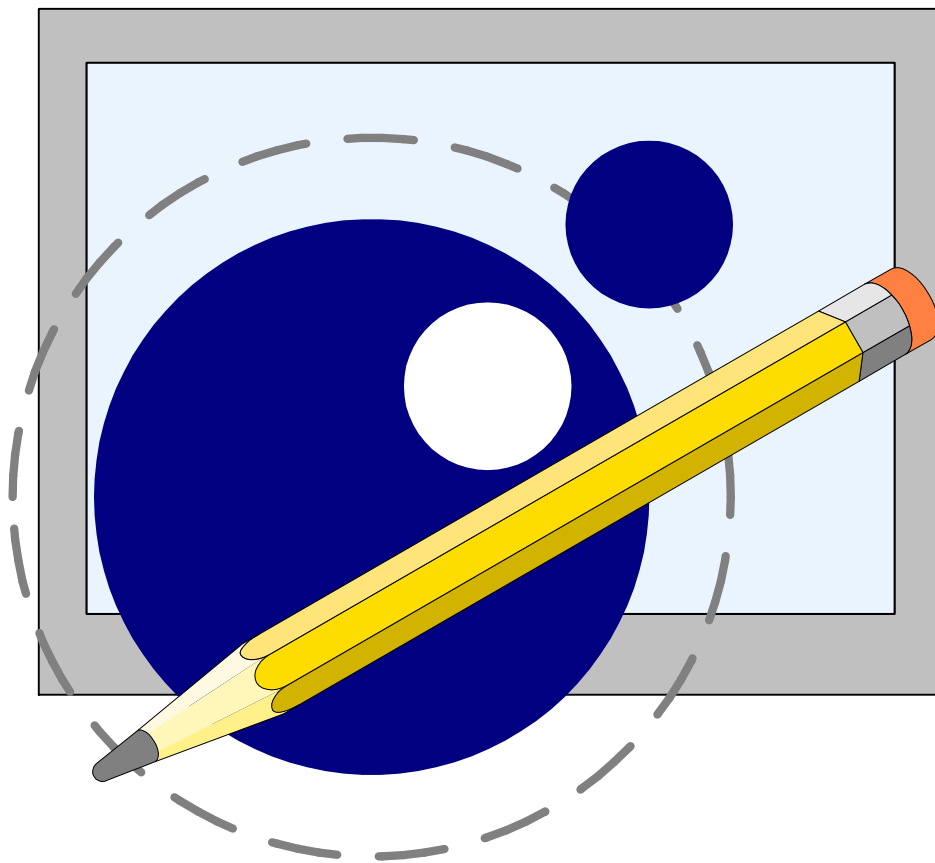


ezLCD+

Lua API Manual



MCours.com

ezLCD+ Documentation Overview

The ezLCD+ documentation consists of:

"ezLCD+10x Manual"

Specific for each ezLCD+ device (ezLCD+101, ezLCD+102, .. 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

Release History

Introduction

Product Features

Quick Start

1 Quick Start: Lua	12
--------------------------	----

ezLCD+ Customization

Drawing on the ezLCD+

1 Print	14
2 Screen Coordinates	15
3 Vector Graphics	16
4 Raster Graphics (Bitmaps)	17
5 Drawing Parameters	18
General	18
Transparency	19
Pen	20
6 Fonts	22
Bitmap Fonts	23
True Type Fonts	24

Programming the ezLCD+ with Lua

1 Constants	26
2 Position Functions	27
SetXY(x, y)	28
SetX(x)	29
SetY(y)	30
GetX()	31
GetY()	32
3 Color Functions	33
RGB(red, green, blue)	34
GetRed(ezLCDcolor)	35
GetGreen(exLCDcolor)	36
GetBlue(ezLCDcolor)	37
SetColor(ezLCDcolor)	38
SetBgColor(ezLCDcolor)	39
ReplaceColor(x, y, width, height, OldColor, NewColor)	40
GetPixel()	41
GetPixel(x, y)	42

4	Transparency Functions	43
	SetAlpha(alpha)	44
	TrColorNone()	45
	SetTrColor(ezLCDcolor)	46
5	Pen Size Functions	47
	SetPenSize(height, width)	48
6	Angle Functions	49
	Deg(degrees)	50
	Rad(radians)	51
7	Button Functions	52
	Overview	53
	Button(ID, iState, iconUp, iconDown, iconDisabled, x, y, width, height)	54
	Button(ID, iState, iconUp, iconDown, iconDisabled, x, y)	55
	Button(iD, iState)	56
	DelButtons()	57
	SetButtonEvent(sButtonHandler)	58
8	Fill Area Functions	59
	Overview	60
	CIs()	61
	CIs(ezLCDcolor)	62
	Fill()	63
	Fill(ezLCDcolor)	64
	Fill(x, y)	65
	Fill(x, y, FillColor)	66
	FillBound(BoundColor)	67
	FillBound(BoundColor, FillColor)	68
	FillBound(x, y, BoundColor)	69
	FillBound(x, y, BoundColor, FillColor)	70
9	Line Drawing Functions	71
	HLine(x2)	72
	HLine(x2, color)	73
	HLine(x1, y1, x2)	74
	HLine(x1, y1, x2, color)	75
	VLine(y2)	76
	VLine(y2, color)	77
	VLine(x1, y1, y2)	78
	VLine(x1, y1, y2, color)	79
	Line(x2, y2)	80
	Line(x2, y2, color)	81
	Line(x1, y1, x2, y2)	82
	Line(x1, y1, x2, y2, color)	83
	LineAng(angle, length)	84
	LineAng(angle, length, color)	85
	LineAng(x1, y1, angle, length)	86
	LineAng(x1, y1, angle, length, color)	87
10	Curve Drawing Functions	88
	Circle(radius)	89
	Circle(radius, color)	90
	Circle(x, y, radius)	91
	Circle(x, y, radius, color)	92
	CircleFill(radius)	93
	CircleFill(radius, color)	94

CircleFill(x, y, radius)	95
CircleFill(x, y, radius, color)	96
Ellipse(a, b)	97
Ellipse(a, b, color)	98
Ellipse(x, y, a, b)	99
Ellipse(x, y, a, b, color)	100
EllipseFill(a, b)	101
EllipseFill(a, b, color)	102
EllipseFill(x, y, a, b)	103
EllipseFill(x, y, a, b, color)	104
Arc(radius, Startang, EndAng)	105
Arc(radius, StartAng, EndAng, color)	106
Arc(x, y, radius, StartAng, EndAng)	107
Arc(x, y, radius, StartAng, EndAng, color)	108
Pie(radius, StartAng, EndAng)	109
Pie(radius, StartAng, EndAng, color)	110
Pie(x, y, radius, StartAng, EndAng)	111
Pie(x, y, radius, StartAng, EndAng, color)	112
EllipseArc(a, b, StartAng, EndAng)	113
EllipseArc(a, b, StartAng, EndAng, color)	114
EllipseArc(x, y, a, b, StartAng, EndAng)	115
EllipseArc(x, y, a, b, StartAng, EndAng, color)	116
EllipsePie(a, b, StartAng, EndAng)	117
EllipsePie(a, b, StartAng, EndAng, color)	118
EllipsePie(x, y, a, b, StartAng, EndAng)	119
EllipsePie(x, y, a, b, StartAng, EndAng, color)	120
11 Polygon Drawing Functions	121
Box(x2, y2)	122
Box(x2, y2, color)	123
Box(x1, y1, x2, y2)	124
Box(x1, y1, x2, y2, color)	125
BoxFill(x2, y2)	126
BoxFill(x2, y2, color)	127
BoxFill(x1, y1, x2, y2)	128
BoxFill(x1, y1, x2, y2, color)	129
Polygon(x1, y1, x2, y2, ... xn, yn)	130
12 Single Pixel Functions	131
Plot()	132
Plot(x, y)	133
Plot(x, y, PlotColor)	134
GetPixel()	135
GetPixel(x, y)	136
13 Font Functions	137
SetBmFont(BitmapFontNo)	138
SetFtFont(FtFontNo, height, width)	139
GetNoOfBmFonts()	140
GetNoOfFtFonts()	141
CacheFtChars(StartChar, EndChar)	142
SetFtUnibase(UnicodeBase)	143
14 Text Orientation Functions	144
TextNorth()	145
TextEast()	146
TextSouth()	147

TextWest()	148
SetFtAngle(Angle)	149
15 Bitmap Functions	150
PutPictNo(PictNo)	151
PutPictNo(x, y, PictNo)	152
GetPictHeight(PictNo)	153
GetPictWidth(PictNo)	154
16 Backlight Functions	155
LightOn()	156
LightOff()	157
LightBright(brightness)	158
17 Screen Capture Functions	159
SdScreenCapture()	160
18 Time Functions	161
Get_ms()	162
Wait_ms(ms)	163
SetTime(time)	164
19 Timer Management Functions	165
Timer(msec, LuaTimerFunc, Id)	166
Timer(msec, LuaTimerFunc)	167
TimerStart(Id)	168
TimerStop(Id)	169
20 Touch Function	170
GetTouchX()	171
GetTouchY()	172
TouchDn()	173
SetTouchEvent(luaTouchFunc)	174
21 Input/Output Functions	175
SD Card Access	176
RS232 Functions	177
RS232 Open: Event Mode	178
Rs232Open(RcvFunc)	178
Rs232Open(RcvFunc, BaudRate)	179
Rs232Open(RcvFunc, BaudRate, Parity)	180
Rs232Open(RcvFunc, BaudRate, Parity, StopBits)	181
Rs232Open(RcvFunc, BaudRate, Parity, StopBits, HandShake)	182
RS232 Open: Buffer Mode	183
Rs232Open()	183
RS232Open(BaudRate)	184
RS232Open(BaudRate, Parity)	185
RS232Open(BaudRate, Parity, StopBits)	186
RS232Open(BaudRate, Parity, StopBits, HandShake)	187
Rs232Close()	188
RS232 Transmit	189
Rs232Tx(Data)	189
Rs232Tx(Data, MaxLen)	190
Rs232TxStr(Str)	191
RS232 Receive	192
Rs232RxLen()	192
Rs232getc()	193
I2C Functions	194
I2Copen(TimeLoNs, TimeHiNs)	195

I2CwriteStart(Address, Data).....	196
I2CwriteStart(Address, Data, Stop).....	197
I2CwriteNext(Data).....	198
I2CwriteNext(Data, Stop).....	199
I2CreadStart(Address).....	200
I2CreadStart(Address, Stop).....	201
I2CreadNext().....	202
I2CreadNext(Stop).....	203
PIN Functions	204
SetPinInp(PinNo).....	205
SetPinInp(PinNo, PullUp).....	206
SetPinsInp(PinsMask).....	207
SetPinsInp(PinsMask, PullUpMask).....	208
SetPinOut(PinNo).....	209
SetPinOut(PinNo, OpenDrain).....	210
SetPinsOut(PinsMask).....	211
SetPinsOut(PinsMask, OpenDrainMask).....	212
SetPinIntr(PinNo, LuaFunction).....	213
RestorePin(PinNo).....	214
RestorePins(PinsMask).....	215
Pin(PinNo).....	216
Pin(PinNo, Value).....	217
Pins(PinsMask).....	218
Pins(PinsMask, Value).....	219
22 Advanced Topics	220
Frame Management Functions	221
SetDispFrame(FrameNo).....	222
SetDispFrame(FrameNo, Sync).....	223
GetDispFrame().....	224
GetNextDispFrame().....	225
SetDrawFrame(FrameNo).....	226
GetDrawFrame().....	227
GetNoOfFrames().....	228
CopyFrame(DestFrame, SourceFrame).....	229
MergeFrame(DestFrame, SourceFrame).....	230
CopyRect(DestFrame, SourceFrame, DestX, DestY, SourceX, SourceY, width, height).....	231
MergeRect(DestFrame, SourceFrame, DestX, DestY, SourceX, SourceY, width, height).....	232
Miscellaneous Functions	233
Peek32(address).....	234
Peek16(address).....	235
Peek8(address).....	236
Poke32(address, data).....	237
Poke16(address, data).....	238
Poke8(address, data).....	239
ExitReq().....	240

GLOSSARY

Release History

Date	Description
03-OCT-2008	Initial Release
09-DEC-2008	Updated for firmware Rev: 2.20: <ul style="list-style-type: none">• Added Constants descriptions• Divided RS232 functions into chapters• Added RS232 Event and Buffer modes descriptions• Modified existing Rs232Open function descriptions and moved them to the new "RS232 Open: Event Mode" chapter.• Added "RS232 Open: Buffer Mode" chapter• Modified description of Rs232Tx (Data) function• Added description of Rs232Tx (Data, MaxLen) function• Added description of Rs232RxLen () function• Added description of Rs232getc () function Corrected and formatted some Lua source code examples

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

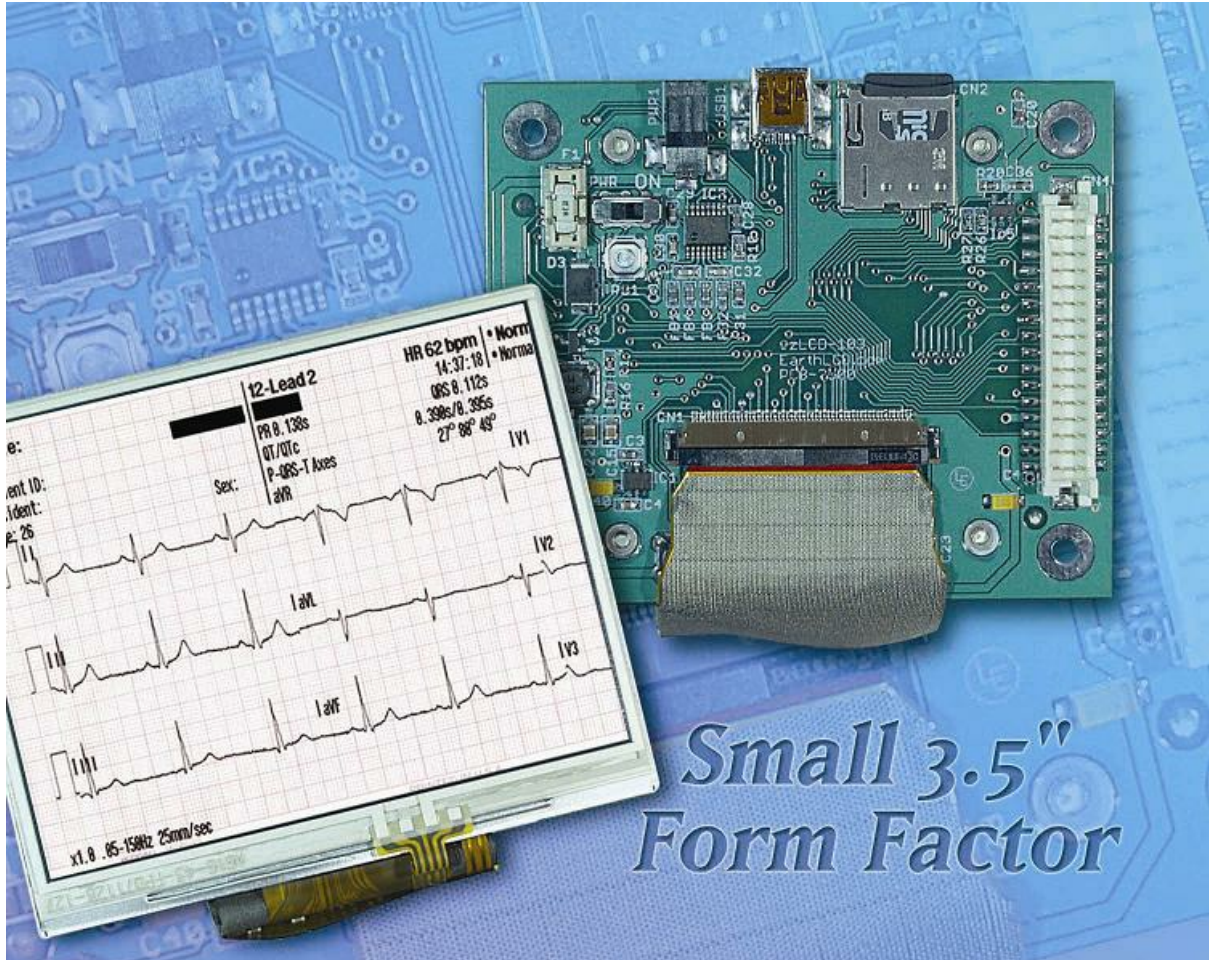
You can download the latest version of this manual at <http://www.ezlcd.com/support/>

