

# **ezLCD+**

## **External Commands Manual**

# 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

## ezLCD+ External Commands Manual

<b>1 About ezLCD+</b> .....	<b>7</b>
<b>2 Quick Start</b> .....	<b>8</b>
Quick Start: External Commands .....	9
<b>3 Drawing on the ezLCD+ Display</b> .....	<b>11</b>
Screen Coordinates .....	11
Vector Graphics .....	12
Raster Graphics (Bitmaps) .....	13
Frames .....	14
Fonts .....	15
Bitmap Fonts .....	16
True Type Fonts .....	17
Drawing Parameters .....	18
General .....	18
Transparency .....	19
Pen .....	20
<b>4 Touch Screen Operations</b> .....	<b>22</b>
Data Protocols .....	22
ezButton .....	23
ezButton Events .....	24
cuButton .....	25
cuButton States .....	26
CalibratedXY .....	27
CalibratedXY Packets .....	28
<b>5 SD Card Operations</b> .....	<b>29</b>
Introduction .....	29
SD File Operations .....	30
SD Raw Operations .....	32
<b>6 Interfacing with Windows USB Driver</b> .....	<b>33</b>
favr-32.dll functions .....	34
<b>7 Source Code Examples</b> .....	<b>40</b>
<b>8 ezLCD+ Commands</b> .....	<b>42</b>
Command Summary .....	43
ARCH .....	48
BOXHH .....	50
BOXHH_FILL .....	51
BUTTON_DEF_LONG .....	53
BUTTON_STATE .....	55
BUTTONS_ALL_UP .....	56
BUTTONS_DELETE_ALL .....	57
CACHE_FT_CHARS .....	58
CACHE_FT_UNICHARS .....	60
CALIBRATE_TOUCH .....	61
CIRCLE_RH .....	62
CIRCLE_RH_FILL .....	63

CLS .....	64
COPY_FRAME .....	65
COPY_RECT .....	66
CRC_PKG .....	68
ELLIPSE_AHBH .....	70
ELLIPSE_AHBH_FILL .....	72
ELLIPSE_ARCH .....	74
ELLIPSE_PIEH .....	76
FILL .....	78
FILL_BOUND .....	79
GET_SERIAL_NO .....	80
H_LINEH .....	81
LIGHT_BRIGHT .....	82
LIGHT_OFF .....	83
LIGHT_ON .....	84
LINE_TO_XHYH .....	85
MERGE_FRAME .....	86
MERGE_RECT .....	87
PIEH .....	89
PING .....	91
PLOT .....	92
PLOT_XHYH .....	93
POLYGON .....	94
PRINT_CHAR .....	97
PRINT_CHAR_BG .....	98
PRINT_FT_UNICHAR .....	99
PRINT_FT_UNISTRING .....	100
PRINT_STRING .....	101
PRINT_STRING_BG .....	102
PUT_BITMAP_RGB .....	103
PUT_PICT_NO .....	105
REPLACE_COLOR .....	106
RESTORE_POSITION .....	107
RUN_LUA .....	108
RUN_LUA_ROM .....	109
RUN_LUA_SD .....	110
SAVE_POSITION .....	112
SD_FILE_CLOSE .....	113
SD_FILE_CLOSE_ALL .....	114
SD_FILE_CREATE .....	115
SD_FILE_DELETE .....	118
SD_FILE_GET_SIZE .....	120
SD_FILE_LIST .....	122
SD_FILE_OPEN .....	125
SD_FILE_READ .....	128
SD_FILE_REWIND .....	131
SD_FILE_SEEK .....	133
SD_FILE_TELL .....	135
SD_FILE_WRITE .....	137
SD_FIND_FIRSTandSD_FIND_NEXT .....	139
SD_FOLDER_CREATE .....	142
SD_FOLDER_DELETE .....	144
SD_FORMAT .....	146
SD_INSERTED .....	148

SD_LOAD_FONT .....	149
SD_PUT_ICON .....	152
SD_RAW_READ .....	154
SD_RAW_WRITE .....	156
SD_SCREEN_CAPTURE .....	159
SD_SIZE .....	160
SD_SPACE_INFO .....	161
SELECT_FONT .....	163
SET_ALPHA .....	164
SET_BG_COLOR_RGB .....	165
SET_COLOR_RGB .....	166
SET_DISP_FRAME .....	167
SET_DRAW_FRAME .....	168
SET_EDIT_RECT .....	169
SET_FT_ANGLE .....	170
SET_FT_FONT .....	173
SET_FT_UNIBASE .....	175
SET_PEN_HEIGHT .....	177
SET_PEN_SIZE .....	178
SET_TR_COLOR_RGB .....	179
SET_XH .....	180
SET_XHYH .....	181
SET_YH .....	182
TEXT_NORTH,SOUTH,EAST,WEST .....	183
TOUCH_PROTOCOL .....	185
TR_COLOR_NONE .....	186
V_LINEH .....	187
<b>Legacy Commands .....</b>	<b>188</b>
ARC .....	189
BOX .....	191
BOX_FILL.....	192
BOXH .....	193
BOXH_FILL .....	194
BUTTON_DEF.....	195
CIRCLE_R.....	197
CIRCLE_R_FILL .....	198
LINE_TO_XHY .....	199
LINE_TO_XY .....	200
PLOT_XHY.....	201
PLOT_XY .....	202
PUT_BITMAP .....	203
PUT_BITMAPH.....	205
PUT_SF_ICON .....	206
SET_BG_COLOR .....	207
SET_BG_COLORH .....	208
SET_COLOR .....	209
SET_COLORH .....	210
SET_X .....	211
SET_XY .....	212
SET_XHY .....	213
SET_Y .....	214
V_LINE .....	215

## GLOSSARY



# ezLCD+ External Commands Manual

## 1 About ezLCD+

### Congratulations on your purchase of the ezLCD+ product!

The ezLCD+ is an all-in-one advanced display panel which includes:

- Display<sup>1</sup>
- Embedded 32bit processor (Atmel AT32AP7000) with LCD Controller
- 4 Mega Bytes of embedded flash for storing custom fonts and bitmaps
- SD/MicroSD<sup>1</sup> Card slot for storing bitmap, fonts and other user data up to 2 Giga Bytes
- Power supply, which generates all the voltages needed by the logic and the display itself
- Touch screen
- Interface drivers and other circuitry

The ezLCD+ communicates with the outside world through several interfaces:

- RS232 Standard<sup>2</sup>
- RS232 TTL
- USB
- I2C
- SPI
- SD/MMC
- Ethernet<sup>2</sup> (10 and 100 Mbit/s)

The ezLCD+ firmware is in-field upgradable and contains an extensive command set:

- Graphic commands
- Double buffering
- True Type and Open Type font rendering
- Bitmap font rendering
- Unicode support
- SD file I/O (FAT12, FAT16, and FAT32)
- Touch Screen commands

The ezLCD+ may be in-field customized by:

- Adding custom fonts
- Adding custom bitmaps
- Customizing startup screen
- Modifying interface parameters like RS232 baudrate, Ethernet MAC and IP address, etc.
- Modifying pin functions

The ezLCD+ is driven by a set of [commands](#), which can be fed through any of the implemented interfaces apart from I2C. Besides that, the ezLCD+ contains an embedded Lua programming language interpreter. The ezLCD+ Lua API is described in a separate manual. The product may be used as an "intelligent" display, or as a stand alone device. There is enough of flash memory left to incorporate additional graphical instructions, or to customize the software for particular tasks. Possible applications include automotive, avionics, nautical, industrial control, hobby, etc.

<sup>1</sup> Refer to H/W Reference Manual of your ezLCD+ product for more details.

<sup>2</sup> Not available on all products. Refer to H/W Reference Manual of your ezLCD+ product for more details

## 2 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

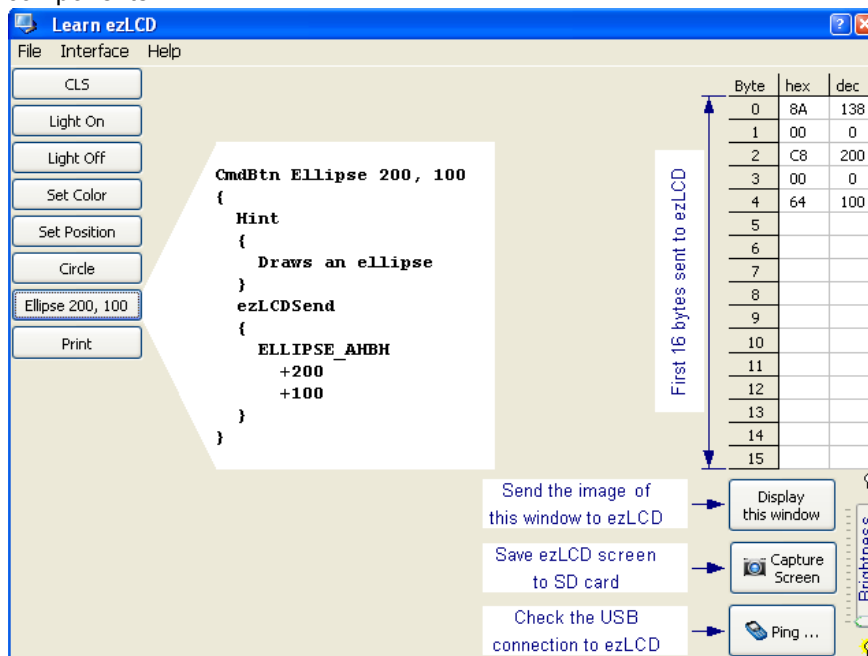
1. Download the latest USB FAVR-32 driver from <http://www.ezlcd.com/support/>
2. Run the downloaded driver installation executable before connecting ezLCD+ to the USB of your computer.
3. Connect ezLCD+ USB to your computer and turn the ezLCD+ power on by sliding the PWR switch into "ON" position. "New 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: External Commands](#)".

## 2.1 Quick Start: External Commands

1. Make sure, that USB FAVR-32 driver is installed on your PC
2. Download the setup of "Learn\_ezLCD" utility from <http://www.ezlcd.com/support/>
3. Install "Learn\_ezLCD" utility by running the downloaded setup
4. Turn-on ezLCD+ and make sure that it is connected to your computer through USB.
5. Run "Learn\_ezLCD" utility. Press "Display this window" button. "Learn\_ezLCD" utility window should appear on the ezLCD.
6. Read "Learn\_ezLCD" Help and experiment with ezLCD commands and "Learn\_ezLCD" buttons

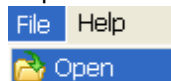
### About Learn\_ezLCD

"Learn\_ezLCD" is a simple utility, designed to help beginners learn how to use ezLCD commands. The picture, below, shows the main window of "Learn\_ezLCD", with some short descriptions of it's components.



Upon start, "Learn\_ezLCD" dynamically generates Command Buttons, based on the data read from the file: Buttons.txt. File Buttons.txt contains the button definitions. It resides in the same folder as Learn\_ezLCD.exe (application executable).

Besides that, the button definitions can be loaded from any other Button Definition File by selecting File->Open from the menu.



Generated buttons are shown in the upper-left part of the picture above.

As an example, the script defining button: `Ellipse 200, 100` is shown to the right of it.

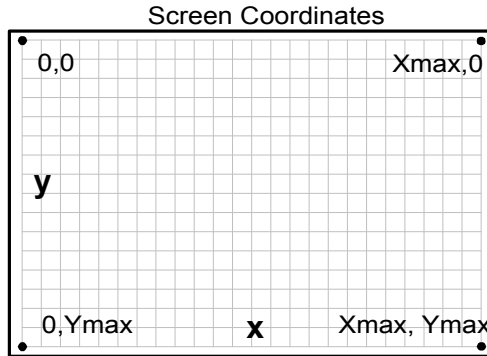
Permanent components of "Learn\_ezLCD" are located in the right part of it's window: They include:

- The Data Sent Table, which shows the first 16 bytes of the data sent to the ezLCD upon pressing the button
- Three Utility Buttons: 'Display this window', 'Capture Screen' and 'Ping ...'
- ezLCD Backlight Control Slider

For more information about ezLCD+ External Commands, please refer to the *"ezLCD+ External Commands Manual"*.

## 3 Drawing on the ezLCD+ Display

### 3.1 Screen Coordinates



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

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

```

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

```

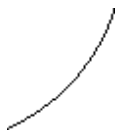
This means that the numbers range

From: -32768 dec = 1000 0000 0000 0000 bin

To: 32767 dec = 0111 1111 1111 1111 bin

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

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



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

## 3.2 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, squares, polygons, ellipses, arcs, etc. The following commands may be used for that purpose: [LINE\\_TO\\_XHYH](#), [BOXHH](#), [POLYGON](#), [ELLIPSE\\_AHBH](#), [ARCH](#), and many more.

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

- Current Position
- Current Color
- Transparency
- Pen

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

### 3.3 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 one of the following commands:

[PUT\\_PICT\\_NO](#), [SD\\_PUT\\_ICON](#)

Additionally, ezLCD+ supports direct pixel drawing on the display. The following commands can be used for that:

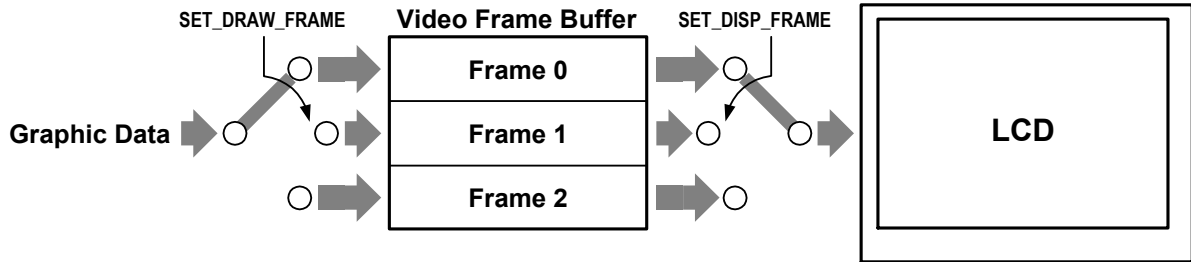
[PLOT](#), [PLOT\\_XHYH](#), [PUT\\_BITMAP\\_RGB](#)

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

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

### 3.4 Frames

In order to support a special effects (animation, double buffering, etc) the whole ezLCD+ video RAM is divided into a few separate frames, as shown on the drawing below. Each frame has the capacity of the full ezLCD+ screen. The number of available frames is displayed on the ezLCD+ System Info page of the Demo Mode, described in the "ezLCD+10x Manual".



The ezLCD+ commands draw on the frame selected by the command: [SET\\_DRAW\\_FRAME](#).  
The screen displays image from the frame selected by the command: [SET\\_DISP\\_FRAME](#).

The same frame can be used to draw in display data. Upon power-up both draw and display frames are set to the Frame 0.

The entire data from one frame can be copied to the another by using one of the following commands:  
[COPY\\_FRAME](#)  
[MERGE\\_FRAME](#)

The portion of one frame can be copied to the another by using one of the following commands:  
[COPY\\_RECT](#)  
[MERGE\\_RECT](#)

### 3.5 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.

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



### 3.5.1 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

### 3.5.2 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*

## 3.6 Drawing Parameters

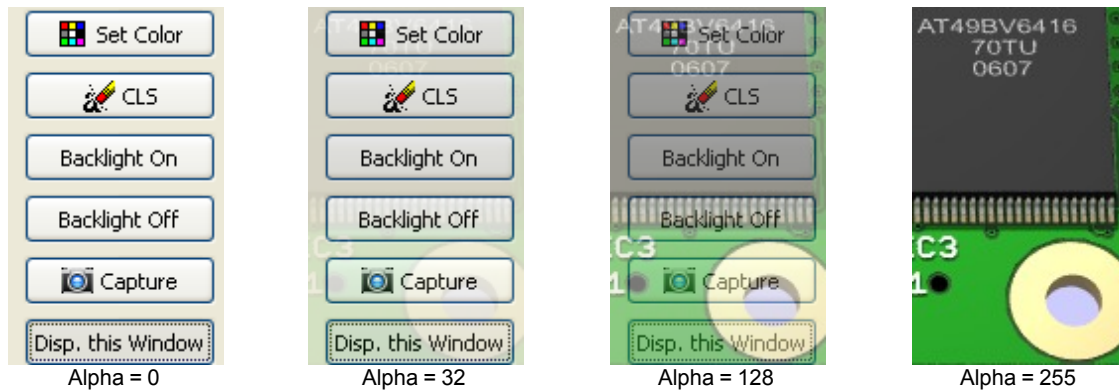
### 3.6.1 General

The graphics is drawn according to the following parameters:

- Current Position.
  - Set by command: [SET\\_XHYH](#)
  - Updated by drawing commands
- Current Color.
  - Set by command: [SET\\_COLOR\\_RGB](#)
  - [Bitmaps](#) are not affected
- Background Color.
  - Set by command: [SET\\_BG\\_COLOR\\_RGB](#)
  - Specifies the background color for Bitmap Font commands: [PRINT\\_CHAR\\_BG](#) and [PRINT\\_STRING\\_BG](#)
  - Only [Bitmap Fonts](#) are affected
- Transparent Color.
  - Set by commands: [SET\\_TR\\_COLOR\\_RGB](#), [TR\\_COLOR\\_NONE](#)
  - Specifies the color, which is ignored during Bitmap drawing
  - Only [Bitmaps](#) are affected (direct pixel drawing is not affected).
- [Transparency](#)
  - Set by command: [SET\\_ALPHA](#)
- [Pen](#)
  - Set by commands: [SET\\_PEN\\_WIDTH](#) and [SET\\_PEN\\_HEIGHT](#)
  - Only the [Vector Graphics](#) is affected
  - Pen Height affects only the drawing of curves (ellipse, circle, arc, etc)

### 3.6.2 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 convex combination 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 the command:

[SET\\_ALPHA](#)

Upon power-up, alpha is set to 255 (fully opaque).

#### Drawing Performance Impact

The drawings are rendered almost 3 time slower, when alpha is set to any value other than 255 or 0.

### 3.6.3 Pen

Vector Graphics is drawn using the Pen. The following Pen parameters are currently supported by ezLCD+ commands:

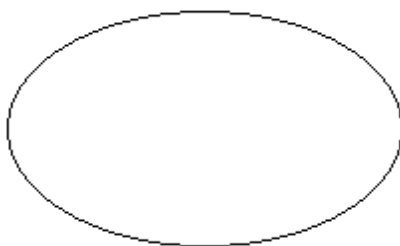
- Pen Width
  - Specifies the horizontal dimension the drawing line (in pixels)
  - Set by command: [SET\\_PEN\\_SIZE](#)
- Pen Height
  - Specifies the vertical dimension of Pen (in pixels), when drawing curves
  - Ignored when drawing straight lines
  - Set by commands: [SET\\_PEN\\_SIZE](#) and [SET\\_PEN\\_HEIGHT](#)

**Notes:** Straight lines are not drawn when Pen Width is set to 0.  
Curves are not drawn when either Pen Width or Height is set to 0.  
[SET\\_PEN\\_SIZE](#) sets both: width and height of Pen  
[SET\\_PEN\\_HEIGHT](#) sets only the height of Pen

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 = 1  
Pen Height = 1



Pen Width = 1  
Pen Height = 40



Pen Width = 40  
Pen Height = 4

## 4 Touch Screen Operations

### 4.1 Data Protocols

Currently, the ezLCD+ can broadcast the touch screen data using the following protocols:

1. [ezButton](#)
  - Touch screen buttons can be defined [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.
  - ezLCD+ sends Button Down and Button Up events for the defined buttons.
  - Easy protocol. Button IDs and events are coded in 1 byte.
  - Events are sent only once per button state change.
2. [cuButton](#)
  - Similar to the ezButton, however the button states are sent continuously, 5 to 20 times per second.
3. [CalibratedXY](#)
  - ezLCD+ sends [TOUCH\\_X](#) and [TOUCH\\_Y](#) packets (X and Y coordinates), when the screen is pressed
  - ezLCD+ sends [PEN\\_UP](#) packets when the touch screen is not pressed.
  - Multi-byte packed oriented protocol.
  - Packets are sent continuously, 5 to 50 times per second.

**Note:** Upon the Power-Up the ezLCD+ does not send any touch screen data until the proper protocol is selected by the TOUCH\_PROTOCOL command.

#### Differences between the [ezButton](#) and [cuButton](#) protocols.

1. [ezButton](#) ezLCD+ sends the event only once per button state change  
[cuButton](#) The button states are reported continuously, 5 to 20 times per second.
2. [ezButton](#) ezLCD+ sends Button Down and Button Up [events](#).  
[cuButton](#) ezLCD+ sends Button Down and Button None [states](#).

### 4.1.1 ezButton

The ezButton (ez = easy) protocol is the easiest way to use the touch screen. All you have to do is:

#### 1. Design the icons of the buttons.

The following button states are supported:

- Button Up
- Button Down
- Button Disabled

Use your favorite software to design the bitmaps of the button states. It is not necessary to design the bitmaps of all the button states. As a matter of fact, the button may exist without the bitmaps assigned to any of the above states.

#### 2. Write the designed icons into the ezLCD+ User ROM.

Use the procedure described in the "ezLCD+10x Manual" to store the bitmaps in the ezLCD+ User ROM. Note which bitmap ID should be assigned to each state of the particular button.

#### 3. In your code, select ezButton protocol.

Send [TOUCH\\_PROTOCOL](#)(1) command to the ezLCD+.

#### 4. In your code, define the buttons using [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

Send the above command for each of the buttons that you want to use. The button definition commands specify:

- Button Number (ID)
- Initial state of the button
- Bitmaps for each of the button states
- The position of the button
- The touch sensitive area of the button (touch zone)

At this point the ezLCD+ starts broadcasting [ezButton events](#) for the defined buttons.

You can respond to those events using ezLCD+ command [BUTTON\\_STATE](#), which will redraw the button in it's new state.

#### Sending of the ezButton events by the ezLCD+

- The ezLCD+ sends the ezButton event only once per button state change. If you need to have the button state continuously updated, please use the [cuButton](#) protocol instead.
- The button cannot be pressed just by sliding the finger onto the button touch zone. The ezLCD+ sends the Button Down Event only if the button is directly pressed.
- When the button is already pressed, it may only be released by removing the finger from the touch screen. Sliding the finger out of the button area will not release the button.

#### 4.1.1.1 ezButton Events

The ezButton events are coded in one byte:

		7	6	5	4	3	2	1	0
<b>Status</b>	<b>Button_Number (0 to 63)</b>								
0	0	Not used (Disregard)							
0	1	Button Down Event							
1	0	Button Up Event							
1	1	Not used (Disregard)							

**Where:**

Button\_Number: The number (ID) of the button, which has caused the event. The button number is specified by the [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

Button Down Event: The button indicated by Button\_Number has just been pressed.

