload images from RAM during program execution. Images may be in .bmp, .jpg, .ezp, or .ezh format

Lengths

By default, all lengths such as radii are in pixels.

Fonts

Fonts are numbered from 0 to 255 and are cached in RAM. The cache is initially loaded via config.txt on the SDCard. The Advanced Cache Management commands may be used to load and/or unload fonts from RAM during program execution. Fonts may be in bitmap (.ezf) or TrueType (.ttf) format.

Example: Create a button

```
ez.button(3, 1, 10, 11, -1, 15, 30)
```

This creates button ID 3 in the UP state (1) using image 10 in the USER ROM for the UP image and image 11 for the DOWN image at screen location (15,30)

8 Constants

Constant	Format	Description
ez.Width	Integer	Width of the screen in pixels
ez.Height	Integer	Height of the screen in pixels
ez.BytesPerPixel	Integer	Number of bytes per pixel
ez.FirmVer	String	ezLCD firmware version
ez.LuaVer	String	Lua version
ez.SerialNo	String	ezLCD Hardware Serial Number
ez.NoOfFrames	Integer	Number of available full-screen frames
ez.NoOfPicts	Integer	Number of pictures loaded in Cache
ez.NoOfBmFonts	Integer	Number of bitmap fonts loaded in Cache
ez.NoOfFtFonts	Integer	Number of true type fonts loaded in Cache

9 Position Functions

The following section details the functions used to manage the current screen position.

Screen Addresses

[Screen Coordinates](#) and object sizes (e.g button height and width) are integers and specified in pixels

9.1 `ez.SetXY(x,y)`**Purpose**

To set the current position to the specified (x, y) location.

Argument List

x	Integer	new current x screen location
y	Integer	new current y screen location

Return Value

None

Reference

[Screen Coordinates](#)

9.2 ez.SetX(x)

Purpose

To set the current x position to the specified x location. The current y location remains unaffected.

Argument List

x	Integer	new current x screen location
---	---------	-------------------------------

Return Value

None

Reference

[Screen Coordinates](#)

9.3 ez.SetY(y)**Purpose**

To set the current y position to the specified y location. The current x location remains unaffected.

Argument List

y	Integer	new current y screen location
---	---------	-------------------------------

Return Value

None

Reference

[Screen Coordinates](#)

9.4 ez.GetX()**Purpose**

To return the current x location.

Argument List

None

Return Value

x	Integer	The current x location
---	---------	------------------------

Reference

[Screen Coordinates](#)

9.5 ez.GetY()**Purpose**

To return the current y location.

Argument List

None

Return Value

y	Integer	The current y location
---	---------	------------------------

Reference

[Screen Coordinates](#)

10 Color Functions

The following sections detail the functions used to affect drawing colors.

The ezLCD display supports 24 bit color. Colors consist of 8 bits of red, green and blue. Where colors are specified as integers, only the low 8 bits (0 - 255) of red, green or blue are used.

10.1 ez.RGB(red, green, blue)**Purpose**

To get the ezLCD color value that corresponds to the specified RGB color values

Argument List

red	Integer	Red component (0 - FF hex)
green	Integer	Green component (0 - FF hex)
blue	Integer	Blue component (0 - FF hex)

Return Value

ezLCDcolor	Integer	ezLCD color value (24 bit) for specified red, green and blue values
------------	---------	---

Notes

Only the low 8 bits of red, green or blue are used.

10.2 ez.GetRed(ezLCDcolor)**Purpose**

To get the Red component of the ezLCD color.

Argument List

ezLCDcolor	Integer	ezLCD color value
------------	---------	-------------------

Return Value

red	Integer	Red component of the specified color
-----	---------	--------------------------------------

10.3 ez.GetGreen(exLCDcolor)**Purpose**

To get the Green component of the ezLCD color.

Argument List

ezLCDcolor	Integer	ezLCD color value
------------	---------	-------------------

Return Value

green	Integer	Green component of the specified color
-------	---------	--

10.4 ez.GetBlue(ezLCDcolor)**Purpose**

To get the RBlueed component of the ezLCD color.

Argument List

ezLCDcolor	Integer	ezLCD color value
------------	---------	-------------------

Return Value

blue	Integer	Blue component of the specified color
------	---------	---------------------------------------

10.5 ez.SetColor(ezLCDcolor)**Purpose**

To set the current color.

Argument List

ezLCDcolor	Integer	ezLCD color code
------------	---------	------------------

Return Value

None

10.6 ez.SetBgColor(ezLCDcolor)**Purpose**

To set the background color.

Argument List

ezLCDcolor	Integer	ezLCD color code
------------	---------	------------------

Return Value

None

10.7 ez.ReplaceColor(x, y, width, height, OldColor,NewColor)**Purpose**

To replace OldColor with NewColor in the specified rectangle.

Argument List

x	Integer	x position of start of rectangle to be affected
y	Integer	y position of start of rectangle to be affected
width	Integer	width of rectangle to be affected
height	Integer	height of rectangle to be affected
OldColor	Integer	ezLCD color value to be replaced
NewColor	Integer	ezLCD color value to be written

Return Value

None

10.8 ez.GetPixel([x,y])**Purpose**

To get the color at the specified screen position.

Argument List

x	Integer	x screen position (default=current position)
y	Integer	y screen position (default=current position)

Return Value

color	Integer	ezLCD color at specified screen position.
-------	---------	---

11 Transparency Functions

The following sections detail the functions used to affect drawing transparency.

11.1 ez.SetAlpha(alpha)

Purpose

To set the value of the transparency alpha. The ezLCD supports transparency by alpha-blending of the pixel being drawn with the background pixel at the particular position. Alpha blending is a technique for combining of two colors allowing for transparency effects in computer graphics. The alpha is a level of opaqueness of the pixel. The value of alpha ranges from 0 to 255, where 0 represents a fully transparent color, and 255 represents a fully opaque color.

Argument List

Alpha	Integer	transparency alpha (0 - 255)
-------	---------	------------------------------

Return Value

None

Notes

Renderings are nearly 3 times slower when Alpha is not 0 or 255. Alpha is 255 by default.

Example

The drawing below shows a picture of electronic circuit drawn over another image using different values of alpha.



Reference

[Transparency](#)

11.2 ez.TrColorNone()**Purpose**

To unset the transparency color to be used when drawing bitmaps. If a call was made to set a bitmap transparency color, use this function to unset that color. This is the system default value.

Argument List

None

Return Value

None

Reference

[Transparency](#)

11.3 ez.SetTrColor(ezLCDcolor)

Purpose

To specify the transparency color to be used when drawing bitmaps. When drawing a bitmap, any color that is the same as TrColor will not be written.

Argument List

ezLCDcolor	Integer	ezLCD color code of bitmap transparency color
------------	---------	---

Return Value

None

Reference

[Transparency](#)

12 Pen Size Functions

The following sections detail the functions used to affect the drawing pen.

12.1 ez.SetPenSize(height, width)**Purpose**

To set the height and width of the drawing pen. This is used in the drawing of vector graphics.

Argument List

height	Integer	pen height
width	Integer	pen width

Return Value

None

Reference

[Pen](#)

13 Angle Functions

The following section details the functions to convert between degrees, radians, and ezLCD angle values.

ezLCD Angles

Angles are orientated clockwise with 0 degrees as straight up (North).

There are 4000 hex (16384 decimal) ezLCD angle units in a full circle, therefore 1 degree is approximately 45.51 units ($16384/360$).

45 degrees would be specified as $45 \times 45.51 = 2048$ decimal (800 hex).

13.1 ez.Deg(degrees)**Purpose**

To get the ezLCD angle value that corresponds to the specified degrees.

Argument List

degrees	Integer	angle specified in degrees
---------	---------	----------------------------

Return Value

ezLCDAngle units	Integer	ezLCD angle units value that corresponds to the specified degrees
------------------	---------	---

13.2 ez.Rad(radians)**Purpose**

To get the ezLCD angle value that corresponds to the specified radians.

Argument List

radians	real	angle specified in radians
---------	------	----------------------------

Return Value

ezLCDAngle units	Integer	ezLCD angle units value that corresponds to the specified radians.
------------------	---------	--

Additional Reference

None

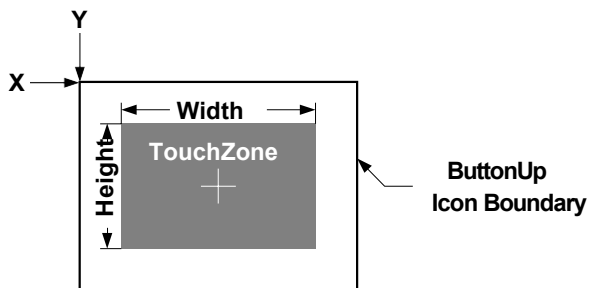
14 Button Functions

The following sections detail the functions used to create buttons, manage button states and process button events.

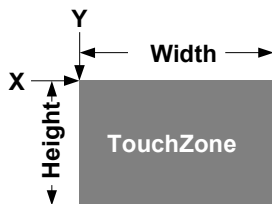
About the Touch Zone:

- The Touch Zone is the active touch response area of the button. Its size is specified by Width and Height .
- If the Button Up Icon is defined (not -1), the Touch Zone is centered on it.
- If the Button Up Icon is none (-1), the position of the upper-left corner of the Touch Zone is specified by X and Y.

Both cases are shown in the drawings below:



Button Up Icon is defined (not = -1)



Button Up Icon is none (= -1)

Overview

The Button functions are used to define a button, change a button state or declare an event handler to be called when a button is pressed. The button state can only be changed programmatically via the button state change function [Button\(iD, iState\)](#).

Button State

The button state is an integer and may be any one of the following values:

UP	1	The button is in the up state
DOWN	2	The button is in the down state
DISABLED	3	The button is visible but pressing it will not affect the button state
NON-VISIBLE	4	The button is hidden and pressing it will not affect the button state
DELETE	5	Delete the button

Button Event

The button event is an integer and may be any one of the following values:

UP	1	The button was released
DOWN	2	The button was pressed

14.1 ez.Button(ID, iState, iconUp, iconDown, iconDisabled, x, y [, width, height])**Purpose**

Define a button

Argument List

ID	Integer	The ID of the created button (0 to 63)
iState	Integer	The initial button state (1: Up, 2: Down, 3: Disabled, 4: Non-Visible, 5: Delete)
iconUp	Integer	The Image Number to be displayed when the button state is UP (1). Specify -1 or FFFF hex for no icon.
iconDown	Integer	The Image Number to be displayed when the button state is DOWN (2). Specify -1 or FFFF hex for no icon.
iconDisabled	Integer	The Image Number to be displayed when the button state is DISABLED (3). Specify -1 or FFFF hex for no icon.
x	Integer	The x (horizontal/left) position where the button and icon start in pixels.
y	Integer	The y (vertical/top) position where the button and icon start in pixels.
width	Integer	The width of the button touch area in pixels. (default=width of iconUp)
height	Integer	The height of the button touch area in pixels. (default=height of iconUp)

Return Value

Success	Boolean	Function successful (TRUE or FALSE)
---------	---------	-------------------------------------

Notes

- The Button is deleted if iState = 5
- If Width and Height are not specified and IconUp is -1 or invalid, no button will be created

14.2 ez.Button(iD,iState)**Function**

Button(ID, iState)

Purpose

To change the state of a button

Argument List

ID	Integer	The ID of the button to affect (0 to 63)
iState	Integer	The new button state (1: Up, 2: Down, 3: Disabled, 4: Non-Visible, 5: Delete)

Return Value

Success	Boolean	Function successful (TRUE or FALSE)
---------	---------	-------------------------------------

14.3 ez.DelButtons()

Purpose

Delete all buttons

Argument List

None

Return Value

None

14.4 ez.SetButtonEvent(sButtonHandler)**Purpose**

To declare the function to be called when a button event occurs.

Argument List

sButtonHandler	String	Name of the button event handler function
----------------	--------	---

Return Value

None

Notes

The handler function (sButtonHandler) will be called asynchronously whenever a button is pressed or released. The handler must be declared to have 2 arguments as follows:

sButtonHandler(ID, iEvent)

ID	Integer	The ID of the button that caused the event.
iEvent	Integer	The button event that occurred (1 = UP or 2 = DOWN).