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+101	ezLCD+102	ezLCD+103	ezLCD+105
Display Size	10.4"	6.4"	3.5"	8.0"
Screen Resolution	640x480	640x480	320x240	800x600
Flash	8MB	8MB	8MB	8MB
DRAM (minimum)	16MB	16MB	16MB	16MB

Quick Start

Quick Start Requirements:

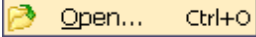
- PC Computer with at least 1 USB 2.0 port
- Windows XP SP2, or Windows Server 2003, or any Windows Vista or Windows Server 2008

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


Quick Start

1. Download the latest USB FAVR-32 driver from <http://www.ezlcd.com/support/>
2. Run the downloaded driver installation executable before connecting ezLCD+ to the USB of your computer.
3. 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.
4. Go to chapter: "[Quick Start: Lua](#)".

1 Quick Start: Lua

1. Make sure, that [USB FAVR-32](#) driver is installed on your PC
2. Download the setup of "ezLuaIDE" from <http://www.ezlcd.com/support/>
3. Install "ezLuaIDE" by running the downloaded setup
4. Turn-on ezLCD+ and make sure that it is connected to your computer through USB.
5. Run "ezLuaIDE". From the Menu, select "File" - "Open"  **Open...** **Ctrl+O**
6. Select HelloWorld.lua file from the folder "Program Files\ezLuaIDE\Examples".

```
>HelloWorld.lua x
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 !")
```

7. Press  **Run** 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+10x Manual" and updates are available at <http://www.ezLCD.com/support> .

Drawing on the ezLCD+

1 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

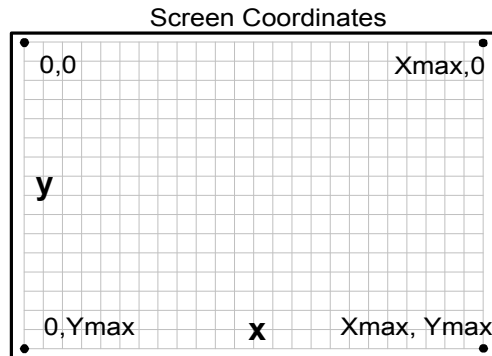
Use

```
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 values 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 [Plot 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 SET [Position Functions](#)
- Updated by drawing commands

Current Color.

- Set by [SetColor](#)
- [Bitmaps](#) are not affected

Background Color.

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

Transparent Color.

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

[Transparency](#)

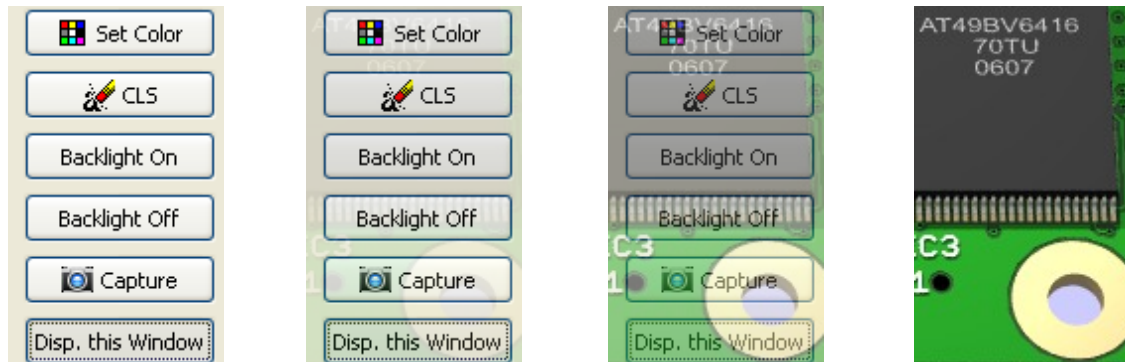
- Set by [SetAlpha](#)

[Pen](#)

- Set by [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 [SetPenSize](#) allows setting of the pen height and width.

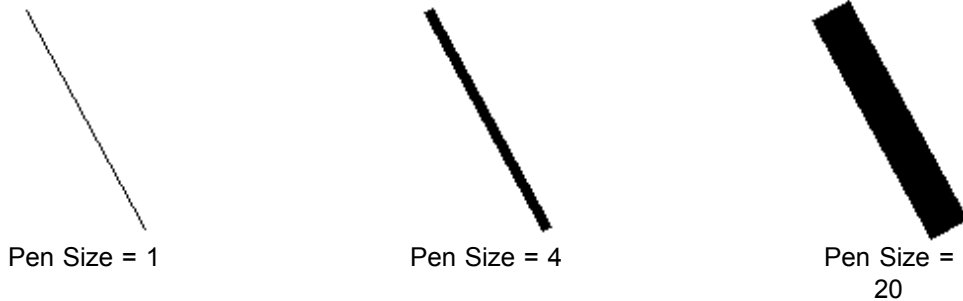
Pen Width specifies the horizontal dimension the drawing line (in pixels).

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

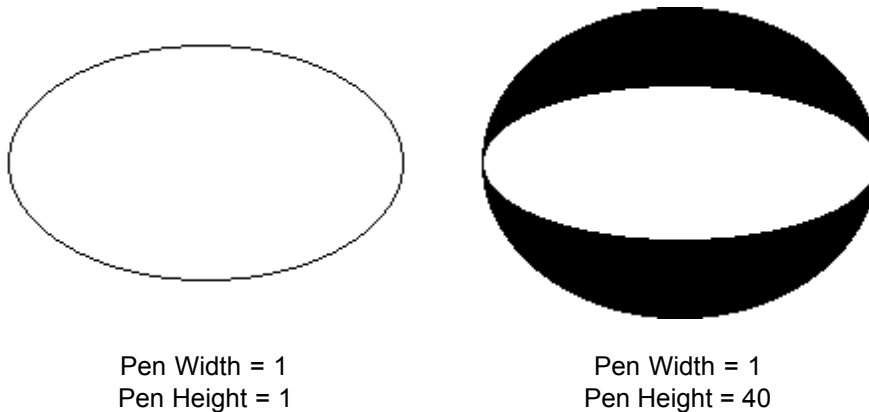
Notes:

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 "*ezLCD+10x 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 "*ezLCD+10x Manual*".

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

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

The following bitmap fonts are installed in the ezLCD+, when it is shipped:

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

6.2 True Type Fonts

The ezLCD+ True Type Fonts can reside in the User ROM, which is described in the *"ezLCD+10x 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 *"ezLCD+10x 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

The following True Type Fonts are installed in the ezLCD+, when it is shipped:

The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog

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.

Icons

ICONS are numbered from 0 to 65534 and are stored in the user ROM. They may be .bmp, .jpg or .ezp.

Lengths

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

Fonts

Fonts are numbered from 0 to 65534 and are stored in the user ROM. They may be bitmap fonts or Free Type (TrueType) fonts.

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)

1 Constants

Name	Type	Description
Width	Integer	Width of the screen in pixels
Height	Integer	Height of the screen in pixels
BytesPerPixel	Integer	Number of bytes per pixel
FirmVer	Number	ezLCD+ firmware version
LuaVer	Number	Lua version
NoOfFrames	Integer	Number of available full-screen frames
NoOfPicts	Integer	Number of pictures in the User ROM
NoOfBmFonts	Integer	Number of bitmap fonts in the User ROM
NoOfFtFonts	Integer	Number of true type fonts in the User ROM
NoOfLuaPgms	Integer	Number of Lua programs User ROM
RomSize	Integer	Total size of the User ROM in bytes
RomUsed	Integer	Number of bytes in the User ROM used for pictures, fonts and Lua programs
RomFree	Integer	Number of bytes available in the User Rom

2 Position Functions

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

Screen Addresses

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

2.1 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](#)

2.2 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](#)

2.3 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](#)

2.4 GetX()

Purpose

To return the current x location.

Argument List

None

Return Value

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

Reference

[Screen Coordinates](#)

2.5 GetY()

Purpose

To return the current y location.

Argument List

None

Return Value

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

Reference

[Screen Coordinates](#)

3 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.

3.1 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.

3.2 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
-----	---------	--------------------------------------

3.3 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
-------	---------	--

3.4 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
------	---------	---------------------------------------

3.5 SetColor(ezLCDcolor)

Purpose

To set the current color.

Argument List

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

Return Value

None

3.6 SetBgColor(ezLCDcolor)

Purpose

To set the background color. This color is used for the functions PrintCharBg and PrintStringBg.

Argument List

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

Return Value

None

3.7 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

3.8 GetPixel()

Purpose

To get the color at the current screen position.

Argument List

None

Return Value

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

3.9 GetPixel(x, y)

Purpose

To get the color at the specified screen position.

Argument List

x	Integer	x screen position
y	Integer	y screen position

Return Value

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

4 Transparency Functions

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

4.1 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

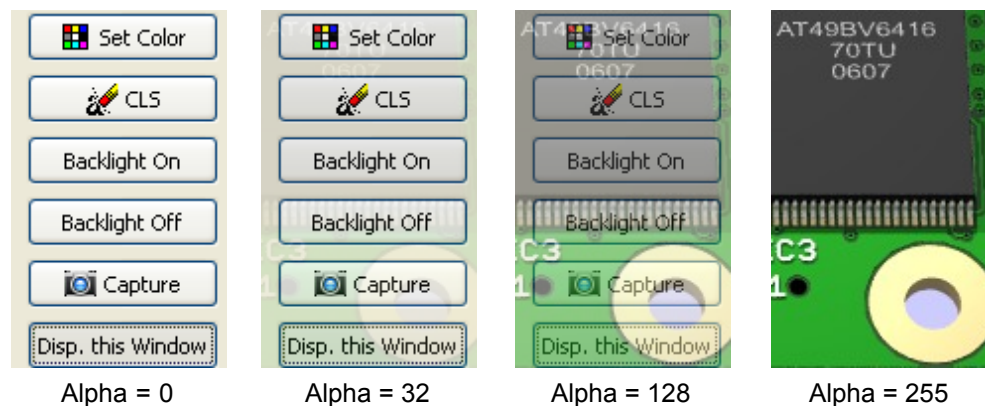
None

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](#)

4.2 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](#)

4.3 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](#)

5 Pen Size Functions

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

5.1 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](#)

6 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).

6.1 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
------------------	---------	---

6.2 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

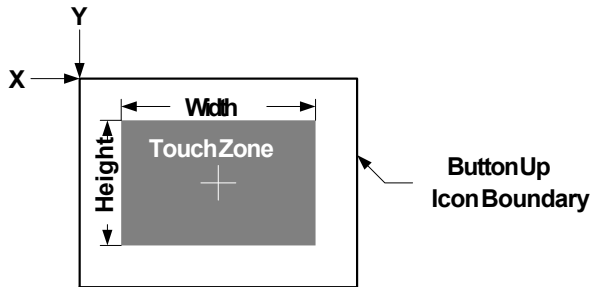
7 Button Functions

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

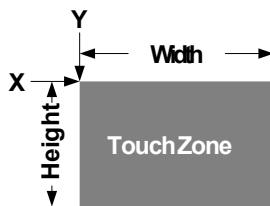
About the Touch Zone:

- 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)

7.1 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

7.2 Button(ID, iState, iconUp, iconDown, iconDisabled, x, y, width, height)

Purpose

Define a button where the button touch area is different than the icon sizes

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 Icon Number in User ROM to be displayed when the button state is UP (1). Specify -1 or FFFF hex for no icon.
iconDown	Integer	The Icon Number in User ROM to be displayed when the button state is DOWN (2). Specify -1 or FFFF hex for no icon.
iconDisabled	Integer	The Icon Number in User ROM 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.
height	Integer	The height of the button touch area in pixels.

Return Value

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

Notes

The Button is deleted if iState = 5

7.3 Button(ID, iState, iconUp, iconDown, iconDisabled, x, y)

Purpose

Define a button where the button touch area are equal to the size of the iconUp image.

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 Icon Number in User ROM to be displayed when the button state is UP (1). Specify -1 or FFFF hex for no icon.
iconDown	Integer	The Icon Number in User ROM to be displayed when the button state is DOWN (2). Specify -1 or FFFF hex for no icon.
iconDisabled	Integer	The Icon Number in User ROM 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.

Return Value

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

Notes

No button will be created when iconUp is -1 or invalid.
The Button is deleted if iState = 5

7.4 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)
---------	---------	-------------------------------------

7.5 DelButtons()

Purpose

Delete all buttons

Argument List

None

Return Value

None

7.6 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
```

8 Fill Area Functions

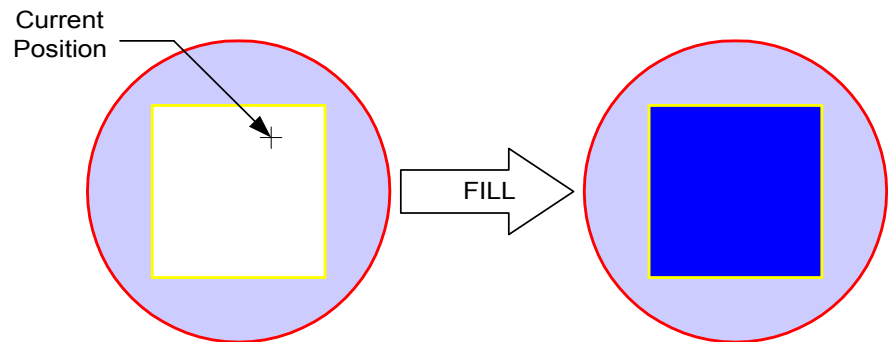
The following section details the functions used to fill screen areas on the ezLCD display

8.1 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.