Button Up Event: The button indicated by Button\_Number has just been released.

**Sending of the ezButton events by the ezLCD+**

- The ezLCD+ sends the ezButton event only once per button state change. If you need to have the button state continuously updated, please use the [cuButton](#) protocol instead.
- The button cannot be pressed just by sliding the finger onto the button touch zone. The ezLCD+ sends the Button Down Event only if the button is directly pressed.
- When the button is already pressed, it may only be released by removing the finger from the touch screen. Sliding the finger out of the button area will not release the button.

### Example:

Let's assume that the ezButton protocol is selected by the [TOUCH\\_PROTOCOL](#) command and the button no 4 is defined by [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

1. Touch Screen: Not pressed  
ezLCD+ Sends: Nothing
2. Touch Screen: Pressed in the Button 4 touch zone  
ezLCD+ Sends: 44hex only 1 time, in the moment when the button 4 become pressed.
3. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: 84hex only 1 time, in the moment when the finger is removed
4. Touch Screen: Pressed outside any of the buttons  
ezLCD+ Sends: Nothing
5. Touch Screen: Finger slides into the Button 4 touch zone  
ezLCD+ Sends: Nothing
6. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: Nothing (because no button has been pressed)
7. Touch Screen: Pressed in the Button 4 touch zone  
ezLCD+ Sends: 44hex only 1 time, in the moment when the button 4 become pressed.
8. Touch Screen: Finger slides out of the Button 4 touch zone  
ezLCD+ Sends: Nothing (the Button 4 is still considered pressed)
9. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: 84hex only 1 time, in the moment when the finger is removed

### 4.1.2 cuButton

The cuButton (cu = continuous update) protocol is an easy way to use the touch screen. All you have to do is:

#### 1. Design the icons of the buttons.

The following button states are supported:

- Button Up
- Button Down
- Button Disabled

Use your favorite software to design the bitmaps of the button states. It is not necessary to design the bitmaps of all the button states. As a matter of fact, the button may exist without the bitmaps assigned to any of the above states.

#### 2. Write the designed icons into the ezLCD+ User ROM.

Use the procedure described in the <%MANUAL\_FIRM%> to store the bitmaps in the ezLCD+ User ROM. Note which bitmap ID should be assigned to each state of the particular button.

#### 3. In your code, select cuButton protocol.

Send [TOUCH\\_PROTOCOL\(2\)](#) command to the ezLCD+.

At this point ezLCD+ starts broadcasting [Button None](#) state, 5 to 20 times per second.

#### 4. In your code, define the buttons using [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

Send the above command for each of the buttons that you want to use. The button definition commands specify:

- Button Number (ID)
- Initial state of the button
- Bitmaps for each of the button states
- The position of the button
- The touch sensitive area of the button (touch zone)

At this point the ezLCD+, broadcasts 5 to 20 times per second:

[Button Down](#) state, if the particular button is pressed.

[Button None](#) state, if no button is pressed.

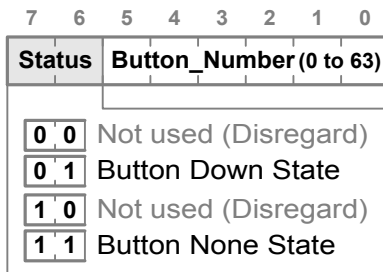
You can respond to the changes in the cuButton states using ezLCD+ command [BUTTON\\_STATE](#), which will redraw the button in it's new state.

#### Sending of the cuButton states by the ezLCD+

- The ezLCD+ sends the cuButton states continuously, 5 to 20 times per second. If you would like to receive the button state only once per event, please use the [ezButton](#) protocol instead.
- The button cannot be pressed just by sliding the finger onto the button touch zone. The ezLCD+ sends the Button Down state only if the button is directly pressed.
- When the button is already pressed, it may only be released by removing the finger from the touch screen. Sliding the finger out of the button area will not release the button.

#### 4.1.2.1 cuButton States

The ezButton events are coded in one byte:



**Where:**

Button\_Number: The number (ID) of the button, which has caused the event. The button number is specified by the [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

Button Down State: The button indicated by Button\_Number is pressed.

Button None State: No button is pressed. Button\_Number for this state is always set to 63.

**Sending of the cuButton states by the ezLCD+**

- The ezLCD+ sends the cuButton states continuously, 5 to 20 times per second. If you would like to receive the button state only once per event, please use the [ezButton](#) protocol instead.
- The button cannot be pressed just by sliding the finger onto the button touch zone. The ezLCD+ sends the Button Down state only if the button is directly pressed.
- When the button is already pressed, it may only be released by removing the finger from the touch screen. Sliding the finger out of the button area will not release the button.

### Example:

Let's assume that the cuButton protocol is selected by the [TOUCH\\_PROTOCOL](#) command and the button no 4 is defined by [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

1. Touch Screen: Not pressed  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second
2. Touch Screen: Pressed in the Button 4 touch zone  
ezLCD+ Sends: 44hex continuously, 5 to 20 times per second
3. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second
4. Touch Screen: Pressed outside any of the buttons  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second
5. Touch Screen: Finger slides into the Button 4 touch zone  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second
6. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second
7. Touch Screen: Pressed in the Button 4 touch zone  
ezLCD+ Sends: 44hex continuously, 5 to 20 times per second
8. Touch Screen: Finger slides out of the Button 4 touch zone  
ezLCD+ Sends: 44hex continuously, 5 to 20 times per second (the Button 4 is still considered pressed)
9. Touch Screen: Finger is removed from the touch screen  
ezLCD+ Sends: FFhex continuously, 5 to 20 times per second

### 4.1.3 CalibratedXY

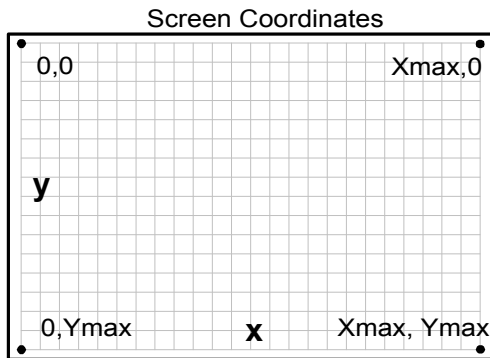
When the CalibratedXY protocol is selected, the ezLCD+:

- ezLCD+ sends [TOUCH\\_X](#) and [TOUCH\\_Y](#) packets (X and Y coordinates), when the screen is pressed
- ezLCD+ sends [PEN\\_UP](#) packets when the touch screen is not pressed.
- Packets are sent continuously, 5 to 50 times per second.

In order to select the CalibratedXY protocol, send [TOUCH\\_PROTOCOL](#)(64) to the ezLCD+.

#### Sending of the touch screen coordinates by the ezLCD+

The touch screen coordinates are sent by the ezLCD+ only when the CalibratedXY protocol is selected.

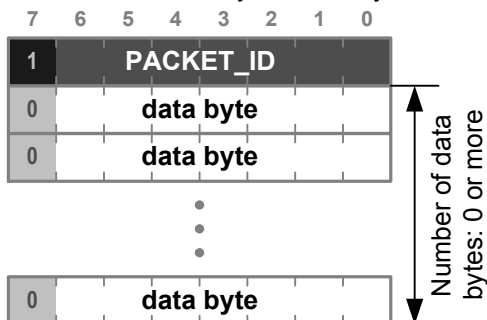


For ezLCD+:

- Xmax = 319
- Ymax= 233

Touch screen coordinates are sent in multi-byte packets:

- Packets are sent in non particular order
- Each packet is starts with the PACKET\_ID byte
- Each packet is identified by it's PACKET\_ID byte
- The PACKET\_ID may be followed by the data bytes, however there are packets which only consist of the PACKET\_ID with no data bytes
- Bit 7 of the PACKET\_ID is always 1
- Bit 7 of the data bytes is always 0



### 4.1.3.1 CalibratedXY Packets

When the CalibratedXY protocol is selected, the touch screen coordinates are sent in multi-byte packets.

#### TOUCH\_X Packet

**Description:** The TOUCH\_X packet represents the touch screen X coordinate. It is sent only if the touch screen is pressed.

**Length:** 3 bytes, including the Packet ID

**Code:** 81hex, 129dec

	7	6	5	4	3	2	1	0
1	TOUCH_X							
0	x6	x5	x4	x3	x2	x1	x0	
0	x13	x12	x11	x10	x9	x8	x7	

**Byte 0: PACKET\_ID (81 hex)**

**Byte 1: bits 0 to 6 of X**

**Byte 2: bits 7 to 14 of X**

#### TOUCH\_Y Packet

**Description:** The TOUCH\_Y packet represents the touch screen Y coordinate. It is sent only if the touch screen is pressed.

**Length:** 3 bytes, including the Packet ID

**Code:** 82hex, 130dec

	7	6	5	4	3	2	1	0
1	TOUCH_Y							
0	y6	y5	y4	y3	y2	y1	y0	
0	y13	y12	y11	y10	y9	y8	y7	

**Byte 0: PACKET\_ID (82 hex)**

**Byte 1: bits 0 to 6 of Y**

**Byte 2: bits 7 to 14 of Y**

#### PEN\_UP Packet

**Description:** The PEN\_UP packet contains no data bytes. It is sent to indicate that the touch screen is not pressed.

**Length:** 1 byte, including the Packet ID

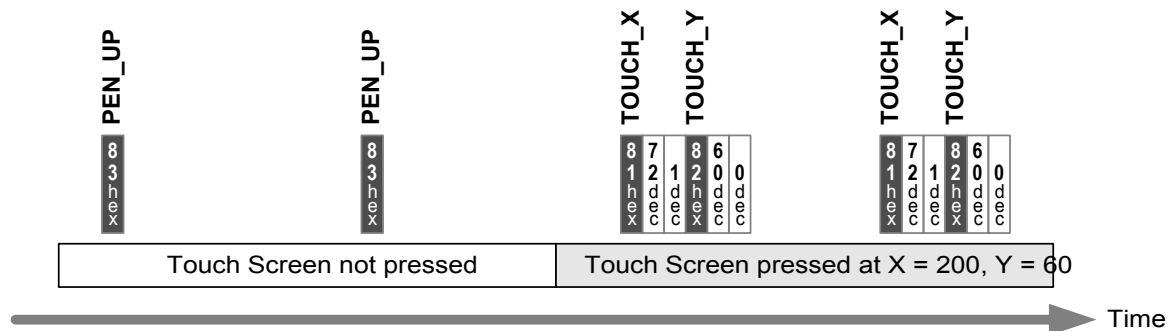
**Code:** 83hex, 131dec

	7	6	5	4	3	2	1	0
1	PEN_UP							

**Byte 0: PACKET\_ID (83 hex)**

### Example:

The drawing below shows an example of the data sent by the ezLCD+, when the CalibratedXY protocol is selected.

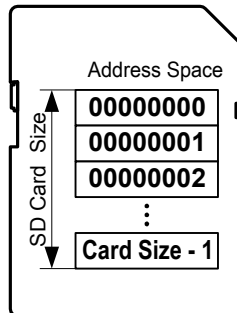


## 5 SD Card Operations

### 5.1 Introduction

The ezLCD+ has a set of [commands](#), which perform write and read operation on the SD (Secure Digital) memory card. The SD Memory Card Specification V1.0 is supported.

The SD Memory Card is "seen" by the ezLCD+ as an external memory (8bit x Card Size), as it is shown on the drawing below



The SD Card Size can be read by using the [SD\\_SIZE](#) command.

The ezLCD+ firmware supports 2 types of SD Card operations:

1. [File Operations](#)

The SD card is treated as a formatted disk. FAT12, FAT16 and FAT32 file systems are supported. The user data is stored in files. Besides the ezLCD+, the files may be read or written by SD Reader/Writer connected to any computer. For more information, please refer to Chapter: [SD File Operations](#).

2. [Raw Operations](#)

The SD card is treated as a memory. The user data is stored at addressed memory locations. For more information, please refer to Chapter: [SD Raw Operations](#).

## 5.2 SD File Operations

The SD card is treated as a formatted disk. The supported file systems are: FAT-12, FAT-16 and FAT32. The user data is stored in files. Besides the ezLCD+, the files may be read or written by SD Reader/Writer connected to any computer.

### Formatting the SD Card.

In order to perform File Operations, the **SD Card has to be formatted in FAT-32, FAT-16 or FAT12**. The ezLCD+ will not perform any File Operations on the unformatted SD Card, nor it will do that on the card formatted with the file system other than FAT-32, FAT-16 or FAT-12.

The SD card can be formatted:

- outside the ezLCD+ by using SD Reader/Writer connected to the PC, or
- inside the ezLCD+ by sending [SD\\_FORMAT](#) command to the ezLCD+, or
- inside the ezLCD+ by using SDformat.exe supplied with the [Source Code Examples](#)

### About the supported file systems

	FAT12	FAT16	FAT32
<b>Full Name</b>	File Allocation Table		
	12-bit version	16-bit version	32-bit version
<b>Introduced</b>	1977	July 1988	August 1996
<b>Max file size</b>	32 MB	2 GB	4 GB
<b>Max number of files</b>	4,077	65,517	268,435,437
<b>Max volume size</b>	32 MB	2 GB	8 TB

### Summary of the SD File Operations commands.

Command	Description
<a href="#">SD_FORMAT</a>	Formats the SD in the specified file system
<a href="#">SD_FILE_LIST</a>	Obtains the list of files and sub-directories which reside in the specified SD Directory.
<a href="#">SD_FIND_FIRST</a> and <a href="#">SD_FIND_NEXT</a>	Obtain the list of SD files and sub-directories (one by one), which match the specified search pattern.
<a href="#">SD_FILE_OPEN</a>	Opens an existing SD file for reading or writing. File Position Index is set to 0. In order to open non-existing, new file, use the command <a href="#">SD_FILE_CREATE</a>
<a href="#">SD_FILE_CREATE</a>	Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.
<a href="#">SD_FILE_CLOSE</a>	Closes SD Flash file.
<a href="#">SD_FILE_CLOSE_ALL</a>	Closes all opened SD Flash files.
<a href="#">SD_FILE_GET_SIZE</a>	Gets the size (in bytes) of the opened SD Flash file.
<a href="#">SD_FILE_READ</a>	Reads the specified number of bytes from the opened SD Flash file, starting from File Position Index.
<a href="#">SD_FILE_WRITE</a>	Writes the specified number of bytes to the opened SD Flash file, starting from File Position Index. File Position Index is incremented by the

Command	Description
	number of the bytes written.
<a href="#">SD_FILE_SEEK</a>	Moves the File Position Index of the opened SD Flash file by the specified number of bytes, from the position specified by the parameter.
<a href="#">SD_FILE_REWIND</a>	Moves the File Position Index to the beginning of the opened SD Flash file.
<a href="#">SD_FILE_TELL</a>	Gets the File Position Index of the opened SD Flash file.
<a href="#">SD_FILE_DELETE</a>	Deletes the SD file.
<a href="#">SD_FOLDER_CREATE</a>	Creates a new folder (directory) on the SD.
<a href="#">SD_FOLDER_DELETE</a>	Deletes an empty folder (directory) on the SD.
<a href="#">SD_SPACE_INFO</a>	Gets the information about the space usage (in bytes) of the formatted SD Card.
<a href="#">SD_PUT_ICON</a>	Reads and displays the bitmap file.
<a href="#">SD_SCREEN_CAPTURE</a>	Saves an image of the displayed screen to the SD as .bmp file.

#### About the File Position Index

The File Position Index specifies the Read/Write position offset (in bytes) from the beginning of the file. Upon opening of the file, the File Position Index is set to 0. The File Position Index is incremented by the subsequent read or write operations on the opened file.

#### About the SD File Path used in the commands:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

#### About the SD Directory/File Path (Search Pattern) used in the [SD\\_FILE\\_LIST](#), [SD\\_FIND\\_FIRST](#) and [SD\\_FIND\\_NEXT](#) commands:

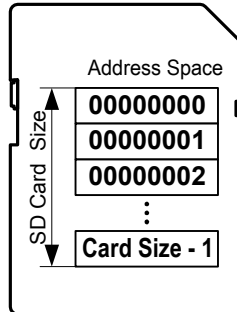
- Directory Path specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: '\*' and '?' are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Directory Path is not case-sensitive. The drive and root directory do not have to be indicated, for example: A:/Cat/Jumped/Over, CAT/jumped/OvEr/ and cat/jumped/over specify the same.
- Long directory names are supported, however the Directory Path + NULL may not exceed 256 bytes.

#### About the Folder Path used in the [SD\\_FOLDER\\_CREATE](#) and [SD\\_FOLDER\\_DELETE](#) commands:

- Folder Path specifies the full path to the directory on the SD.
- Directories (folders) should be separated by: / (**not by:** \ like in Windows and DOS).
- Long names are supported, however the Folder Path (+ NULL) may not exceed 256 bytes.

### 5.3 SD Raw Operations

The SD card is treated as an external memory (8bit x Card Size). The user data is stored at addressed memory locations. Obviously, the SD Card does not have to be formatted.



#### Summary of the SD Raw Operations commands.

Command	Description
<a href="#">SD_SIZE</a>	Gets the physical size (in bytes) of the SD Card.
<a href="#">SD_RAW_READ</a>	Reads the data from SD starting from the specified SD address.
<a href="#">SD_RAW_WRITE</a>	Writes the data on SD starting from the specified SD address.
<a href="#">SD_INSERTED</a>	Checks if the SD card is inserted

## 6 Interfacing with Windows USB Driver

File favr-32.dll is the Dynamic Link Library, containing functions, which facilitate interfacing with the USB driver. File favr-32.dll is installed in the system during USB driver installation.

### Supporting Files

favr-32.h	- prototypes of favr-32.dll <a href="#">functions</a> .
favr-32_x86.lib	- favr-32.dll type library for 32-bit Windows (all processors). Format: COFF (Microsoft, PellesC, GCC)
favr-32_omf_x86.lib	- favr-32.dll type library for 32-bit Windows (all processors). Format: OMF (Borland)
favr-32_amd64.lib	- favr-32.dll type library for 64-bit run by amd64 processor. Format: COFF (Microsoft, PellesC, GCC)
favr-32_ia64.lib	- favr-32.dll type library for 64-bit run by ia64 processor. Format: COFF (Microsoft, PellesC, GCC)

All of the above files are supplied with [Source Code Examples](#).

**Note:** Since the above libraries are only of type definition kind, it may be possible that 32-bit Windows libraries are compatible with all other platforms.

## 6.1 favr-32.dll functions

### Standard Functions



#### Favr32GetDeviceCount

**Description:** Gets the number of connected FAVR-32 devices

**Prototype:** `DWORD __stdcall Favr32GetDeviceCount(VOID);`

**Returns:** Number of connected FAVR-32 devices

---



#### Favr32Open

**Description:** Opens FAVR-32 for write and read operations

**Prototype:** `BOOL __stdcall Favr32Open(DWORD dev_no);`

**Arguments:** dev\_no - Favr-32 device no (0 - 126)

**Returns:** TRUE - success  
FALSE - failure

---



#### Favr32Close

**Description:** Closes FAVR-32 and frees the allocated memory

**Prototype:** `BOOL __stdcall Favr32Close(DWORD dev_no);`

**Arguments:** dev\_no - Favr-32 device no (0 - 126)

**Returns:** TRUE - success  
FALSE - failure

---



#### Favr32CloseAll

**Description:** Closes FAVR-32 and frees the allocated memory

**Prototype:** `VOID __stdcall Favr32CloseAll(VOID);`

---

 **Favr32SetRefreshFunction**

**Description:** Sets the PC Display Refresh Function, which should be periodically called during the waiting periods, to prevent temporary freezing of the windows display. The Display Refresh Function is set to NULL upon loading of favr-32.dll.

**Prototype:** `VOID __stdcall Favr32SetRefreshFunction(fp_ptr_t refresh);`

**Arguments:** refresh - Display Refresh Function (NULL for no refresh).  
fp\_ptr\_t is defined as:  
`typedef VOID (CALLBACK *fp_ptr_t)(VOID);`

**Example:**

```
VOID CALLBACK MyRefresh(VOID)
{
    Application->ProcessMessages();
}

void Whatever(void)
{
    ... // Some code here
    Favr32SetRefreshFunction(MyRefresh);
    ... // Some code here
}
```

 **Favr32GetPath**

**Description:** Gets a path string of the opened FAVR-32 device. This is mostly useless function

**Prototype:** `LPCTSTR __stdcall Favr32GetPath(DWORD dev_no);`

**Arguments:** dev\_no - Favr-32 device no (0 - 126)

**Returns:** string in case of success, NULL in case of failure.

 **Favr32WriteBfr**

**Description:** Writes a buffer into the FAVR-32. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `DWORD __stdcall Favr32WriteBfr(DWORD dev_no, PVOID bfr, DWORD length);`

**Arguments:** dev\_no - Favr-32 device no (0 - 126)  
bfr - buffer to write  
length - number of bytes to write

**Returns:** Number of bytes written



### Favr32WriteByte

**Description:** Writes a single byte into the FAVR-32. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `DWORD __stdcall Favr32WriteByte(DWORD dev_no, UCHAR d);`

**Arguments:** `dev_no` - Favr-32 device no (0 - 126)  
`d` - byte to write

**Returns:** Number of bytes written

---



### Favr32ReadBfrWithTimeout

**Description:** Reads data from FAVR-32 into a buffer. Timeout is specified. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `DWORD __stdcall Favr32ReadBfrWithTimeout(DWORD dev_no, PVOID bfr, DWORD length, DWORD timeout);`

**Arguments:** `dev_no` - Favr-32 device no (0 - 126)  
`bfr` - buffer to read to  
`length` - number of bytes to read  
`timeout` - timeout in milliseconds

**Returns:** Number of bytes read

---



### Favr32ReadBfr

**Description:** Reads data from FAVR-32 into a buffer. Timeout is calculated automatically based on the number of bytes to be read. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `DWORD __stdcall Favr32ReadBfr(DWORD dev_no, PVOID bfr, DWORD length);`

**Arguments:** `dev_no` - Favr-32 device no (0 - 126)  
`bfr` - buffer to read to  
`length` - number of bytes to read

**Returns:** Number of bytes read

---



### Favr32WaitForByte

**Description:** Waits for the particular byte from FAVR-32 or timeout. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `INT32 __stdcall Fav32WaitForByte(DWORD dev_no, UCHAR d, DWORD timeout);`

**Arguments:**

- dev\_no - Fav32 device no (0 - 126)
- d - byte to wait for
- timeout - timeout in milliseconds

**Returns:**

- 1 - Success (byte received)
- 0 - Failure (timeout)

### Fav32WaitForByteOrError

**Description:** Waits for the particular data or error byte from FAVR-32, or timeout. If FAVR-32 is closed, it is opened automatically.