Example

```
-- Define the Button Event Handler
function ProcessButtons(id, event)
    -- TODO: Insert your button processing code here

    -- Display the image which corresponds to the event
    ez.Button(id, event)
end

-- Main Program - define a few sample buttons
ez.Button(0, 1, 0, 1, -1, 10, 10, 50, 50)
ez.Button(1, 1, 2, 3, -1, 60, 10, 50, 50)
ez.Button(2, 1, 4, 5, -1, 10, 60, 50, 50)
ez.Button(3, 1, 6, 7, -1, 60, 60, 50, 50)

-- Start to receive button events
ez.SetButtonEvent("ProcessButtons")

-- Infinite loop to stay in Lua
while true do
end
```


15 Fill Area Functions

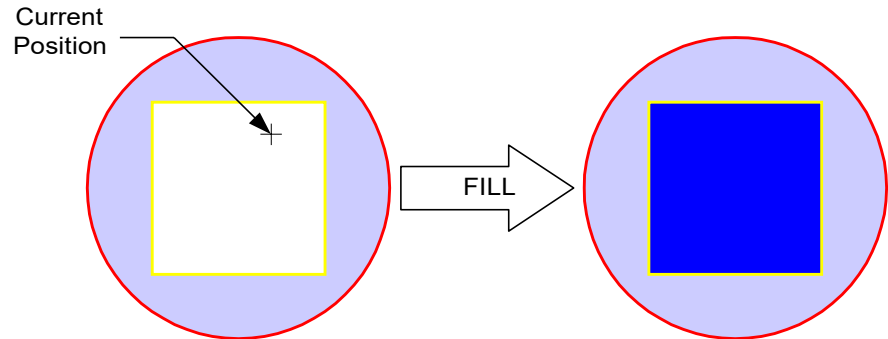
The following section details the functions used to fill screen areas on the ezLCD display

Overview

Fill Functions

The Fill functions will change the pixel at the start position and all adjoining pixels of the same color to a new color.

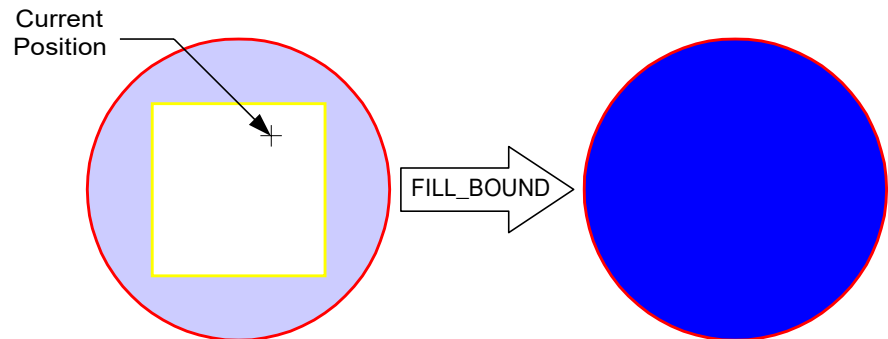
In the following example, the start position is located in the white area and the fill color is blue.



FillBound Functions

The FillBound functions will change the pixel at the start position and all adjoining pixels bounded by the bound color to a new color.

In the following example, the start position is located in the white area (although could be in the yellow area also) with red as the bound color and the fill color is blue.



15.1 ez.Cls([ezLCDcolor])**Purpose**

To clear the entire screen by writing the specified color to the screen.

Argument List

ezLCDcolor	Integer	ezLCD color code (default=foreground color)
------------	---------	--

Return Value

None

15.2 ez.Fill([x,y,] [FillColor])**Purpose**

To fill an area with the specified color. The Fill area begins at the (x, y) position specified and includes all adjoining pixels that are the same color as the existing (pre-filled) color.

Argument List

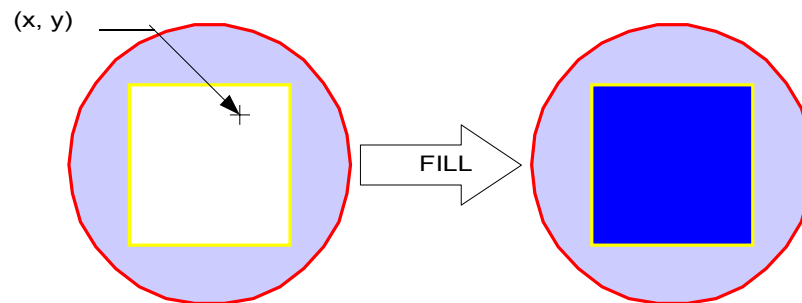
x	Integer	x screen position where fill is to begin (default=current position)
y	Integer	y screen position where fill is to begin (default=current position)
FillColor	Integer	ezLCDcolor code (default=foreground color)

Return Value

None

Example

In the following example, the initial position could be anywhere in the white area and the fill color is blue.



15.3 ez.FillBound([x, y,] BoundColor [,FillColor])**Purpose**

To fill an area of the screen with the specified FillColor. The Fill area is defined as the specified (x, y) position and includes all adjoining pixels, bounded by the pixels that are the same color as the BoundColor.

Argument List

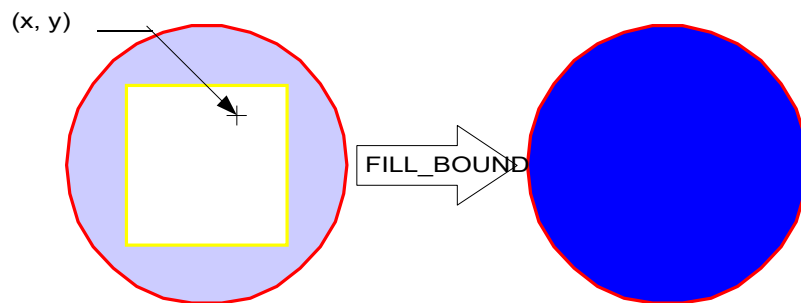
x	Integer	x screen position where fill is to begin (default=current position)
y	Integer	y screen position where fill is to begin (default=current position)
BoundColor	Integer	ezLCD color used to define the bounding area
FillColor	Integer	ezLCD color to be used for filling (default=foreground color)

Return Value

None

Example

In the following example, the initial position could be anywhere inside the red circle and the fill color is blue.



16 Line Drawing Functions

The following section details the functions used to draw lines.

The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

16.1 ez.HLine([x1, y1,] x2 [,color])**Purpose**

To draw a horizontal line using the specified color from the specified position (x1, y1) to the the position (x2, y1). The line size is defined by the width setting of the pen which is configured in [ez.SetPenSize](#).

Argument List

x1	Integer	x position where to start the line.
y1	Integer	y position where to start the line.
x2	Integer	x position where drawn line ends.
color	Integer	ezLCD color value of line to draw

Return Value

None

16.2 ez.VLine([x1, y1,] y2 [,color])**Purpose**

To draw a vertical line using the specified color from the specified position (x1, y1) to the the position (x1, y2). The line size is defined by the width setting of the pen which is configured in [ez.SetPenSize](#).

Argument List

x1	Integer	x position where to start the line. (default=current position)
y1	Integer	y position where to start the line. (default=current position)
y2	Integer	y position where drawn line ends.
color	Integer	ezLCD color value of line to draw (default=foreground color)

Return Value

None

16.3 ez.Line([x1, y1,] x2, y2 [,color])**Purpose**

To draw a line using the specified color from the specified position (x1, y1) to the specified position (x2, y2).

Argument List

x1	Integer	x position where to start the line. (default=current position)
y1	Integer	y position where to start the line. (default=current position)
x2	Integer	x position where drawn line ends.
y2	Integer	y position where drawn line ends.
color	Integer	ezLCD color value of line to draw (default=foreground color)

Return Value

None

Notes

The line height and width are set by the [ez.SetPenSize](#) function

16.4 ez.LineAng([x1, y1,] angle, length [, color])**Purpose**

To draw a line using the specified color from the specified position (x1, y1) at the angle specified in [ezLCD Angles](#) for the specified length.

Argument List

x1	Integer	line x start position (default=current position)
y1	Integer	line y start position (default=current position)
angle	Integer	angle to draw line in ezLCD angle units.
length	Integer	length of line to be drawn in pixels.
color	Integer	ezLCD color to be used to draw line. (default=foreground color)

Return Value

None

Notes

The line height and width are set by the [ez.SetPenSize](#) function

17 Curve Drawing Functions

The following section details the functions used to draw curves.

The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

17.1 ez.Circle([x, y,] radius [, color])**Purpose**

To draw a circle using the specified color centered at the specified position with a radius of radius. The thickness of the drawn line is computed using the pen width and height which is configured in [ez.SetPenSize](#).

Argument List

x	Integer	x location of circle center (default=current position)
y	Integer	y location of circle center (default=current position)
radius	real	radius of circle.
color	Integer	ezLCD color (default=foreground color)

Return Value

None

17.2 ez.CircleFill([x,y,]radius [,color])**Purpose**

To draw a filled circle using the specified color centered at the specified position with the specified radius.

Argument List

x	Integer	x location of circle center (default=current position)
y	Integer	y location of circle center (default=current position)
radius	real	radius of circle.
color	Integer	ezLCD color (default=foreground color)

Return Value

None

17.3 ez.Ellipse([x, y,] a, b [, color])**Purpose**

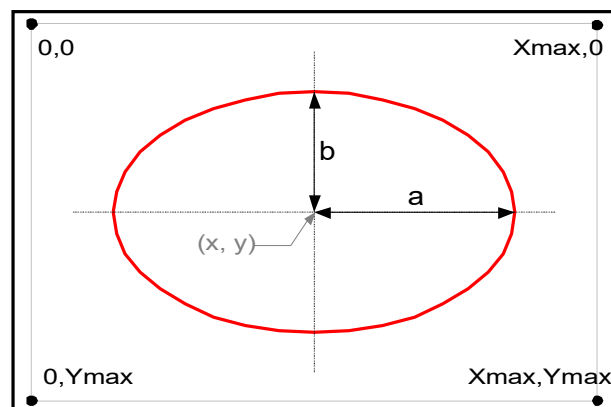
To draw an ellipse using the specified color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b. The thickness of the drawn line is computed using the pen width and height which is configured in [ez.SetPenSize](#).

Argument List

x	Integer	x position of ellipse center. (default=current position)
y	Integer	y position of ellipse center. (default=current position)
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color of ellipse. (default=foreground color)

Return Value

None



17.4 ez.EllipseFill([x, y,] a, b [, color])**Purpose**

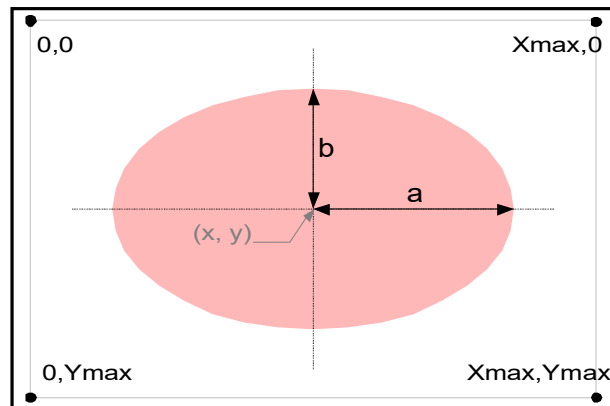
To draw a filled ellipse using the specified color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

x	Integer	x position of ellipse center. (default=current position)
y	Integer	y position of ellipse center. (default=current position)
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color of ellipse. (default=foreground color)

Return Value

None



17.5 ez.Arc([x, y,] radius, StartAng, EndAng [, color])

Purpose

To draw an arc using the specified color centered at the specified position with the specified radius from angle StartAng to EndAng specified in [ezLCD angle units](#). The thickness of the drawn line is computed using the pen width and height which is configured in [ez.SetPenSize](#).

Argument List

x	Integer	x location of arc center
y	Integer	y location of arc center
radius	Integer	radius of arc.
StartAng	Integer	Angle to begin arc at specified in ezLCD angle units.
EndAng	Integer	Angle to end arc at specified in ezLCD angle units.
color	Integer	ezLCD color

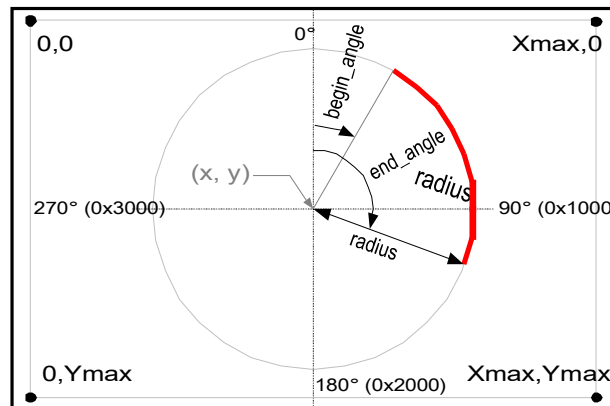
Return Value

None

Notes

Angles are orientated clockwise with 0 degrees as straight up (North).