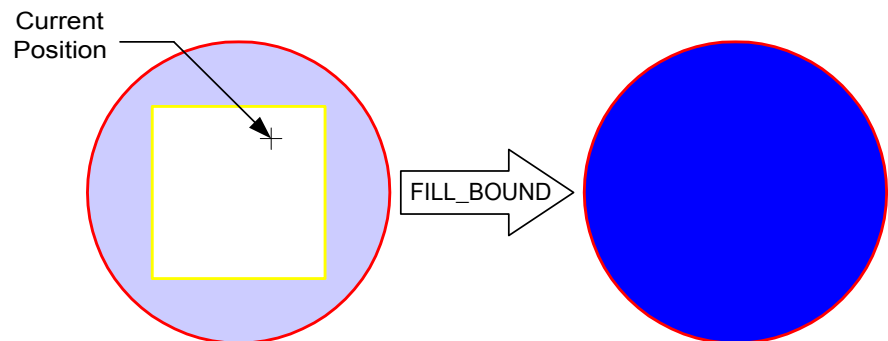
In the following example, the start position is located in the white area and the fill color is blue.



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.



8.2 Cls()

Purpose

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

Argument List

None

Return Value

None

8.3 CIs(ezLCDcolor)

Purpose

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

Argument List

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

Return Value

None

8.4 Fill()

Purpose

To fill an area with the current color. The Fill area is defined as the current position and includes all adjoining pixels that are the same color as the existing (pre-filled) color.

Argument List

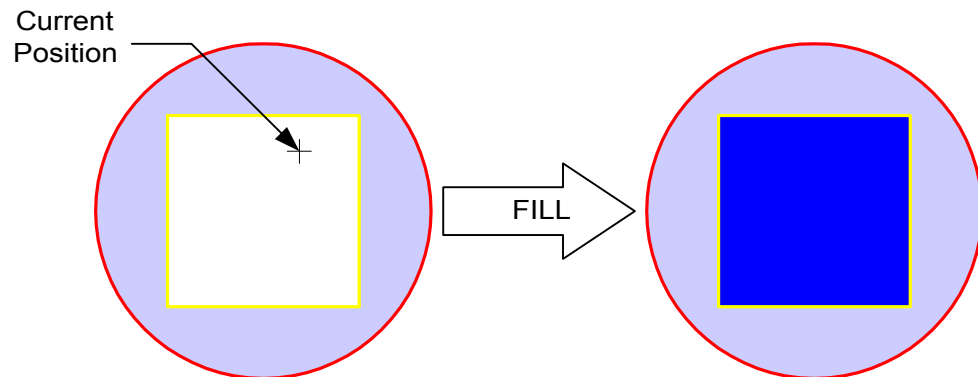
None

Return Value

None

Example

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



8.5 Fill(ezLCDcolor)

Purpose

To fill an area with the specified color. The Fill area is defined as the current position and includes all adjoining pixels that are the same color as the existing (pre-filled) color.

Argument List

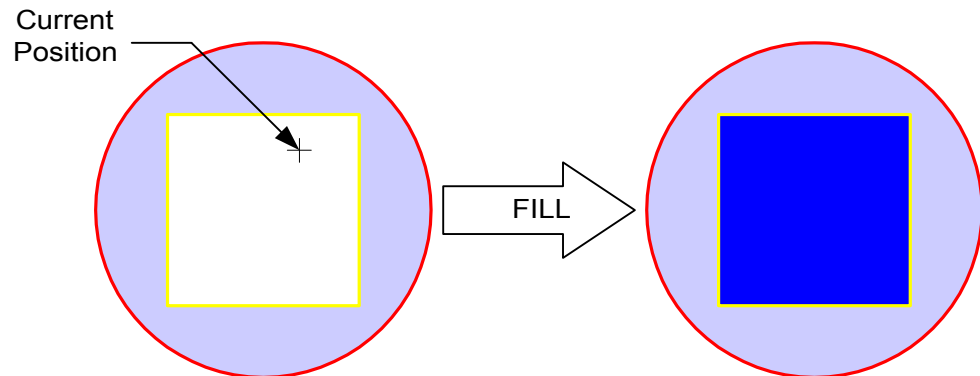
ezLCDcolor	Integer	RGB color code
------------	---------	----------------

Return Value

None

Example

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



8.6 Fill(x, y)

Purpose

To fill an area with the current 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

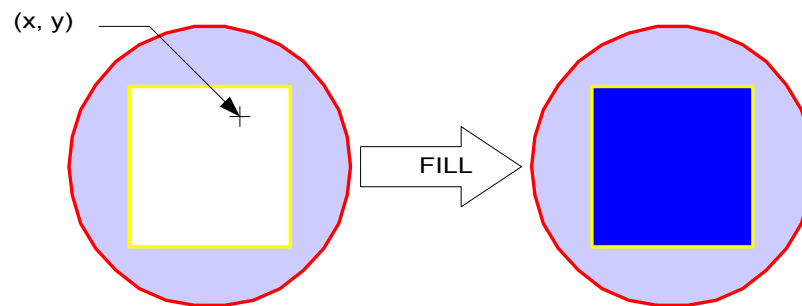
x	Integer	x screen position where fill is to begin
y	Integer	y screen position where fill is to begin

Return Value

None

Example

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



8.7 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

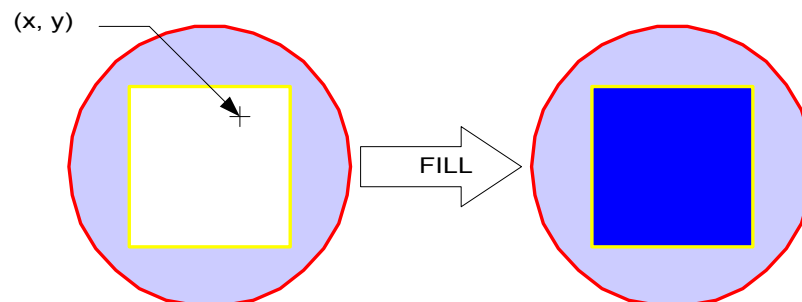
x	Integer	x screen position where fill is to begin
y	Integer	y screen position where fill is to begin
FillColor	Integer	ezLCDcolor code

Return Value

None

Example

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



8.8 FillBound(BoundColor)

Purpose

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

Argument List

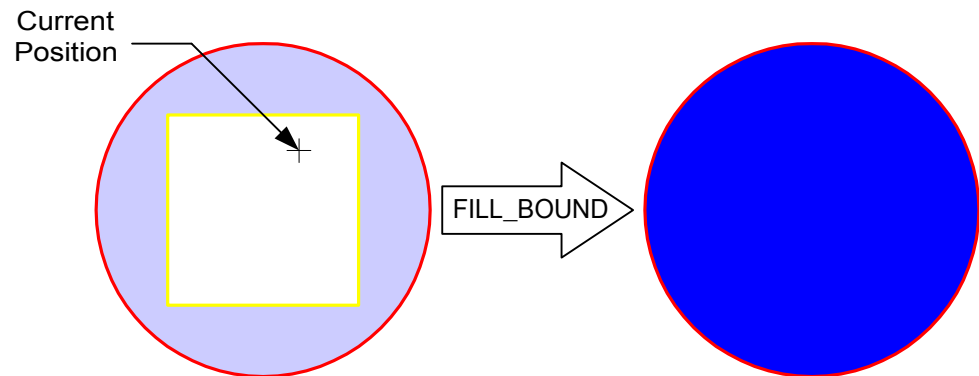
BoundColor	Integer	ezLCD color code
------------	---------	------------------

Return Value

None

Example

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



8.9 FillBound(BoundColor, FillColor)

Purpose

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

Argument List

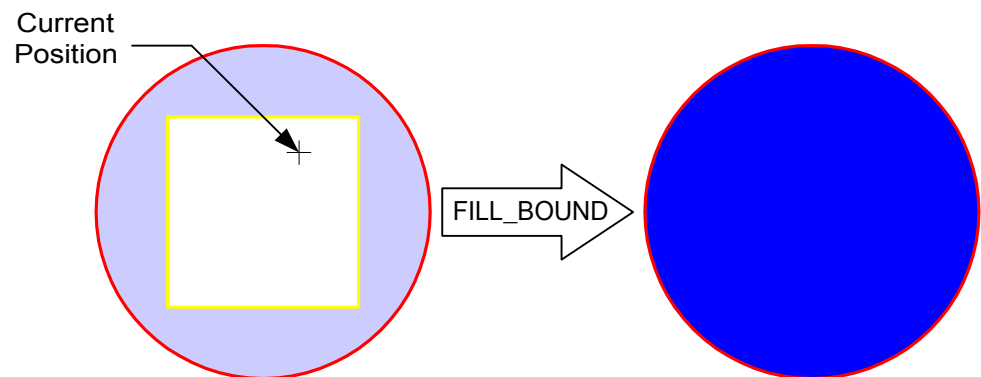
BoundColor	Integer	ezLCD color used to define the bounding area
FillColor	Integer	ezLCD color to be used for filling

Return Value

None

Example

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



8.10 FillBound(x, y, BoundColor)

Purpose

To fill an area of the screen with the current color. 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

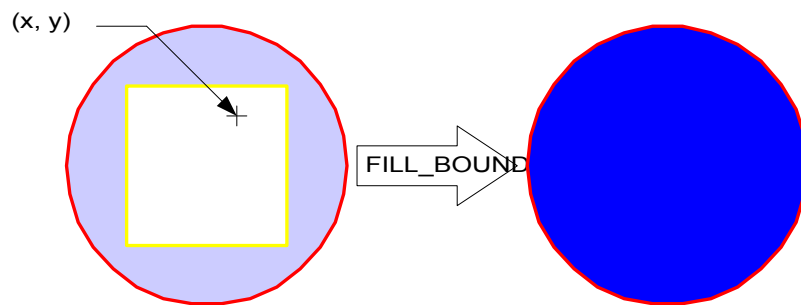
x	Integer	x screen position where fill is to begin
y	Integer	y screen position where fill is to begin
BoundColor	Integer	ezLCD color used to define the bounding area

Return Value

None

Example

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



8.11 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

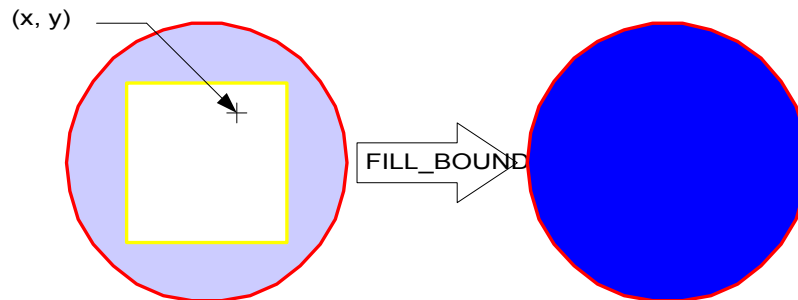
x	Integer	x screen position where fill is to begin
y	Integer	y screen position where fill is to begin
BoundColor	Integer	ezLCD color used to define the bounding area
FillColor	Integer	ezLCD color to be used for filling

Return Value

None

Example

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



9 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](#).

9.1 HLine(x2)

Purpose

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

Argument List

x2	Integer	x position where drawn line ends.
----	---------	-----------------------------------

Return Value

None

9.2 HLine(x2, color)

Purpose

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

Argument List

x2	Integer	x position where drawn line ends.
color	Integer	ezLCD color value to draw

Return Value

None

9.3 HLine(x1, y1, x2)

Purpose

To draw a horizontal line using the current 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 [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.

Return Value

None

9.4 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 [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

9.5 VLine(y2)

Purpose

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

Argument List

y2	Integer	y position where drawn line ends.
----	---------	-----------------------------------

Return Value

None

9.6 VLine(y2, color)

Purpose

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

Argument List

y2	Integer	y position where drawn line ends.
color	Integer	RGB color value to draw

Return Value

None

9.7 VLine(x1, y1, y2)

Purpose

To draw a vertical line using the current 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 [SetPenSize](#).

Argument List

x1	Integer	x position where to start the line.
y1	Integer	y position where to start the line.
y2	Integer	y position where drawn line ends.

Return Value

None

9.8 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 [SetPenSize](#).

Argument List

x1	Integer	x position where to start the line.
y1	Integer	y position where to start the line.
y2	Integer	y position where drawn line ends.
color	Integer	ezLCD color value of line to draw

Return Value

None

9.9 Line(x2, y2)

Purpose

To draw a line using the current color from the current position (x, y) to the specified position (x2, y2). The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

x2	Integer	x position where drawn line ends.
y2	Integer	y position where drawn line ends

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.10 Line(x2, y2, color)

Purpose

To draw a line using the specified color from the current position (x, y) to the specified position (x2, y2). The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

x2	Integer	x position where drawn line ends
y2	Integer	y position where drawn line ends.
color	Integer	ezLCD color value to draw

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.11 Line(x1, y1, x2, y2)

Purpose

To draw a line using the current color from the specified position (x1, y1) to the specified position (x2, y2). The line size is defined by the width setting of the pen which is configured in [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.
y2	Integer	y position where drawn line ends.

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.12 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). The line size is defined by the width setting of the pen which is configured in [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.
y2	Integer	y position where drawn line ends.
color	Integer	ezLCD color value of line to draw

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.13 LineAng(angle, length)

Purpose

To draw a line using the current color from the current position (x, y) at the angle specified in [ezLCD angle units](#) for the specified length. The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

angle	Integer	angle to draw line in ezLCD angle units.
length	Integer	length of line to be drawn in pixels.

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.14 LineAng(angle, length, color)

Purpose

To draw a line using the specified color from the current position (x, y) at the angle specified in [ezLCD angle units](#) in degrees for the specified length. The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

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.

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.15 LineAng(x1, y1, angle, length)

Purpose

To draw a line using the current color from the specified position (x1, y1) at the angle specified in [ezLCD angle units](#) for the specified length. The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

x1	Integer	line x start position
y1	Integer	line y start position
angle	Integer	angle to draw line in ezLCD angle units.
length	Integer	length of line to be drawn in pixels.