**Prototype:** `INT32 __stdcall Fav32WaitForByteOrError(DWORD dev_no, UCHAR d, UCHAR e, DWORD timeout);`

**Arguments:**

- dev\_no - Fav32 device no (0 - 126)
- d - byte to wait for
- e - error byte
- timeout - timeout in milliseconds

**Returns:**

- 1 - Success (byte received)
- 0 - Failure (timeout)
- 1 - Error byte received

### Fav32FlushRead

**Description:** Empties the USB driver Rx buffer and clears the cache.

**Prototype:** `VOID __stdcall Fav32FlushRead(DWORD dev_no);`

**Arguments:** dev\_no - Fav32 device no (0 - 126)

### Fav32GetSN

**Description:** Gets the value of the User Device Serial Number (Configuration Key: SerialNo). This works faster than ezLCD+ command: GET\_SERIAL\_NO, since the serial number is retrieved from the device path without querying ezLCD+. Configuration Keys are described in "ezLCD+10x Manual".

**Prototype:** `DWORD __stdcall Favr32GetSN(DWORD dev_no);`

**Arguments:** `dev_no` - Favr-32 device no (0 - 126)

**Returns:** User Device Serial Number

---



### Favr32GetDeviceNoFromSN

**Description:** Gets first found Favr-32 device no (0 -126) of the unit which User Device Serial Number (Configuration Key: SerialNo) matches the one specified in the argument. This function is useful when more than one ezLCDs are connected to the computer through USB port. Configuration Keys are described in *"ezLCD+10xManual"*.

**Prototype:** `DWORD __stdcall Favr32GetDeviceNoFromSN(DWORD sn);`

**Arguments:** `sn` - Favr-32 User Device Serial No

**Returns:** Favr-32 device no (0 - 126), or any other number if the device is not connected

**Example:** The following code sends CLS command to the device with S/N: 1234

```
int dev_no;
dev_no = Favr32GetDeviceNoFromSN(1234);
if (dev_no < 127)
{
    Favr32WriteByte(dev_no, 0x21);
    Favr32Close(dev_no);
}
```

---

## Primitive Functions

The functions described below are internally called by the standard functions (described above). Descriptions of primitive functions are given for reference only.

### Favr32Write

**Description:** Provides an overlapped writing primitive to OUT pipe of the opened Favr-32 device.  
Caution: This is a "primitive" function. Consider using the following functions instead:

```
Favr32WriteBfr  
Favr32WriteByte
```

**Prototype:** `BOOL __stdcall Favr32Write(DWORD dev_no, PVOID pData, DWORD dwLen, PDWORD pLength, DWORD dwMilliseconds);`

**Arguments:**

dev_no	- Favr-32 device no (0 - 126)
pData	- buffer to write
dwLen	- number of bytes to write
pLength	- pointer to number of bytes written
dwMilliseconds	- timeout in milliseconds

**Returns:**

TRUE	- success
FALSE	- failure

---

### Favr32Read

**Description:** Provides an overlapped reading primitive from IN pipe of the opened Favr-32 device.  
Caution: This is a "primitive" function. Consider using the following functions instead:

```
Favr32ReadBfr  
Favr32ReadBfrWithTimeout  
Favr32WaitForByte  
Favr32WaitForByteOrError
```

**Prototype:** `BOOL __stdcall Favr32Read(DWORD dev_no, PVOID pData, DWORD dwLen, PDWORD pLength, DWORD dwMilliseconds);`

**Arguments:**

dev_no	- Favr-32 device no (0 - 126)
pData	- buffer to read to
dwLen	- number of bytes to read
pLength	- pointer to number of bytes read
dwMilliseconds	- timeout in milliseconds

**Returns:**

TRUE	- success
FALSE	- failure

## 7 Source Code Examples

Source Code Examples are provided in order to illustrate:

- sending commands to the ezLCD+
- receiving ezLCD+ response
- interfacing with the ezLCD+ USB driver
- reading and writing information on the SD flash card attached to the ezLCD+

All of the examples show the usage of the SD access commands, because most of those commands require a response from the ezLCD+, thus providing better educational value.

**Note:** The examples are provided on the "as is" base.  
There is no warranty whatsoever.

ezLCD+ Interface

All of the examples use USB to communicate with the ezLCD+. They use the [favr-32.dll](#) to interface with the USB driver. The favr-32.dll is installed with USB driver. The header file (favr-32.h) and type libraries for favr-32.dll are provided with the examples.

### Development Environment

All examples are written in portable 'C'.

'MS Visual C++ 6.0', 'Borland C++ Builder 6.0' and 'Pelles C' projects are provided with the examples, however any windows C compiler should be able to build the examples without any errors.

Pelles C is a lightweight freeware integrated development environment for Windows and Pocket PC programming in the C language. It is built and maintained by Mr. Pelle Orinius. Pelles C is available for FREE from:

<http://www.pellesc.de/>

'Borland C++ Builder 6.0' projects can also be opened and compiled by the newest 'Borland Turbo C++ Explorer' available for FREE from: <http://www.turboexplorer.com/cpp>

### Folders

Exe - build executables  
Sources - 'C' sources  
VisualC - 'MS Visual C++ 6.0' projects and type libraries  
Borland - 'Borland C++ Builder 6.0' projects and type libraries  
PellesC - 'Pelles C' projects and type libraries  
Lib - Type libraries for favr-32.dll

### Projects

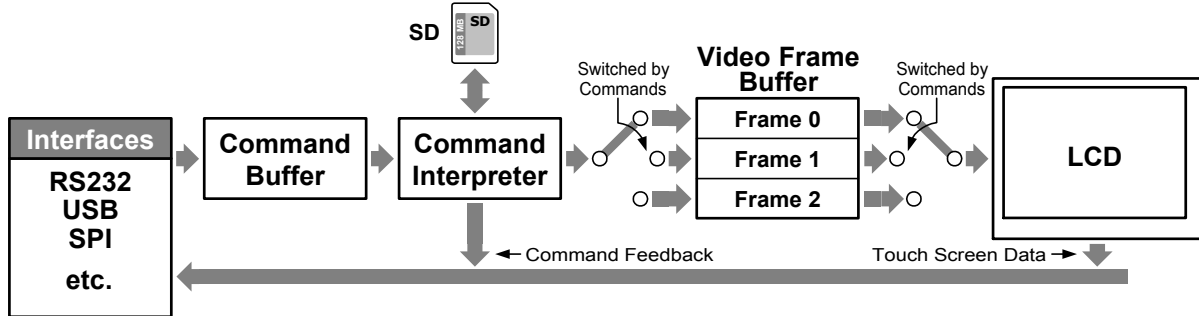
SDicon - Displays a bitmap from the SD card on the ezLCD+ screen  
Sources: icon.c  
Headers: CmdCodes.h, favr-32.h

SDsize - Reads and prints the size of the SD card  
Sources: size.c  
Headers: CmdCodes.h, favr-32.h

- SDflist - Reads and prints the list the files which reside in the particular folder of the SD card  
Sources: flist.c  
Headers: CmdCodes.h, favr-32.h
- SDfsize - Reads and displays the size of the file from SD card  
Sources: fsize.c  
Headers: CmdCodes.h, favr-32.h
- SDfget - Copies the file from SD Card to the PC  
Sources: fget.c  
Headers: CmdCodes.h, favr-32.h
- SDfput - Copies the file from the PC to SD Card  
Sources: fput.c  
Headers: CmdCodes.h, favr-32.h
- SDfdel - Deletes the file from the SD Card  
Sources: fdel.c  
Headers: CmdCodes.h, favr-32.h
- SDrawrd - Reads and prints the raw data from the SD Card  
Sources: rawrd.c  
Headers: CmdCodes.h, favr-32.h
- SDformat - Formats SD card in FAT32  
Sources: format.c  
Headers: CmdCodes.h, favr-32.h

## 8 ezLCD+ Commands

The ezLCD+ can be externally driven by a set of [commands](#). The commands can be sent to the ezLCD+ through any of the implemented interfaces (USB, RS232, SPI, etc.)



Each of the implemented interfaces uses the same set of [Commands](#). The ezLCD+ Commands implement Canvas-type drawing on the LCD screen surface.

Upon arrival, the ezLCD+ Commands are stored into the 1 Mega byte long **Command Buffer** as shown above.

All interfaces use the same Command Buffer. The **Command Interpreter**, picks up byte-by-byte the commands stored in the Command Buffer and draws the corresponding graphics on one of the three frames of the **Video Frame Buffer**. The drawing frame is selected by the command: [SET\\_DRAW\\_FRAME](#). The LCD screen displays the image drawn one of the frames of the **Video Frame Buffer**. The drawing frame is selected by the command: [SET\\_DISP\\_FRAME](#). The commands are processed on a First-In, First-Out principle.

### Example:

The following commands will draw a green circle with a radius of 60 pixels, and centered at position:  $x = 160, y = 100$ .

Data sent to the ezLCD+ byte by byte (Columns: Value and Format):

Mnemonic	Value	Format	Comment
<a href="#">SET_COLOR_RGB</a>	31	hex	Set the drawing color to:
Red	0	hex	green
Green	FF	hex	
Blue	0	hex	
<a href="#">SET_XHYH</a>	33	hex	Set the drawing position to:
x MSB	0	dec	x (column) = 160
x LSB	160	dec	
y MSB	0	dec	y (row) = 100
y LSB	100	dec	
<a href="#">CIRCLE_RH</a>	89	hex	Draw the circle with the radius of:
r MSB	0	dec	60 pixels
r LSB	60	dec	

## 8.1 Command Summary

### Current Drawing Position

Command	Description
<a href="#">SET_XHYH</a>	Sets the Current Position
<a href="#">SET_XH</a>	Sets only the X-coordinate of the Current Position
<a href="#">SET_YH</a>	Sets only the Y-coordinate of the Current Position
<a href="#">SAVE_POSITION</a>	Stores the Current Position to the Position ID
<a href="#">RESTORE_POSITION</a>	Restores the Current Position saved by the <a href="#">SAVE_POSITION</a> command

### Drawing Colors

Command	Description
<a href="#">SET_COLOR_RGB</a>	Sets the Current Drawing Color
<a href="#">SET_ALPHA</a>	Sets value of the transparency <a href="#">Alpha</a>
<a href="#">SET_TR_COLOR_RGB</a>	Specifies the color, which is ignored during <a href="#">Bitmap</a> drawing
<a href="#">TR_COLOR_NONE</a>	Sets the <a href="#">transparent color</a> of bitmaps to none
<a href="#">SET_BG_COLOR_RGB</a>	Sets the Background Color for <a href="#">PRINT_CHAR_BG</a> and <a href="#">PRINT_STRING_BG</a>
<a href="#">REPLACE_COLOR</a>	Replaces color in the Edit Rectangle set by <a href="#">SET_EDIT_RECT</a> command.

### Drawing Pen

Command	Description
<a href="#">SET_PEN_SIZE</a>	Sets the size of drawing <a href="#">Pen</a> (both width and height)
<a href="#">SET_PEN_HEIGHT</a>	Sets height of the drawing <a href="#">Pen</a> . Affects only drawing of curves

### Plotting Single Pixels

Command	Description
<a href="#">PLOT</a>	Plots a point at Current Position in Current Color.
<a href="#">PLOT_XHYH</a>	Plots a point in Current Color at the specified position.

### Lines

Command	Description
<a href="#">LINE_TO_XHYH</a>	Draws a line in Current Color, from Current Position to the specified position
<a href="#">H_LINEH</a>	Quickly draws a horizontal line from the Current Position to the column specified by the parameter
<a href="#">V_LINEH</a>	Quickly draws a vertical line from Current Position, to the row specified by the parameter

### Curves

Command	Description
<a href="#">CIRCLE_RH</a>	Draws a circle in Current Color centered at Current Position

Command	Description
<a href="#">CIRCLE_RH_FILL</a>	Draws a circle filled with Current Color centered at Current Position
<a href="#">ARCH</a>	Draws an arc in Current Color, with the center at Current Position, starting on Begin Angle and ending on End Angle
<a href="#">PIEH</a>	Draws a pie filled with Current Color with the center at Current Position, starting on Begin Angle and ending on End Angle
<a href="#">ELLIPSE_AHBH</a>	Draws an ellipse with a Pen in Current Color centered at Current Position
<a href="#">ELLIPSE_AHBH_FILL</a>	Draws an ellipse filled with Current Color and centered at Current Position
<a href="#">ELLIPSE_ARCH</a>	Draws an ellipse arc with a Pen in Current Color centered at Current Position starting from Begin Angle and ending on End Angle.
<a href="#">ELLIPSE_PIEH</a>	Draws an ellipse pie filled with Current Color centered at Current Position starting from Begin Angle and ending on End Angle

### Polygons

Command	Description
<a href="#">BOXHH</a>	Draws a rectangle
<a href="#">BOXHH_FILL</a>	Draws a rectangle filled with Current Color
<a href="#">POLYGON</a>	Draws a polygon filled with the Current Color. The first vertex is located at the Current Position

### Filling

Command	Description
<a href="#">CLS</a>	Clears the screen by filling it with the Current Color
<a href="#">FILL</a>	Fills an area with the Current Color. Filling seed is located at the Current Position. The fill area is restricted by the connected pixels, which color is different than the seed pixel.
<a href="#">FILL_BOUND</a>	Fills an area with the Current Color. Filling seed is located at the Current Position. The fill area is restricted by the connected pixels, which color is specified by the Bound Color parameter.

### Editing

Command	Description
<a href="#">SET_EDIT_RECT</a>	Sets the rectangle for editing (replacing colors, etc).
<a href="#">REPLACE_COLOR</a>	Replaces color in the Edit Rectangle set by <a href="#">SET_EDIT_RECT</a> command.

### Frames

Command	Description
<a href="#">SET_DISP_FRAME</a>	Sets the <a href="#">Frame</a> to be displayed on the screen
<a href="#">SET_DRAW_FRAME</a>	Sets the <a href="#">Frame</a> that ezLCD+ commands draw on
<a href="#">COPY_FRAME</a>	Copies all the contents of the source <a href="#">Frame</a> to the destination Frame
<a href="#">COPY_RECT</a>	Copies the rectangular portion of one frame to the other
<a href="#">MERGE_FRAME</a>	Merges the contents of the source <a href="#">Frame</a> with the destination Frame

Command	Description
	according to the value of transparency <a href="#">Alpha</a> . The resulting image is held in the destination Frame.
<a href="#">MERGE_RECT</a>	Merges the rectangular portion of one frame with the other according to the value of transparency <a href="#">Alpha</a> .

### Displaying Bitmap Images

Command	Description
<a href="#">PUT_BITMAP_RGB</a>	Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP
<a href="#">PUT_PICT_NO</a>	Displays a bitmap (icon) with its upper-left corner positioned at the Current Position. The bitmap is read from the User ROM
<a href="#">SD_PUT_ICON</a>	Displays an icon with its upper-left corner positioned at the Current Position. The icon is read from the file on the SD card attached to the SD/MMC interface.

### Fonts

Command	Description
<a href="#">SELECT_FONT</a>	Selects the <a href="#">Bitmap Font</a> from User ROM
<a href="#">SET_FT_FONT</a>	Selects and configures <a href="#">True Type Font</a> from User ROM.
<a href="#">SD_LOAD_FONT</a>	Loads and selects font file from SD card. Both, <a href="#">Bitmap Fonts</a> and <a href="#">True Type Fonts</a> are supported.
<a href="#">SET_FT_UNIBASE</a>	Sets the base code for the Unicode characters, so they can be printed using 8-bit codes. Only the <a href="#">True Type Fonts</a> are affected.

### Printing Text

Command	Description
<a href="#">PRINT_CHAR</a>	Prints an ASCII-coded character at the Current Position
<a href="#">PRINT_FT_UNICHAR</a>	Prints a 16-bit Unicode-coded <a href="#">True Type</a> character at the Current Position
<a href="#">PRINT_STRING</a>	Prints a null-terminated ASCII-coded string starting at the Current Position
<a href="#">PRINT_FT_UNISTRING</a>	Prints a null-terminated 16-bit Unicode-coded <a href="#">True Type</a> string starting at the Current Position
<a href="#">PRINT_CHAR_BG</a>	Prints a bitmap character at Current Position on the background specified by the <a href="#">SET_BG_COLORH</a> command
<a href="#">PRINT_STRING_BG</a>	Prints null-terminated string of bitmap characters starting at Current Position on the background specified by <a href="#">SET_BG_COLORH</a> command
<a href="#">CACHE_FT_CHARS</a>	Renders <a href="#">True Type</a> characters in memory and puts the resulting glyphs into cache memory. Characters are specified using 8-bit ASCII codes. Cached characters can be displayed instantly, because they do not need any special processing.
<a href="#">CACHE_FT_UNICHARS</a>	Renders <a href="#">True Type</a> characters in memory and puts the resulting glyphs into cache memory. Characters are specified using 16-bit Unicode codes. Cached characters can be displayed instantly, because they do not need any special processing.

Command	Description
<a href="#">SET_FT_ANGLE</a>	Sets the angle, in which the <a href="#">True Type Font</a> characters and strings are printed
<a href="#">TEXT_NORTH</a> , <a href="#">TEXT_SOUTH</a> , <a href="#">TEXT_EAST</a> , <a href="#">TEXT_WEST</a>	Set the orientation of the text.

### Backlight

Command	Description
<a href="#">LIGHT_ON</a>	Turns on the screen backlight
<a href="#">LIGHT_OFF</a>	Turns off the screen backlight
<a href="#">LIGHT_BRIGHT</a>	Sets the brightness of the screen backlight

### Touch Screen

Command	Description
<a href="#">CALIBRATE_TOUCH</a>	Starts the touch screen calibration procedure
<a href="#">TOUCH_PROTOCOL</a>	Changes the default behavior of the ezLCD+ touch control function
<a href="#">BUTTON_DEF_LONG</a>	Defines and draws a touch button
<a href="#">BUTTON_STATE</a>	Changes the state of a previously defined touch button
<a href="#">BUTTONS_ALL_UP</a>	Changes the state of all defined touch buttons to Button Up
<a href="#">BUTTONS_DELETE_ALL</a>	Deletes all touch buttons

### SD I/O

Command	Description
<a href="#">SD_FORMAT</a>	Formats the SD in the specified file system
<a href="#">SD_FILE_LIST</a>	Obtains the list of files and sub-directories which reside in the specified SD Directory.
<a href="#">SD_FIND_FIRST</a> and <a href="#">SD_FIND_NEXT</a>	Obtain the list of SD files and sub-directories (one by one), which match the specified search pattern.
<a href="#">SD_FILE_OPEN</a>	Opens an existing SD file for reading or writing. File Position Index is set to 0. In order to open non-existing, new file, use the command <a href="#">SD_FILE_CREATE</a>
<a href="#">SD_FILE_CREATE</a>	Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.
<a href="#">SD_FILE_CLOSE</a>	Closes SD Flash file.
<a href="#">SD_FILE_CLOSE_ALL</a>	Closes all opened SD Flash files.
<a href="#">SD_FILE_GET_SIZE</a>	Gets the size (in bytes) of the opened SD Flash file.
<a href="#">SD_FILE_READ</a>	Reads the specified number of bytes from the opened SD Flash file, starting from File Position Index.
<a href="#">SD_FILE_WRITE</a>	Writes the specified number of bytes to the opened SD Flash file, starting from File Position Index. File Position Index is incremented by the number of the bytes written.

Command	Description
<a href="#">SD_FILE_SEEK</a>	Moves the File Position Index of the opened SD Flash file by the specified number of bytes, from the position specified by the parameter.
<a href="#">SD_FILE_REWIND</a>	Moves the File Position Index to the beginning of the opened SD Flash file.
<a href="#">SD_FILE_TELL</a>	Gets the File Position Index of the opened SD Flash file.
<a href="#">SD_FILE_DELETE</a>	Deletes the SD file.
<a href="#">SD_FOLDER_CREATE</a>	Creates a new folder (directory) on the SD.
<a href="#">SD_FOLDER_DELETE</a>	Deletes an empty folder (directory) on the SD.
<a href="#">SD_SPACE_INFO</a>	Gets the information about the space usage (in bytes) of the formatted SD Card.
<a href="#">SD_PUT_ICON</a>	Reads and displays the bitmap file.
<a href="#">SD_SCREEN_CAPTURE</a>	Saves an image of the displayed screen to the SD as .bmp file.
<a href="#">SD_SIZE</a>	Gets the physical size (in bytes) of the SD Card.
<a href="#">SD_RAW_READ</a>	Reads the data from SD starting from the specified SD address.
<a href="#">SD_RAW_WRITE</a>	Writes the data on SD starting from the specified SD address.
<a href="#">SD_INSERTED</a>	Checks if the SD card is inserted

#### Lua Scripts

Command	Description
<a href="#">RUN_LUA</a>	Runs a Lua script
<a href="#">RUN_LUA_ROM</a>	Runs a Lua script from the User ROM

#### CRC Protection

Command	Description
<a href="#">CRC_PKG</a>	Sends CRC-protected data package to the ezLCD+

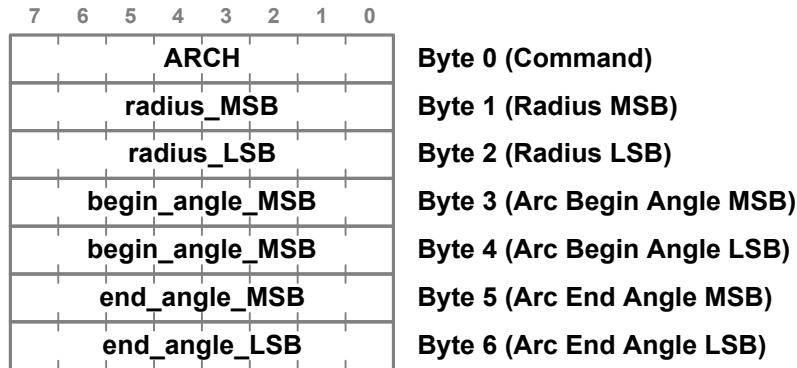
#### System

Command	Description
<a href="#">PING</a>	Checks if the ezLCD+ is connected and ready to receive commands
<a href="#">GET_SERIAL_NO</a>	Gets the value of the User Device Serial Number (Configuration Key: SerialNo)

## 8.2 ARCH

**Description:** Draws an arc in Current Color, with the center at Current Position, starting on Begin Angle and ending on End Angle.