There are 4000 hex (16384 decimal) ezLCD angle units in a full circle, therefore 1 degree is 45.51 units. For example, 45 degrees would be specified as $45 \times 45.51 = 2048$ decimal (800 hex).



17.6 ez.Pie([x, y,] radius, StartAng, EndAng [, color])**Purpose**

To draw a pie filled arc using the specified color centered at the specified position with the specified radius from angle StartAng to EndAng specified in [ezLCD angle units](#).

Argument List

x	Integer	x location of pie center (default=current position)
y	Integer	y location of pie center (default=current position)
radius	real	radius of pie.
StartAng	Integer	Angle to begin arc at specified in ezLCD angle units.
EndAng	Integer	Angle to end arc at specified in ezLCD angle units.
color	Integer	ezLCD color (default-foreground color)

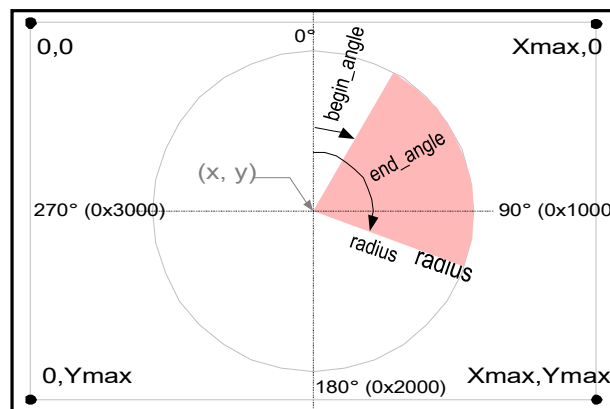
Return Value

None

Notes

Angles are orientated clockwise with 0 degrees as straight up (North).

There are 4000 hex (16384 decimal) ezLCD angle units in a full circle, therefore 1 degree is 45.51 units. For example, 45 degrees would be specified as 45 x 45.51 = 2048 decimal (800 hex).



17.7 ez.EllipseArc([x, y,] a, b, StartAng, EndAng [, color])

Purpose

To draw an Ellipse arc using the specified color centered at the specified position from angle StartAng to EndAng specified in [ezLCD Angles](#) where the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b. The thickness of the drawn line is computed using the pen width and height which is configured in [ez.SetPenSize](#).

Argument List

x	Integer	x location of ellipse center (default=current position)
y	Integer	y location of ellipse center (default=current position)
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
StartAng	Integer	Angle to begin arc at specified in ezLCD angle units.
EndAng	Integer	Angle to end arc at specified in ezLCD angle units.
color	Integer	ezLCD color (default=foreground color)

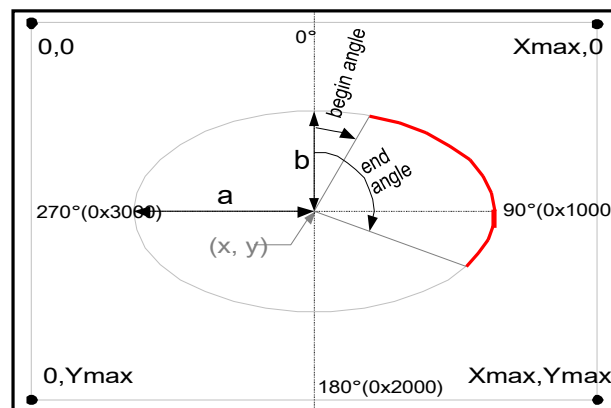
Return Value

None

Notes

Angles are orientated clockwise with 0 degrees as straight up (North).

There are 4000 hex (16384 decimal) ezLCD angle units in a full circle, therefore 1 degree is 45.51 units. For example, 45 degrees would be specified as 45 x 45.51 = 2048 decimal (800 hex).



17.8 ez.EllipsePie([x, y,] a, b, StartAng, EndAng [, color])**Purpose**

To draw an Ellipse pie using the specified color centered at the specified position from angle StartAng to EndAng specified in [ezLCD Angles](#) where the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

x	Integer	x location of ellipse center (default=current position)
y	Integer	y location of ellipse center (default=current position)
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
StartAng	Integer	Angle to begin arc at specified in ezLCD angle units.
EndAng	Integer	Angle to end arc at specified in ezLCD angle units.
color	Integer	ezLCD color (default=current foreground color)

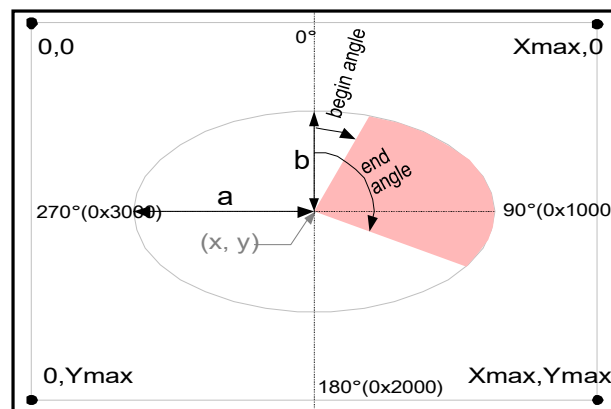
Return Value

None

Notes

Angles are orientated clockwise with 0 degrees as straight up (North).

There are 4000 hex (16384 decimal) ezLCD angle units in a full circle, therefore 1 degree is 45.51 units. For example, 45 degrees would be specified as 45 x 45.51 = 2048 decimal (800 hex).



18 Polygon Drawing Functions

The following section details the functions used to draw polygons.

18.1 ez.Box([x1, y1,] x2, y2 [,color])**Purpose**

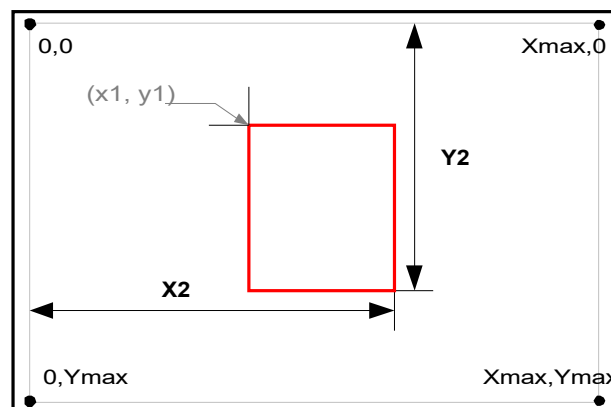
To draw a box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color and the line size will be the current pen width.

Argument List

x1	Integer	x screen position of starting corner (default=current position)
y1	Integer	y screen position of starting corner (default=current position)
x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner
color	Integer	ezLCD color to use for drawing box (default=foreground color)

Return Value

None

**Additional Reference**

[ez.SetPenSize](#)

18.2 ez.BoxFill([x1, y1,] x2, y2 [, color])**Purpose**

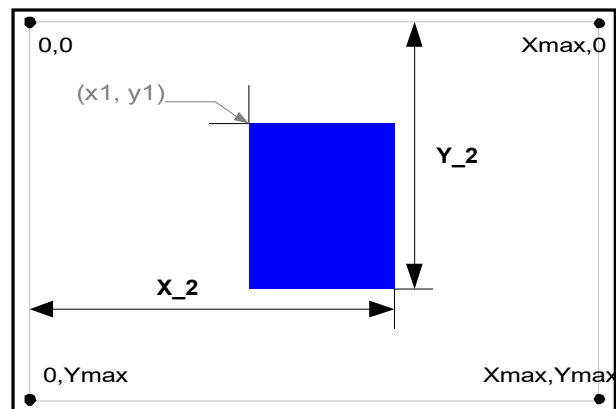
To draw a filled in box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color.

Argument List

x1	Integer	x screen position of starting corner (default=current position)
y1	Integer	y screen position of starting corner (default=current position)
x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner
color	Integer	ezLCD color to use for drawing box (default=foreground color)

Return Value

None



18.3 ez.Polygon(x1, y1, x2, y2, ... xn, yn)

Purpose

To draw a Polygon using the specified vertices list (x1, y1), (x2, y2), ... (xn, yn). The polygon will be filled using the specified color.

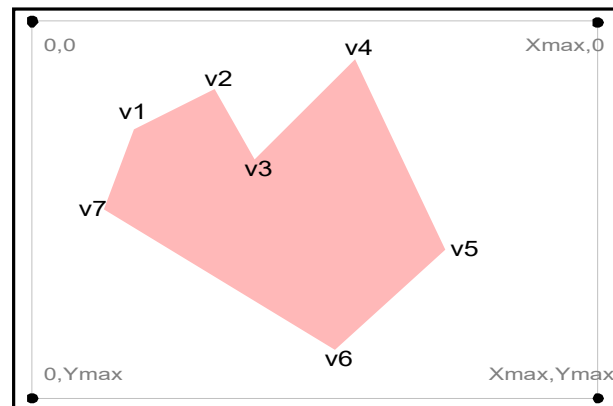
Argument List

x1	Integer	x screen position of vertex 1
y1	Integer	y screen position of vertex 1
x2	Integer	x screen position of vertex 2
y2	Integer	y screen position of vertex 2
xn	Integer	x screen position of vertex n
yn	Integer	y screen position of vertex n

Return Value

None

Example



19 Single Pixel Functions

The following section details the functions used to single pixel operations.

19.1 ez.Plot([x, y [,PlotColor]])**Purpose**

To draw a pixel at the specified location using the specified color.

Argument List

x	Integer	x screen position (default=current position)
y	Integer	y screen position (default=current position)
PlotColor	Integer	ezLCD color (default=foreground color)

Return Value

None

19.2 ez.GetPixel([x, y])**Purpose**

To get the ezLCD color value at the specified location

Argument List

x	Integer	x screen position (default=current position)
y	Integer	y screen position (default=current position)

Return Value

ezLCDcolor	Integer	The ezLCD color value at the current location
------------	---------	---

20 Font Functions

The following section details the functions used to manipulate fonts.

Use the native Lua [Print](#) function to print strings on the ezLCD display.

20.1 ez.SetBmFont(BitmapFontNo)**Purpose**

Sets the current font to the specified bitmap font from the user ROM

Argument List

BitmapFontNo	Integer	bitmap font number
--------------	---------	--------------------

Return Value

Success	Boolean	true if bitmap font number exists in the user ROM.
---------	---------	--

20.2 ez.SetFtFont(FtFontNo, height, width)**Purpose**

Sets the current font to the specified Free Type (TrueType) font from the user ROM

Argument List

FtFontNo	Integer	FreeType font number
height	Integer	FreeType font height in pixels (1 - 255)
width	Integer	FreeType font width in pixels (0 - 255). A value of zero will cause the width to be computed automatically.

Return Value

Success	Boolean	true if FreeType font number exists in the user ROM.
---------	---------	--

Notes

Rendering is faster when width is computed automatically (set to 0).

20.3 ez.GetNoOfBmFonts()**Purpose**

Gets the number of bitmap fonts in the user ROM

Argument List

None

Return Value

Count	Integer	number of bitmaps fonts in the user ROM.
-------	---------	--

20.4 ez.GetNoOfFtFonts()**Purpose**

Gets the number of Free Type (TrueType) fonts in the user ROM

Argument List

None

Return Value

Count	Integer	number of FreeType fonts in the user ROM.
-------	---------	---

20.5 ez.CacheFtChars(StartChar, EndChar)**Purpose**

Caches the specified character number range from the current Free Type (TrueType) font. This will cause the characters to be rendered approximately 100 times faster.

Argument List

StartChar	Integer	first unicode character number to be cached
EndChar	Integer	last unicode character number to be cached

Return Value

None

Notes

Characters are cached on first use. By using this function, you can pre-cache the specified character range so that they are rendered at the same speed every time.

Font Cache Details

- Holds the bitmap glyphs of the characters.
- Only the True Type characters are cached.
- Cache memory is dynamically allocated. Characters are not cached when there is no memory left.
- Each time the True Type font (or its size) is changed, the font cache is cleared.

20.6 ez.SetFtUnibase(UnicodeBase)**Purpose**

Sets the Free Type (True Type) base page to the specified Unicode page.

Argument List

UnicodeBase	Integer	FreeType font Unicode base page number
-------------	---------	--

Return Value

None

21 Text Orientation Functions

The following section details the functions used to set the TrueType font character orientation.