Return Value

None

Notes

The line height and width are set by the SetPenSize function

9.16 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 angle units](#) for the specified length. The line size is defined by the width setting of the pen which is configured in [SetPenSize](#).

Argument List

x1	Integer	line x start position
y1	Integer	line y start 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.

Return Value

None

Notes

The line height and width are set by the SetPenSize function

10 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](#).

10.1 Circle(radius)

Purpose

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

Argument List

radius	real	radius of circle to be drawn
--------	------	------------------------------

Return Value

None

10.2 Circle(radius, color)

Purpose

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

Argument List

radius	real	radius of circle.
color	Integer	ezLCD color of circle

Return Value

None

10.3 Circle(x, y, radius)

Purpose

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

Argument List

x	Integer	x location of circle center
y	Integer	y location of circle center
radius	real	radius of circle.

Return Value

None

10.4 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 [SetPenSize](#).

Argument List

x	Integer	x location of circle center
y	Integer	y location of circle center
radius	real	radius of circle.
color	Integer	ezLCD color

Return Value

None

10.5 CircleFill(radius)

Purpose

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

Argument List

radius	real	radius of circle.
--------	------	-------------------

Return Value

None

10.6 CircleFill(radius,color)

Purpose

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

Argument List

radius	real	radius of circle.
color	Integer	ezLCD color of circle

Return Value

None

10.7 CircleFill(x, y, radius)

Purpose

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

Argument List

x	Integer	x location of circle center
y	Integer	y location of circle center
radius	real	radius of circle.

Return Value

None

10.8 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
y	Integer	y location of circle center
radius	real	radius of circle.
color	Integer	ezLCD color

Return Value

None

10.9 Ellipse(a, b)

Purpose

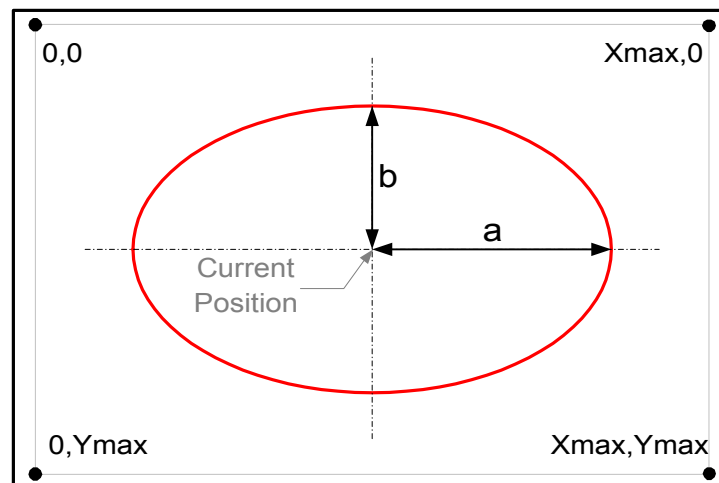
To draw an ellipse using the current color centered at the current position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b . The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

Argument List

a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.

Return Value

None



10.10 Ellipse(a, b, color)

Purpose

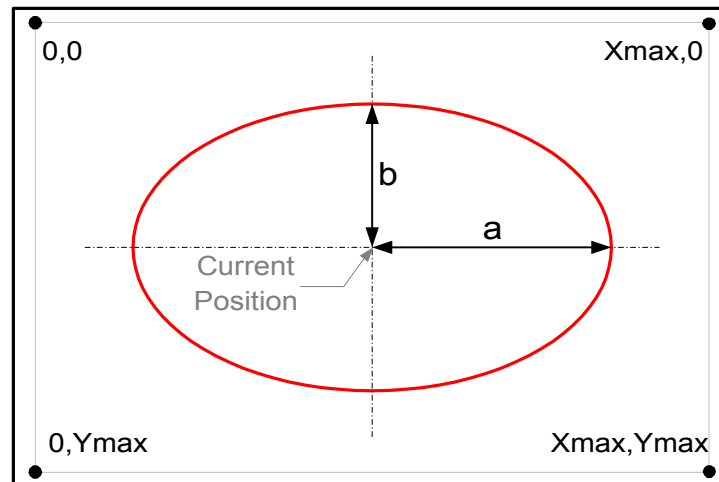
To draw an ellipse using the specified color centered at the current position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b. The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

Argument List

a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color

Return Value

None



10.11 Ellipse(x, y, a, b)

Purpose

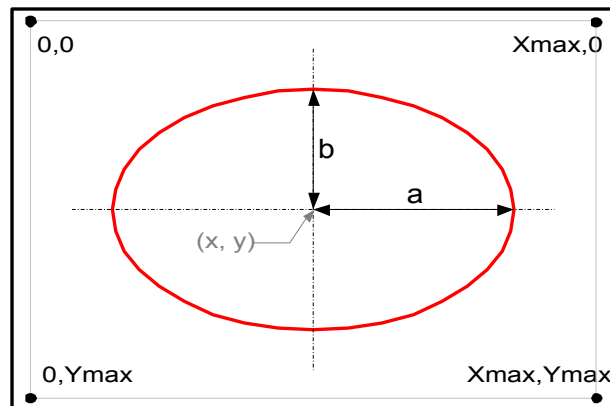
To draw an ellipse using the current color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b. The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

Argument List

x	Integer	x position of ellipse center.
y	Integer	y position of ellipse center.
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.

Return Value

None



10.12 Ellipse(x, y, a, b, color)

Purpose

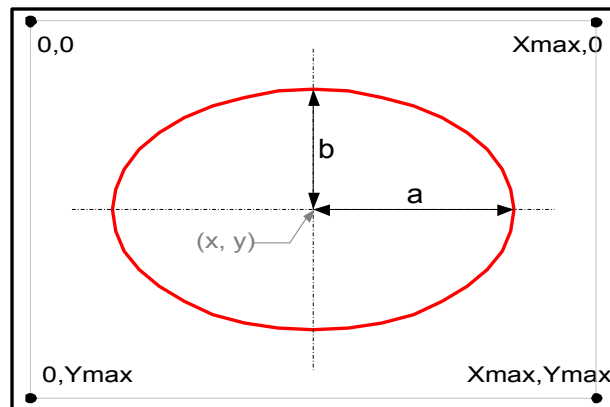
To draw an ellipse using the specified color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b . The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

Argument List

x	Integer	x position of ellipse center.
y	Integer	y position of ellipse center.
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color of ellipse.

Return Value

None



10.13 EllipseFill(a, b)

Purpose

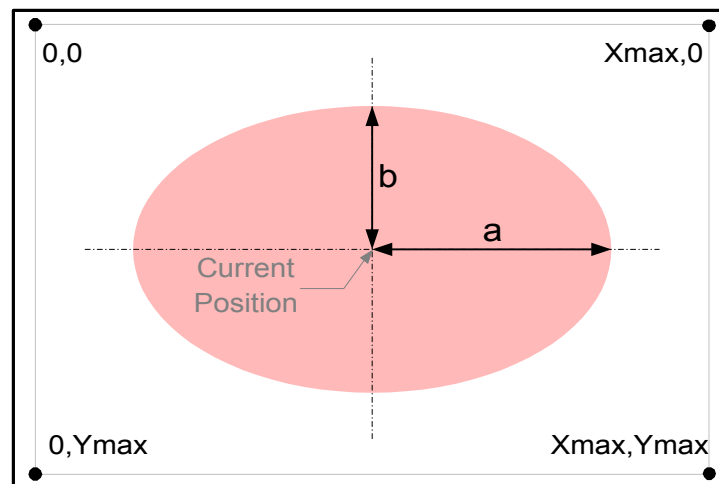
To draw a filled ellipse using the current color centered at the current position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.

Return Value

None



10.14 EllipseFill(a, b, color)

Purpose

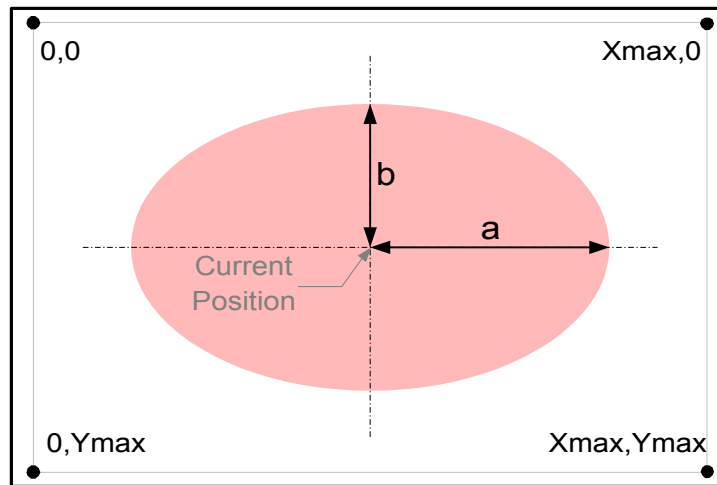
To draw a filled ellipse using the specified color centered at the current position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color of ellipse.

Return Value

None



10.15 EllipseFill(x, y, a, b)

Purpose

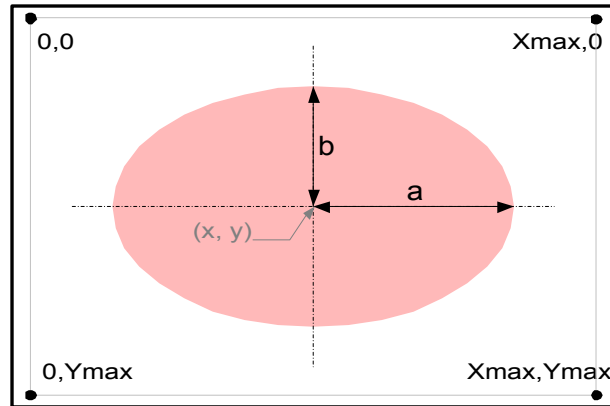
To draw a filled ellipse using the current color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

x	Integer	x position of ellipse center.
y	Integer	y position of ellipse center.
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.

Return Value

None



10.16 EllipseFill(x, y, a, b, color)

Purpose

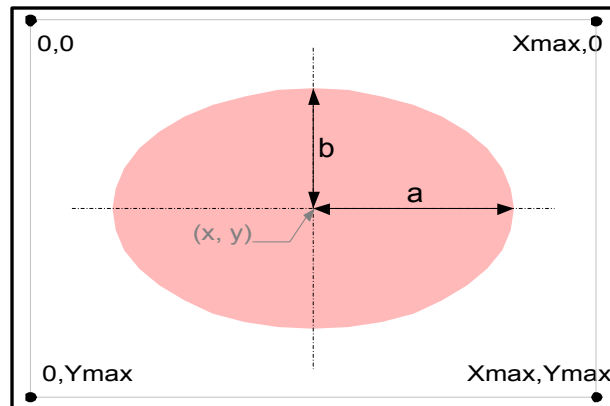
To draw a filled ellipse using the specified color centered at the specified position with the length of the horizontal semi-axis of a and the length of the vertical semi-axis of b.

Argument List

x	Integer	x position of ellipse center.
y	Integer	y position of ellipse center.
a	Integer	horizontal semi-axis length.
b	Integer	vertical semi-axis length.
color	Integer	ezLCD color of ellipse.

Return Value

None



10.17 Arc(radius, StartAng, EndAng)

Purpose

To draw an arc using the current color centered at the current 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 [SetPenSize](#).

Argument List

radius	real	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.

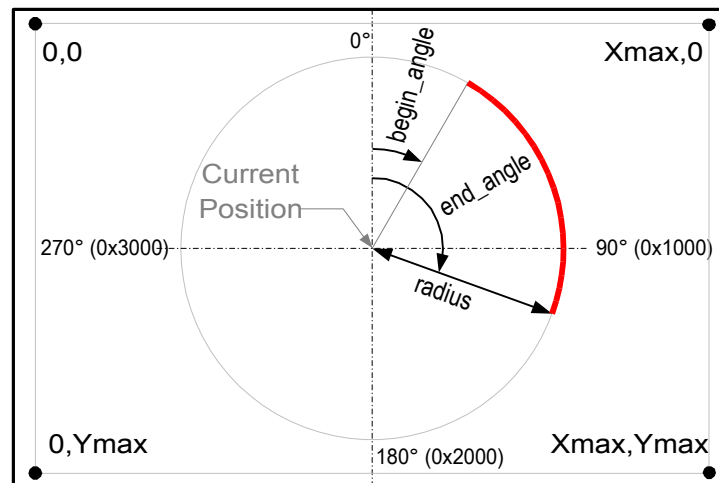
Return Value

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).



10.18 Arc(radius, StartAng, EndAng, color)

Purpose

To draw an arc using the specified color centered at the current 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 [SetPenSize](#).

Argument List

radius	real	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

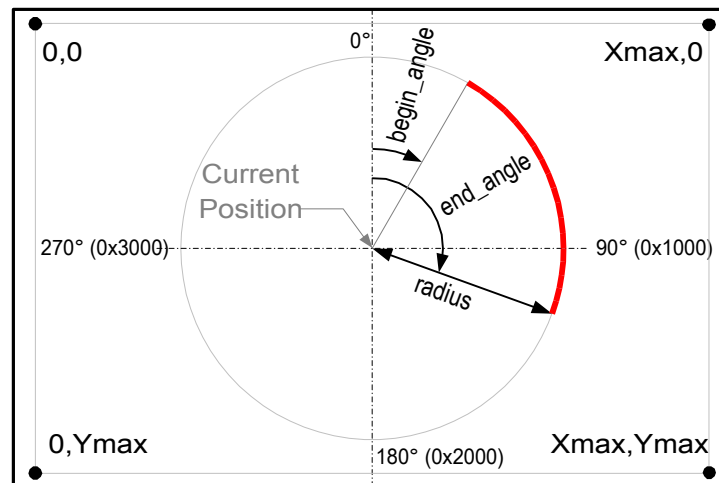
Return Value

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).



10.19 Arc(x, y, radius, StartAng, EndAng)

Purpose

To draw an arc using the current 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 [SetPenSize](#).

Argument List

x	Integer	x location of arc center
y	Integer	y location of arc center
radius	real	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.

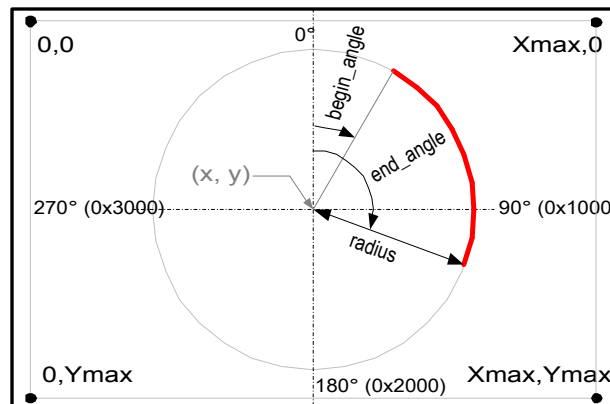
Return Value

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).