**Code:** **8F**hex, **143**dec



**See Also:** [PIEH](#), [CIRCLE\\_RH](#), [ELLIPSE\\_ARCH](#)

**Angle Coding:** The full angle ( $360^\circ$ ) is equal to 4000hex (16384dec).

To transform degrees to ARC angle units:

$$\text{Angle\_lcd} = \text{Angle\_deg} \times 2048 / 45$$

For example:

$$2048\text{dec} = 800\text{hex} = 45^\circ$$

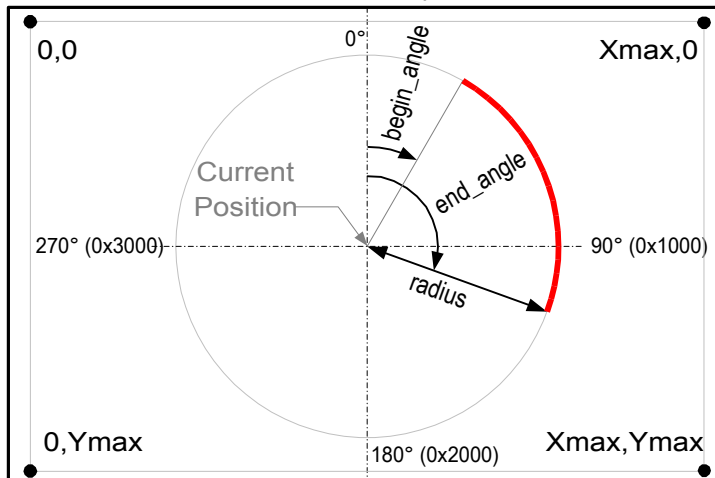
$$4096\text{dec} = 1000\text{hex} = 90^\circ$$

$$8192\text{dec} = 2000\text{hex} = 180^\circ$$

$$12288\text{dec} = 3000\text{hex} = 270^\circ$$

$$16384\text{dec} = 4000\text{hex} = 360^\circ = 0^\circ$$

The angle is oriented clockwise with the zero positioned at the top of the screen, as it is shown on the picture below



**Example:**

The following sequence will draw a green arc from 45 to 225 degrees with the center positioned at (160, 117) and a radius of 80.  
 $225 \times 2048 / 45 = 10240$  (2800hex)

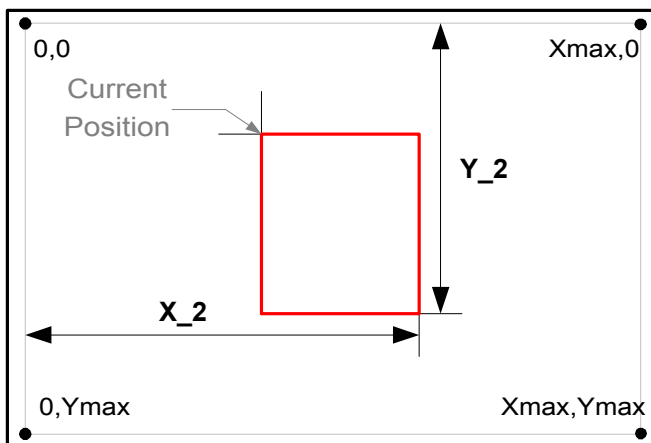
```
SET_COLOR_RGB 31 hex
  Red          0
  Green       FF
  Blue        0 hex
SET_XHYH      33 hex
  0            0 dec (x MSB)
  160         160 dec (x LSB)
  0           0 dec (y MSB)
  117        117 dec (y LSB)
ARCH        8F hex
  0           0 dec (radius MSB)
  80          80 dec (radius LSB)
  08          08 hex (begin_angle MSB)
  00          00 hex (begin_angle LSB)
  28          28 hex (end_angle MSB)
  00          00 hex (end_angle LSB)
```

### 8.3 BOXHH

**Description:** Draws a rectangle.

**Code:** **A4**hex, **164**dec

7	6	5	4	3	2	1	0	
<b>BOXHH</b>								<b>Byte 0 (Command)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x2 MSB)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x2 LSB)</b>
y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 3 (y2 MSB)</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 4 (y2 LSB)</b>



Ref: [Screen Coordinates](#)

See Also: [SET\\_XHYH](#), [BOXHH\\_FILL](#)

#### Example:

The following sequence will draw a rectangle with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

```

SET_XHYH      33 hex
  0           0 dec (x MSB)
 95          95 dec (x LSB)
  0           0 dec (y MSB)
 10          10 dec (y LSB)
BOXHH        A4 hex
  0           0 dec (X_2 MSB)
180          180 dec (X_2 LSB)
  0           0 dec (Y_2 MSB)
120          120 dec (Y_2 LSB)

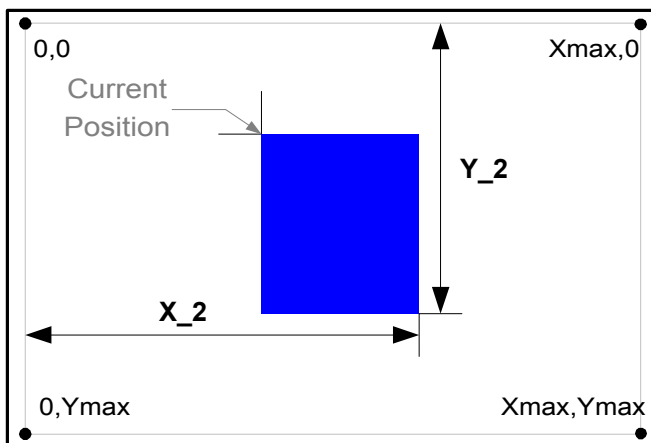
```

## 8.4 BOXHH\_FILL

**Description:** Draws a rectangle filled with Current Color.

**Code:** **A5**hex, **165**dec

7	6	5	4	3	2	1	0	
<b>BOXHH_FILL</b>								<b>Byte 0 (Command)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x2 MSB)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x2 LSB)</b>
y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 3 (y2 MSB)</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 4 (y2 LSB)</b>



**Note:** Since this is a filled figure, [Pen](#) parameters are ignored.

**Ref:** [Screen Coordinates](#)

**See Also:** [SET\\_XHYH](#), [BOXHH](#)

### Example:

The following sequence will draw a blue filled rectangle, with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

```

SET_COLOR_RGB 31 hex
Red           0
Green        0
Blue         FF hex
SET_XHYH     33 hex
0             0 dec (x MSB)
95           95 dec (x LSB)
0             0 dec (y MSB)
10           10 dec (y LSB)
BOXHH_FILL  A5 hex
0             0 dec (X_2 MSB)
180          180 dec (X_2 LSB)
0             0 dec (Y_2 MSB)

```

120

120 dec (Y\_2 LSB)

## 8.5 BUTTON\_DEF\_LONG

**Description:** Defines and draws a touch button

**Code:** B5<sub>hex</sub>, 181<sub>dec</sub>

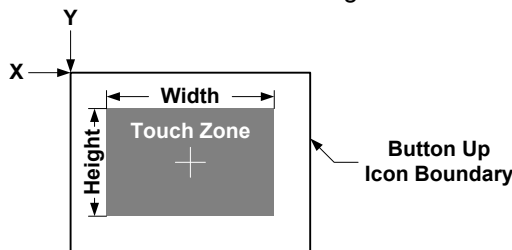
7 6 5 4 3 2 1 0	<b>BUTTON_DEF_LONG</b>	<b>Byte 0: Command</b>
	button_no	<b>Byte 1: Button No (0 to 63)</b>
	state	<b>Byte 2: Initial State (1: Up, 2: Down, 3: Disabled, 4: Non-Visible)</b>
	buton Up icon MSB	<b>Byte 3: } Byte 4: } Icon No in User ROM for button Up (FFFF<sub>hex</sub>= none)</b>
	buton Up icon LSB	
	buton Down icon MSB	<b>Byte 5: } Byte 6: } Icon No in User ROM for button Down (FFFF<sub>hex</sub>= none)</b>
	buton Down icon LSB	
	button Disabled icon MSB	<b>Byte 7: } Byte 8: } Icon No in User ROM for button Disabled (FFFF<sub>hex</sub>= none)</b>
	button Disabled icon LSB	
x15 x14 x13 x12 x11 x10 x9 x8		<b>Byte 9: } Byte 10: } Button upper-left corner X-coordinate</b>
x7 x6 x5 x4 x3 x2 x1 x0		
y15 y14 y13 y12 y11 y10 y9 y8		<b>Byte 11: } Byte 12: } Button upper-left corner Y-coordinate</b>
y7 y6 y5 y4 y3 y2 y1 y0		
	touch_zone_width	<b>Byte 13: Touch Zone width</b>
	touch_zone_height	<b>Byte 14: Touch Zone height</b>

### About the Touch Zone:

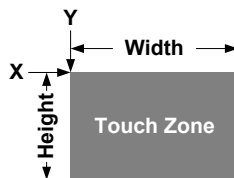
Touch Zone is the active touch response area of the button. It is specified by **With** (Byte 13) and **Height** (Byte 14).

- If the Button Up Icon is defined (Bytes 3 and 4 are not 255), the Touch Zone is centered on it.
- If the Button Up Icon is none (Bytes 3 and 4 are both 255), the position of the upper-left corner of the Touch Zone is specified by **X** (Bytes: 9 and 10) and **Y** (Bytes: 11 and 12).

Both cases are shown on the drawings below:



Button Up Icon is defined (Byte 3 is not 255)



Button Up Icon is none (Byte 3 = 255)

See Also: [BUTTON\\_STATE](#), [BUTTONS\\_ALL\\_UP](#), [BUTTONS\\_DELETE\\_ALL](#), [TOUCH\\_PROTOCOL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Scree Operations](#)
- [ezButton](#)
- [cuButton](#)

### Example:

The following sequence will define the Button No. 4 with the following bitmaps:

- Button Up Icon in User ROM: 8
- Button Down Icon in User ROM: 9
- No Icon for Button Disabled state

The button will be positioned at X = 260 and Y = 170.

It's Touch Zone will have the width of 40 and the height of 30.

The button will be initially drawn using Button Up icon.

```

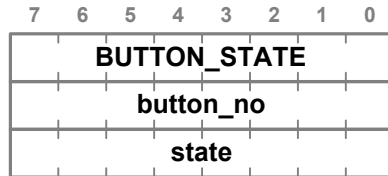
BUTTON_DEF_LONG  B5 hex (Command)
  4              4 dec (Button No)
  1              1 dec (Initial State: Button Up)
  8              0 dec (Button Up Icon No MSB)
  8              8 dec (Button Up Icon No LSB)
  8              0 dec (Button Down Icon No MSB)
  9              9 dec (Button Down Icon No LSB)
 255             255 dec (Button Disabled Icon No MSB)
 255             255 dec (Button Disabled Icon No LSB)
  1              1 dec (Upper-left corner X MSB)
  4              4 dec (Upper-left corner X LSB)
  0              0 dec (Upper-left corner Y MSB)
 170             170 dec (Upper-left corner Y LSB)
  40             40 dec (Width of the Touch Zone)
  30             30 dec (Height of the Touch Zone)

```

## 8.6 BUTTON\_STATE

**Description:** Changes the state of a previously defined touch button

**Code:** **B1**hex, **177**dec



**Byte 0: Command**

**Byte 1: Button No (0 to 63)**

**Byte 2: Button State**

- 0 - Delete permanently
- 1 - Up
- 2 - Down
- 3 - Disabled
- 4 - Non-Visible

### About the Button State:

The button is automatically redrawn after it's state has been changed, if the icon for the new state has been defined by the [BUTTON\\_DEF](#) command.

Deleting the button (Byte 2 = 0) will not erase the button image from the screen. The ezLCD+ just stops reacting to the deleted button events.

Changing the button state to Non-Visible (Byte 2 = 4) will also not erase the button image from the screen. The Non-Visible (Byte 2 = 4) state should mainly be used with the [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command, if we do not wish the button to be initially drawn.

**See Also:** [BUTTON\\_DEF](#), [BUTTON\\_DEF\\_LONG](#), [BUTTONS\\_ALL\\_UP](#), [BUTTONS\\_DELETE\\_ALL](#), [TOUCH\\_PROTOCOL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Screen Operations](#)
- [ezButton](#)
- [cuButton](#)

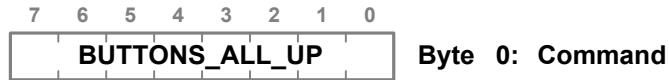
### Example:

The following sequence will change the state of the Button No. 4 to the Button Down. The button will be redrawn using Button Down Icon.

```
BUTTON_STATE  B1 hex (Command)
                4      4 dec (Button No)
                2      2 dec (Button Down)
```

## 8.7 BUTTONS\_ALL\_UP

**Description:** Changes the state of all defined touch buttons to Button Up  
**Code:** **B3**<sub>hex</sub>, **179**<sub>dec</sub>



**Note:**

The button will be automatically redrawn, if the icon for the Button Up has been defined by the [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.

**See Also:** [BUTTON\\_DEF](#), [BUTTON\\_DEF\\_LONG](#), [BUTTON\\_STATE](#), [BUTTONS\\_DELETE\\_ALL](#), [TOUCH\\_PROTOCOL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Screen Operations](#)
- [ezButton](#)
- [cuButton](#)

### Example:

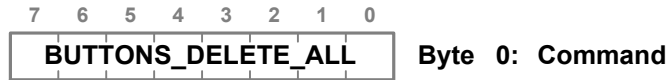
The following sequence will change the state of all Buttons to Up.

```
BUTTONS_ALL_UP B3 hex (Command)
```

## 8.8 BUTTONS\_DELETE\_ALL

**Description:** Deletes all touch buttons

**Code:** **B4**hex, **180**dec



**Note:** Deleting the buttons will not erase their image from the screen. The ezLCD+ will just stop reacting to the button events.

**See Also:** [BUTTON\\_DEF](#), [BUTTON\\_DEF\\_LONG](#), [BUTTON\\_STATE](#), [BUTTONS\\_ALL\\_UP](#), [TOUCH\\_PROTOCOL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Screen Operations](#)
- [ezButton](#)
- [cuButton](#)

### Example:

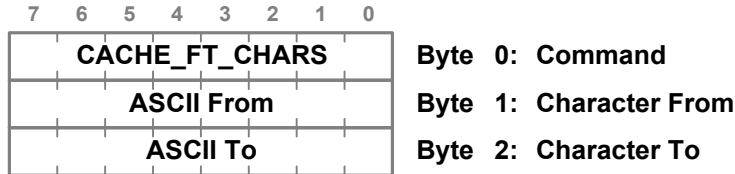
The following sequence will delete all Buttons.

**BUTTONS\_DELETE\_ALL** **B4 hex** (Command)

## 8.9 CACHE\_FT\_CHARS

**Description:** Renders [True Type](#) characters in memory and puts the resulting glyphs into cache memory. Characters are specified using 8-bit ASCII codes. Cached characters can be displayed instantly, because they do not need any special processing.

**Code:** **95**<sub>hex</sub>, **149**<sub>dec</sub>



This command can also cache the Unicode characters:

Unicode Character = Unicode Base + ASCII character.

Please, refer to commands: [SET\\_FT\\_UNIBASE](#)

This command will not cache any characters unless the True Type Font is selected using [SET\\_FT\\_FONT](#) or [SD\\_LOAD\\_FONT](#).

In order to cache characters using 16-bit Unicode codes, use command: [CHACHE\\_FT\\_UNICHARS](#).

### About the font cache:

- Holds the bitmap glyphs of the characters.
- Only the True Type characters are cached.
- Commands: [PRINT\\_STRING](#), [PRINT\\_CHAR](#), [PRINT\\_UNISTRING](#) and [PRINT\\_UNICHAR](#) automatically put the printed characters into cache.
- Cache memory is dynamically allocated. Characters are not cached when there is no memory left.
- Each time the True Type font (or it's size) is changed, the font cache is cleared.
- The [SET\\_FT\\_ANGLE](#) command clears the font glyph cache. No cache is used after this command is issued. In order to re-enable cache use one of the following commands: [TEXT\\_NORTH](#), [TEXT\\_SOUTH](#), [TEXT\\_WEST](#), [TEXT\\_EAST](#)

**See Also:** [CHACHE\\_FT\\_UNICHARS](#), [SET\\_FT\\_UNIBASE](#), [SET\\_FT\\_FONT](#)

### Example:

The following sequence will cache the characters from 'a' to 'z'.

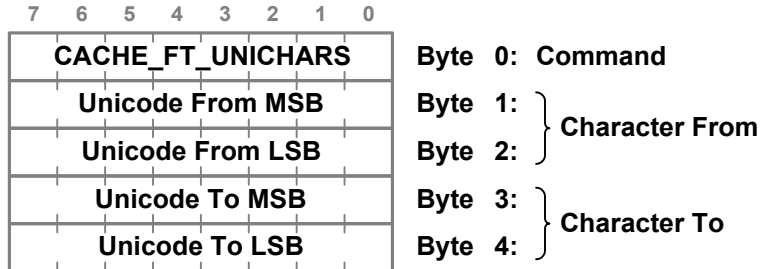
```
SET_FT_FONT      91 hex
  0              0 dec (Font No = 0)
 24             24 dec (Height = 24)
  0             0      (Width = automatic)
SET_FT_UNIBASE  93 hex
  0              0 (Unicode Base MSB)
  0              0 (Unicode Base LSB)
CACHE_FT_CHARS 95 hex
  'a'           61 hex (ASCII from)
  'z'           7A hex (ASCII to)
```



## 8.10 CACHE\_FT\_UNICHARS

**Description:** Renders [True Type](#) characters in memory and puts the resulting glyphs into cache memory. Characters are specified using 16-bit Unicode codes. Cached characters can be displayed instantly, because they do not need any special processing.

**Code:** **96**hex, **150**dec



This command will not cache any characters unless the True Type Font is selected using [SET\\_FT\\_FONT](#) or [SD\\_LOAD\\_FONT](#).

### About the font cache:

- Holds the bitmap glyphs of the characters.
- Only the True Type characters are cached.
- Commands: [PRINT\\_STRING](#), [PRINT\\_CHAR](#), [PRINT\\_UNISTRING](#) and [PRINT\\_UNICHAR](#) automatically put the printed characters into cache.
- Cache memory is dynamically allocated. Characters are not cached when there is no memory left.
- Each time the True Type font (or it's size) is changed, the font cache is cleared.
- The [SET\\_FT\\_ANGLE](#) command clears the font glyph cache. No cache is used after this command is issued. In order to re-enable cache use one of the following commands: [TEXT\\_NORTH](#), [TEXT\\_SOUTH](#), [TEXT\\_WEST](#), [TEXT\\_EAST](#)

See Also: [CACHE\\_FT\\_CHARS](#), [SET\\_FT\\_FONT](#)

### Example:

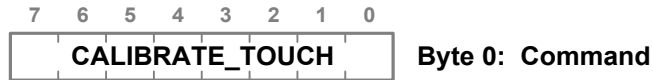
The following sequence will cache the Unicode characters from U+0430 to U+044F.

```
SET_FT_FONT          91 hex
  0                  0 dec (Font No = 0)
 24                  24 dec (Height = 24)
  0                  0      (Width = automatic)
CACHE_FT_UNICHARS  95 hex
  04                 04 hex (Unicode from MSB)
  30                 30 hex (Unicode from LSB)
  04                 04 hex (Unicode to MSB)
  4F                 4F hex (Unicode to LSB)
```

## 8.11 CALIBRATE\_TOUCH

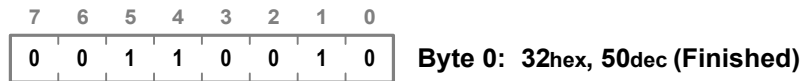
**Description:** Starts the touch screen calibration procedure

**Code:** B6<sub>hex</sub>, 182<sub>dec</sub>



### ezLCD+ Response

When touch screen calibration is finished, the ezLCD+ responds with:



**Note:** It is recommended to wait for the finish of the touch calibration procedure, before sending more commands to the ezLCD+.

### Example:

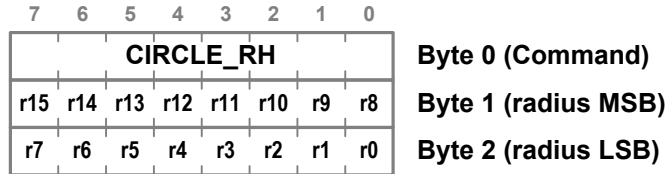
The following sequence will start the touch screen calibration procedure

CALIBRATE\_TOUCH      B6 hex

## 8.12 CIRCLE\_RH

**Description:** Draws a circle in Current Color centered at Current Position.

**Code:** **89**hex, **137**dec



**See Also:** [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#), [ELLIPSE\\_AHBH](#)

### Example:

The following sequence will draw a green circle with the center positioned at (160, 117).

```

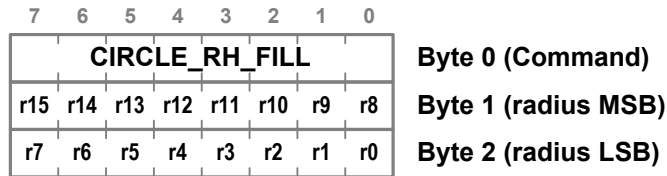
SET_COLOR_RGB 31 hex
Red           0
Green        FF
Blue         0 hex
SET_XHYH     33 hex
0             0 dec (x MSB)
160          160 dec (x LSB)
0            0 dec (y MSB)
117         117 dec (y LSB)
CIRCLE_RH    89 hex
0            0 dec (radius MSB)
80          80 dec (radius LSB)

```

## 8.13 CIRCLE\_RH\_FILL

**Description:** Draws a circle filled with Current Color centered at Current Position.

**Code:** 99hex, 153dec



**Note:** Since this is a filled figure, [Pen](#) parameters are ignored.

**See Also:** [SET\\_XHY](#), [SET\\_COLORH](#), [ELLIPSE\\_AHBH\\_FILL](#)

### Example:

The following sequence will draw a red filled circle with the center positioned at (160, 117).

```

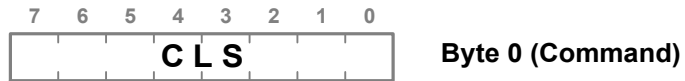
SET_COLOR_RGB  31 hex
Red            FF hex
Green          0
Blue           0 hex
SET_XHYH       33 hex
0              0 dec (x MSB)
160            160 dec (x LSB)
0              0 dec (y MSB)
117            117 dec (y LSB)
CIRCLE_RH_FILL 99 hex
0              0 dec (radius MSB)
80             80 dec (radius LSB)

```

## 8.14 CLS

**Description:** Clears the screen by filling it with the Current Color.

**Code:** 21hex, 33dec



**Note:** CLS is not affected by the transparency [Alpha](#)

**See Also:** [SET\\_COLOR\\_RGB](#)

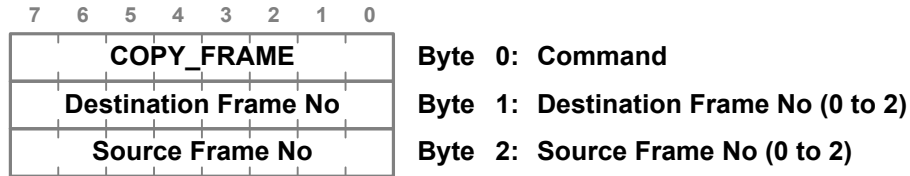
### Example:

The following sequence will clear the screen (and fill it with white).

```
SET_COLOR_RGB 31 hex
Red           FF hex
Green        FF hex
Blue         FF hex
CLS          21 hex
```

## 8.15 COPY\_FRAME

**Description:** Copies all the contents of the source [Frame](#) to the destination Frame.  
**Code:** 53<sub>hex</sub>, 83<sub>dec</sub>



The contents of the destination frame will be completely replaced by the contents of the source frame. This command is not affected by the transparency [Alpha](#). In order to make one frame transparent over another use the command: [MERGE\\_FRAME](#).