Use the native Lua [Print](#) function to print strings on the ezLCD display.

21.1 ez.TextNorth()

Purpose

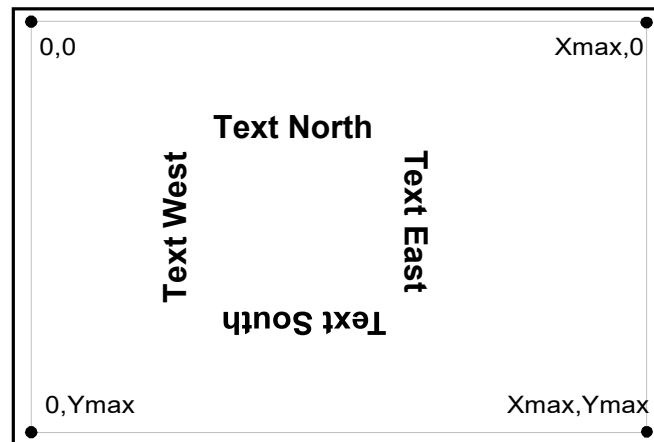
Sets the text orientation to North

Argument List

None

Return Value

None



21.2 ez.TextEast()**Purpose**

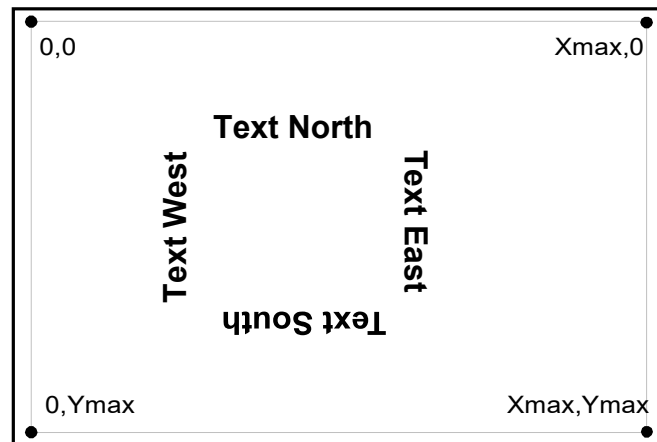
Sets the text orientation to East

Argument List

None

Return Value

None



21.3 ez.TextSouth()

Purpose

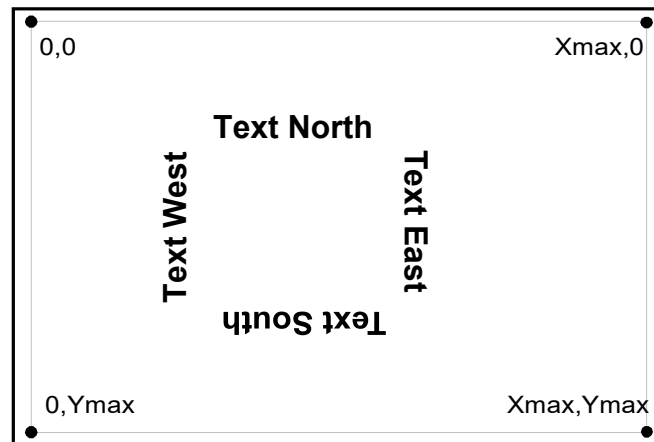
Sets the text orientation to South

Argument List

None

Return Value

None



21.4 ez.TextWest()**Purpose**

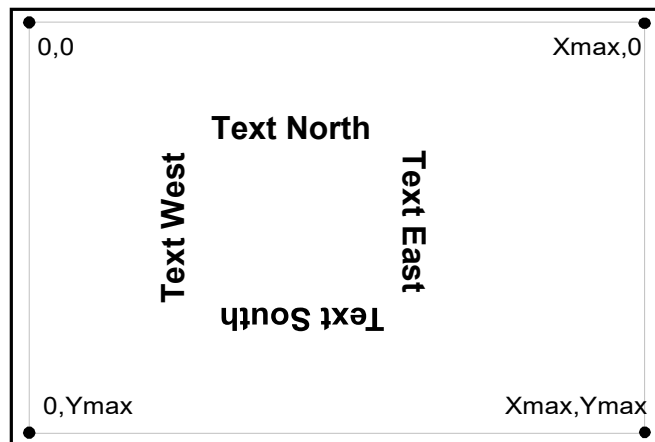
Sets the text orientation to West

Argument List

None

Return Value

None



21.5 ez.SetFtAngle(Angle)**Purpose**

Sets the angle to draw Free Type (True Type) Font characters

Argument List

Angle	Integer	Angle specified in ezLCD Angle Units
-------	---------	--

Return Value

None

22 Bitmap Functions

The following section details the functions used to display a bitmap.

22.1 ez.PutPictNo([x, y,] PictNo)

NOTE: Legacy deprecated command. Recommended Replacement: ez.PutPictFile

Purpose

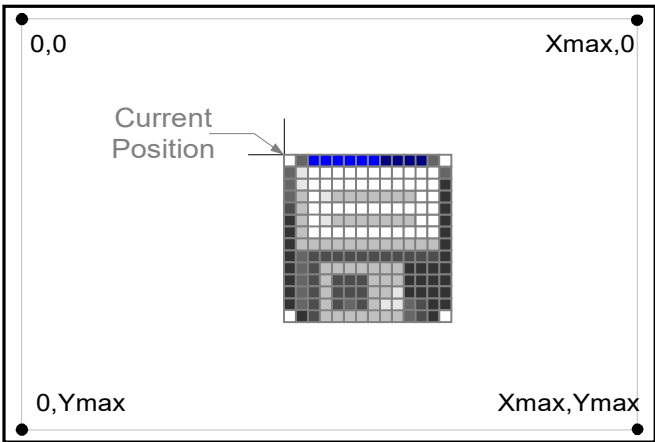
Display the specified bitmap with upper left corner being positioned at the specified position.

Argument List

x	Integer	x position to display bitmap (default=current position)
y	Integer	y position to display bitmap (default=current position)
PictNo	Integer	image number from config.txt

Return Value

Success	Boolean	true if image number exists
---------	---------	-----------------------------



22.2 ez.PutPictFile([x, y,] filename)**Purpose**

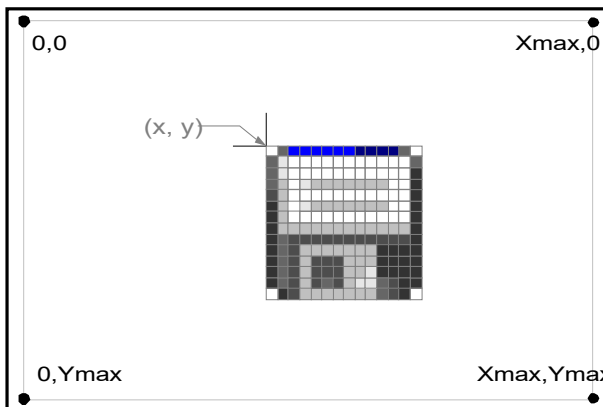
Display the specified image with upper left corner being positioned at the specified position. Supports JPG, BMP, EZH, and EZIP images. JPG is recommended

Argument List

x	Integer	x position to display image (default=current position)
y	Integer	y position to display image (default=current position)
filename	string	filename of image to display

Return Value

Success	Boolean	true if the image was successfully displayed.
---------	---------	---



22.3 ez.GetPictHeight(PictNo or Filename)

NOTE: PictNo variation is deprecated.

Purpose

Return the height in pixels of the specified image in config.txt

Argument List

PictNo	Integer	image number in config.txt
Filename	String	filename of image

Return Value

Height	Integer	image of bitmap in pixels
--------	---------	---------------------------

22.4 ez.GetPictWidth(PictNo or filename)

NOTE: PictNo variation is deprecated.

Purpose

Return the width in pixels of the specified image in config.txt

Argument List

PictNo	Integer	image number in config.txt
filename	String	Filename of image

Return Value

Width	Integer	width of image in pixels
-------	---------	--------------------------

23 QRCode Function

The following command permits display of industry-standard QR Codes.

QRCode Library credits:

Copyright (c) Project Nayuki. (MIT License)
<https://www.nayuki.io/page/qr-code-generator-library>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
- The Software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the Software or the use or other dealings in the Software.

23.1 ez.PutQRCode(text [, ecl [, minVer [, maxVer [, mask [, boostEcl]]]])**Purpose**

Display an industry-standard QRCode on the screen at the current cursor position, using the current foreground color

Argument List

text	string	Text to display
ecl	Integer	Error Correction Level: 0=7%, 1=15% (default), 2=25%, 3=30%
minVer	Integer	QRCode Min Version (size) Range=1-40 (default=1)
maxVer	Integer	QRCode Max Version (size) Range=1-40 (default=40)
mask	Integer	Mask Pattern Range=-1, 0-7 Default=-1 (auto)
boostEcl	integer	0=no boost; 1=If larger size is possible without version increase, make larger (default)

Return Value

result	Integer	0 if Success
--------	---------	--------------

Notes

- For further details on QRCode see https://en.wikipedia.org/wiki/QR_code

Example

The following code prints the QR Code for "Hello World" on the screen

```
ez.SetXY(100,100)
ez.PutQRCode("Hello World")
```

24 Backlight Functions

The following section details the functions used to affect the screen backlight.

24.1 ez.LightOn()**Purpose**

To turn on the screen backlight.

Argument List

None

Return Value

None

24.2 ez.LightOff()

Purpose

To turn off the screen backlight.

Argument List

None

Return Value

None

24.3 ez.LightBright(brightness)**Purpose**

To set the screen backlight level

Argument List

brightness	Integer	brightness value between 0 - 255
------------	---------	----------------------------------

Return Value

None

25 Screen Capture Functions

The following section details the functions used to capture the screen contents.

25.1 ez.SdScreenCapture()**Purpose**

Saves an image of the screen to the SD card as a bitmap file (.bmp) in the top level folder named Scr_Cap. Files are named scr_xxxx.bmp where xxxx is a 4 digit number starting at scr_0001.bmp.

Argument List

None

Return Value

Success	boolean	true = image was successfully saved.
---------	---------	--------------------------------------

26 Time Functions

The following section details the system time functions.

26.1 ez.Get_ms()**Purpose**

Gets the number of milliseconds since system power on.

Argument List

None

Return Value

ms	Integer	milliseconds since system power on.
----	---------	-------------------------------------

26.2 ez.Wait_ms(ms)**Purpose**

Pauses the program specified number of milliseconds.

Argument List

ms	Integer	number of milliseconds to pause.
----	---------	----------------------------------

Return Value

None

26.3 ez.SetTime(hour, min, sec, month, day, year)**Purpose**

Set the system time.

Argument List

hour	Integer	range=0-23
min	Integer	range=0-59
sec	Integer	range=0-59
month	Integer	range=1-12
day	Integer	range=1-31
year	Integer	range=2000-2099

Return Value

result	Integer	0 if Success
--------	---------	--------------

Notes

- ezLCD hardware does not support daylight savings time or time zones directly. But the developer can implement them manually in their code if needed
- The clock on ezLCD can be battery backed-up by connecting a battery to the proper terminals and changing the battery select jumper to the battery position. See model-specific hardware documentation for details.
- The clock may be read via the standard Lua **os.time()** and **os.date()** functions. Please refer to the standard Lua documentation for details: <https://www.lua.org/pil/22.1.html>

Example

The following code sets the system time to July 3, 2008, 3:14pm

```
-- Set system time
ez.SetTime(15, 14, 0, 7, 3, 2008)
```


27 Timer Management Functions

The following section details the functions used to generate an asynchronous timer event.

Up to 16 timers may be active at any one time.

27.1 ez.Timer(msec, LuaTimerFunc, Id)**Purpose**

Sets or resets a timer to execute the specified function after the specified time elapses.

Argument List

msec	Integer	number of ms between each call to LuaTimerFunc
LuaTimerFunc	String	name of function to execute
Id	Integer	Id of the timer (0 - 15)

Return Value

Success	Boolean	true on success
---------	---------	-----------------

27.2 ez.Timer(msec, LuaTimerFunc)**Purpose**

Sets a timer to execute the specified function after the specified time elapses.

Argument List

msec	Integer	number of ms between each call to LuaTimerFunc.
LuaTimerFunc	String	name of function to execute

Return Value

TimerId	Integer	Timer ID of newly created timer (0 - 15) or -1 if no timer ID's are available.
---------	---------	--

27.3 ez.TimerStart(Id)**Purpose**

Restarts the specified timer.