10.20 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 [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

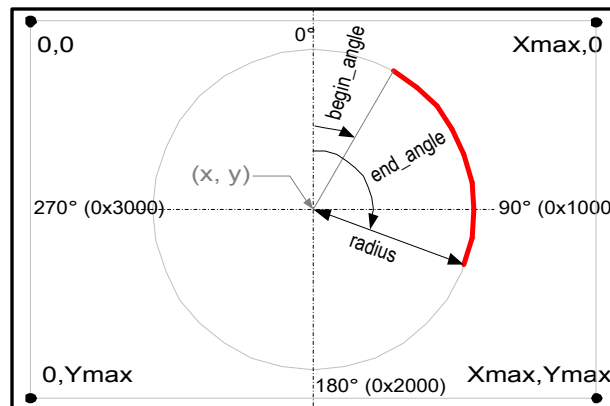
Return Value

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).



10.21 Pie(radius, StartAng, EndAng)

Purpose

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

Argument List

radius	real	radius of pie.
StartAng	Integer	Angle to begin pie at specified in ezLCD angle units.
EndAng	Integer	Angle to end pie at specified in ezLCD angle units.

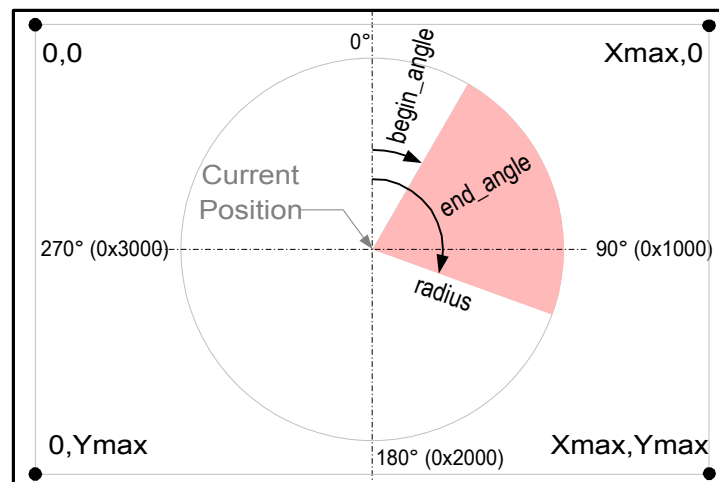
Return Value

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).



10.22 Pie(radius, StartAng, EndAng, color)

Purpose

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

Argument List

radius	real	radius of pie.
StartAng	Integer	Angle to begin pie at specified in ezLCD angle units.
EndAng	Integer	Angle to end pie at specified in ezLCD angle units.
color	Integer	ezLCD color

Return Value

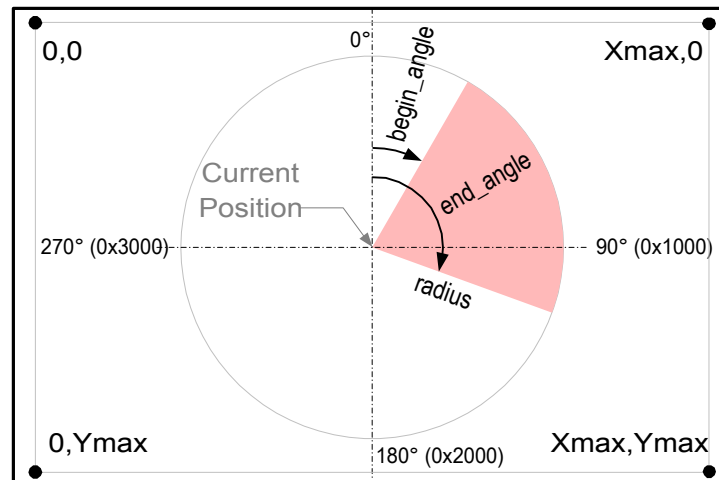
None

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).

Example



10.23 Pie(x, y, radius, StartAng, EndAng)

Purpose

To draw a pie filled arc using the current 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
y	Integer	y location of pie center
radius	real	radius of pie.
StartAng	Integer	Angle to begin pie at specified in ezLCD angle units.
EndAng	Integer	Angle to end pie at specified in ezLCD angle units.

Return Value

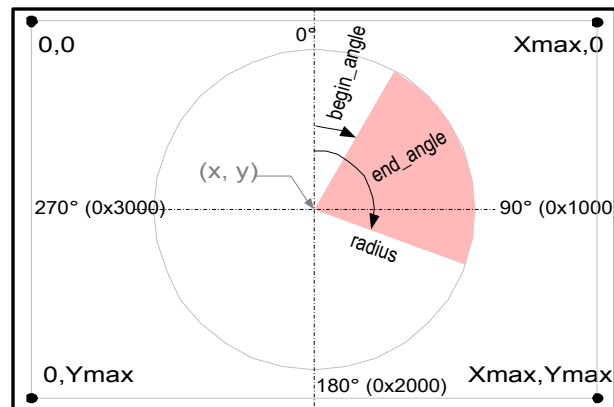
None

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).

Example



10.24 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
y	Integer	y location of pie center
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

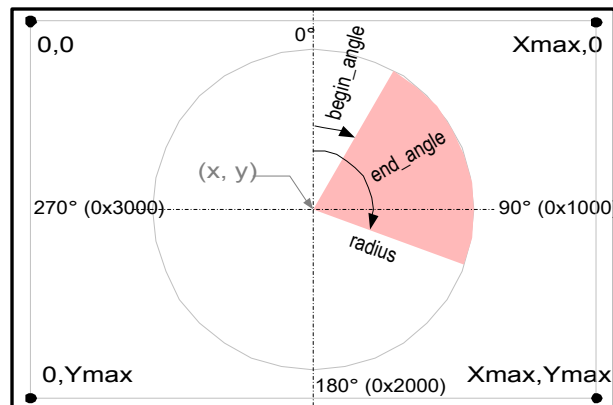
Return Value

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).



10.25 EllipseArc(a, b, StartAng, EndAng)

Purpose

To draw an Ellipse arc using the current color centered at the current position from angle StartAng to EndAng specified in [ezLCD angle units](#) 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 [SetPenSize](#).

Argument List

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.

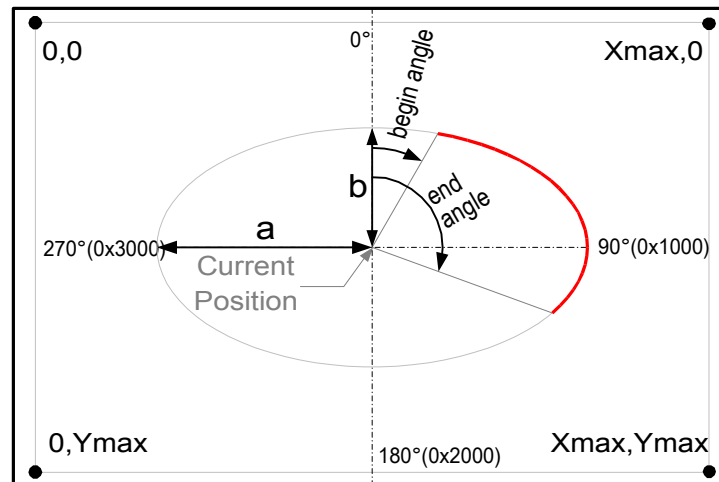
Return Value

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).



10.26 EllipseArc(a, b, StartAng, EndAng, color)

Purpose

To draw an Ellipse arc using the specified color centered at the current position from angle StartAng to EndAng specified in [ezLCD angle units](#) 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 [SetPenSize](#).

Argument List

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

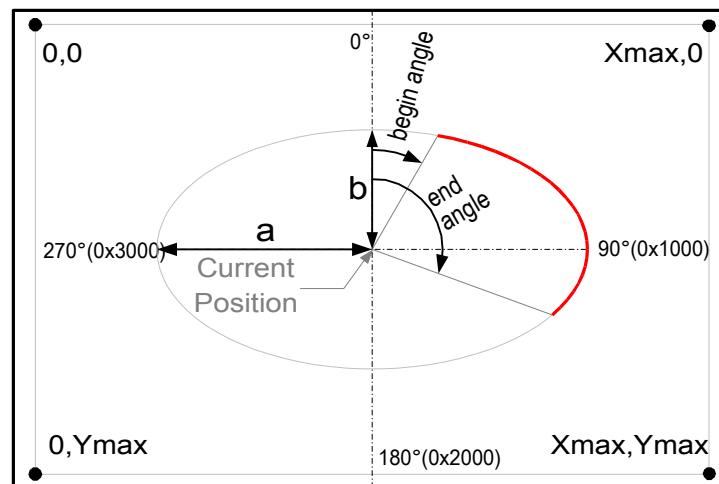
Return Value

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).



10.27 EllipseArc(x, y, a, b, StartAng, EndAng)

Purpose

To draw an Ellipse arc using the current color centered at the specified position from angle StartAng to EndAng specified in [ezLCD angle units](#) 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 [SetPenSize](#).

Argument List

x	Integer	x location of ellipse center
y	Integer	y location of ellipse center
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.

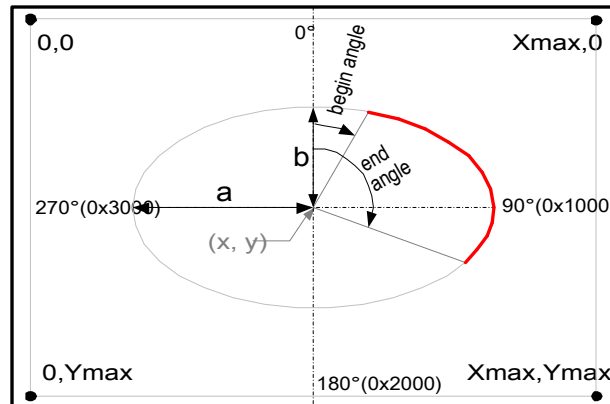
Return Value

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).



10.28 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 angle units](#) where the length of the horizontal semi-axis is a and the length of the vertical semi-axis is b. The thickness of the drawn line is computed using the pen width and height which is configured in [SetPenSize](#).

Argument List

x	Integer	x location of ellipse center
y	Integer	y location of ellipse center
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

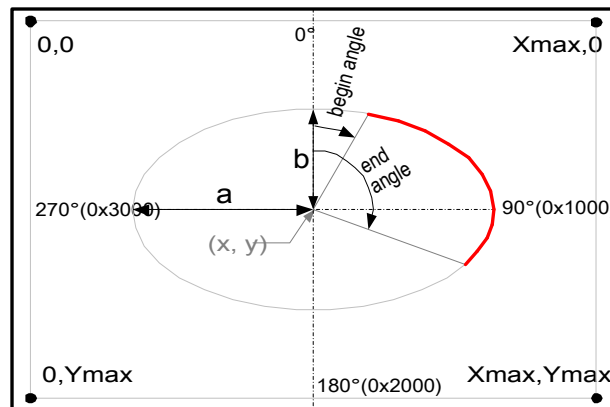
Return Value

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).



10.29 EllipsePie(a, b, StartAng, EndAng)

Purpose

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

Argument List

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.

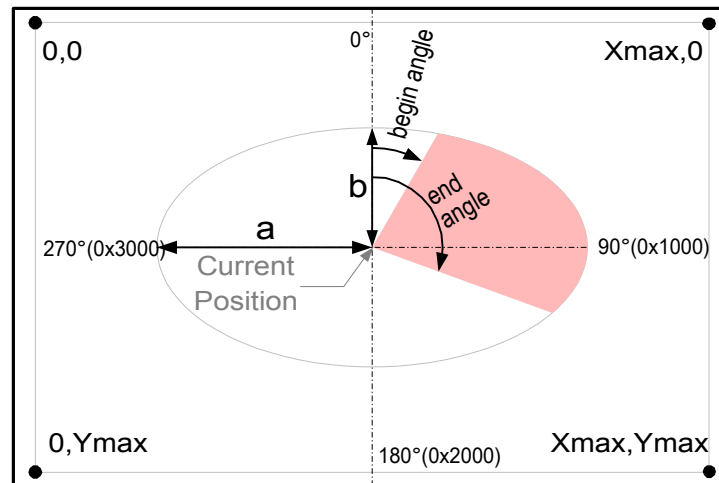
Return Value

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).



10.30 EllipsePie(a, b, StartAng, EndAng, color)

Purpose

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

Argument List

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

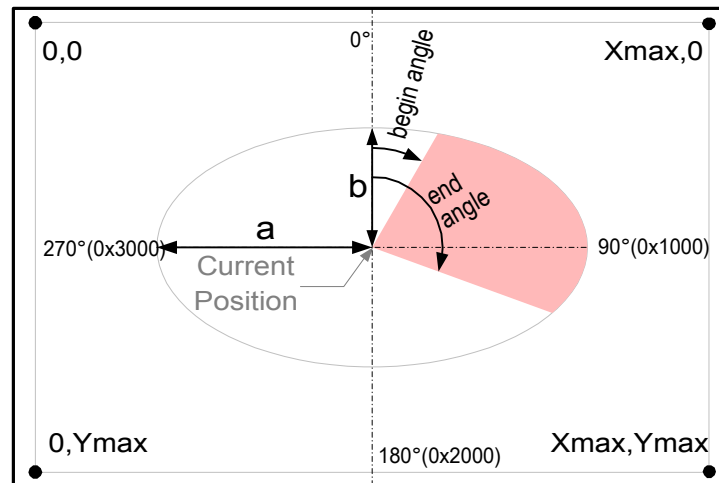
Return Value

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).



10.31 EllipsePie(x, y, a, b, StartAng, EndAng)

Purpose

To draw an Ellipse pie using the current color centered at the specified position from angle StartAng to EndAng specified in [ezLCD angle units](#) 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
y	Integer	y location of ellipse center
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.

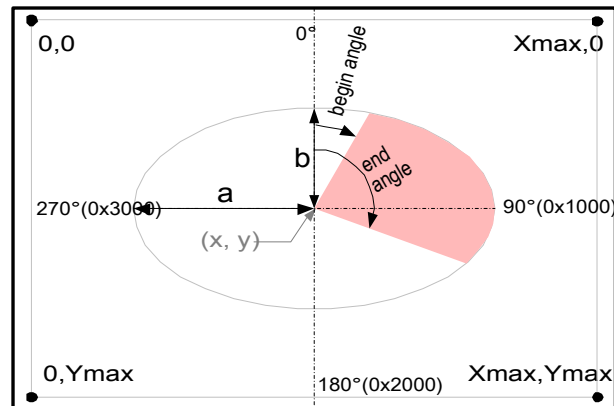
Return Value

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).