When [Alpha](#) = 255, both commands (COPY\_FRAME and MERGE\_FRAME) yield identical results.

**Note:** Copying a frame to itself may yield unpredictable results.

**Ref:** [Frames](#)

**See Also:** [MERGE\\_FRAME](#), [COPY\\_RECT](#), [MERGE\\_RECT](#), [SET\\_DISP\\_FRAME](#), [SET\\_DRAW\\_FRAME](#)

### Example:

The following sequence will copy the contents of Frame 1 to Frame 0

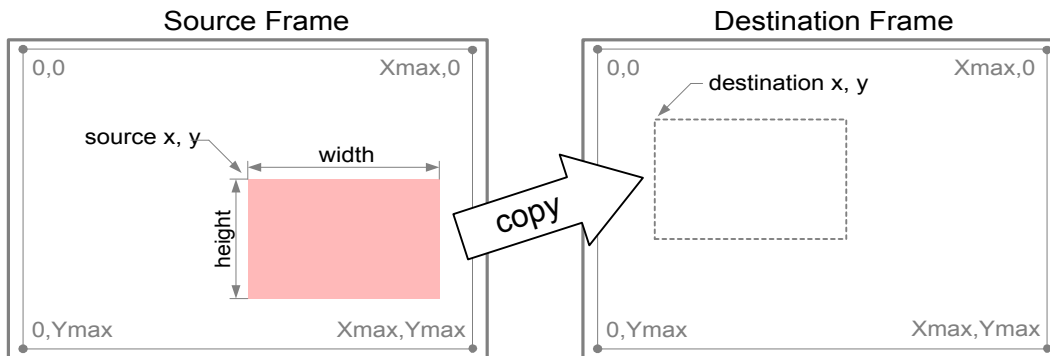
```
COPY_FRAME 53 hex
  0          0 (Destination Frame No)
  1          1 (Source Frame No)
```

## 8.16 COPY\_RECT

**Description:** Copies the rectangular portion of one frame to the other.

**Code:** 55<sub>hex</sub>, 85<sub>dec</sub>

7	6	5	4	3	2	1	0	
<b>COPY_RECT</b>								
Destination Frame No								<b>Byte 0: Command</b>
Source Frame No								<b>Byte 1: Destination Frame No (0 to 2)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 2: Source Frame No (0 to 2)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 3: } Destination Rectangle upper-left corner X-coordinate</b>
y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 4: } Destination Rectangle upper-left corner X-coordinate</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 5: } Destination Rectangle upper-left corner Y-coordinate</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 6: } Destination Rectangle upper-left corner Y-coordinate</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 7: } Source Rectangle upper-left corner X-coordinate</b>
y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 8: } Source Rectangle upper-left corner X-coordinate</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 9: } Source Rectangle upper-left corner Y-coordinate</b>
Width MSB								<b>Byte 10: } Source Rectangle upper-left corner Y-coordinate</b>
Width LSB								<b>Byte 11: } Rectangle Width</b>
Height MSB								<b>Byte 12: } Rectangle Width</b>
Height LSB								<b>Byte 13: } Rectangle Height</b>
								<b>Byte 14: } Rectangle Height</b>



The contents of the destination rectangle will be completely replaced by the contents of the source rectangle. This command is not affected by the transparency [Alpha](#). In order to make one rectangle transparent over another use the command: [MERGE\\_RECT](#).

When [Alpha](#) = 255, both commands (COPY\_RECT and MERGE\_RECT) yield identical results.

**Note:** Copying a rectangle to itself may yield unpredictable results.

**Ref:** [Frames](#)

**See Also:** [MERGE\\_RECT](#), [COPY\\_FRAME](#), [MERGE\\_FRAME](#), [SET\\_DISP\\_FRAME](#), [SET\\_DRAW\\_FRAME](#)

**Example:**

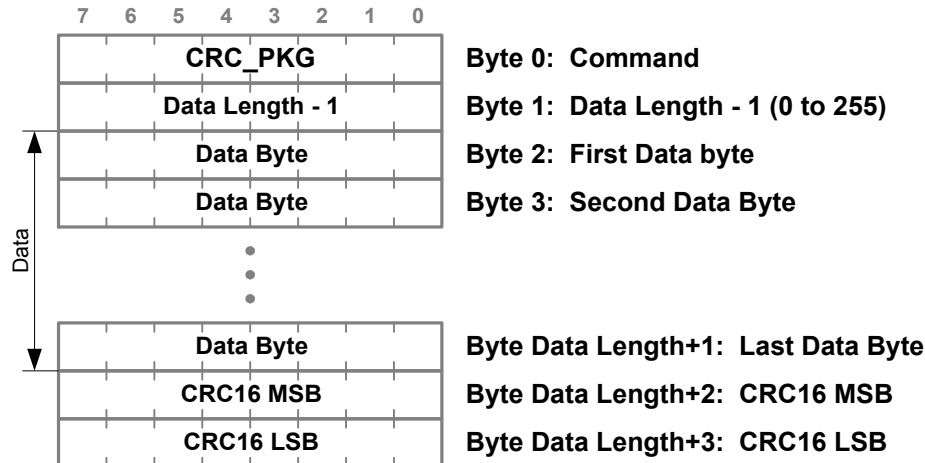
The following sequence will copy the rectangle size 100x50 from Frame 1 (320, 240) to Frame 0 (40, 60)

```
COPY_RECT    55 hex
  0           0 (Destination Frame No)
  1           1 (Source Frame No)
  0           0 dec (Destination X MSB)
  40          40 dec (Destination X LSB)
  0           0 dec (Destination Y MSB)
  60          60 dec (Destination Y LSB)
  1           1 dec (Source X MSB)
  64          64 dec (Source X LSB)
  0           0 dec (Source Y MSB)
  240         240 dec (Source Y LSB)
  0           0 dec (Width MSB)
  100         100 dec (Width LSB)
  0           0 dec (Height MSB)
  50          50 dec (Height LSB)
```

## 8.17 CRC\_PKG

**Description:** Sends CRC-protected data package to the ezLCD+  
Data will be executed by ezLCD+ only if CRCs match.

**Code:** **CC**<sub>hex</sub>, **204**<sub>dec</sub>



### CRC16:

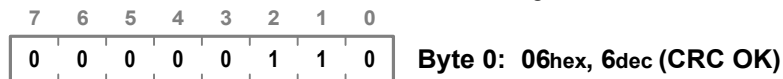
- Calculated over Data Bytes
- CCITT standard CRC-16:  
16 bit, non-reflected CRC  
Polynomial:  $1021_{hex}$   
Seed:  $FFFF_{hex}$   
Final xor: 0

**Note:** The data consists of commands and their arguments. The data has to end on the command boundary.

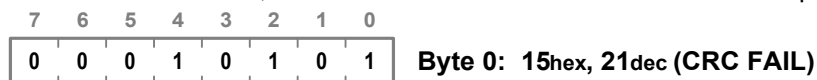
### ezLCD+ Response

After receiving the CRC\_PKG command, the ezLCD+ calculates CRC and compares it to the received CRC.

If both CRCs are the same, the ezLCD+ starts executing the received data, and responds with:



In case of the CRCs mismatch, the ezLCD+ erases the received data and responds with:



The ezLCD+ response is sent through the same interface, which received the CRC\_PKG command.

**Note:** The 'C' source code example of calculating CCITT standard CRC-16 is shown on the next page.

```

// CCITT standard CRC-16
// =====
// 16 bit, non-reflected CRC using:
// polynomial: 0x1021
// seed: 0xFFFF
// final xor: 0
// Endian: independent
#define CRC16_SEED      0xFFFF
#define CRC16_FINAL_XOR 0x0000
#define CRC16_START    CRC16_SEED
#define CRC16_NEXT(crc, next_byte) (((crc) << 8) ^ Crcl6Table[((crc) >> 8) ^ (next_byte)])
#define CRC16_END(crc) ((crc) ^ CRC16_FINAL_XOR)
const unsigned short Crcl6Table[256] =
{
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcdbc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdcc, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

unsigned short Crcl6(const void *data, unsigned int length)
{
    unsigned short crc;
    const unsigned char *bptr;
    bptr = (const unsigned char *) data;
    crc = CRC16_START;
    while (length--)
    {
        crc = CRC16_NEXT(crc, *bptr++);
    }
    return CRC16_END(crc);
}

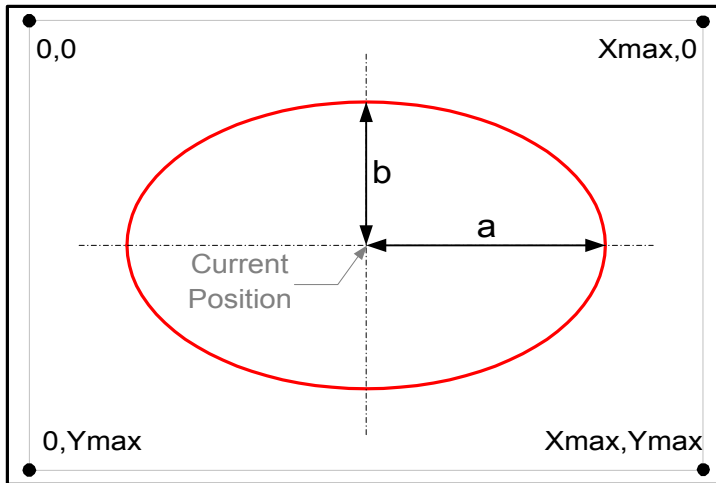
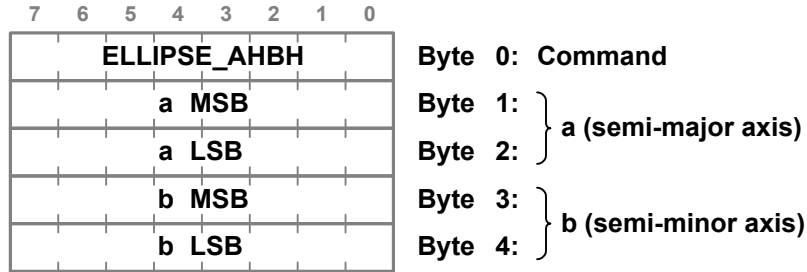
```

## 8.18 ELLIPSE\_AHBH

**Description:** Draws an ellipse with a Pen in Current Color centered at Current Position.

**Parameters:** a - length of semi-major axis  
b - length of semi-minor axis

**Code:** **8A**hex, **138**dec



See Also: [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#), [CIRCLE\\_RH](#)

### Example:

The following sequence will draw a red ellipse with the center positioned at (160, 117).

```

SET_COLOR_RGB 31 hex
Red          FF hex
Green       0
Blue        0 hex
SET_XHYH    33 hex
0           0 dec (x MSB)
160        160 dec (x LSB)
0           0 dec (y MSB)
117       117 dec (y LSB)
ELLIPSE_AHBH 8A hex
0           0 dec (semi-major axis MSB)
80         80 dec (semi-major axis LSB)
0           0 dec (semi-minor axis MSB)
50         50 dec (semi-minor axis LSB)

```

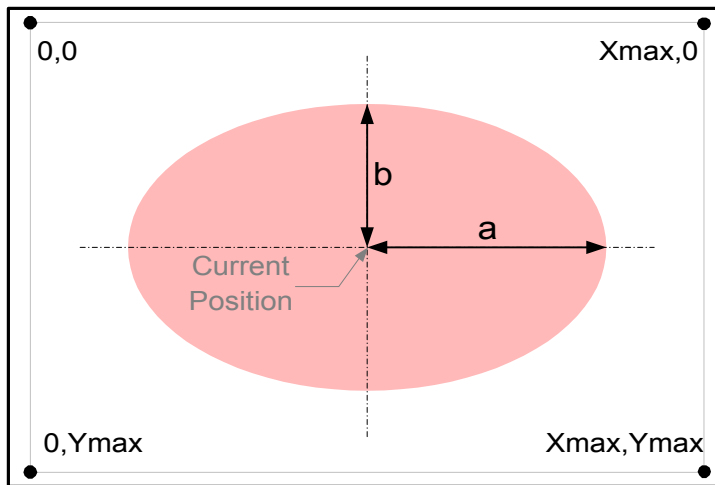
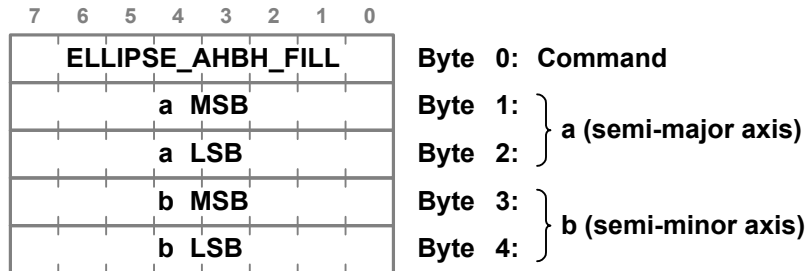


## 8.19 ELLIPSE\_AHBH\_FILL

**Description:** Draws an ellipse filled with Current Color and centered at Current Position.

**Parameters:** a - length of semi-major axis  
b - length of semi-minor axis

**Code:** **8B**hex, **139**dec



See Also: [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#), [CIRCLE\\_RH\\_FILL](#)

### Example:

The following sequence will draw an ellipse filled with the red color and centered at (160, 117).

```

SET_COLOR_RGB      31 hex
Red                FF hex
Green              0
Blue               0 hex
SET_XHYH           33 hex
0                  0 dec (x MSB)
160                160 dec (x LSB)
0                  0 dec (y MSB)
117                117 dec (y LSB)
ELLIPSE_AHBH_FILL 8B hex
0                  0 dec (semi-major axis MSB)
80                 80 dec (semi-major axis LSB)
0                  0 dec (semi-minor axis MSB)
50                 80 dec (semi-minor axis LSB)

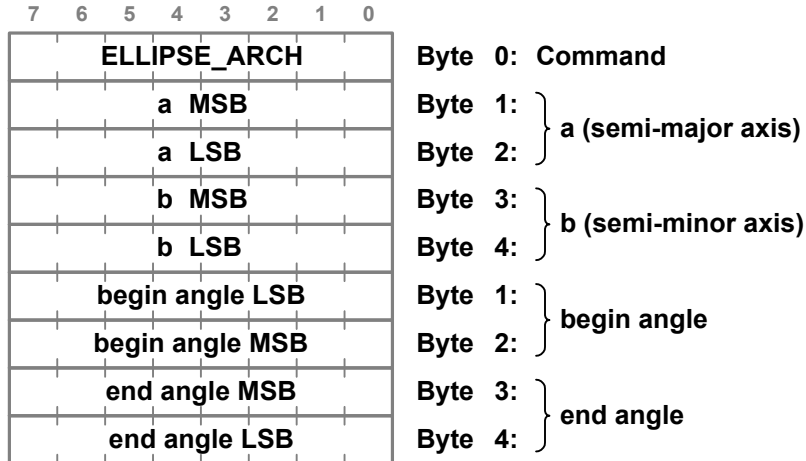
```



## 8.20 ELLIPSE\_ARCH

**Description:** Draws an ellipse arc with a [Pen](#) in Current Color centered at Current Position starting from Begin Angle and ending on End Angle.

**Code:** **8C**hex, **140**dec



**See Also:** [ARCH](#), [ELLIPSE\\_PIEH](#)

**Angle Coding:** The full angle ( $360^\circ$ ) is equal to **4000hex** (**16384dec**).

To transform degrees to ARC angle units:

**Angle\_lcd = Angle\_deg x 2048 / 45**

For example:

**2048dec = 800hex =  $45^\circ$**

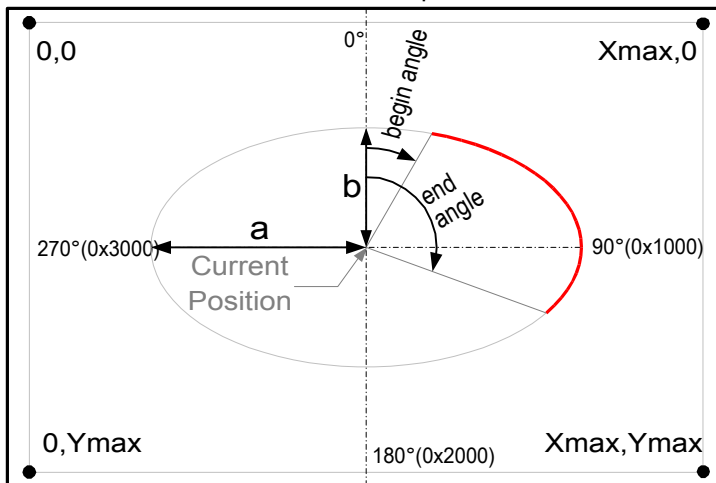
**4096dec = 1000hex =  $90^\circ$**

**8192dec = 2000hex =  $180^\circ$**

**12288dec = 3000hex =  $270^\circ$**

**16384dec = 4000hex =  $360^\circ = 0^\circ$**

The angle is oriented clockwise with the zero positioned at the top of the screen, as it is shown on the picture below



**Example:**

The following sequence will draw an ellipse arc from 22.5 to 225 degrees and a = 80, b = 40.

$22.5 \times 2048 / 45 = 1024$  (400hex)

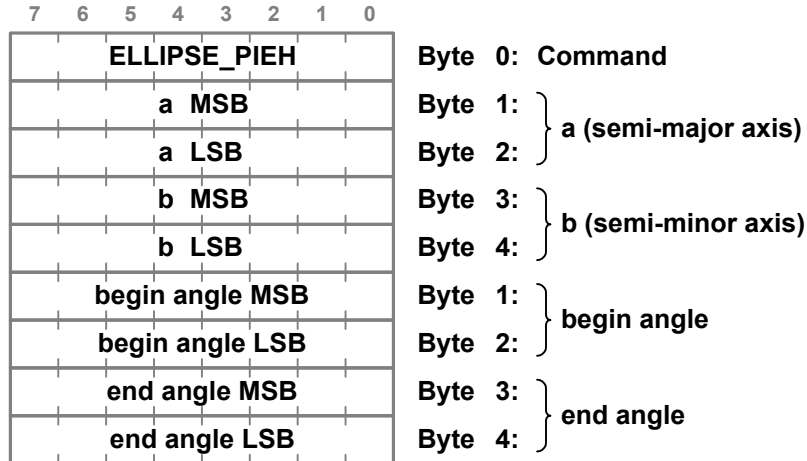
$225 \times 2048 / 45 = 10240$  (2800hex)

```
ELLIPSE_ARCH  8C hex
  0            0 dec (a MSB)
 80           80 dec (a LSB)
  0            0 dec (b MSB)
 40           40 dec (b LSB)
 04           04 hex (begin_angle MSB)
 00           00 hex (begin_angle LSB)
 28           28 hex (end_angle MSB)
 00           00 hex (end_angle LSB)
```

## 8.21 ELLIPSE\_PIEH

**Description:** Draws an ellipse pie filled with Current Color centered at Current Position starting from Begin Angle and ending on End Angle.

**Code:** 8Ehex, 142dec



See Also: [PIEH](#), [ELLIPSE\\_ARCH](#)

**Angle Coding:** The full angle (360°) is equal to 4000hex (16384dec).

To transform degrees to ARC angle units:

$\text{Angle\_lcd} = \text{Angle\_deg} \times 2048 / 45$

For example:

2048dec = 800hex = 45°

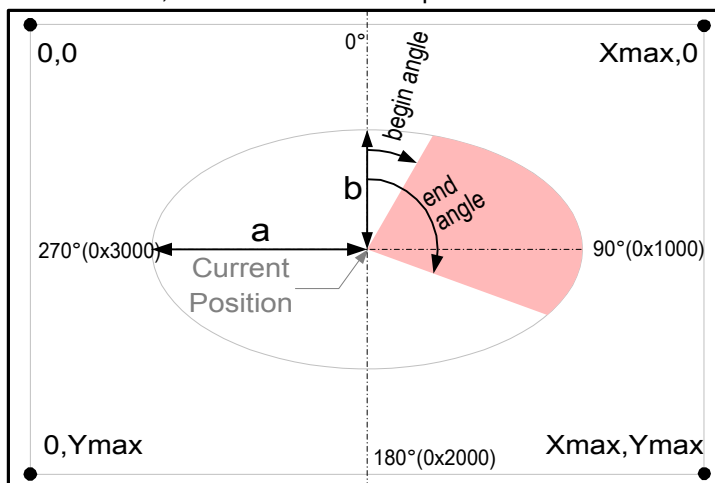
4096dec = 1000hex = 90°

8192dec = 2000hex = 180°

12288dec = 3000hex = 270°

16384dec = 4000hex = 360° = 0°

The angle is oriented clockwise with the zero positioned at the top of the screen, as it is shown on the picture below



**Example:**

The following sequence will draw an ellipse pie from 22.5 to 225 degrees and a = 80, b = 40.

$22.5 \times 2048 / 45 = 1024$  (400hex)

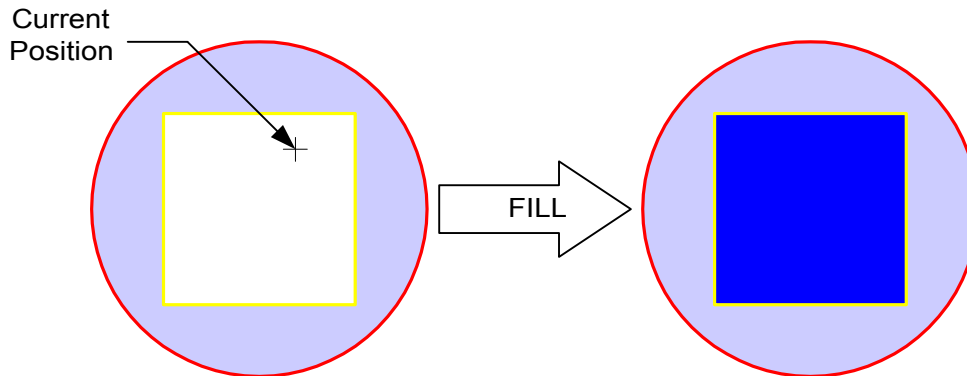
$225 \times 2048 / 45 = 10240$  (2800hex)

```
ELLIPSE_PIEH  8E hex
  0           0 dec (a MSB)
 80          80 dec (a LSB)
  0           0 dec (b MSB)
 40          40 dec (b LSB)
 04          04 hex (begin_angle MSB)
 00          00 hex (begin_angle LSB)
 28          28 hex (end_angle MSB)
 00          00 hex (end_angle LSB)
```

## 8.22 FILL

**Description:** Fills an area with the Current Color. Filling seed is located at the Current Position. The fill area is restricted by the connected pixels, which color is different than the seed pixel.