Argument List

Id	Integer	Id of the timer (0 - 15)
----	---------	--------------------------

Return Value

None

Notes

The delay time will be the last delay period that the specified timer was set to.

27.4 ez.TimerStop(Id)**Purpose**

Stops the specified timer.

Argument List

Id	Integer	Id of the timer (0 - 15)
----	---------	--------------------------

Return Value

None

28 Touch Functions

The following section details the functions used to manage screen touches.

28.1 ez.GetTouchX()**Purpose**

Return the x position of the last screen touch.

Argument List

None

Return Value

x	Integer	x position of the last screen touch
---	---------	-------------------------------------

28.2 ez.GetTouchY()**Purpose**

Return the y position of the last screen touch.

Argument List

None

Return Value

y	Integer	y position of the last screen touch
---	---------	-------------------------------------

28.3 ez.TouchDn()**Purpose**

To determine if the screen is currently being pressed.

Argument List

None

Return Value

press	Boolean	true = screen is pressed, false = not pressed
-------	---------	---

28.4 ez.SetTouchEvent(luaTouchFunc)

Purpose

To set up an event handler to be called when the screen is pressed or released.

Argument List

luaTouchFunc	String	Function to be called when screen is pressed or nil to disable this event.
--------------	--------	--

Return Value

None

Notes

SetTouchEvent (**nil**) will disable future events.

Example

```
-- Function to be called on screen touch change
function MyTouchHandler(bTouch)
  -- bTouch will be true if screen is currently pressed or false if not.
end

-- Set up the event handler
ez.SetTouchEvent("MyTouchHandler")
```

29 SD Card Access

The ezLCD has an SD Card which is a full file system. Instead of implementing functions in the ezLCD + API library, the SD Card can be accessed as a standard file system using the native Lua file I/O functions such as `io.open`, `io.read`, `io.write`, `io.close`, etc. See your Lua programming manual for a list of all Lua I/O functions.

As the I/O functions are part of the standard Lua I/O library, make sure to ***not*** prepend "ez." in front of these functions.

Example

Use

```
io.open "myfile.txt"
```

not

```
ez.open "myfile.txt"
```

Other useful functions (see Lua documentation for details):

```
os.remove("myfile.txt")
```

```
os.rename("oldfile.txt", "newfile.txt")
```

Notes

Directories should be separated by forward slash ('/'), not by a backslash ('\') as in Windows and DOS.

The File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.

Long file names are supported, however the length of the complete File Path (directory + filename + extension) may not exceed 255 characters.

30 RS232 Functions

The following section details the functions used to manipulate the RS232 Interface.

RS232 Input Modes

Data can be received by RS232 in 2 ways:

- **Event Mode** - User defined event function is automatically called for each received byte.
- **Buffer Mode** - Incoming bytes are stored in the internal buffer. User can retrieve stored bytes from the buffer. The buffer has a size of 64 kBytes (65536 bytes) and is automatically allocated by ezLCD.

The input mode is decided by the type of the first parameter of the Rs232Open function. If the first parameter specifies an event function, the RS232 is opened in the Event Mode. Otherwise, the RS232 is opened in the Buffer Mode. RS232 interface should be closed first (Rs232Close function) before switching from one input mode to the other.

Obviously, the Input Modes affect only the way data is received. They do not have any effect on the way the data is sent.

30.1 ez.SerialOpen(RcvFunc [, PortNo [, BaudRate [, DataBits [, Parity [, StopBits [, HandShake]]]]]])**Purpose**

To open RS232 port in the [Event Mode](#).

Argument List

RcvFunc	String	Name of the Lua function to be called when a byte has been successfully received on the RS232port.
PortNo	Integer	Which RS232 port to open (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000. Default=9600
DataBits	Integer	Number of data bits (7, 8 [default], or 9)
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=None [default], 1=Even, 2=Odd
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit [Default], 1=1.5bit, 2=2bits, 3=0.5bit
HandShake	Integer	Handshake value to configure the RS232 port to accept. Valid values are: 0=None [Default], 1=h/w (RTS/CTS), 2=XON/XOFF, 3=HW (RTS), 4=HW (CTS), 5=RS485.

Return Value

Success	Boolean	true = port successfully opened.
---------	---------	----------------------------------

Example

In the example below, the ezLCD will stay in loop until it receives number 3.

```
bStop = false
```

```
-- Event Function
```

```
function ReceiveFunction(byte)
```

```
    if (byte == 3) then
```

```
        bStop=true
```

```
    end
```

```
end
```

```
-- open the RS-232 port
```

```
ez.SerialOpen("ReceiveFunction", 0, 9600)
```

```
-- Loop until number 3 is received by RS-232
```

```
while not bStop do
```

```
end ez.SerialClose()
```

30.2 ez.SerialOpen([PortNo [, BaudRate [, DataBits [, Parity [, StopBits [, HandShake]]]]]])**Purpose**

To open RS232 port in the [Buffer Mode](#).

Argument List

PortNo	Integer	Which RS232 port to open (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000. Default=9600
DataBits	Integer	Number of data bits (7, 8 [default], or 9)
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=None [default], 1=Even, 2=Odd.
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit [default], 1=1.5bit, 2=2bits, 3=0.5bit.
HandShake	Integer	Handshake value to configure the RS232 port to accept. Valid values are: 0=None [default], 1=h/w (RTS/CTS), 2=XON/XOFF, 3=HW (RTS), 4=HW (CTS), 5=RS485.

Return Value

Success	Boolean	true = port successfully opened.
---------	---------	----------------------------------

Example

In the example below, the ezLCD will stay in loop until it receives number 3.

```

bStop = false

-- open the RS-232 port
ez.SerialOpen()
-- loop until a 0 byte is sent down the RS-232 port
while (not bStop)
    n=ez.SerialRxLen()
    for i=1,n do
        if (ez.Serialgetc() == 3) then
            bStop=true
        end
    end
end
ez.SerialClose()

```

30.3 ez.SerialClose([PortNo])**Purpose**

To close the RS232 port and free allocated resources.

Argument List

PortNo	Integer	Which RS232 port to close (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0
--------	---------	---

Return Value

None

30.4 ez.SerialTx(Data [, PortNo [, MaxLen]])**Purpose**

To transmit data out, through the RS232 port.

Argument List

The behaviour of this function depends on the type of the argument 'Data'

Data	Integer	integer to be sent Number of bytes to be sent (maximum 4) depends on the value of the argument 'MaxLen'.	
		MaxLen	Bytes Sent (0 = LSB, 3 = MSB)
		1	Byte 0
		2	first Byte 1, Byte 0 last
		3	first Byte 2, Byte 1, Byte 0 last
		4	first Byte 3, Byte 2, Byte 1, Byte 0 last
	String	string to be sent Number of bytes to be sent depends on the value of the argument 'MaxLen'. Default=string length	
	Table	table to be sent Only integer and string elements of the table are sent. Integers are treated as bytes. If the integer value is bigger than 255, only it's least significant byte is sent. Total number bytes to be sent is limited by the value of the argument 'MaxLen'	
	light userdata	pointer to the data to be sent Number of bytes to be sent is specified by the value of the argument 'MaxLen'	
MaxLen	Integer	maximum number of bytes to be sent	
PortNo	Integer	Which RS232 port to access (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0	

Return Value

Success	Boolean	true = data successfully transmitted.
---------	---------	---------------------------------------

30.5 ez.SerialRxLen([PortNo])**Purpose**

To find out how many unread bytes are in the RS232 Input Buffer. Makes sense only if RS232 port is opened in the [BufferMode](#).

Argument List

PortNo	Integer	Which RS232 port to access (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0
--------	---------	--

Return Value

NoOfBytes	Integer	number of unread bytes in the RS232 Input Buffer Bytes are read from the RS232 Input Buffer by using function Rs232getc
-----------	---------	--

Example

In the example below, the ezLCD will stay in loop until it receives number 3.

```

bStop = false
-- open the RS-232 port
ez.SerialOpen(9600)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
    n = ez.SerialRxLen()
    for i = 1,n do
        if (ez.Serialgetc() == 3) then
            bstop = true
        end
    end
end
end
ez.SerialClose()

```

30.6 ez.Serialgetc([PortNo])

Purpose

To read bytes from the RS232 Input Buffer. Makes sense only if RS232 port is opened in the [Buffer Mode](#).

Argument List

PortNo	Integer	Which RS232 port to access (see hardware docs for locations): 0=Main UART (default), 1=Second UART, 2=USB Virtual Comm Port, 3=ESP32 UART0
--------	---------	--

Return Value

ReadByte	Integer	0 to 255 : byte read from the RS232 Input Buffer -1 : RS232 Input Buffer is empty
----------	---------	--

Example

In the example below, the ezLCD will stay in loop until it receives number 3.

```
bStop = false
-- open the RS-232 port
ez.SerialOpen(9600)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
    n = ez.SerialRxLen()
    for i = 1,n do
        if (ez.Serialgetc() == 3) then
            bstop = true
        end
    end
end
end
ez.SerialClose()
```

31 I2C Functions

The following section details the functions used to manipulate the I2C Interface.

The I2C Interface is used by the ezLCD to communicate with I2C devices like temperature sensors, serial EEPROMS, Analog to Digital Converters, etc.

I2C is a Master-Slave type interface, which is used to communicate with peripherals using a special type of low-speed serial protocol. ezLCD always operates as I2C Master.

More information about I2C can be found at: <http://en.wikipedia.org/wiki/I%C2%B2C>

I2C specification is available at: http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

31.1 ez.I2CopenMaster([SpeedMode, [Freq, [DigFilter, [AnalogFilter, [RiseTime, [FallTime]]]]]])**Purpose**

To initialize the I2C Interface in Master mode

Argument List

SpeedMode	Integer	0 = Standard (default) 1 = Fast 2 = FastPlus
Freq	Integer	IC Transmission/Receive Frequency in KHz Range/Defaults: <ul style="list-style-type: none"> • SpeedMode Standard: 1 - 100 (100 is default) • SpeedMode Fast: 1 - 400 (400 is default) • SpeedMode FastPlus: 1 - 1000 (1000 is default)
DigFilter	Integer	Digital Filter Coefficient (0-15) Higher # = more filtering <ul style="list-style-type: none"> • 0 = Digital Filter Off (default)
AnalogFilter	Boolean	Analog Filter Enabled (Default=FALSE)
RiseTime	Integer	I2C pulse rise time in ns. Range: 0 – 1000. (default=0)
FallTime	Integer	I2C pulse fall time in ns. Range 0 – 1000 (default=0)

Return Value

Success	Boolean	true = port successfully opened. false = failure (optional error message set)
---------	---------	--

Supported Models

ezLCD-408	Not supported (see alternate command format TBD)
ezLCD-4056	Not supported (see alternate command format TBD)
ezLCD-105B	Supported
ezLCD-5035	Supported
ezLCD-5104	Supported
ezLCD-5121	Supported

For details on advanced I2C topics such as speeds and filtering, please refer to the STM32H757 Reference Manual, located at https://www.st.com/resource/en/reference_manual/dm00176879-stm32h745755-and-stm32h747757-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

31.2 ez.I2CWrite(Address, Register, Data [, regSize [, timeout]])**ez.I2CWrite(Address, Register, IstringData, dataSize [, regSize [, timeout]])****Purpose**

To transmit data to I2C device

Argument List

Address	Integer	Slave address 0 - 127. (do not include the direction bit)
Register	Integer	I2C Register (AKA memory address)
Data	Integer	Data byte to be written (for single byte version of the command). Range: 0-255
IstringData	IString	byte string to send (non-zero terminated; zero bytes permitted in string)
dataSize	Integer	number of bytes in IString to send
regSize	Integer	size of Register Number (8 or 16 bits) Default=8
timeout	Integer	I2C timeout setting in ms (1 to 48). Default=48

Return Value

Success	Boolean	true = success false = write failed (optional error message set)
---------	---------	---

31.3 ez.I2Cread(Address, Register [, dataSize [, regSize [, timeout]]])**Purpose**

To read from an I2C device

Argument List

Address	Integer	Slave address 0 - 127. (do not include direction bit)
Register	Integer	I2C Register (AKA memory address)
dataSize	Integer	Number of bytes to read (default=1)
regSize	Integer	size of Register Number (8 or 16 bits) Default=8
timeout	Integer	I2C timeout setting in ms (1 to 48). Default=48

Return Value

data	IString	success=IString containing dataSize bytes failure=nil. (optional error message set)
------	---------	--

32 GPIO Functions

The following section details the functions used to manipulate the GPIO interface. Note that the Pins are numbered starting at zero (0). Please see applicable ezLCD Hardware Guide for pin numbering assignments

32.1 ez.SetPinInp(PinNo [, PullUp [, PullDn]])**Purpose**

Configures the I/O pin as discrete input.