10.32 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 angle units](#) 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
y	Integer	y location of ellipse center
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

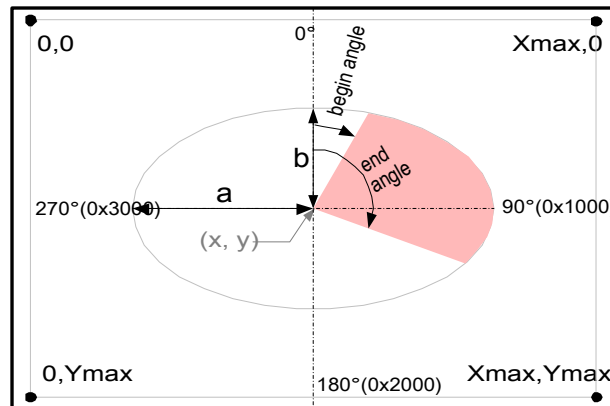
Return Value

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).



11 Polygon Drawing Functions

The following section details the functions used to draw polygons.

11.1 Box(x2, y2)

Purpose

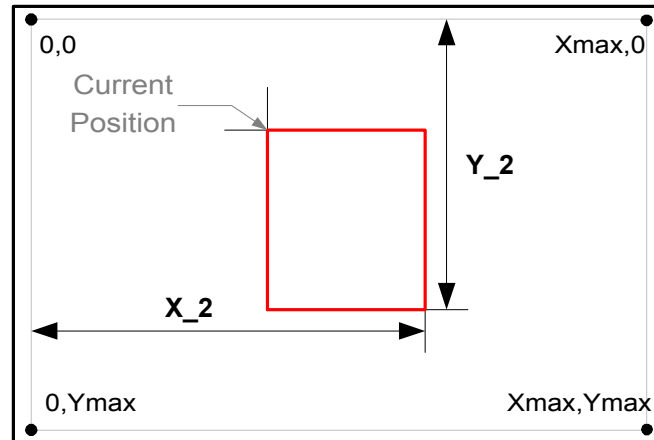
To draw a box using the current position as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the current color and the line size will be the current pen width.

Argument List

x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner

Return Value

None



Additional Reference

[SetPenSize](#)

11.2 Box(x2, y2, color)

Purpose

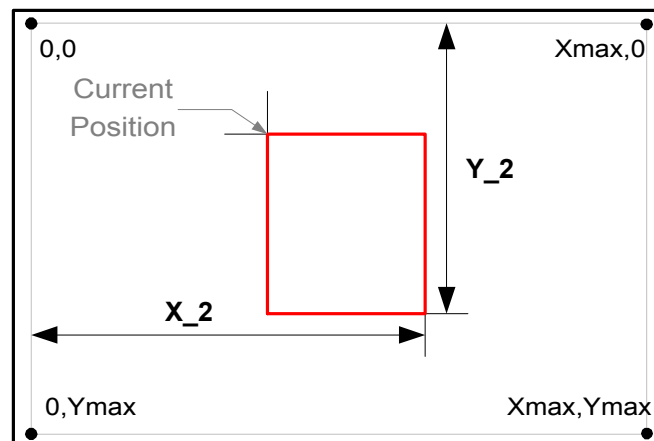
To draw a box using the current position as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color and the line size will be the current pen width.

Argument List

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

Return Value

None



Additional Reference

[SetPenSize](#)

11.3 Box(x1, y1, x2, y2)

Purpose

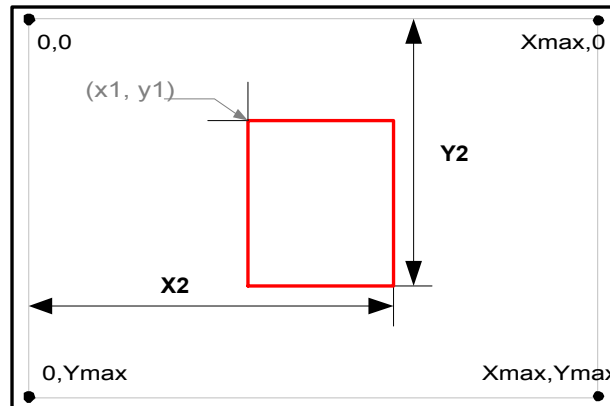
To draw a box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the current color and the line size will be the current pen width.

Argument List

x1	Integer	x screen position of starting corner
y1	Integer	y screen position of starting corner
x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner

Return Value

None



Additional Reference

[SetPenSize](#)

11.4 Box(x1, y1, x2, y2, color)

Purpose

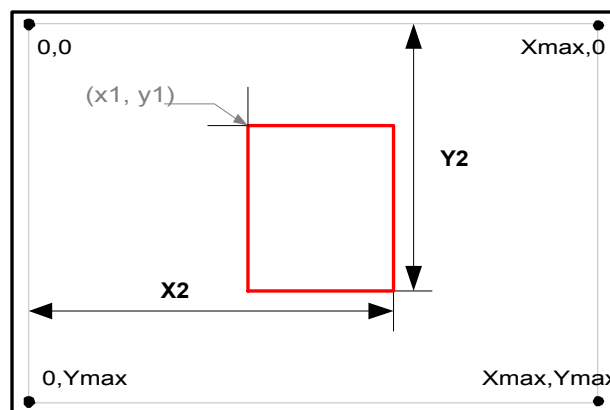
To draw a box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color and the line size will be the current pen width.

Argument List

x1	Integer	x screen position of starting corner
y1	Integer	y screen position of starting corner
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

Return Value

None



Additional Reference

[SetPenSize](#)

11.5 BoxFill(x2, y2)

Purpose

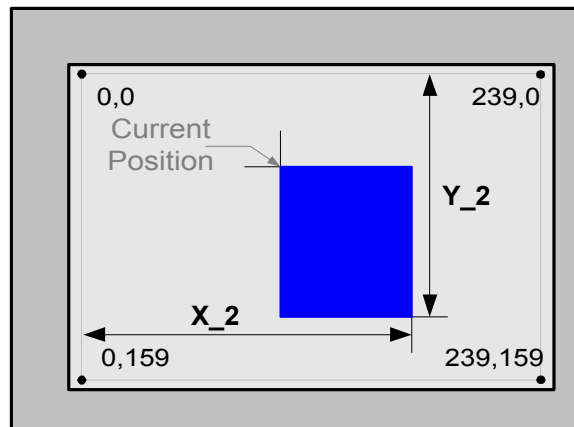
To draw a filled in box using the current position as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the current color.

Argument List

x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner

Return Value

None



11.6 BoxFill(x2, y2, color)

Purpose

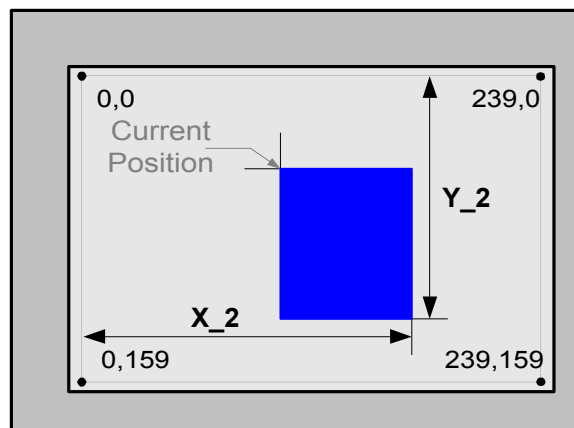
To draw a filled in box using the current position as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color.

Argument List

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

Return Value

None



11.7 BoxFill(x1, y1, x2, y2)

Purpose

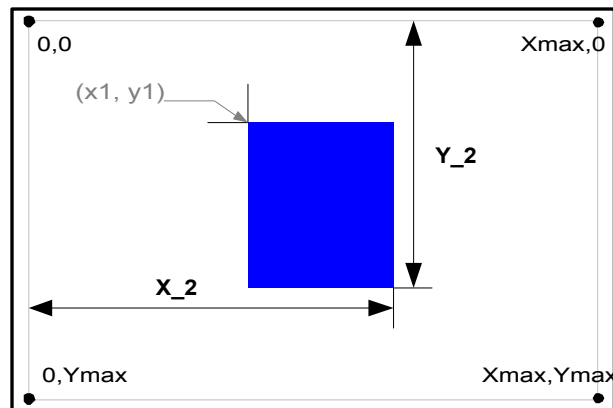
To draw a filled in box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the current color.

Argument List

x1	Integer	x screen position of starting corner
y1	Integer	y screen position of starting corner
x2	Integer	x screen position of ending corner
y2	Integer	y screen position of ending corner

Return Value

None



11.8 BoxFill(x1, y1, x2, y2, color)

Purpose

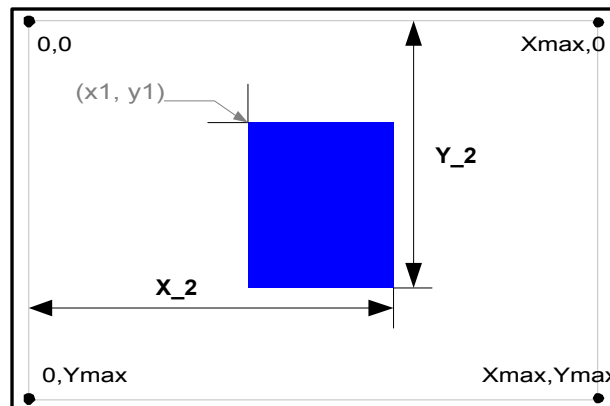
To draw a filled in box using the specified position (x1, y1) as the starting corner and the specified position (x2, y2) as the ending corner. The box will be drawn using the specified color.

Argument List

x1	Integer	x screen position of starting corner
y1	Integer	y screen position of starting corner
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

Return Value

None



11.9 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.

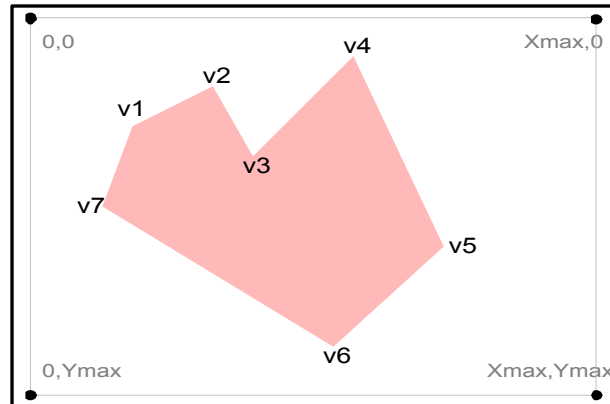
Argument List

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



12 Single Pixel Functions

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

12.1 Plot()

Purpose

To draw a pixel at the current position using the current color

Argument List

None

Return Value

None

12.2 Plot(x, y)

Purpose

To draw a pixel at the specified position using the current color.

Argument List

x	Integer	x screen location
y	Integer	y screen location

Return Value

None

12.3 Plot(x, y, PlotColor)

Purpose

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

Argument List

x	Integer	x screen position
y	Integer	y screen position
PlotColor	Integer	ezLCD color

Return Value

None

12.4 GetPixel()

Purpose

To get the ezLCD color value at the current location

Argument List

None

Return Value

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

12.5 GetPixel(x, y)

Purpose

To get the ezLCD color value at the specified location

Argument List

x	Integer	x screen position
y	Integer	y screen position

Return Value

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

13 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.

13.1 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.
---------	---------	--

Notes

The following bitmap fonts are shipped with ezLCD+

The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

The quick brown fox jumps over a lazy dog

13.2 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).

Notes

The following Free Type fonts are shipped with ezLCD+

The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
 The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog
The quick brown fox jumps over a lazy dog

13.3 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.
-------	---------	--

13.4 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.
-------	---------	---

13.5 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

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

13.6 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

14 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.