**Code:** **9B**hex, **155**dec



**Notes:** Please, refer to the [FILL\\_BOUND](#) command, which uses the different way to define the fill area. FILL is not affected by the transparency [Alpha](#).

**See Also:** [FILL\\_BOUND](#), [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#)

### Example:

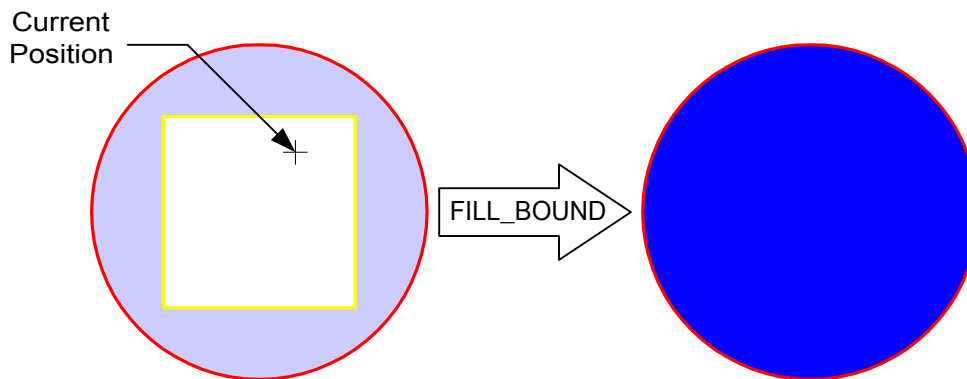
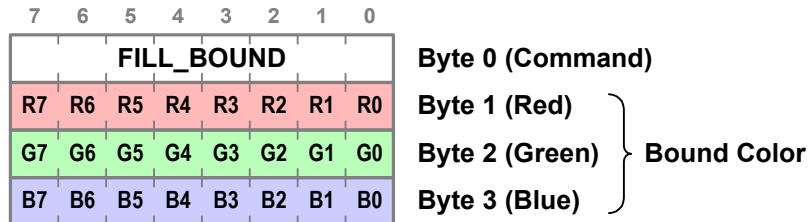
The following sequence will clear the area surrounding point at (160, 117) with blue (ref: drawing above).

```
SET_COLOR_RGB 31 hex
  Red          0
  Green        0
  Blue        FF hex
SET_XHYH      33 hex
  0            0 dec (x MSB)
  160         160 dec (x LSB)
  0           0 dec (y MSB)
  117        117 dec (y LSB)
FILL       9B hex
```

## 8.23 FILL\_BOUND

**Description:** Fills an area with the Current Color. Filling seed is located at the Current Position. The fill area is restricted by the connected pixels, which color is specified by the Bound Color parameter.

**Code:** **9C**hex, **156**dec



**Notes:** Please, refer to the [FILL](#) command, which uses the different way to define the fill area. FILL\_BOUND is not affected by the transparency [Alpha](#).

**See Also:** [FILL](#), [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#)

### Example:

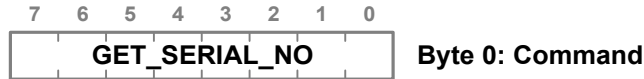
The following sequence will clear the area surrounding point at (160, 117) with blue, using red as a Bound Color (ref: drawing above).

```

SET_COLOR_RGB 31 hex
Red           0
Green        0
Blue         FF hex
SET_XHYH     33 hex
0             0 dec (x MSB)
160          160 dec (x LSB)
0            0 dec (y MSB)
117         117 dec (y LSB)
FILL_BOUND   9B hex
Red          FF hex
Green        0
Blue         0
  
```

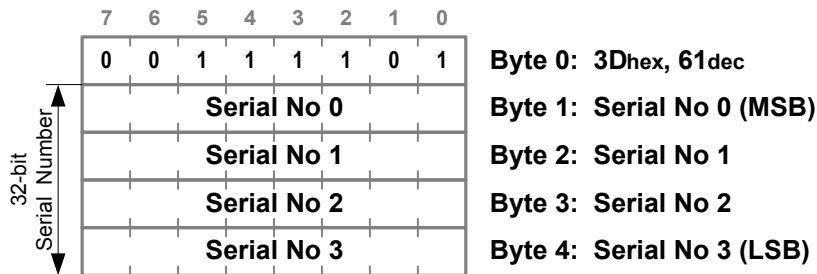
## 8.24 GET\_SERIAL\_NO

**Description:** Gets the value of the User Device Serial Number (Configuration Key: SerialNo).  
**Code:** 9D<sub>hex</sub>, 157<sub>dec</sub>



### ezLCD+ Response

After receiving the GET\_SERIAL\_NO command, the ezLCD+ responds with:



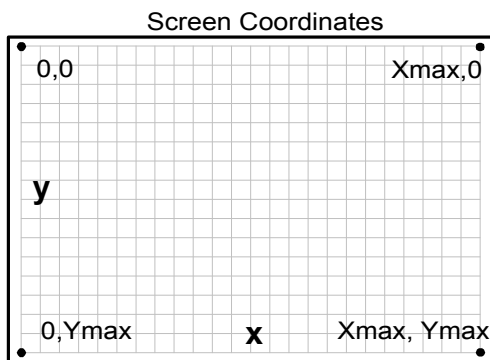
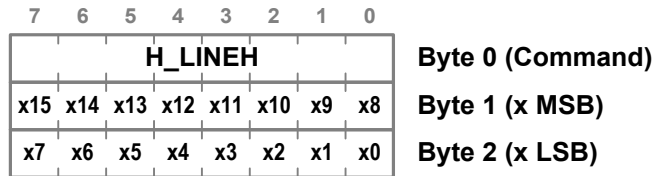
The ezLCD+ response is sent through the same interface, which received the GET\_SERIAL\_NO command.

Configuration Keys are described in "ezLCD+10xManual".

## 8.25 H\_LINEH

**Description:** Quickly draws a horizontal line from the Current Position to the column specified by the parameter.

**Code:** **A0**hex, **160**dec



See Also: [V\\_LINE](#), [SET\\_XHY](#)

### Example:

The following sequence will draw a green horizontal line from (20, 60) to (170, 60).

```

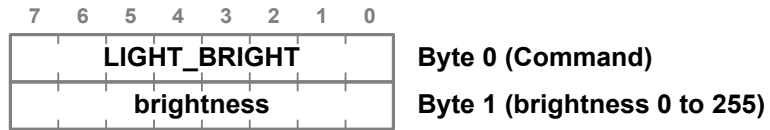
SET_COLOR_RGB 31 hex
Red           0
Green        FF
Blue         0 hex
SET_XHYH     33 hex
0             0 dec (x MSB)
20           20 dec (x LSB)
0             0 dec (y MSB)
60           60 dec (y LSB)
H_LINEH      A0 hex
0             0 dec (x MSB)
170          170 dec (x LSB)

```

## 8.26 LIGHT\_BRIGHT

**Description:** Sets the brightness of the screen backlight.

**Code:** 80hex, 128dec



**Note:** The default brightness is 255

See Also: [LIGHT\\_ON](#), [LIGHT\\_OFF](#)

### Example:

The following sequence will set the backlight to 25% of its full brightness.

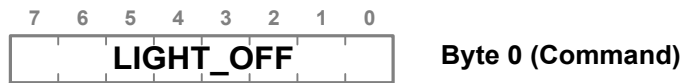
LIGHT\_BRIGHT 80 hex  
64

64 dec

## 8.27 LIGHT\_OFF

**Description:** Turns off the screen backlight.

**Code:** 23<sub>hex</sub>, 35<sub>dec</sub>



See Also: [LIGHT\\_ON](#), [LIGHT\\_BRIGHT](#)

### Example:

The following sequence will turn off the screen backlight.

```
LIGHT_OFF 23 hex
```

## 8.28 LIGHT\_ON

**Description:** Turns on the screen backlight.

**Code:** 22<sub>hex</sub>, 34<sub>dec</sub>



See Also: [LIGHT\\_OFF](#), [LIGHT\\_BRIGHT](#)

### Example:

The following sequence will turn on the screen backlight.

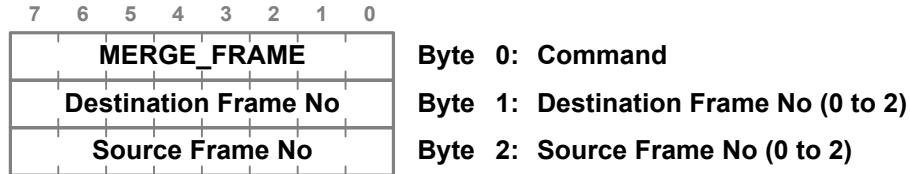
LIGHT\_ON 22 hex



### 8.30 MERGE\_FRAME

**Description:** Merges the contents of the source [Frame](#) with the destination Frame according to the value of transparency [Alpha](#). The resulting image is held in the destination Frame.

**Code:** 54hex, 84dec



When [Alpha](#) = 255, both commands (COPY\_FRAME and MERGE\_FRAME) yield identical results.

**Note:** Merging a frame with itself may yield unpredictable results.

**Ref:** [Frames](#)

**See Also:** [COPY\\_FRAME](#), [COPY\\_RECT](#), [MERGE\\_RECT](#), [SET\\_DISP\\_FRAME](#), [SET\\_DRAW\\_FRAME](#)

#### Example:

The following sequence will merge the contents of Frame 1 with Frame 0

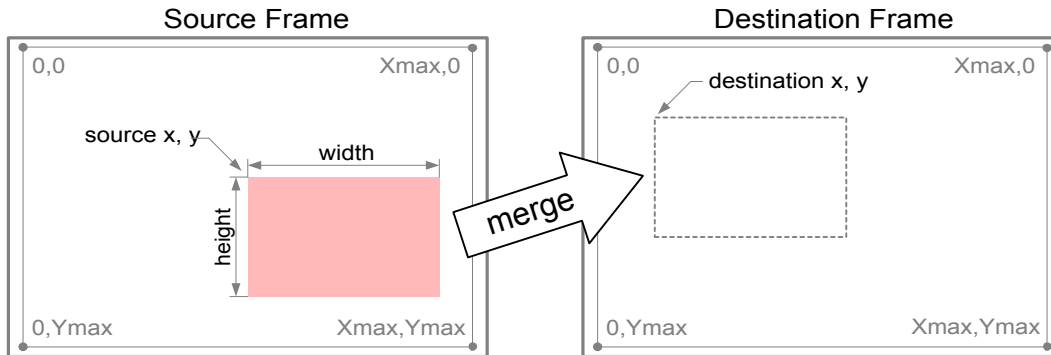
```
COPY_FRAME 53 hex
  0          0 (Destination Frame No)
  1          1 (Source Frame No)
```

### 8.31 MERGE\_RECT

**Description:** Merges the rectangular portion of one frame with the other according to the value of transparency [Alpha](#).

**Code:** 56hex, 86dec

7 6 5 4 3 2 1 0	<b>MERGE_RECT</b>	
Destination Frame No		<b>Byte 0: Command</b>
Source Frame No		<b>Byte 1: Destination Frame No (0 to 2)</b>
x15 x14 x13 x12 x11 x10 x9 x8		<b>Byte 2: Source Frame No (0 to 2)</b>
x7 x6 x5 x4 x3 x2 x1 x0		<b>Byte 3: } Destination Rectangle upper-left corner X-coordinate</b>
y15 y14 y13 y12 y11 y10 y9 y8		
y7 y6 y5 y4 y3 y2 y1 y0		<b>Byte 4: } Destination Rectangle upper-left corner Y-coordinate</b>
x15 x14 x13 x12 x11 x10 x9 x8		
x7 x6 x5 x4 x3 x2 x1 x0		<b>Byte 5: } Source Rectangle upper-left corner X-coordinate</b>
y15 y14 y13 y12 y11 y10 y9 y8		
y7 y6 y5 y4 y3 y2 y1 y0		<b>Byte 6: } Source Rectangle upper-left corner Y-coordinate</b>
Width MSB		
Width LSB		<b>Byte 8: } Rectangle Width</b>
Height MSB		<b>Byte 9: } Rectangle Height</b>
Height LSB		
		<b>Byte 11: } Rectangle Height</b>
		<b>Byte 12: } Rectangle Height</b>
		<b>Byte 13: } Rectangle Height</b>
		<b>Byte 14: } Rectangle Height</b>



When [Alpha](#) = 255, both commands (COPY\_RECT and MERGE\_RECT) yield identical results.

**Note:** Merging a rectangle with itself may yield unpredictable results.

**Ref:** [Frames](#)

**See Also:** [COPY\\_RECT](#), [COPY\\_FRAME](#), [MERGE\\_FRAME](#), [SET\\_DISP\\_FRAME](#), [SET\\_DRAW\\_FRAME](#)

**Example:**

The following sequence will merge the rectangle size 100x50 from Frame 1 located at (320, 240) with rectangle in Frame 0 located at (40, 60)

```
MERGE_RECT    56 hex
  0            0 (Destination Frame No)
  1            1 (Source Frame No)
  0            0 dec (Destination X MSB)
 40            40 dec (Destination X LSB)
  0            0 dec (Destination Y MSB)
 60            60 dec (Destination Y LSB)
  1            1 dec (Source X MSB)
 64            64 dec (Source X LSB)
  0            0 dec (Source Y MSB)
240            240 dec (Source Y LSB)
  0            0 dec (Width MSB)
100            100 dec (Width LSB)
  0            0 dec (Height MSB)
 50            50 dec (Height LSB)
```

## 8.32 PIEH

**Description:** Draws a pie filled with Current Color with the center at Current Position, starting on Begin Angle and ending on End Angle.

**Code:** 90<sub>hex</sub>, 144<sub>dec</sub>

7	6	5	4	3	2	1	0
<b>PIEH</b>							
radius_MSB							
radius_LSB							
begin_angle_MSB							
begin_angle_LSB							
end_angle_MSB							
end_angle_LSB							

<b>Byte 0 (Command)</b>
<b>Byte 1 (Radius MSB)</b>
<b>Byte 2 (Radius LSB)</b>
<b>Byte 3 (Pie Begin Angle MSB)</b>
<b>Byte 4 (Pie Begin Angle LSB)</b>
<b>Byte 5 (Pie End Angle MSB)</b>
<b>Byte 6 (Pie End Angle LSB)</b>

**Note:** Since this is a filled figure, [Pen](#) parameters are ignored.

**See Also:** [ARCH](#), [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#), [CIRCLE\\_RH](#), [ELLIPSE\\_PIEH](#)

**Angle Coding:** The full angle (360°) is equal to 4000<sub>hex</sub> (16384<sub>dec</sub>).

To transform degrees to ARC angle units:

$$\text{Angle\_lcd} = \text{Angle\_deg} \times 2048 / 45$$

For example:

$$2048_{\text{dec}} = 800_{\text{hex}} = 45^\circ$$

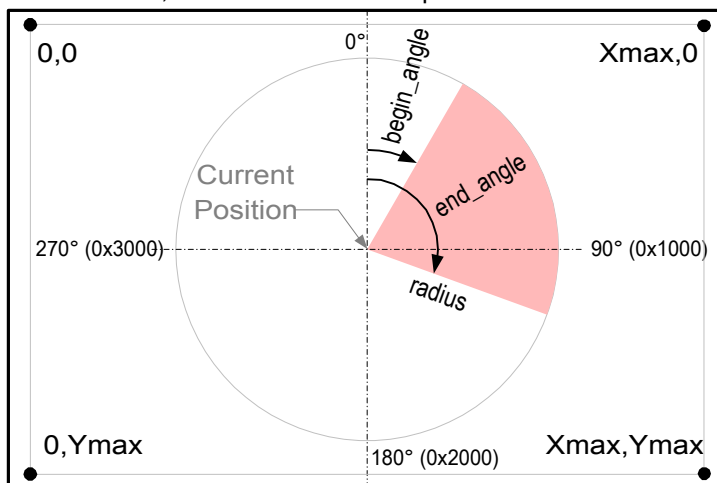
$$4096_{\text{dec}} = 1000_{\text{hex}} = 90^\circ$$

$$8192_{\text{dec}} = 2000_{\text{hex}} = 180^\circ$$

$$12288_{\text{dec}} = 3000_{\text{hex}} = 270^\circ$$

$$16384_{\text{dec}} = 4000_{\text{hex}} = 360^\circ = 0^\circ$$

The angle is oriented clockwise with the zero positioned at the top of the screen, as it is shown on the picture below



**Example:**

The following sequence will draw a green pie from 45 to 225 degrees with the center positioned at (160, 117) and a radius of 80.

$225 \times 2048 / 45 = 10240$  (2800hex).

```
SET_COLOR_RGB 31 hex
Red           0
Green        FF
Blue         0 hex
SET_XHYH     33 hex
0             0 dec (x MSB)
160          160 dec (x LSB)
0            0 dec (y MSB)
117         117 dec (y LSB)
PIEH       90 hex
0            0 dec (radius MSB)
80           80 dec (radius LSB)
08           08 hex (begin_angle MSB)
00           00 hex (begin_angle LSB)
28           28 hex (end_angle MSB)
00           00 hex (end_angle LSB)
```

### 8.33 PING

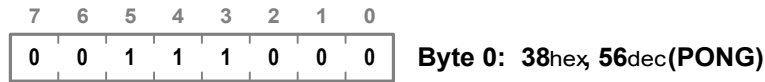
**Description:** Checks if the ezLCD+ is connected and ready to receive commands.

**Code:** 83<sub>hex</sub>, 131<sub>dec</sub>



#### ezLCD+ Response

After receiving the PING command, the ezLCD+ responds with the PONG (38<sub>hex</sub>, 56<sub>dec</sub>) byte:



The ezLCD+ response is sent through the same interface, which received the PING command.

#### Example:

The following sequence will check if the ezLCD+ is OK:

PING 83 hex

If the ezLCD+ is connected and ready to receive commands, it responds with:

38 hex

## 8.34 PLOT

**Description:** Plots a point at Current Position in Current Color.

**Code:** **26**hex, **38**dec



**Note:** This command draws a pixel-size point. It is not affected by the [Pen](#) parameters.

**See Also:** [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#), [PLOT\\_XHYH](#)

### Example:

The following sequence will put a blue point at (160, 117).

```
SET_COLOR_RGB 31 hex
Red           0
Green        0
Blue         FF hex
SET_XHYH     33 hex
0             0 dec (x MSB)
160          160 dec (x LSB)
0            0 dec (y MSB)
117          117 dec (y LSB)
PLOT         26 hex
```

## 8.35 PLOT\_XHYH

**Description:** Plots a point in Current Color at the specified position.

**Code:** 3Ehex, 62dec

	7	6	5	4	3	2	1	0	
	<b>PLOT_XHYH</b>								
	x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 0 (Command)</b>
	x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 1 (x MSB)</b>
	y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 2 (x LSB)</b>
	y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 3 (y MSB)</b>
									<b>Byte 4 (y LSB)</b>

**Note:** This command draws a pixel-size point. It is not affected by the [Pen](#) parameters.

**Ref:** [Screen Coordinates](#)

**See Also:** [SET\\_COLOR\\_RGB](#), [PLOT](#)

### Example:

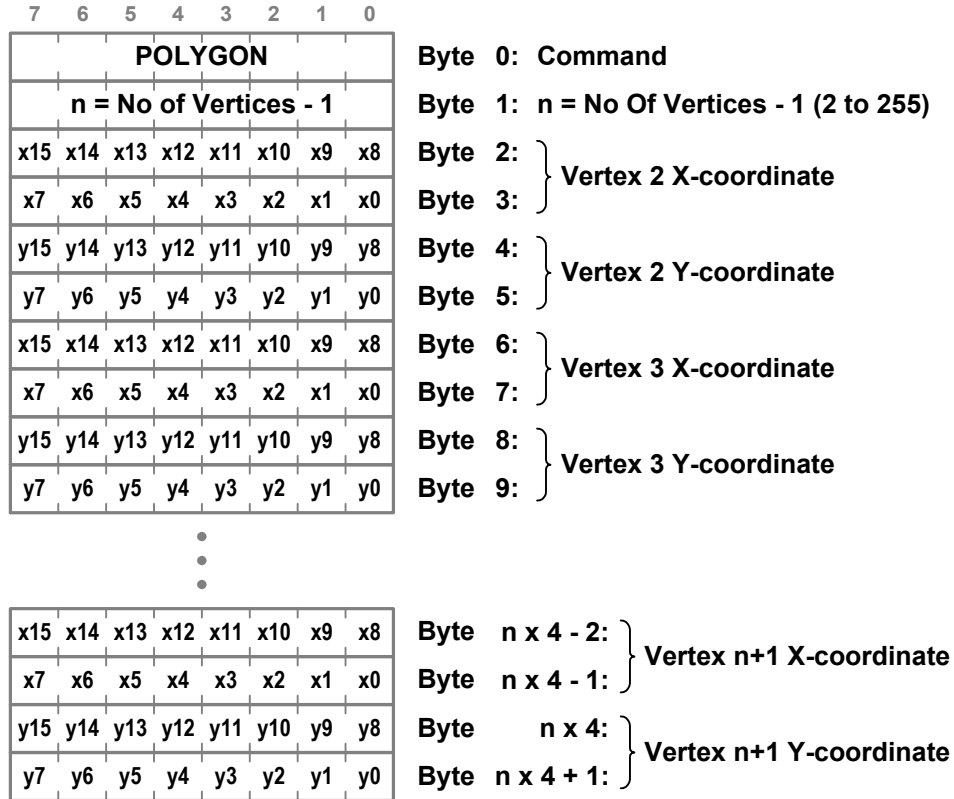
The following sequence will put a red point at (310, 117).

```
SET_COLOR_RGB 31 hex
Red          FF hex
Green       0
Blue        0
PLOT_XHYH   3E hex
  1          1 dec (x MSB)
 60         160 dec (x LSB 1*256+54=310)
  0          0 dec (y MSB)
117        117 dec (y LSB)
```

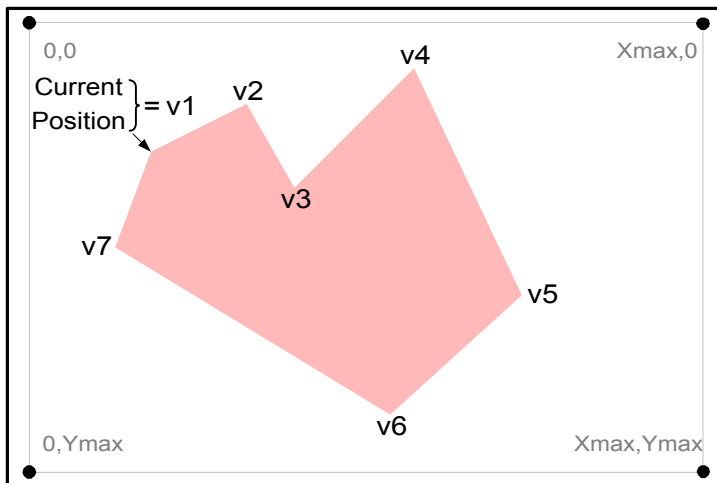
### 8.36 POLYGON

**Description:** Draws a polygon filled with the Current Color. The first vertex is located at the Current Position.

**Code:** **A6**hex, **166**dec



There is no need to specify the first vertex, because it is located at the Current Position. The Byte: 1 of the POLYGON command specifies the number of vertices, which are defined in the following bytes, starting with the second vertex.