Argument List

PinNo	Integer	Pin Number to be configured as a discrete input. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
PullUp	Boolean	Enable PullUp resistor on pin. Default=pullup disabled
PullDn	Boolean	Enable PullDown resistor on pin. Default=pulldown disabled. NOTE: if PullUp==true, PullDn setting is ignored (always false)

Return Value

None

Notes

If the reconfigured pin is part of any interface, this function may change the functionality of such interface.

For example, if pin normally assigned as RS232 Transmit is reconfigured, the RS232 will stop transmitting any data.

When Lua exits, all pins are automatically restored to their default configurations.

Example

```
-- Configuring pin 0 as input with no pullup or pulldown resistor  
ez.SetPinInp(0)
```


32.2 ez.SetPinsInp(PinsMask [, PullUpMask [, PullDnMask]])

Purpose

Configures all the specified I/O pins as discrete input.

Argument List

PinsMask	Integer	Bit Mask where each bit that is set to 1 is set to be a discrete input. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
PullUpMask	Integer	Bit Mask where each bit that is set to 1 enables the pull up resistor on that pin (if set as input)
PullDnMask	Integer	Bit Mask where each bit that is set to 1 enables the pulldown resistor on that pin (if set as input). 1 Bits ignored on any pin set as PullUp above

Return Value

None

Notes

If the reconfigured pin is part of any interface, this function may change the functionality of such interface.

For example, if pin normally assigned as RS232 Transmit is reconfigured, the RS232 will stop transmitting any data.

When Lua exits, all pins are automatically restored to their default configurations.

Example

```
-- Configuring pins 1, 2 and 5 as inputs with no pull up resistor  
ez.SetPinsInp(0x26) -- 0x26 = 0010 0110
```

32.3 ez.SetPinOut(PinNo [, InitialState [, OpenDrain [, PullUp [, PullDn [, Speed]]]])**Purpose**

Configures the I/O pin as discrete output.

Argument List

PinNo	Integer	Pin Number to be configured as a discrete output. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
InitialState	Boolean	Initial Pin State (0/1) Default=0
OpenDrain	Boolean	True=Open Drain mode False=Push-Pull mode (default)
PullUp	Boolean	Enable PullUp resistor on pin. Default=pullup disabled
PullDn	Boolean	Enable PullDown resistor on pin. Default=pulldown disabled. NOTE: if PullUp==true, PullDn setting is ignored (always false)
Speed	Integer	Pin slew rate (rise/fall time) 0=Low 1=Medium 2=High 3=Very High (default)

Return Value

None

Notes

If the reconfigured pin is part of any interface, this function may change the functionality of such interface.

For example, if pin normally assigned as RS232 Transmit is reconfigured, the RS232 will stop transmitting any data.

When Lua exits, all pins are automatically restored to their default configurations.

Example

```
-- Configuring pin 2 as push-pull output, no pullup/down, very high slew
rate
ez.SetPinOut(2)
```

32.4 ez.SetPinsOut(PinsMask [, InitialStateMask [, OpenDrainMask [, PullUpMask [, PullDnMask]]]])**Purpose**

Configures all the specified I/O pins as discrete output.

Argument List

PinsMask	Integer	Bit Mask where each bit that is set to 1 is set to be a discrete output. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
InitialStateMask	Integer	Bit Mask where each bit represents the initial state of the corresponding pin. Ignored for any pin not set as Output Default=0
OpenDrainMask	Integer	Bit Mask where each bit that is set to 1 indicates corresponding pin is Open Drain Mode. 0 bits indicate Push Pull mode. Ignored for any pin not set as Output
PullUpMask	Integer	Bit Mask where each bit that is set to 1 indicates corresponding pin has a PullUp resistor enabled. Ignored for any pin not set as Output
PullDnMask	Integer	Bit Mask where each bit that is set to 1 indicates corresponding pin has a PullDown resistor enabled. Ignored for any pin set to PullUp above

Return Value

None

Notes

This command configures each selected output pin as Slew Rate Very High. If you require a different setting, initialize the pin(s) individually using the SetPinOut command instead.

If the reconfigured pin is part of any interface, this function may change the functionality of such interface.

For example, if pin normally assigned as RS232 Transmit is reconfigured, the RS232 will stop transmitting any data.

When Lua exits, all pins are automatically restored to their default configurations.

Example

```
-- Configuring pins 3 and 6 as outputs
ez.SetPinsOut(0x48) -- 0x48 = 0100 1000
```

32.5 ez.SetPinIntr(PinNo, LuaFunction [, edgeSelect [, pullUp [, pullDn]]])**Purpose**

Defines an interrupt handler function that is automatically executed by the logic level change of the specified discrete input pin.

Argument List

PinNo	Integer	Pin Number. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
LuaFunction	String	Name of Lua function to execute on pin logic level change. Specify nil to disable further processing.
edgeSelect	Integer	0 = Rising and Falling Edge (default) 1 = Rising Edge Only 2 = Falling Edge Only
pullUp	Boolean	Enable PullUp resistor on pin. Default=pullup disabled
pullDn	Boolean	Enable PullDown resistor on pin. Default=pulldown disabled. NOTE: if PullUp==true, PullDn setting is ignored (always false)

Return Value

None

Notes

Some pins are not available for use to generate interrupts. Refer to your ezLCD Hardware Manual for details

Example

```
-- The following code counts the changes of logical level
-- on input pin 1 and stops when the level has changed 10 times
function MyInterrupt(pin_no)
    count = count + 1
    if (count = 10) then
        ez.SetPinIntr(1, nil)
    end
end

-- Configure pin 1 as input with pull up resistor enabled
ez.SetPinInp(1, true)

-- Assign on change interrupt to pin 1
count = 0
ez.SetPinIntr(1, "MyInterrupt")
```

32.6 ez.RestorePin(PinNo)

Purpose

To restore the discrete I/O pin to the default configuration

Argument List

PinNo	Integer	Lua I/O Pin Number. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
-------	---------	---

Return Value

None

Notes

This function also disables the associated interrupt set by SetPinIntr.

32.7 ez.RestorePins(PinsMask)**Purpose**

To restore the discrete I/O pins to the default configuration

Argument List

PinsMask	Integer	Bit Mask where each bit that is set to 1 is set to be restored. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
----------	---------	---

Return Value

None

Notes

This function also disables associated interrupts set by SetPinIntr.

32.8 ez.Pin(PinNo)

Purpose

Retrieve the logic level on the specified pin.

Argument List

PinNo	Integer	Lua I/O Pin Number. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
-------	---------	---

Return Value

Value	Integer	0 = low >0 = high <0 = invalid pin number
-------	---------	---

32.9 ez.Pin(PinNo, Value)**Purpose**

Set the logic level on the specified pin.

Argument List

PinNo	Integer	Lua I/O Pin Number. Refer to your ezLCD Hardware Manual for I/O Pin assignments.
Value	Integer	0 = low >0 = high

Return Value

None

32.10 ez.Pins(PinsMask)**Purpose**

Retrieve the logic level on the specified pins.

Argument List

PinsMask	Integer	Bitmask of I/O Pin Numbers Refer to your ezLCD Hardware Manual for I/O Pin assignments.
----------	---------	--

Return Value

ValueMask	Integer	Each bit will be 0 (low) or 1 (high) depending on the level on each corresponding pin.
-----------	---------	--

32.11 ez.Pins(PinsMask, Value)**Purpose**

Set the logic level on each of the specified pins.

Argument List

PinsMask	Integer	Bitmask of I/O Pin Numbers Refer to your ezLCD Hardware Manual for I/O Pin assignments.
Value	Integer	The level on each pin will be set to low (bit value = 0) or high (bit value = 1).

Return Value

None

33 Quadrature Encoder Interface

The quadrature encoder interface permits the connection of a quadrature encoder to select models of ezLCD.

The quadrature encoder's position can be read either on demand or via an interrupt.

Pin Assignments – ezLCD5035:

Encoder	Use	Connect to
A	Output Phase A	I2S2_MCK (User 2 / Pin 20)
B	Output Phase B	GPIO6 (User 2 / Pin 26)
Switch	Pushbutton Switch (optional)	Any available GPIO pin
V+	3.3v	+3.3v DO NOT EXCEED 3.3v
GND	GND	Any available ground pin

More information about Quadrature Encoders can be found at:

https://en.wikipedia.org/wiki/Incremental_encoder#Quadrature_outputs

33.1 **ez.QuadOpen([maxValue])**
 ez.QuadOpen(LuaFunction [, maxValue])

Purpose

Open the Quadrature Encoder interface

Argument List

LuaFunction	function	If specified, this function will be called whenever the quadrature encoder count changes
maxValue	Integer	Specifies the maximum count value (default=100). Range: 2-32767

Return Value

True = Success

33.2 ez.QuadClose()**Purpose**

Close the Quadrature Encoder interface

Argument List

None

Return Value

None

33.3 ez.QuadGetValue()**Purpose**

Get the current Quadrature Encoder count

Argument List

None

Return Value

Current Quadrature Encoder Count (range = 0 to maxValu specified in QuadOpen())

34 CANBus Interface

CANBus is a widely-used 2-wire multidrop networking protocol, found in many applications such as in the automotive, marine, and industrial industries. ezLCD has 2 levels of CANBus support: high-level support for specific CANBus implementations and low-level support to allow developers to interface with virtually any CANBus implementation not covered as part of high-level support.

Supported High-level protocols:

- OBD2
- SAE J1939
- NMEA2000
- CANOpen
- DeviceNet

All implementations of CANBus require a CANBus transceiver chip external to the ezLCD. MCP2551 is one example of a suitable transceiver. All CANBus networks also require 120 ohm resistor termination on each physical end of the network.

For more information on CANBus: https://en.wikipedia.org/wiki/CAN_bus

34.1 Low-Level CANBus Interface

The Low-Level CANBus interface routines are utilized to allow support of any implementation of CANBus that is not supported using the high-level functions included in the Lua API.

CANBus packets are stored in specialized on-CPU-chip RAM. On the STM32H7 series chips, this RAM is 10K in size and is shared by both CANBus controllers. The division of CANBus RAM is controlled using the CANBusOpen command, via the use of numerous optional parameters.

CANBus RAM is divided into several functions:

Tx FIFO	Default Transmission storage: Packets get transmitted in order that they are buffered
Rx FIFO	Default Receive storage: Packets get stored in the order that they were received. Two Rx FIFOs are available
Tx Buffers or Queue	TBD
Rx Buffers	Filtered Packets can be stored in specific buffers regardless of reception order
Tx Events	Optionally store events related to individual Tx Packets
Standard Filters	Stores definitions for Rx Filters based on 11-bit Message IDs
Extended Filters	Stores definitions for Rx Filters based on 29-bit Message IDs

Filters are utilized to accept or reject received CANBus packets based on a criteria. Additionally, a filter can specify that the received packet be stored in a specific Rx Buffer or FIFO. Filters can also mark a packet as High Priority, to be processed via a separate High-Priority Rx Event Handler.

A global filter also exists, which determines the disposition of any received packet that does not match one of the Standard or Extended Filters.

For more details regarding the low-level CANBus protocol functionality, please refer to the **STM32H757 Reference Manual**

Purpose

Argument List

Return Value

Notes

Example

© 2022 Earth Computer Technologies, Inc.

34.1.2 ez.CANBusClose([BusNo])**Purpose**

Closes the low-level CANBus Interface

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- Close CANBus interface 1 with default settings  
ez.CANBusClose()
```

34.1.3 ez.CANBusStart([BusNo])**Purpose**

Starts the low-level CANBus Interface after configuration is complete.

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- Open and Start CANBus interface 1 with default settings  
ez.CANBusOpen()  
ez.CANBusStart()
```

34.1.4 ez.CANBusStop([BusNo])**Purpose**

Stops the low-level CANBus Interface. Once stopped, additional configuration commands may be executed.