14.1 TextNorth()

Purpose

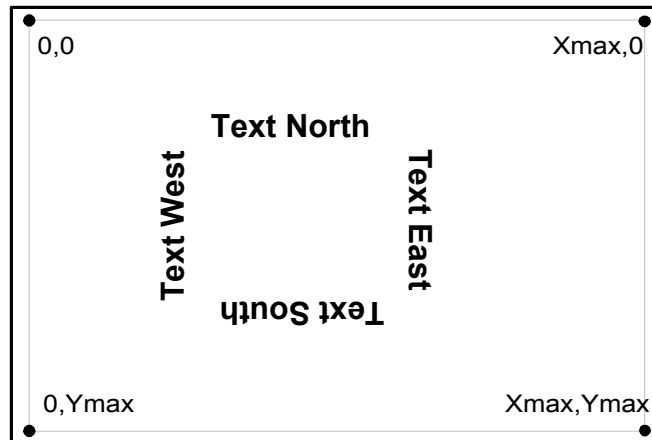
Sets the text orientation to North

Argument List

None

Return Value

None



14.2 TextEast()

Purpose

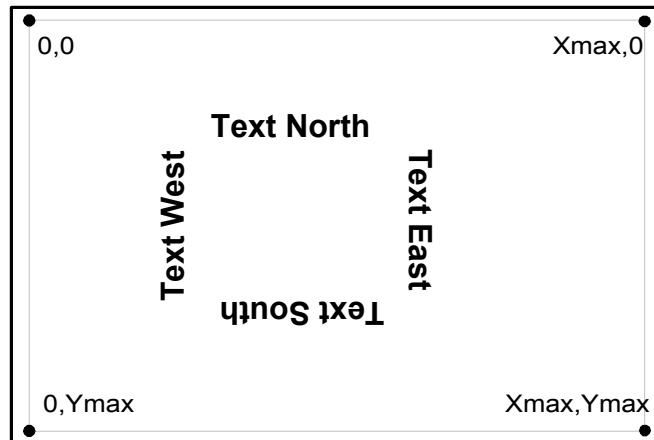
Sets the text orientation to East

Argument List

None

Return Value

None



14.3 TextSouth()

Purpose

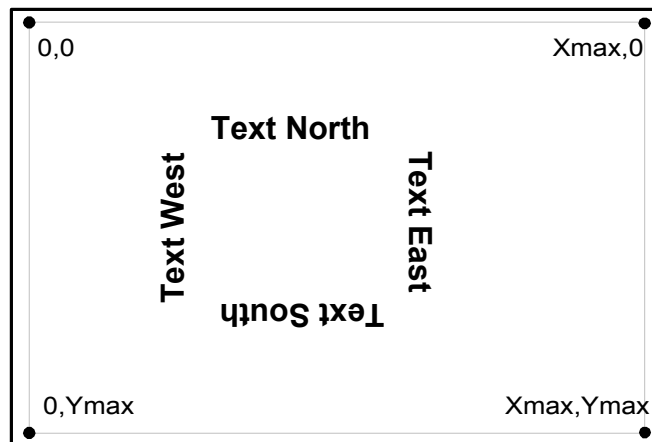
Sets the text orientation to South

Argument List

None

Return Value

None



14.4 TextWest()

Purpose

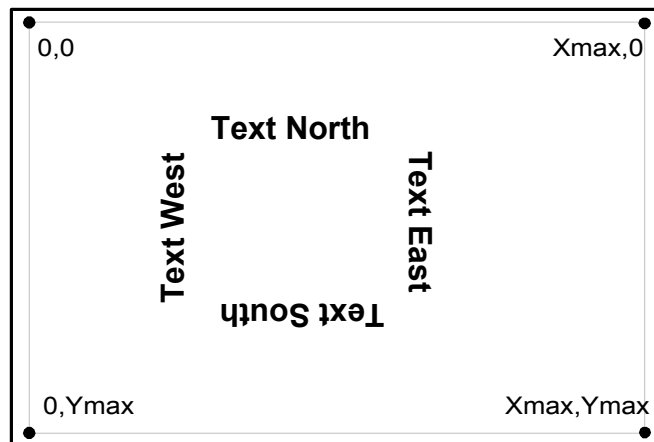
Sets the text orientation to West

Argument List

None

Return Value

None



14.5 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

15 Bitmap Functions

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

15.1 PutPictNo(PictNo)

Purpose

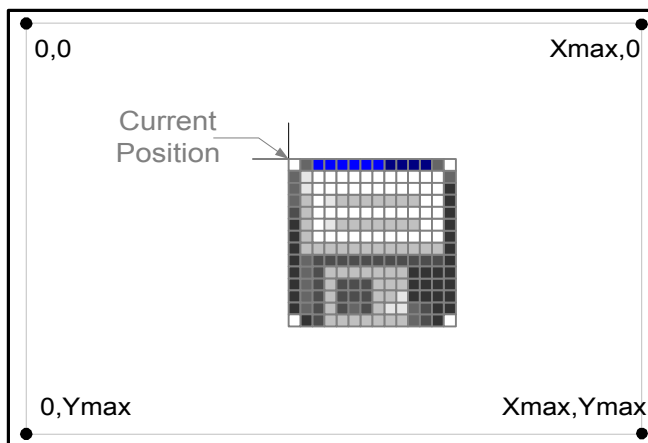
Display the specified bitmap with upper left corner being positioned at the current position.

Argument List

PictNo	Integer	bitmap number in the user ROM
--------	---------	-------------------------------

Return Value

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



15.2 PutPictNo(x, y, PictNo)

Purpose

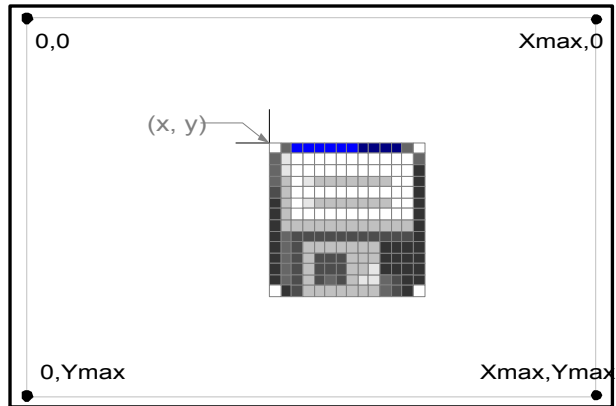
Display the specified bitmap with upper left corner being positioned at the specified position.

Argument List

x	Integer	x position to display bitmap
y	Integer	y position to display bitmap
PictNo	Integer	bitmap number in the user ROM

Return Value

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



15.3 GetPictHeight(PictNo)

Purpose

Return the height in pixels of the specified bitmap in the user ROM

Argument List

PictNo	Integer	bitmap number in the user ROM
--------	---------	-------------------------------

Return Value

Height	Integer	height of bitmap in pixels
--------	---------	----------------------------

15.4 GetPictWidth(PictNo)

Purpose

Return the width in pixels of the specified bitmap in the user ROM

Argument List

PictNo	Integer	bitmap number in the user ROM
--------	---------	-------------------------------

Return Value

Width	Integer	width of bitmap in pixels
-------	---------	---------------------------

16 Backlight Functions

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

MCours.com

16.1 LightOn()

Purpose

To turn on the screen backlight.

Argument List

None

Return Value

None

16.2 LightOff()

Purpose

To turn off the screen backlight.

Argument List

None

Return Value

None

16.3 LightBright(brightness)

Purpose

To set the screen backlight level

Argument List

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

Return Value

None

17 Screen Capture Functions

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

17.1 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.
---------	---------	--------------------------------------

18 Time Functions

The following section details the system time functions.

18.1 Get_ms()

Purpose

Gets the number of milliseconds since system power on.

Argument List

None

Return Value

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

18.2 Wait_ms(ms)

Purpose

Pauses the program specified number of milliseconds.

Argument List

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

Return Value

None

18.3 SetTime(time)

Purpose

Set the system time. The time value is the number of seconds since Jan 1, 1970 00:00:00.

Argument List

time	Integer	number of seconds
------	---------	-------------------

Return Value

None

Example

The following code sets the system time to July 3, 2008, 2:14pm using the Lua function `os.time`.

```
-- Set system time (Ref: OS library)
ez.SetTime(os.time{year=2008, month=7, day=3, hour=14, min=14, sec=0})
```

19 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.

19.1 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
---------	---------	-----------------

19.2 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.
---------	---------	--

19.3 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.

19.4 TimerStop(Id)

Purpose

Stops the specified timer.

Argument List

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

Return Value

None

20 Touch Function

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

20.1 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
---	---------	-------------------------------------

20.2 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
---	---------	-------------------------------------

20.3 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
-------	---------	---

20.4 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
SetTouchEvent("MyTouchHandler")
```

21 Input/Output Functions

The following section details the functions to read and write to the ezLCD screen, SD card and external devices (RS-232, I2C, PIN).

21.1 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"
```

Notes

1. Directories should be separated by forward slash ('/'), not by a backslash ('\') as in Windows and DOS.
2. 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.
3. Long file names are supported, however the length of the complete File Path (directory + filename + extension) may not exceed 255 characters.

21.2 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.

21.2.1 RS232 Open: Event Mode

21.2.1.1 Rs232Open(RcvFunc)

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 RS232 port.
---------	--------	---

Return Value

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

Note

The baud rate, parity, stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

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
end

-- open the RS-232 port
ez.Rs232Open("ReceiveFunction")

-- loop until number 3 is received by RS-232
while not bStop do
end
ez.Rs232Close()

```

21.2.1.2 Rs232Open(RcvFunc, BaudRate)

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 RS232 port.
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000

Return Value

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

Note

The parity, stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

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.Rs232Open("ReceiveFunction", 9600)

-- loop until number 3 is received by RS-232
while not bStop do
end
ez.Rs232Close()
```

21.2.1.3 Rs232Open(RcvFunc, BaudRate, Parity)

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 RS232 port.
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.

Return Value

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

Note

The stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

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
end

-- open the RS-232 port
ez.Rs232Open("ReceiveFunction", 9600, 5)

-- loop until number 3 is received by RS-232
while not bStop do
end
ez.Rs232Close()

```

21.2.1.4 Rs232Open(RcvFunc, BaudRate, Parity, StopBits)

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 RS232 port.
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit, 1=1.5bit, 2=2bits.

Return Value

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

Note

The handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

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
end

-- open the RS-232 port
ez.Rs232Open("ReceiveFunction", 9600, 5, 0)

-- loop until number 3 is received by RS-232
while not bStop do
end
ez.Rs232Close()

```

21.2.1.5 Rs232Open(RcvFunc, BaudRate, 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 RS232 port.
BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit, 1=1.5bit, 2=2bits.
HandShake	Integer	Handshake value to configure the RS232 port to accept. Valid values are: 1=h/w, 2=XON/XOFF, Other=None.

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.Rs232Open("ReceiveFunction", 9600, 5, 0, 1)

-- loop until number 3 is received by RS-232
while not bStop do
end
ez.Rs232Close()

```

21.2.2 RS232 Open: Buffer Mode

21.2.2.1 Rs232Open()

Purpose

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

Argument List

None

Return Value

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

Note

The baud rate, parity, stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

Example

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

```
bStop = false
-- open the RS-232 port
ez.Rs232Open()
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
ez.Rs232Close()
```

21.2.2.2 RS232Open(BaudRate)

Purpose

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

Argument List

BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
----------	---------	--

Return Value

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

Note

The parity, stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

Example

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

```

bStop = false
-- open the RS-232 port
ez.Rs232Open(9600)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
ez.Rs232Close()

```


21.2.2.3 RS232Open(BaudRate, Parity)

Purpose

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

Argument List

BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.

Return Value

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

Note

The stop bits and handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

Example

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

```

bStop = false
-- open the RS-232 port
ez.Rs232Open(9600, 4)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
ez.Rs232Close()

```

21.2.2.4 RS232Open(BaudRate, Parity, StopBits)

Purpose

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

Argument List

BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit, 1=1.5bit, 2=2bits.

Return Value

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

Note

The handshake will default to the values specified in the User Configuration. User Configuration is described in the "ezLCD+10x Manual" Chapter: "ezLCD+ Customization/User Configuration".

Example

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

```

bStop = false
-- open the RS-232 port
ez.Rs232Open(9600, 4, 0)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxFLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
end
ez.Rs232Close()

```

21.2.2.5 RS232Open(BaudRate, Parity, StopBits, HandShake)**Purpose**

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

Argument List

BaudRate	Integer	Baud rate to configure the RS232 port to accept. Valid values are: 150 - 350,000
Parity	Integer	Parity value to configure the RS232 port to accept. Valid values are: 0=Even, 1=Odd, 2=Space, Other=None.
StopBits	Integer	Number of stop bits to configure the RS232 port to accept. Valid values are: 0=1bit, 1=1.5bit, 2=2bits.
HandShake	Integer	Handshake value to configure the RS232 port to accept. Valid values are: 1=h/w, 2=XON/XOFF, Other=None.

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.Rs232Open(9600, 4, 0, 4)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
end
ez.Rs232Close()

```

21.2.3 Rs232Close()**Purpose**

To close the RS232 port and free allocated resources.

Argument List

None

Return Value

None

21.2.4 RS232 Transmit

21.2.4.1 Rs232Tx(Data)

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	byte to be sent If the integer value is bigger than 255, only it's least significant byte is sent.
	String	string to be sent
	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.

Return Value

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

21.2.4.2 Rs232Tx(Data, 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'.										
		<table border="1"> <thead> <tr> <th>MaxLen</th> <th>Bytes Sent (0 = LSB, 3 = MSB)</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Byte 0</td> </tr> <tr> <td>2</td> <td>first Byte 1, Byte 0 last</td> </tr> <tr> <td>3</td> <td>first Byte 2, Byte 1, Byte 0 last</td> </tr> <tr> <td>4</td> <td>first Byte 3, Byte 2, Byte 1, Byte 0 last</td> </tr> </tbody> </table>	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
		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'.											
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										

Return Value

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

21.2.4.3 Rs232TxStr(Str)

Purpose

To transmit an ASCII text string out through the RS232 port

Note: This function is depreciated (starting with firmware 2.20). Use [Rs232Tx \(Str\)](#) instead.

Argument List

Str	String	string to be sent
-----	--------	-------------------

Return Value

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

21.2.5 RS232 Receive

21.2.5.1 Rs232RxLen()

Purpose

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

Argument List

None

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.Rs232Open(9600)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
ez.Rs232Close()

```


21.2.5.2 Rs232getc()

Purpose

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

Argument List

None

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.Rs232Open(9600)
-- loop until a 0 byte is sent down the RS-232 port
while( not bStop )
  n = ez.Rs232RxLen()
  for i = 1,n do
    if (ez.Rs232getc() == 3) then
      bstop = true
    end
  end
end
ez.Rs232Close()
```

21.3 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

21.3.1 I2Copen(TimeLoNs, TimeHiNs)

Purpose

To initialize the I2C Interface

Argument List

TimeLoNs	Integer	The low period of the SCL clock in nanoseconds.
TimeHiNs	Integer	The high period of the SCL clock in nanoseconds.

Return Value

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

21.3.2 I2CwriteStart(Address, Data)**Purpose**

To initialize and transmit data across the I2C Interface

Argument List

Address	Integer	Slave address 0 - 127. Note that addresses 80 - 87 are restricted.
Data	Integer	Data byte to be written

Return Value

Success	Boolean	true = port successfully opened. false = write failed (invalid address or NACK from I2C interface).
---------	---------	---

Notes

A stop is not sent and the interface remains open.

21.3.3 I2CwriteStart(Address, Data, Stop)

Purpose

To initialize and transmit data across the I2C Interface

Argument List

Address	Integer	Slave address 0 - 127. Note that addresses 80 - 87 are restricted.
Data	Integer	Data byte to be written
Stop	Boolean	true = send stop (NACK) after writing data, false = don't send stop.

Return Value

Success	Boolean	true = port successfully opened. false = write failed (invalid address or NACK from I2C interface).
---------	---------	---

21.3.4 I2CwriteNext(Data)

Purpose

To transmit data across the I2C Interface

Argument List

Data	Integer	Data byte to be written
------	---------	-------------------------

Return Value

Success	Boolean	true = port successfully opened. false = write failed (invalid address or NACK from I2C interface).
---------	---------	---

Notes

A stop is not sent and the interface remains open.

21.3.5 I2CwriteNext(Data, Stop)

Purpose

To transmit data across the I2C Interface

Argument List

Data	Integer	Data byte to be written
Stop	Boolean	true = send stop after writtng data, false = don't send stop.

Return Value

Success	Boolean	true = port successfully opened. false = write failed (invalid address or NACK from I2C interface).
---------	---------	---

21.3.6 I2CreadStart(Address)**Purpose**

To initialize and read a byte of data out of the I2C Interface.

Argument List

Address	Integer	Slave address 0 - 127. Note that addresses 80 - 87 are restricted.
---------	---------	--

Return Value

Data	Integer	0 - 255 = valid data byte, < 0 = read error
------	---------	---

Notes

A stop is not sent and the interface remains open.

21.3.7 I2CreadStart(Address, Stop)**Purpose**

To initialize and read a byte of data out of the I2C Interface.