**See Also:** [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#)

**Example:**

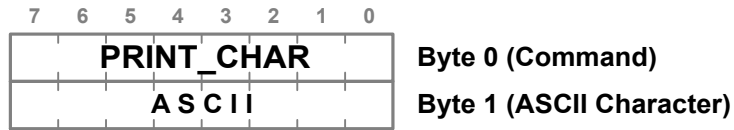
The following sequence will draw a pink triangle.

```
SET_COLOR_RGB 31 hex
Red           FF hex
Green        80 hex
Blue         80 hex
SET_XHYH     33 hex
0            0 dec (x MSB)
100         100 dec (x LSB)
0           0 dec (y MSB)
50          50 dec (y LSB)
POLYGON      A6 hex
2           2 (3 vertices - 1 = 2)
0           0 dec (second vertex x MSB)
150        150 dec (second vertex x LSB)
0           0 dec (second vertex y MSB)
150        150 dec (second vertex y LSB)
0           0 dec (third vertex x MSB)
50          50 dec (third vertex x LSB)
0           0 dec (third vertex y MSB)
150        150 dec (third vertex y LSB)
```

## 8.37 PRINT\_CHAR

**Description:** Prints an ASCII-coded character at the Current Position.

**Code:** 2Chex, 44dec



**Note:** This command can also print the Unicode characters. Please, refer to commands: [SET\\_FT\\_UNIBASE](#) and [SET\\_FT\\_FONT](#)

**See Also:** [SELECT\\_FONT](#), [SET\\_FT\\_FONT](#), [SET\\_FT\\_UNIBASE](#), [PRINT\\_STRING](#)

### Example:

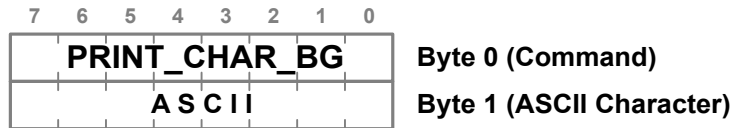
The following sequence will print character 'M'.

```
PRINT_CHAR    2C hex
'M'           4D hex
```

### 8.38 PRINT\_CHAR\_BG

**Description:** Prints a bitmap character at Current Position on the background specified by the [SET\\_BG\\_COLORH](#) command.

**Code:** **3C**hex, **60**dec



**Note:** This command cannot print the [True Type Fonts](#).

**See Also:** [SELECT\\_FONT](#), [SET\\_BG\\_COLORH](#), [PRINT\\_STRING\\_BG](#)

#### Example:

The following sequence will print the character 'M' in white on a black background using Font 2.

```

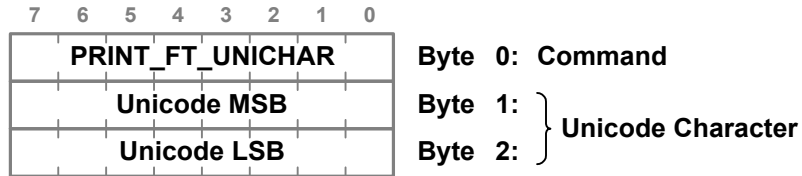
SELECT_FONT      2B hex
2                2 dec
SET_BG_COLOR_RGB 32 hex
Red              0
Green            0
Blue             0
SET_COLOR_RGB   31 hex
Red              FF hex
Green            FF hex
Blue             FF hex
PRINT_CHAR_BG   3C hex
'M'              4D hex

```

### 8.39 PRINT\_FT\_UNICHAR

**Description:** Prints a 16-bit Unicode-coded [True Type](#) character at the Current Position.

**Code:** 97hex, 151dec



**See Also:** [SET\\_FT\\_FONT](#), [PRINT\\_FT\\_UNISTRING](#)

**Note:** This command will not print any characters unless the True Type Font is selected using [SET\\_FT\\_FONT](#) or [SD\\_LOAD\\_FONT](#).

#### Example:

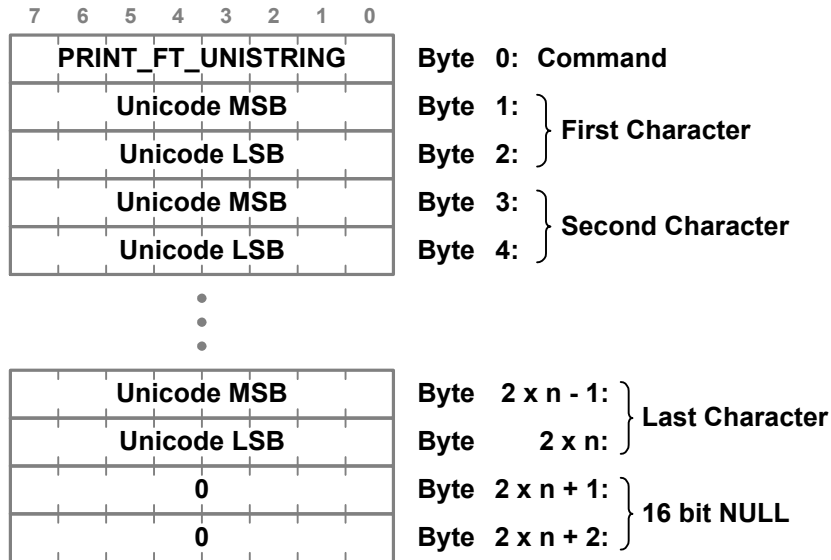
The following sequence will print character 'M'.

```
PRINT_UNICHAR 97 hex
0             0 hex (Unicode 'M' MSB)
'M'          4D hex (Unicode 'M' LSB)
```

## 8.40 PRINT\_FT\_UNISTRING

**Description:** Prints a null-terminated 16-bit Unicode-coded [True Type](#) string starting at the Current Position.

**Code:** 98hex, 152dec



**Note:** This command will not print any characters unless the True Type Font is selected using [SET\\_FT\\_FONT](#) or [SD\\_LOAD\\_FONT](#).

**See Also:** [SET\\_FT\\_FONT](#), [PRINT\\_FT\\_UNICHAR](#)

### Example:

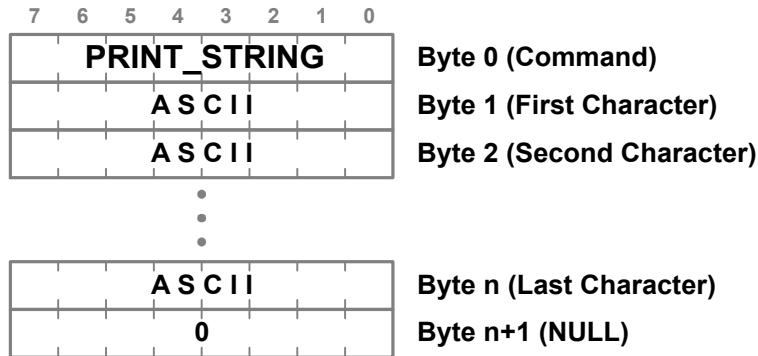
The following sequence will print "Дождь" (rain) in Russian.

```
PRINT_FT_UNISTRING 98 hex
04                04 hex (' ' MSB)
14                14 hex (' ' LSB)
04                04 hex ('>' MSB)
3E                3E hex ('>' LSB)
04                04 hex ('6' MSB)
36                36 hex ('6' LSB)
04                04 hex ('4' MSB)
34                34 hex ('4' LSB)
04                04 hex ('L' MSB)
4C                4C hex ('L' LSB)
NULL              0 (NULL)
NULL              0 (NULL)
```

## 8.41 PRINT\_STRING

**Description:** Prints a null-terminated ASCII-coded string starting at the Current Position.

**Code:** **2D**hex, **45**dec



**Note:** This command can also print the Unicode characters. Please, refer to commands: [SET\\_FT\\_UNIBASE](#) and [SET\\_FT\\_FONT](#)

**See Also:** [SELECT\\_FONT](#), [SET\\_FT\\_FONT](#), [SET\\_FT\\_UNIBASE](#), [PRINT\\_CHAR](#)

### Example:

The following sequence will print "LCD" in purple.

```

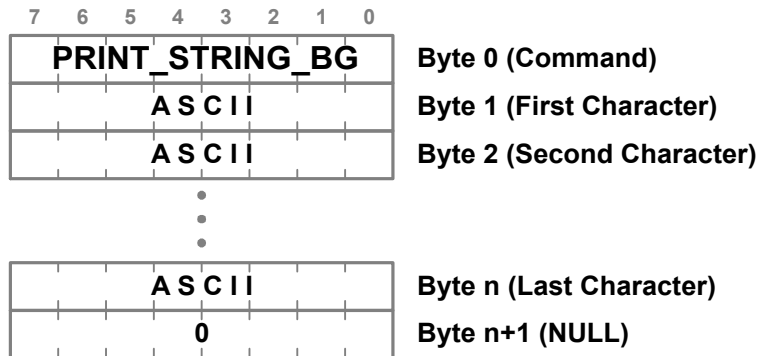
SET_COLOR_RGB 31 hex
Red           80 hex
Green        0
Blue         80 hex
PRINT_STRING 2D hex
'L'          4C hex
'C'          43 hex
'D'          44 hex
NULL         0 hex

```

## 8.42 PRINT\_STRING\_BG

**Description:** Prints null-terminated string of bitmap characters starting at Current Position on the background specified by [SET\\_BG\\_COLORH](#) command.

**Code:** **3D**hex, **61**dec



**Note:** This command cannot print the [True Type Fonts](#).

**See Also:** [SELECT\\_FONT](#), [SET\\_BG\\_COLORH](#), [PRINT\\_CHAR\\_BG](#)

### Example:

The following sequence will print "LCD" in yellow on a navy background, using Bitmap Font 0.

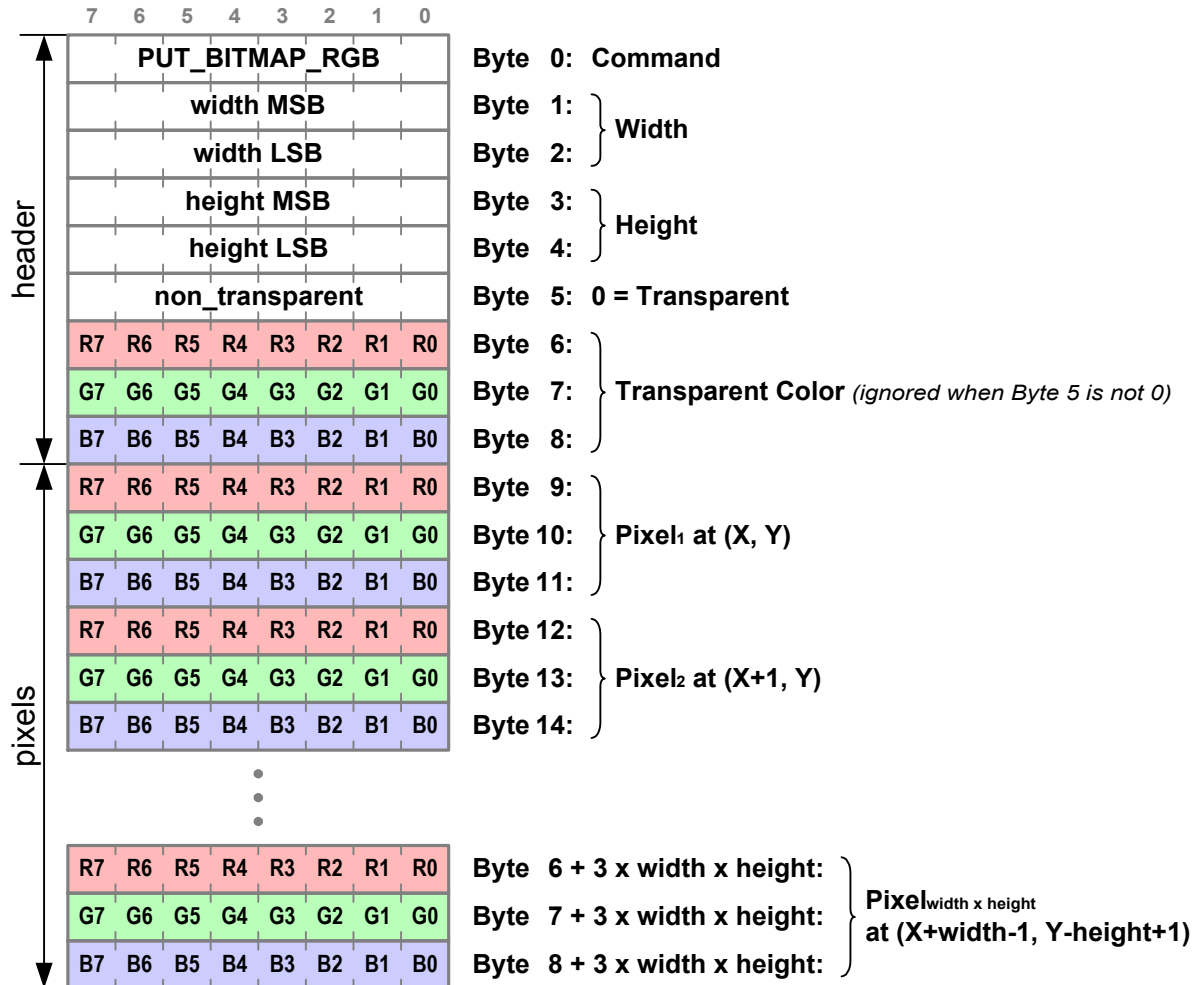
```

SET_BG_COLOR_RGB 32 hex
Red              0
Green            0
Blue            80 hex
SET_COLOR_RGB   31 hex
Red             FF hex
Green           FF hex
Blue            0
SELECT_FONT     2B hex
0               0 dec
PRINT_STRING_BG 3D hex
'L'            4C hex
'C'            43 hex
'D'            44 hex
NULL           0 hex
  
```

## 8.43 PUT\_BITMAP\_RGB

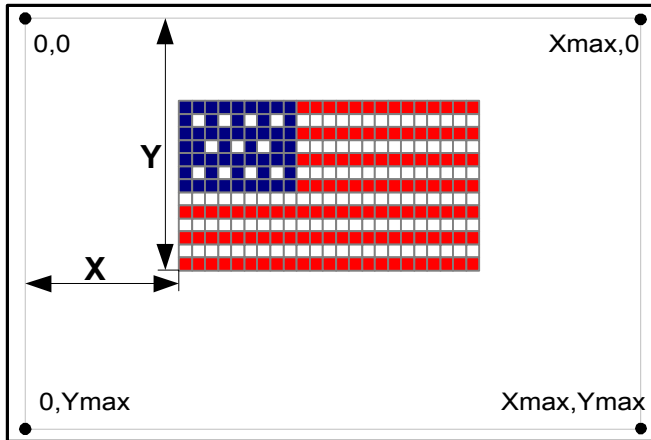
**Description:** Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP

**Code:** 9F<sub>hex</sub>, 159<sub>dec</sub>



**Notes:** 1. The total number of bytes is:  $3 \times \text{width} \times \text{height} + 9$

2. When Byte 5 = 0, Bytes 6, 7 and 8 specify local transparent color. Pixels equal to local transparent color are ignored during bitmap drawing. The drawing is not affected by the global Transparent Color set by command `SET_TR_COLOR_RGB`. All pixels are drawn when Byte 5 is not 0.



Example of the order of the pixels in case of the 4x3 bitmap.

9	10	11	12
5	6	7	8
1	2	3	4

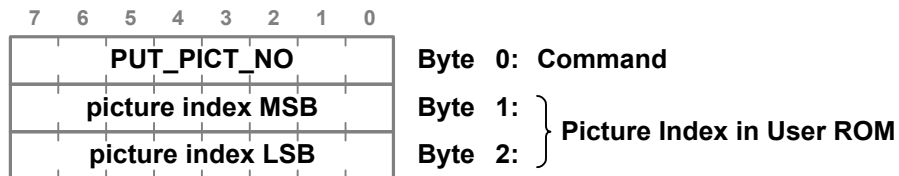
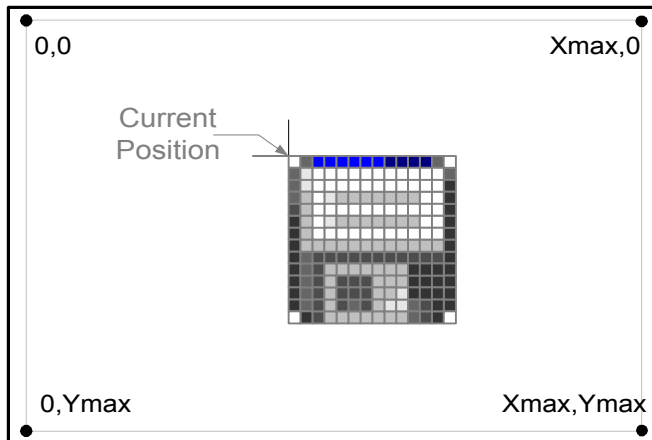
Pixel 1

See Also: [SET\\_XHYH](#), [SET\\_COLOR\\_RGB](#)

## 8.44 PUT\_PICT\_NO

**Description:** Displays a bitmap (icon) with its upper-left corner positioned at the Current Position. The bitmap is read from the User ROM.

**Code:** 59hex, 89dec



See Also: [SD\\_PUT\\_ICON](#)

### Example:

The following sequence will display Icon No. 3 with its upper-left corner positioned at (60, 43).

```

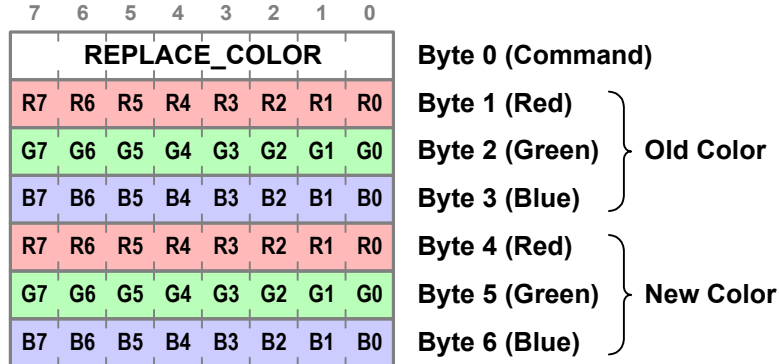
SET_XHYH    33 hex
  0          0 dec (x MSB)
 60         60 dec (x LSB)
  0          0 dec (y MSB)
 43         43 dec (y LSB)
PUT_PICT_NO 59 hex
  0          0 dec (index MSB)
  3          3 dec (index LSB)

```

## 8.45 REPLACE\_COLOR

**Description:** Replaces color in the Edit Rectangle set by [SET\\_EDIT\\_RECT](#) command.

**Code:** **5D**hex, **93**dec



The "Old Color" is replaced by the "New Color".

**See Also:** [SET\\_EDIT\\_RECT](#)

### Example:

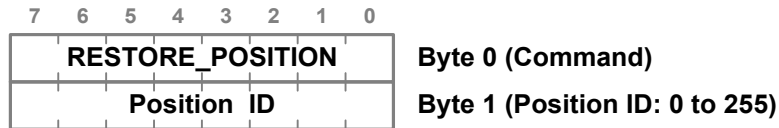
The following sequence will replace color red with green inside the rectangle size of 100x50 and positioned at (320, 240).

```
SET_EDIT_RECT 5C hex
  1          1 dec (X MSB)
 64         64 dec (X LSB)
  0          0 dec (Y MSB)
240        240 dec (Y LSB)
  0          0 dec (Width MSB)
100        100 dec (Width LSB)
  0          0 dec (Height MSB)
 50         50 dec (Height LSB)
REPLACE_COLOR 5D hex
Red         FF hex (Old color red component)
Green       0      (Old color green component)
Blue        0      (Old color blue component)
Red         0      (New color red component)
Green       FF hex (New color green component)
Blue        0      (New color blue component)
```

## 8.46 RESTORE\_POSITION

**Description:** Restores the Current Position saved by the [SAVE\\_POSITION](#) command.

**Code:** 36hex, 54dec



See Also: [SAVE\\_POSITION](#), [SET\\_XHYH](#)

### Example:

The following sequence will draw 3 lines with the common starting point: (160, 117)

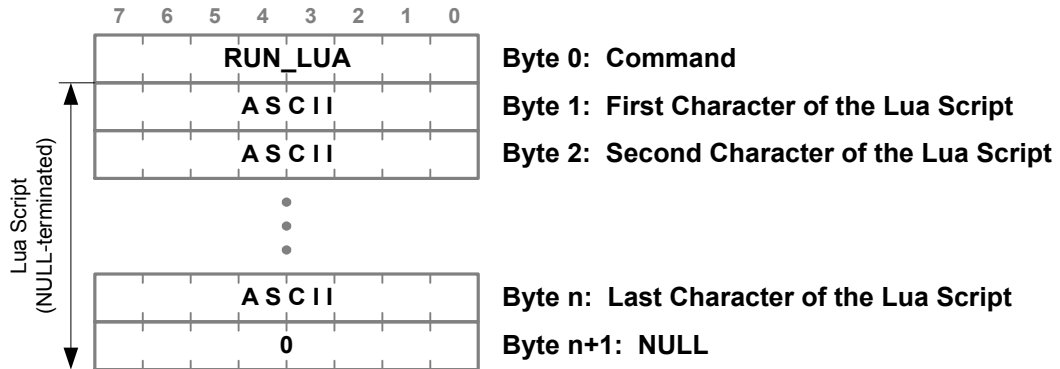
```

SET_XHYH      33 hex
  0            0 dec (x MSB)
 160          160 dec (x LSB)
  0            0 dec (y MSB)
 117          117 dec (y LSB)
SAVE_POSITION 35 hex
  12          12 dec (Position ID)
LINE_TO_XHYH  3F hex
  0            0 dec (x MSB)
 247          247 dec (x LSB)
  0            0 dec (y MSB)
 67           67 dec (y LSB)
RESTORE_POSITION 36 hex
 12           12 dec (Position ID)
LINE_TO_XHYH  3F hex
  0            0 dec (x MSB)
 73           73 dec (x LSB)
  0            0 dec (y MSB)
 67           67 dec (y LSB)
RESTORE_POSITION 36 hex
 12           12 dec (Position ID)
V_LINEH      A1 hex
  0            0 dec (y MSB)
 217         217 dec (y LSB)

```

## 8.47 RUN\_LUA

**Description:** Runs a Lua script.  
**Code:** **A7**hex, **167**dec



See Also: [RUN\\_LUA\\_ROM](#), [RUN\\_LUA\\_SD](#)

### ezLCD+ Response

After receiving the RUN\_LUA command, the ezLCD+ runs the Lua script. Upon successful execution of the string the ezLCD+ responds with character '1' (31hex, 49dec),

or

if the script contains syntax or run-time errors, the ezLCD+ responds with character '0' (30hex, 48dec) followed by a null-terminated error string.