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- Stop CANBus interface 1  
ez.CANBusStop()
```

34.1.5 ez.CANBusRegisterHandler(handlerType, handlerFunction [, BusNo])**Purpose**

Closes the low-level CANBus Interface

Argument List

Parameter	Integer	Default	Description
handlerType	Integer	N/A	0=RxFIFO0 1=RxFIFO1 2=RxBuffer 3=RxHighPriMessage 4=TxEvent 5=TxComplete 6=TxAbortComplete 7=TxFIFOEmpty 8=Error State Passive 9=Error State Warning 10=Bus Off
handlerFunction	String	N/A	Lua function name to be executed as handler
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- Close CANBus interface 1 with default settings  
ez.CANBusClose()
```

34.1.6 `ez.CANBusConfigFilter(BusNo, filterIndex, IDType, config=0)`
`ez.CANBusConfigFilter(BusNo, filterIndex, IDType, config [, filterType [, ID1 [, ID2 [, buffIdx [, calib]]]]])`

Purpose

Configures CANBus filters to determine disposition of matching frames.

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	N/A	CANBus Number (1 or 2)
filterIndex	Integer	N/A	Index number of filter slot
IDType	Integer	N/A	0=11-Bit Address, 1=29-Bit Address
config	Integer	N/A	0=Disable Filter 1=Store in Rx FIFO0 2=Store in Rx FIFO1 3=Reject 4=Set as high Priority 5=Set as high Priority and store in Rx FIFO0 6=Set as high Priority and store in Rx FIFO1 7=Store in Rx Buffer (specify buffer number below)
filterType	Integer	0	Matching ID Type: 0=Range from ID1 – ID2 1=ID1 or ID2 2=ID1 using ID2 as a mask (TODO CHECK) 3=Range from ID1 – ID2 with no mask
ID1	Integer	0	Message ID 1
ID2	Integer	0	Message ID 2
buffIdx	Integer	0	Rx Buffer to store in of config=7
calib	Integer	0	0=Not Calib Message, 1=Calib message

Return Value

0 = Success, -1 if Error

Example

```
-- Store frames from ID 0x7e8 in Rx Buffer 2
ez.CANBusConfigFilter(1, 0, 0, 7, 0, 0x7e8, 0x7e8, 2)
```

34.1.7 ez.CANBusGlobalFilter([NonMatchStd [, NonMatchExt [, RejectRTRStd [, RejectRTRExt [, BusNo]]]])

Purpose

Configures the disposition of any received CANBus frames not matching a filter

Argument List

Parameter	Type	Default	Description
NonMatchStd	Integer	0	Sets disposition for 11-bit frames not matching a filter: 0=Store in Rx FIFO0 1=Store in Rx FIFO1 2=Reject
NonMatchExt	Integer	0	Sets disposition for 29-bit frames not matching a filter: 0=Store in Rx FIFO0 1=Store in Rx FIFO1 2=Reject
RejectRTRStd	Integer	0	Sets disposition for 11-bit frames with RTR (remote request) bit set: 0=Accept 1=Reject
RejectRTRExt	Integer	0	Sets disposition for 29-bit frames with RTR (remote request) bit set: 0=Accept 1=Reject
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- Configure Global filter to reject all non-matching frames
ez.CANBusGlobalFilter( 2, 2, 1, 1 )
```

34.1.8 ez.CANBusEnableISOMode([BusNo [, boschMode]])**Purpose**

Configures CANFD Frame Format standards specification

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	1	CANBus Number (1 or 2)
boschMode	Integer	1	ISO-11898-1 Mode Enable: 0=Bosch CANFD Spec 1.0 1=ISO-11898-1

Return Value

None

Note

This setting only affects CAN-FD Mode frames

Example

```
-- Configure CANFD Bosch mode  
ez.CANBusClose( 1, 0 )
```


34.1.9 ez.CANBusTx(msgID, IDType, data [, length [, BusNo [, frameType [, txEventEnable [, MM [, txBuffer [, format [, errState [, BRS]]]]]]]]])

Purpose

Queues a Frame for CANBus Transmission

Argument List

Parameter	Integer	Default	Description
msgID	Integer	N/A	Message ID (Address). 11-bit or 29-bit depending on IDType setting
IDType	Integer	N/A	0=11-Bit, 1=29-Bit
data	String	N/A	Data to be transmitted. Embedded NULLs accepted (not a null-terminated string)
length	Integer	8	Data length
BusNo	Integer	1	CANBus Number (1 or 2)
frameType	Integer	0	0=Data Frame 1=Remote Request (RTR bit Set)
txEventEnable	Integer	0	0=Tx Event Disabled 1=Generate Tx Events
MM	Integer	0	Message Marker (stored in Tx Events for reference)
txBuffer	Integer	-1	Where to buffer frame: -1 = TX FIFO 0-31 = Tx Buffer 0-31
format	Integer	0	0=Classic CAN 1=FD-CAN (CAN 2.0)
errState	Integer	0	0=Error State Active 1=Error State Passive
BRS	Integer	0	NOTE: Only for FDCAN 0=Bit Rate Switching Off 1=Bit Rate Switching On

Return Value

0 = Success, -1 if Error

Notes

Example

```
-- TBD
```

34.1.10 ez.CANBusTxAbort(buffIdx, [BusNo])**Purpose**

Cancels a pending Buffer-stored CANBus Transmission

Argument List

Parameter	Integer	Default	Description
buffIdx	Integer	N/A	TX Buffer Number to Abort (0-63)
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

0 = Success, -1 if Error

Example

```
-- TBD
```

34.1.11 ez.CANBusRx

MsgID, IDType, frameType, Data, Length, errStatus, format, BRS, filterMatch, filterIdx = ez.CANBusRx(buffIdx, [BusNo])

Purpose

Receives a received CANBus frame from specified location. This function is used for polling and certain event-based buffer operations.

Argument List

Parameter	Integer	Default	Description
buffIdx	Integer	0	Where to retrieve Rx Frame from: 0-63=Rx Buffer 0-63 64=Rx FIFO0 65=Rx FIFO1
BusNo	Integer	1	CANBus Number (1 or 2)

Return Values

MsgID	Integer	Message ID (Address)
IDType	Integer	0=11-Bit Address 1=29-Bit Address
frameType	Integer	0=Data Frame 1=Remote Request (RTR bit Set)
Data	String	Received data portion of CANBus Frame (can contain embedded NULLs). Use Length value to determine number of bytes
Length	Integer	Size of Data field in bytes
errStatus	Integer	0=Error State Passive 1=Error State Active
format	Integer	0=Classic CAN 1=FD-CAN (CAN 2.0)
BRS	Integer	0=BRS Off 1=BRS On
filterMatch	Integer	0=No filter matched 1=Filter matched
filterIdx	Integer	If Filter matched, filter index of matching filter

Example

```
-- Read received CANBus packet from Rx Buffer 0 on Bus 1
MsgID, IDType, frameType, Data, Length, errStatus, format, BRS, filterMatch, filterIdx =
ez.CANBusRx ()
```

34.1.12 ez.CANBusIsTxBuffFree(bufferNo [, BusNo])**Purpose**

Closes the low-level CANBus Interface

Argument List

Parameter	Integer	Default	Description
bufferNo	Integer	N/A	Tx Buffer to check (0 – 63)
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

True = Tx Buffer is empty

False=Tx Buffer contains a pending frame

Example

```
-- Close CANBus interface 1 with default settings  
ez.CANBusClose()
```

34.1.13 ez.CANBusIsRxMsgAvail(bufferNo [, BusNo])**Purpose**

Closes the low-level CANBus Interface

Argument List

Parameter	Integer	Default	Description
bufferNo	Integer	N/A	Rx Buffer to check (0 – 63)
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

False = Rx Buffer is empty

True = Rx Buffer contains a pending frame that can be read via ez.CANBusRx()

Example

```
-- Close CANBus interface 1 with default settings  
ez.CANBusClose()
```

34.1.14 ez.CANBusGetFIFOLevel(fifoToCheck [, BusNo])**Purpose**

Report number of FIFO elements of the specified type are free

Argument List

Parameter	Integer	Default	Description
fifoToCheck	Integer	N/A	Specify FIFO to report the status of: 0=Tx FIFO 1=Rx FIFO 0 2=Rx FIFO 1
BusNo	Integer	1	CANBus Number (1 or 2)

Return Value

Integer: number of FIFO Elements that are free

Example

```
-- Get the number of FIFO elements free in the Tx FIFO  
N = ez.CANBusGetFIFOLevel(0)
```

34.1.15 ez.CANBusGetErrCnt
txErr, rxErr, rxErrPassive, errLogging = ez.CANBusGetErrCnt([BusNo])

Purpose

Closes the low-level CANBus Interface

Argument List

Parameter	Integer	Default	Description
BusNo	Integer	1	CANBus Number (1 or 2)

Return Values

txError	Integer	Number of Transmit errors (0 – 255)
rxErr	Integer	Number of Receive errors (0 – 127)
rxErrPassive	Integer	0=Rx Error counter is below the passive level (128) 1=Rx Error counter has reached the passive level (128)
errLogging	Integer	Number of Rx and Tx FD-CAN protocol errors (0-255)

Example

```
-- Get current error counters for CANBus port 1 (default)
txErr, rxErr, rxErrPassive, errLogging = ez.CANBusIsGetErrCnt()
```

35 DMX512 Interface

The DMX512 interface utilizes the RS485 physical interface with a specialized API to support the DMX lighting control protocol. This industry-standard protocol is widely used to control theatrical and commercial lighting systems.

ezLCD currently supports 1 DMX Universe with a maximum of 512 channels. The frequency that DMX packets are transmitted is adjustable to accommodate lighting instruments that may not completely follow the official specification (especially common with older and DJ-class lighting).

Currently, only transmission of DMX channel data is supported; extensions such as RDM are not presently supported. Art-Net (DMX512 over Ethernet) is not presently supported.

A DMX network consists of daisy-chained RS485 connections (2-wire plus ground). Data is transmitted at fixed a bitrate of 250kbaud. Each end of the network must be terminated with a 120 ohm resistor. Up to 32 devices (loads) may be connected to a single cable of up to 1300 feet in total length, without the use of repeaters. Each device uses one or more consecutive channels and the starting channel number is configured per-device via either dipswitches or a control panel on each individual device. Channel values range from 0 – 255. Details of channel usage are device-specific and are contained within the manufacturer's documentation.

NOTE that the ezLCD is not capable of directly driving a DMX cable. Identical to RS-485, a suitable driver chip must be employed, such as SN75176 or . The USART pins on the ezLCD are 5-volt tolerant so either a 5v driver chip can be used safely.

Pin Assignments – ezLCD5035:

SN75176, xxx, or equiv	Use	Connect to
TD	DMX Data TX	USART3 TX
RD	DMX Data RX (currently unused)	USART3 RX
Pins 2, 3 (DE,RE)	Data Transmit Enable	USART3 DE
GND	GND	Any available ground pin

More information on this protocol can be found at <https://en.wikipedia.org/wiki/DMX512>

35.1 ez.DMXOpen([maxChannels [, txDelay [, portNo]]])**Purpose**

Opens and starts the DMX512 interface

Argument List

maxChannels	Integer	Number of channels to transmit per packet. Default=512
txDelay	Integer	Delay time between transmission of DMX packets. Default=TBD
portNo	Integer	UART Port used for DMX universe. (Default=1) Currently, only port 1 is supported (USART3)

Return Value

0 = Success

Notes

Upon the completion of this command, the DMX port will immediately begin transmission of DMX channel data on the selected port. Initially, all channel levels are set to 0, until DMXSetLevel is used.

When Lua exits, the DMX port is automatically closed and all DMX packets cease to be transmitted.

Example

```
-- Start DM512 interface on the default port  
ez.DMXOpen()
```

35.2 ez.DMXClose([portNo])**Purpose**

Stops and closes the DMX512 interface

Argument List

portNo	Integer	UART Port used for DMX universe. (Default=1) Currently, only port 1 is supported (USART3)
--------	---------	--

Return Value

0 = Success

Notes

Upon the completion of this command, the DMX port stop transmission of DMX channel data on the selected port and the port will be available for assignment to other functions as necessary.

The behavior of instruments on the DMX network once transmission is terminated is device-dependent. Some instruments can be configured to adjust their parameters to preset values upon loss of signal. Others will remain at the last values that they received. Consult the respective instrument manuals for more details.

Example

```
-- Stop DM512 interface on the default port  
ez.DMXClose()
```

35.3 ez.DMXSetLevel(channel, value [, rampTime [, portNo]])**Purpose**

Sets the level of a DMX channel, to be transmitted by an open DMX512 port.

Argument List

channel	Integer	Channel number to set (range 1-maxChannels). maxChannels is specified in DMXOpen or if unspecified, defaults to 512
level	Integer	Level to set the channel to. Range=0-255. Exact usage depends on individual lighting instrument manufacturer's specifications.
rampTime	Integer	Time in seconds to ramp up or down level to new value. If 0, change is instant. (default=0)
portNo	Integer	UART Port used for DMX universe. (Default=1) Currently, only port 1 is supported (USART3)

Return Value

0 = Success

Notes

Upon the completion of this command, the DMX port stop transmission of DMX channel data on the selected port and the port will be available for assignment to other functions as necessary.

The behavior of instruments on the DMX network once transmission is terminated is device-dependent. Some instruments can be configured to adjust their parameters to preset values upon loss of signal. Others will remain at the last values that they received. Consult the respective instrument manuals for more details.

Example