Argument List

Address	Integer	Slave address 0 - 127. Note that addresses 80 - 87 are restricted.
Stop	Boolean	true = send stop (NACK) after reading data, false = don't send stop

Return Value

Data	Integer	0 - 255 = valid data byte, < 0 = read error
------	---------	---

21.3.8 I2CreadNext()**Purpose**

To read a byte of data out of the I2C Interface

Argument List

None

Return Value

Data	Integer	0 - 255 = valid data byte, < 0 = read error
------	---------	---

Notes

A stop is not sent and the interface remains open.

21.3.9 I2CreadNext(Stop)

Purpose

To read a byte of data out of the I2C Interface

Argument List

Stop	Boolean	true = send stop (NACK) after reading data, false = don't send stop
------	---------	---

Return Value

Data	Integer	0 - 255 = valid data byte, < 0 = read error
------	---------	---

21.4 PIN Functions

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

Note that the Pins are numbered starting at zero (0).

21.4.1 SetPinInp(PinNo)

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 product manual for Lua I/O Pins assignments.
-------	---------	---

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 pull up resistor  
SetPinInp(0)
```

21.4.2 SetPinInp(PinNo, PullUp)

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 product manual for Lua I/O Pins assignments.
PullUp	Boolean	True = enable internal pull up resistor, false = no pull up resistor (default) for the specified pin.

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 pull up resistor enabled  
SetPinInp(0, true)
```

21.4.3 SetPinsInp(PinsMask)

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 product manual for Lua I/O Pins assignments.
----------	---------	--

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  
SetPinsInp(0x26)    -- 0x26 = 0010 0110
```

21.4.4 SetPinsInp(PinsMask, PullUpMask)

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 product manual for Lua I/O Pins assignments.
PullUpMask	Integer	Bit Mask where each bit that is set to 1 enables that input pin as a pull up resistor. If the bit is 0, then pull up is not enabled.

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 pins 2 and 5 with pull up resistors
SetPinsInp(0x26, 0x24)    -- 0x26 = 0010 0110
                          -- 0x24 = 0010 0100
```


21.4.5 SetPinOut(PinNo)

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 product manual for Lua I/O Pins assignments.
-------	---------	--

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  
SetPinOut(2)
```

21.4.6 SetPinOut(PinNo, OpenDrain)

Purpose

Configures the I/O pin as discrete output.

Argument List

PinNo	Integer	Pin Number to be configured as a discrete input. Refer to your product manual for Lua I/O Pins assignments.
OpenDrain	Boolean	True = open drain output, false = push-pull output (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 an open drain output  
SetPinOut(2, true)
```

21.4.7 SetPinsOut(PinsMask)

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 product manual for Lua I/O Pins assignments.
----------	---------	---

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 3 and 6 as outputs  
SetPinsOut(0x48) -- 0x48 = 0100 1000
```

21.4.8 SetPinsOut(PinsMask, OpenDrainMask)

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 product manual for Lua I/O Pins assignments.
OpenDrainMask	Integer	Bit Mask where each bit that is set to 1 sets that output pin as an open drain. If the bit is 0, then it is push-pull.

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 3 and 6 as outputs with pin 6 as open drain  
SetPinsOut(0x48, 0x40)  -- 0x48 = 0100 1000  
                        -- 0x40 = 0100 0000
```

21.4.9 SetPinIntr(PinNo, LuaFunction)**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 product manual for Lua I/O Pins assignments.
LuaFunction	String	Name of Lua function to execute on pin logic level change. Specify nil to disable further processing.

Return Value

None

Notes

PinNo must have been defined as in input pin.

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
        SetPinIntr(1, nil)
    end
end

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

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

```

21.4.10 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 product manual for Lua I/O Pins assignments).
-------	---------	---

Return Value

None

Notes

This function also disables the associated interrupt set by SetPinIntr.

21.4.11 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 product manual for Lua I/O Pins assignments.
----------	---------	--

Return Value

None

Notes

This function also disables associated interrupts set by SetPinIntr.

21.4.12 Pin(PinNo)**Purpose**

Retrieve the logic level on the specified pin.

Argument List

PinNo	Integer	Lua I/O Pin Number (refer to your product manual for Lua I/O Pins assignments).
-------	---------	---

Return Value

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

21.4.13 Pin(PinNo, Value)**Purpose**

Set the logic level on the specified pin.

Argument List

PinNo	Integer	Lua I/O Pin Number (refer to your product manual for Lua I/O Pins assignments).
Value	Integer	0 = low >0 = high

Return Value

None

21.4.14 Pins(PinsMask)**Purpose**

Retrieve the logic level on the specified pins.

Argument List

PinsMask	Integer	Bitmask of I/O Pin Numbers (refer to your product manual for Lua I/O Pins assignments).
----------	---------	---

Return Value

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

21.4.15 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 product manual for Lua I/O Pins assignments).
Value	Integer	The level on each pin will be set to low (bit value = 0) or high (bit value = 1).

Return Value

None

22 Advanced Topics

The following sections describe ezLCD+ advanced topics.

22.1 Frame Management Functions

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

Frame Management

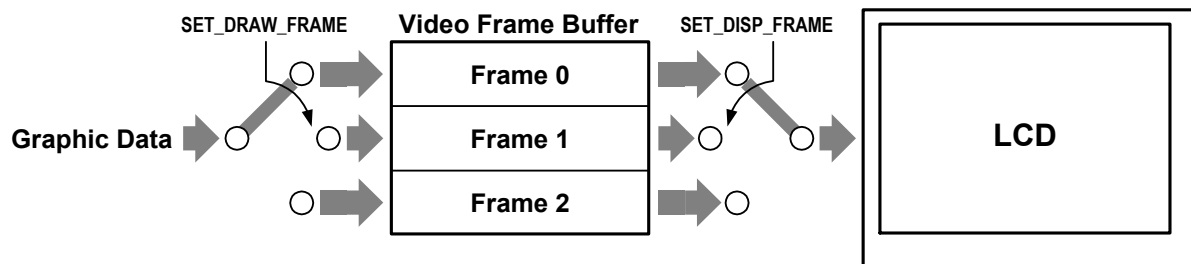
ezLCD devices consist of 1 or more frames. A frame is a portion of the ezLCD memory that can be displayed on the screen. The number of available frames depends on the amount of memory that has been installed and the height/width in pixels of the screen; however, there are at least 2 frames in each ezLCD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

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



22.1.1 SetDispFrame(FrameNo)**Purpose**

Sets the frame to be displayed on the screen

Argument List

FrameNo	Integer	frame number to display
---------	---------	-------------------------

Return Value

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

22.1.2 SetDispFrame(FrameNo, Sync)

Purpose

Sets the frame to be displayed on the screen

Argument List

FrameNo	Integer	frame number to display
Sync	Boolean	Change the frame on the next vertical sync (=true) or change immediately (=false)

Return Value

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

Notes

The ezLCD+ screen is refreshed 30 to 70 times per second, depending of the LCD type. vSync (Vertical Synchronization) is an internal LCD signal which is active between screen refreshes. It signals, that the full screen refresh has just ended and the display is about to start the new screen refresh cycle.

If the Display Frame is changed while the old frame is being refreshed, the screen may show a combination of both frames, producing a page tearing artifact partway down the image.

22.1.3 GetDispFrame()**Purpose**

Gets the currently displayed frame number.

Argument List

None

Return Value

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

22.1.4 GetNextDispFrame()

Purpose

Gets the frame, which will be displayed after the next vSync.

Argument List

None

Return Value

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

Notes

The ezLCD+ screen is refreshed 30 to 70 times per second, depending of the LCD type. vSync (Vertical Synchronization) is an internal LCD signal which is active between screen refreshes. It signals, that the full screen refresh has just ended and the display is about to start the new screen refresh cycle.

If the Display Frame is changed while the old frame is being refreshed, the screen may show a combination of both frames, producing a page tearing artifact partway down the image.

22.1.5 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
---------	---------	----------------------------------

22.1.6 GetDrawFrame()**Purpose**

Gets the current draw frame number.

Argument List

None

Return Value

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

22.1.7 GetNoOfFrames()**Purpose**

Gets the number of ezLCD+ display frames.

Note: This function is depreciated (starting with firmware 2.20). Use [NoOfFrames](#) constant instead.

Argument List

None

Return Value

Count	Integer	number of frames in the ezLCD+
-------	---------	--------------------------------

22.1.8 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
---------	---------	---------------------------------

22.1.9 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
---------	---------	----------------------------------

22.1.10 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

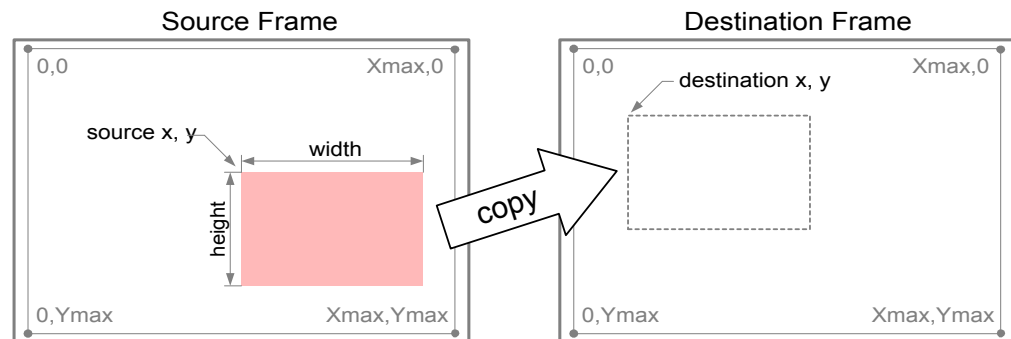
DestFrame	Integer	frame number to be copied to
SourceFrame	Integer	frame number of be copied from
DestX	Integer	x position in DestFrame to place copied data
DestY	Integer	y position in DestFrame to place copied data
SourceX	Integer	x position in SourceFrame to copy data from
SourceY	Integer	y position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

22.1.11 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

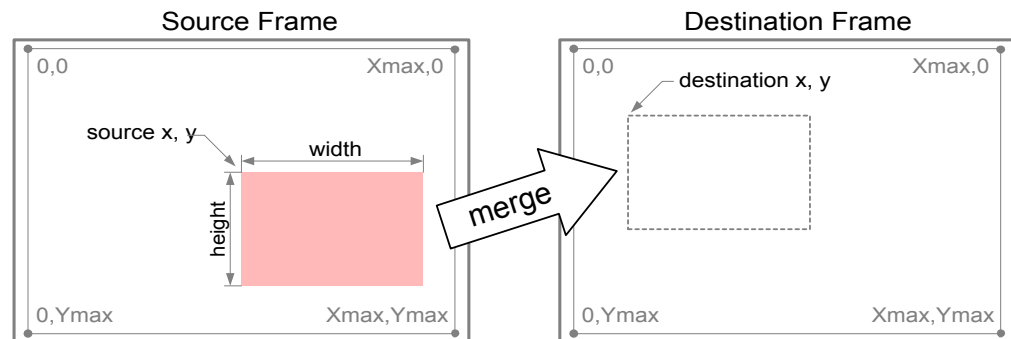
DestFrame	Integer	frame number to be copied to
SourceFrame	Integer	frame number of be copied from
DestX	Integer	x position in DestFrame to place copied data
DestY	Integer	y position in DestFrame to place copied data
SourceX	Integer	x position in SourceFrame to copy data from
SourceY	Integer	y position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



22.2 Miscellaneous Functions

The following section details miscellaneous functions.

WARNING!

Reading or writing directly to memory may have unexpected results.

22.2.1 Peek32(address)**Purpose**

Read a 32 bit value from the memory address specified.

Argument List

address	Integer	memory address to read data from
---------	---------	----------------------------------

Return Value

value	Integer	32 bit value read from memory
-------	---------	-------------------------------

Note

1. The address specified should be on a long word (32-bit) boundary. If it is not, the lowest 2 address bits are ignored.
2. Reading certain addresses that are mapped to status registers may actually cause their value to be reset and therefore have undesirable effects.

22.2.2 Peek16(address)

Purpose

Read a 16 bit value from the memory address specified.

Argument List

address	Integer	memory address to read data from
---------	---------	----------------------------------

Return Value

value	Integer	16 bit value read from memory
-------	---------	-------------------------------

Note

1. The address specified should be on a word (16-bit) boundary. If it is not, the lowest address bit is ignored.
2. Reading certain addresses that are mapped to status registers may actually cause their value to be reset and therefore have undesirable effects.

22.2.3 Peek8(address)

Purpose

Read a 8 bit value from the memory address specified.

Argument List

address	Integer	memory address to read data from
---------	---------	----------------------------------

Return Value

value	Integer	8 bit value read from memory
-------	---------	------------------------------

Note

Reading certain addresses that are mapped to status registers may actually cause their value to be reset and therefore have undesirable effects.

22.2.4 Poke32(address, data)

Purpose

Write a 32 bit value from the memory address specified.

Argument List

address	Integer	memory address to write data to
data	Integer	32 bit value to write to address

Return Value

None

Notes

1. The address specified should be on a long word (32-bit) boundary. If it is not, the low 2 address bits are ignored.
2. Direct writing to memory may have undesired results.

22.2.5 Poke16(address, data)**Purpose**

Write a 16 bit value from the memory address specified.

Argument List

address	Integer	memory address to write data to
value	Integer	16 bit value to write to address

Return Value

None

Notes

1. The address specified should be on a word (16-bit) boundary. If it is not, the lowest address bit is ignored.
2. Direct writing to memory may have undesired results.

22.2.6 Poke8(address, data)

Purpose

Write a 8 bit value from the memory address specified.

Argument List

address	Integer	memory address to write data to
data	Integer	8 bit value to write to address

Return Value

None

Notes

Direct writing to memory may have undesired results.

22.2.7 ExitReq()

Purpose

Checks if exit from Lua has been externally requested through the USB Interface.

Argument List

None

Return Value

value	Boolean	True = exit requested, False = no exit request
-------	---------	--

Example

```
-- This function may be used to exit from a loop upon the request sent by the  
while not ExitReq() do  
end
```


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. Described in the " <i>ezLCD+10x Manual</i> ".
ezLCD+ Customization	Modification of the default power-up parameters. Addition of custom fonts, bitmaps, Lua programs, etc. Described in the " <i>ezLCD+10x Manual</i> ".
Firmware	Operating software of the ezLCD+. Can be in-field upgraded. Described in the " <i>ezLCD+10x Manual</i> ".
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. Described in the " <i>ezLCD+10x Manual</i> ".
User ROM	Part of the ezLCD+ Customization. A place in the ezLCD+ flash, where user can store custom fonts, bitmaps, Lua programs, etc. Described in the " <i>ezLCD+10x Manual</i> ".

MCours.com