```
-- Open DMX port with default values and start transmitting,
-- initializing all levels to 0
ez.DMXOpen()

-- Set DMX Channel 5 instantly to level 127 (50%)
ez.DMXSetLevel(5, 127)

-- Set DMX Channel 6 to level 255 (100%) over the course of 5 seconds
-- NOTE as this is the first level set for Channel 6, ramp is from 0 to 255
ez.DMXSetLevel(6, 255, 5)
```

35.4 ez.DMXGetLevel(channel [, portNo])**Purpose**

Sets the level of a DMX channel, to be transmitted by an open DMX512 port.

Argument List

channel	Integer	Channel number to set (range 1-maxChannels). maxChannels is specified in DMXOpen or if unspecified, defaults to 512
portNo	Integer	UART Port used for DMX universe. (Default=1) Currently, only port 1 is supported (USART3)

Return Value

Level channel is currently set to (0-255)

Notes

If the selected channel is currently transitioning via the rampRate setting of DMXSetLevel, the value returned will be the current level being transmitted to the instrument, not the final target value.

Example

```
-- Open DMX port with default values and start transmitting,  
-- initializing all levels to 0  
ez.DMXOpen()  
  
-- Set DMX Channel 5 instantly to level 127 (50%)  
ez.DMXSetLevel(5, 127)  
  
-- Get the value of channel 5 (will return 127)  
Level = ez.DMXGetLevel(5)
```

36 Advanced Topics

The following sections describe ezLCD advanced topics.

36.1 Frame Management Functions

The following section details the functions used to manipulate ezLCD display frames.

Frame Management

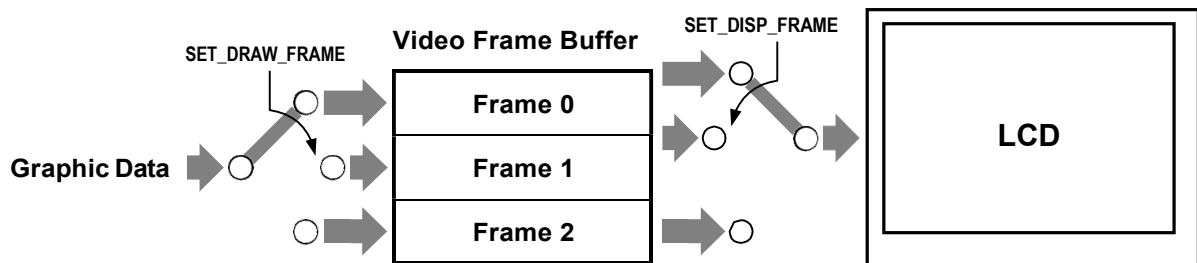
ezLCD devices consist of 1 or more frames. A frame is a portion of the ezLCD memory that can be displayed on the screen. The number of available frames depends on the amount of memory that has been installed, configuration settings, and the height/width in pixels of the screen; however, there are at least 2 frames in each ezLCD device.

Frames are numbered starting at zero (0).

ezLCD devices have the concept of "draw" frames and a "display" frame. Functions that affect the contents of the draw frame will not affect the contents of the display frame **unless** the draw frame and the display frame are identical.

Draw frames are virtual displays. By writing to different draw frames, the ezLCD can have preloaded screens which can be instantly displayed by changing the current display frame to one of these preloaded draw frames.

The Frame management functions allow the draw frame and display frame to be changed.



Layers

On ezLCD-4xxx and ezLCD-5x there are two Layers (foreground and background) which are combined to form the final image. A different frame is assigned to each of these two layers. This functionality can be useful for things such as image overlays and display of gauges where most of the gauge is static (the background) while the pointer changes (foreground).

The visibility of each layer can also be controlled

By default (after boot) the Foreground Layer is set to Frame 1 and the Background Layer is set to Frame 0 and both are visible.

36.1.1 ez.SetDispFrame(FrameNo, LayerNo)**Purpose**

Sets the frame to be displayed as the specified layer (background or foreground) on the screen

Argument List

FrameNo	Integer	frame number to display
LayerNo	Integer	Select Display Layer (0=Background or 1=Foreground). Default=1 (Foreground)

Return Value

Success	Boolean	true if the FrameNo can be displayed
---------	---------	--------------------------------------

Startup Values

At boot, Layer 1 (Foreground) is set to Frame 1; Layer 0 (Background) is set to Frame 0

36.1.2 ez.GetDispFrame(LayerNo)**Purpose**

Gets the currently displayed frame number for the specified layer

Argument List

LayerNo	Integer	Select Layer (0=Background or 1=Foreground). Default=1 (Foreground)
---------	---------	---

Return Value

FrameNo	Integer	Current display frame number
---------	---------	------------------------------

Startup Values

At boot, Layer 1 (Foreground) is set to Frame 1; Layer 0 (Background) is set to Frame 0

36.1.3 ez.GetLayerVisibility(LayerNo)**Purpose**

Gets the visibility status of the specified layer (foreground or background) on the screen

Argument List

LayerNo	Integer	Select Display Layer (0=Background or 1=Foreground). Default=1 (Foreground)
---------	---------	---

Return Value

visible	boolean	Is the specified layer visible on the screen?
---------	---------	---

Startup Values

At boot, Layer 1 (Foreground) is Visible and Layer 0 (Background) is not visible

36.1.4 ez.SetLayerVisibility(layerNo, visible)**Purpose**

Sets the frame to be displayed as the Foreground Layer on the screen

Argument List

layerNo	integer	Layer number being referenced (0=Background or 1=Foreground)
visible	boolean	Should the specified layer be visible on the screen?

Return Value

None

Startup Values

At boot, Layer 1 (Foreground) is Visible and Layer 0 (Background) is not visible

36.1.5 ez.SetDrawFrame(FrameNo)**Purpose**

Sets the frame to be used for drawing commands.

Argument List

FrameNo	Integer	frame number to draw on
---------	---------	-------------------------

Return Value

Success	Boolean	true if the FrameNo can be found
---------	---------	----------------------------------

Startup Values

At boot, Layer 1 (Foreground) is the drawing frame

36.1.6 ez.GetDrawFrame()**Purpose**

Gets the current draw frame number.

Argument List

None

Return Value

FrameNo	Integer	Current draw frame number
---------	---------	---------------------------

Startup Values

At boot, Layer 1 (Foreground) is the drawing frame

36.1.7 ez.CopyFrame(DestFrame, SourceFrame)**Purpose**

Copy the contents of the SourceFrame to the DestFrame

Argument List

DestFrame	Integer	frame number to be copied to
SourceFrame	Integer	frame number of be copied from

Return Value

Success	Boolean	true if the copy was successful
---------	---------	---------------------------------

36.1.8 ez.MergeFrame(DestFrame, SourceFrame)**Purpose**

Merge the contents of the SourceFrame with the DestFrame and store the results in DestFrame using the current [Alpha](#) as the transparency setting.

Argument List

DestFrame	Integer	frame number to be merged into
SourceFrame	Integer	frame number of the source frame

Return Value

Success	Boolean	true if the merge was successful
---------	---------	----------------------------------

36.1.9 ez.CopyRect(DestFrame, SourceFrame, DestX, DestY, SourceX, SourceY, width, height)**Purpose**

Copy the contents of a rectangular region from the SourceFrame to the DestFrame.

Argument List

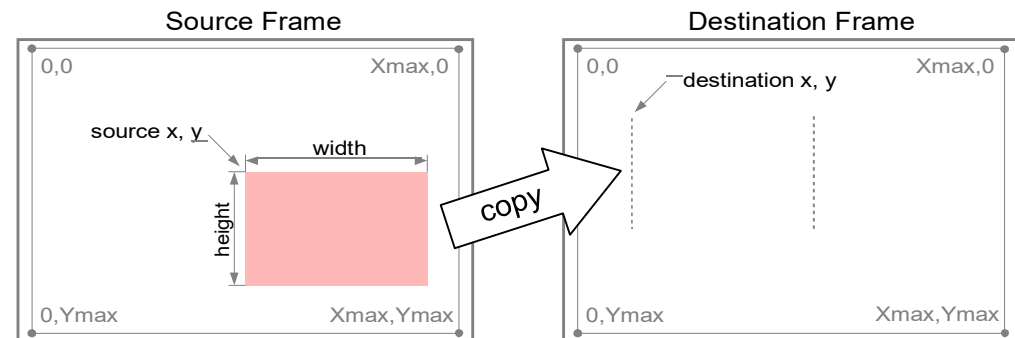
DestFrame	Integer	frame number to be copied to
SourceFrame	Integer	frame number of be copied from
DestX	Integer	x position in DestFrame to place copied data
DestY	Integer	y position in DestFrame to place copied data
SourceX	Integer	x position in SourceFrame to copy data from
SourceY	Integer	y position in SourceFrame to copy data from
width	Integer	width of area to be copied
height	Integer	height of area to be copied

Return Value

Success	Boolean	true if the copy was successful
---------	---------	---------------------------------

Notes

Copy a rectangle sized portion width by height from the frame SourceFrame starting at position (SourceX, SourceY) to frame DestFrame starting at position (DestX, DestY)

Example

36.1.10 ez.MergeRect(DestFrame, SourceFrame, DestX, DestY, SourceX, SourceY, width, height)

Purpose

Merge the contents of a rectangular region from the SourceFrame to the DestFrame using the current transparency value [Alpha](#).

Argument List

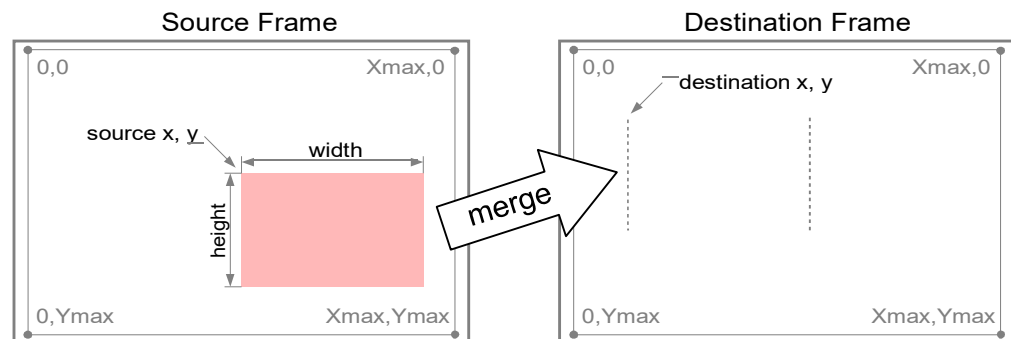
DestFrame	Integer	frame number to be copied to
SourceFrame	Integer	frame number of be copied from
DestX	Integer	x position in DestFrame to place copied data
DestY	Integer	y position in DestFrame to place copied data
SourceX	Integer	x position in SourceFrame to copy data from
SourceY	Integer	y position in SourceFrame to copy data from
width	Integer	width of area to be copied
height	Integer	height of area to be copied

Return Value

Success	Boolean	true if the copy was successful
---------	---------	---------------------------------

Notes

Merge a rectangle sized portion width by height from the frame SourceFrame starting at position (SourceX, SourceY) to frame DestFrame starting at position (DestX, DestY)



GLOSSARY

Configuration Keys	Part of ezLCD Customization. Set of text words and values assigned to them. They are specifying the <i>User Configuration</i> . Similar, in concept, to the keys used in Windows .ini files.
ezLCD Customization	Modification of the default power-up parameters. Addition of custom fonts, bitmaps, Lua programs, etc.
Firmware	Operating software of the ezLCD. Can be in-field upgraded.
Lua	Powerful, fast, light-weight, embeddable scripting language. By embedding Lua interpreter, the ezLCD become a true independent system (computer), which does not need any external host to drive it. Described in the " <i>ezLCD Lua API Manual</i> ".
User Configuration	Part of ezLCD Customization. Modifies some of the ezLCD default parameters like: communication parameters, start-up screen, etc. Upon the power-up the ezLCD CPU configures the ezLCD according to the data read from the User Configuration.