#### Error string format:

empty string

or

[*txt1*]:*n*: *txt2*

Where:

*txt1* - script ident text (ignore)

*n* - script error line number

*txt2* - error text

For example:

```
[string "ezCmdLua"]:23: attempt to index global 'BigShip' (a nil value)
```

The ezLCD+ response is sent through the same interface, which received the RUN\_LUA command.

### Example:

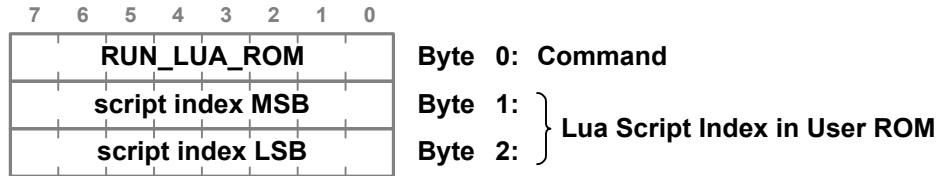
The following sequence will print "Hello World!" in purple at x = 100 and y = 50.

```
RUN_LUA  A7 hex
          ez.Cls(0xFFFFFFFF)
          ez.SetXY(100, 50)
          ez.SetColor(ez.RGB(255,0,255))
          print("Hello World!")
NULL     0 hex
```

## 8.48 RUN\_LUA\_ROM

**Description:** Runs a Lua script from the User ROM.

**Code:** **A8**hex, **168**dec



See Also: [RUN\\_LUA](#), [RUN\\_LUA\\_SD](#)

### ezLCD+ Response

After receiving the RUN\_LUA command, the ezLCD+ runs the Lua script. Upon successful execution of the string the ezLCD+ responds with character '1' (31hex, 49dec),

or

if the script contains syntax or run-time errors, the ezLCD+ responds with character '0' (30hex, 48dec) followed by a null-terminated error string.

#### Error string format:

empty string

or

[*txt1*]:*n*: *txt2*

Where:

*txt1* - script ident text (ignore)

*n* - script error line number

*txt2* - error text

For example:

```
[string "ezCmdLua"]:23: attempt to index global 'BigShip' (a nil value)
```

The ezLCD+ response is sent through the same interface, which received the RUN\_LUA\_ROM command.

### Example:

The following sequence will run Lua script no 2

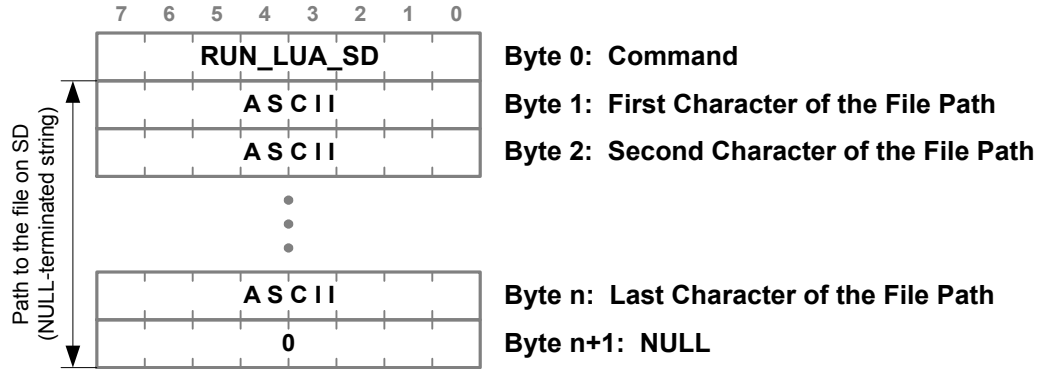
```
RUN_LUA_ROM A8 hex
0          0 dec (index MSB)
2          2 dec (index LSB)
```

## 8.49 RUN\_LUA\_SD

**Description:** Runs Lua script from the SD card attached to the SD/MMC interface.

**Supported file systems:** FAT12, FAT16, FAT32

**Code:** **A9**<sub>hex</sub>, **169**<sub>dec</sub>



**Note:** SD card has to be formatted in the supported file system.

**See Also:** [RUN\\_LUA](#), [RUN\\_LUA\\_ROM](#)

### About the File Path:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

### ezLCD+ Response

After receiving the RUN\_LUA command, the ezLCD+ runs the Lua script. Upon successful execution of the string the ezLCD+ responds with character '1' (31hex, 49dec),

or

if the script contains syntax or run-time errors, the ezLCD+ responds with character '0' (30hex, 48dec) followed by a null-terminated error string.

### Error string format:

empty string

or

[*txt1*]:*n*: *txt2*

Where:

*txt1* - script ident text (ignore)

*n* - script error line number

*txt2* - error text

For example:

[string "ezCmdLua"]:23: attempt to index global 'BigShip' (a nil value)

The ezLCD+ response is sent through the same interface, which received the RUN\_LUA\_SD command.

**Example:**

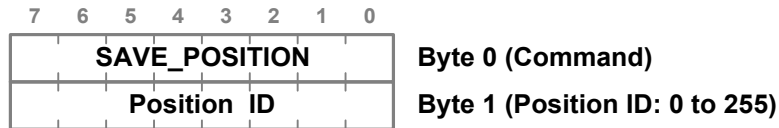
The following sequence will run Lua script from SD file: ezLCD.lua

RUN_LUA_SD	A9 hex
'e'	65 hex
'z'	7A hex
'L'	4C hex
'C'	43 hex
'D'	44 hex
'.'	2E hex
'l'	6C hex
'u'	75 hex
'a'	61 hex
NULL	0 hex

## 8.50 SAVE\_POSITION

**Description:** Stores the Current Position to the Position ID.  
The saved position may be later restored by the [RESTORE\\_POSITION](#) command.

**Code:** **35**hex, **53**dec



See Also: [RESTORE\\_POSITION](#), [SET\\_XHYH](#)

### Example:

The following sequence will draw 3 lines with the common starting point: (160, 117)

```

SET_XHYH      33 hex
  0            0 dec (x MSB)
 160          160 dec (x LSB)
  0            0 dec (y MSB)
 117          117 dec (y LSB)
SAVE_POSITION  35 hex
  12           12 dec (Position ID)
LINE_TO_XHYH  3F hex
  0            0 dec (x MSB)
 247          247 dec (x LSB)
  0            0 dec (y MSB)
 67           67 dec (y LSB)
RESTORE_POSITION 36 hex
 12           12 dec (Position ID)
LINE_TO_XHYH  3F hex
  0            0 dec (x MSB)
 73           73 dec (x LSB)
  0            0 dec (y MSB)
 67           67 dec (y LSB)
RESTORE_POSITION 36 hex
 12           12 dec (Position ID)
V_LINEH      A1 hex
  0            0 dec (y MSB)
 217          217 dec (y LSB)

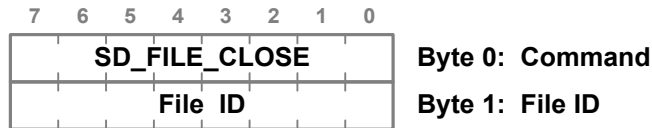
```

## 8.51 SD\_FILE\_CLOSE

**Description:** Closes SD Flash file. Re-enables the touch screen if no other SD files are opened.

[Supported file systems](#): FAT12, FAT16, FAT32

**Code:** 72<sub>hex</sub>, 114<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CREATE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

### Example:

The following sequence will close SD file 1.

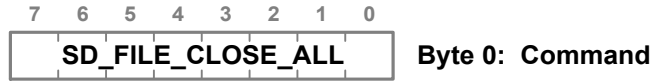
```
SD_FILE_CLOSE 72 hex
1 1 dec (File ID)
```

## 8.52 SD\_FILE\_CLOSE\_ALL

**Description:** Closes all opened SD Flash files and re-enables the touch screen.

[Supported file systems](#): FAT12, FAT16, FAT32

**Code:** 73<sub>hex</sub>, 115<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CREATE](#), [SD\\_FILE\\_CLOSE](#)

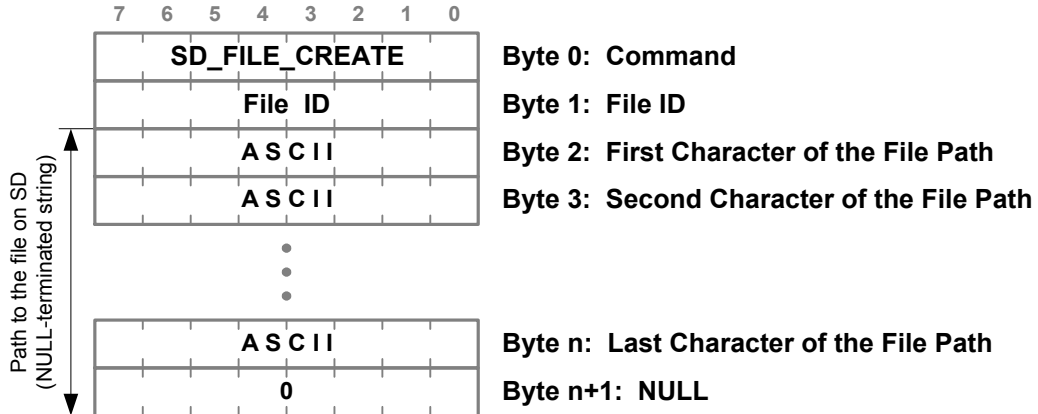
### Example:

The following sequence will close all opened SD files and re-enable the touch screen.

SD\_FILE\_CLOSE\_ALL    73 hex

### 8.53 SD\_FILE\_CREATE

**Description:** Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.  
**Supported file systems:** FAT12, FAT16, FAT32  
**Code:** 76hex, 118dec



**Notes:** SD card has to be formatted in the supported file system.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

**About the File ID:**

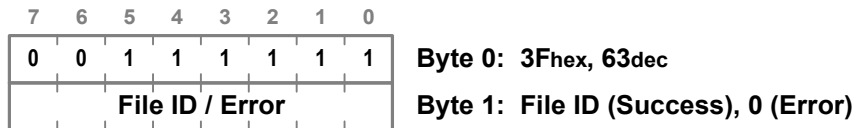
File ID identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

**About the File Path:**

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A: /Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

**ezLCD+ Response**

After receiving the SD\_FILE\_CREATE command, the ezLCD+ responds with the following sequence:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_CREATE command.

**Touch Screen Processing**

SD\_FILE\_CREATE command temporarily disables the touch screen. The touch screen will be automatically re-enabled when all files are closed. This can be done by issuing the [SD\\_FILE\\_CLOSE](#) or [SD\\_FILE\\_CLOSE\\_ALL](#) command.

**Note:** The touch screen is temporary disabled, even if due to error no file is created. If this is the case,

issuing "dummy" [SD\\_FILE\\_CLOSE](#) or [SD\\_FILE\\_CLOSE\\_ALL](#) command will re-enable the touch screen.

**Example:**

The following sequence will create and open file MyFile.dat

```
SD_FILE_CREATE  76 hex
  1              1 dec (File ID)
'M'             4D hex
'y'             79 hex
'F'             46 hex
'i'             69 hex
'l'             6C hex
'e'             65 hex
'.'             2E hex
'd'             64 hex
'a'             63 hex
't'             74 hex
NULL            0 hex
```

If the file has successfully been created, the ezLCD+ responds with the following sequence:

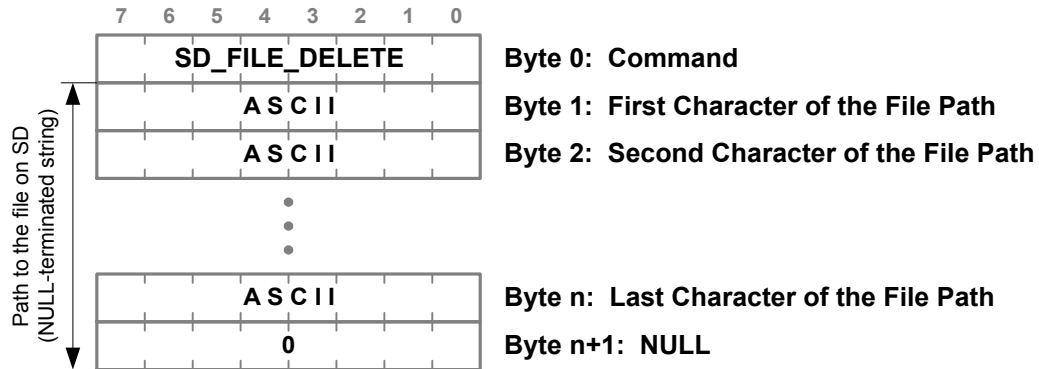
```
3F hex
  1 dec
```

In case of the failure, the following sequence will be sent by the ezLCD+:

```
3F hex
  0 dec
```

## 8.54 SD\_FILE\_DELETE

**Description:** Deletes the SD file  
**Supported file systems:** FAT12, FAT16, FAT32  
**Code:** **7D**<sub>hex</sub>, **125**<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.  
 A read-only or opened file cannot be deleted.

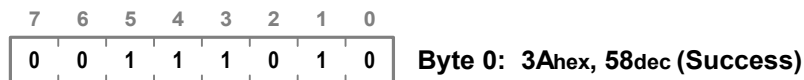
### About the File Path:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.
- Wildcards are not allowed.

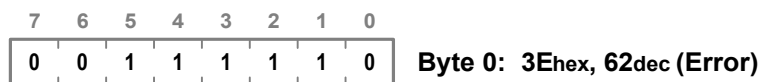
### ezLCD+ Response

After receiving the SD\_FILE\_DELETE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_DELETE command.

*SD\_FILE\_DELETE example is shown on the next page*

**Example:**

The following sequence will delete file MyFile.dat

SD_FILE_DELETE	7D hex
'M'	4D hex
'y'	79 hex
'F'	46 hex
'i'	69 hex
'l'	6C hex
'e'	65 hex
'.'	2E hex
'd'	64 hex
'a'	63 hex
't'	74 hex
NULL	0 hex

If the file has successfully been deleted, the ezLCD+ responds with the following sequence:  
3A hex

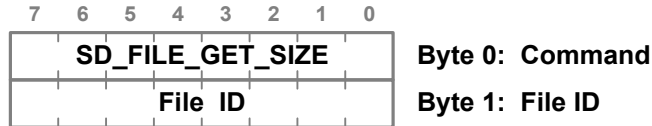
In case of the failure, the following sequence will be sent by the ezLCD+:  
3E hex

## 8.55 SD\_FILE\_GET\_SIZE

**Description:** Gets the size (in bytes) of the opened SD Flash file.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 74<sub>hex</sub>, 116<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.  
This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

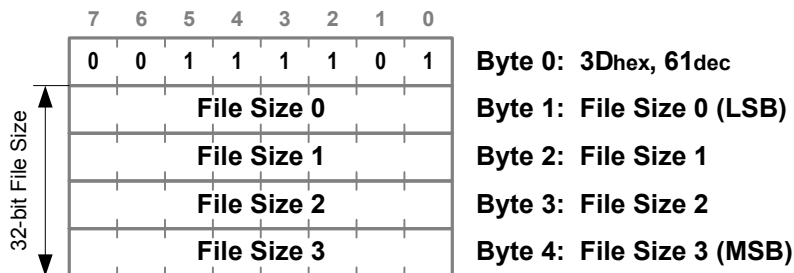
### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

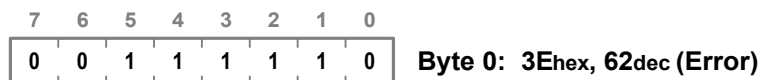
### ezLCD+ Response

After receiving the SD\_FILE\_GET\_SIZE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:

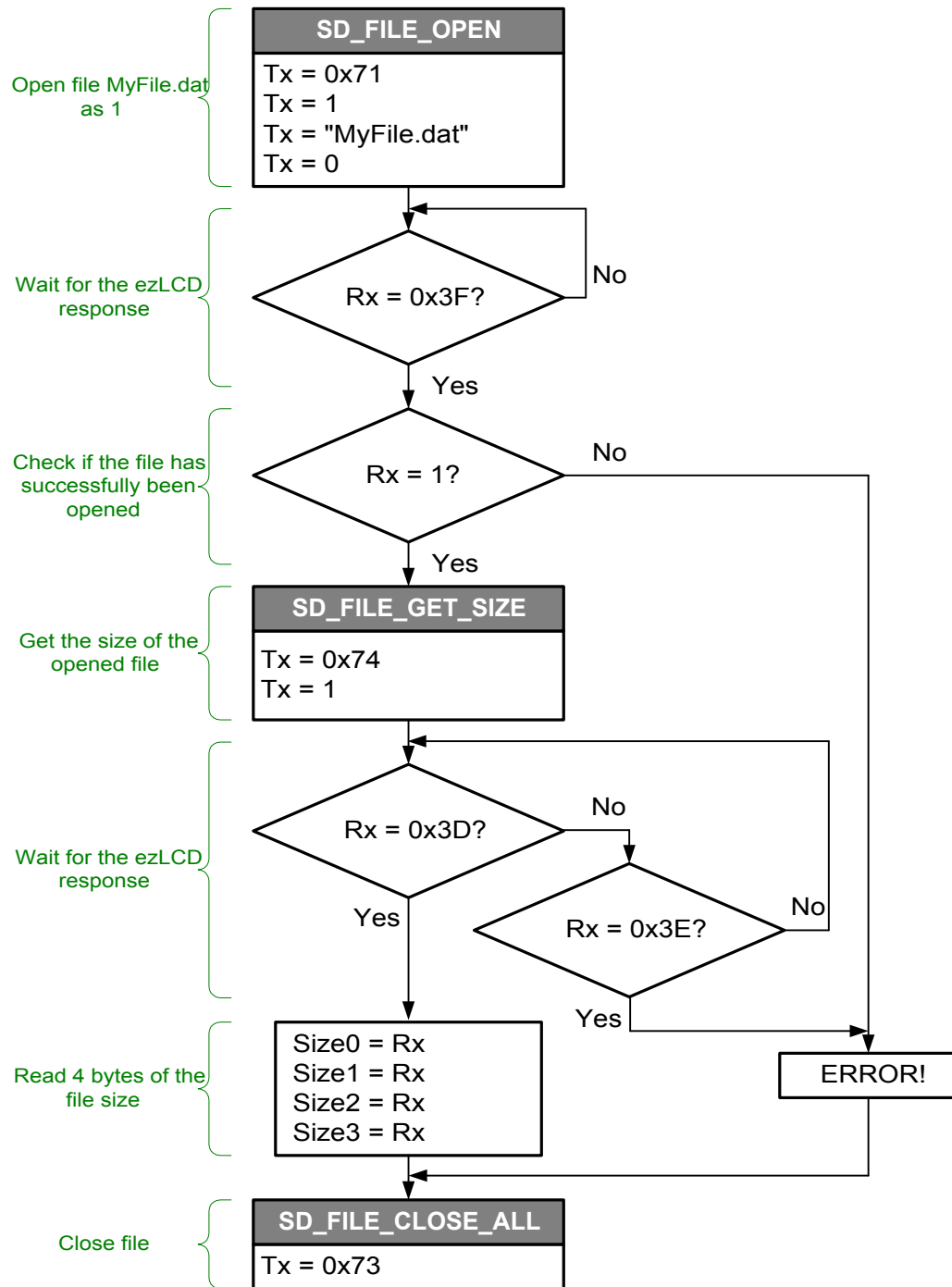


The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_GET\_SIZE command.

*SD\_FILE\_GET\_SIZE example is shown on the next page.*

**Example:**

The following flow chart shows an example of getting the size of the file MyFile.dat

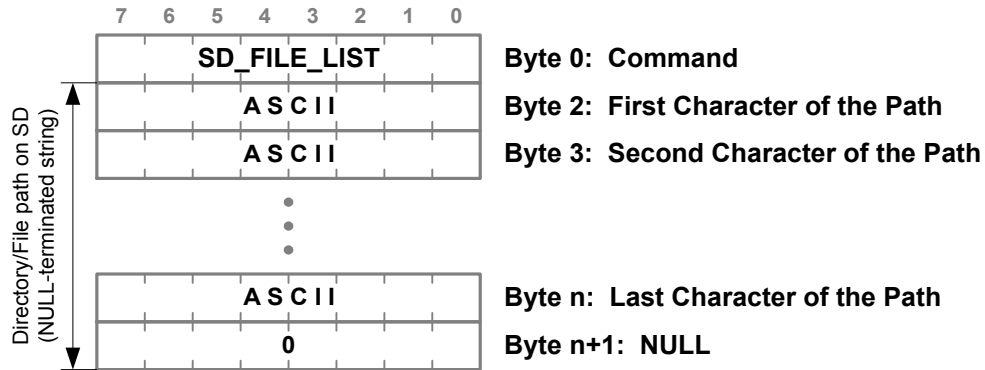


## 8.56 SD\_FILE\_LIST

**Description:** Obtains the list of files and sub-directories which reside in the specified SD Directory. This command is similar to the DOS "dir" command.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 79hex, 121dec



This command obtains all the names at once. To obtain the names one by one please refer to: [SD\\_FIND\\_FIRST](#) and [SD\\_FIND\\_NEXT](#)

**Note:** SD card has to be formatted in the supported file system.

### About the SD Directory/File Path:

- Directory Path specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: '\*' and '?' are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Directory Path is not case-sensitive. The drive and root directory do not have to be indicated, for example: A:/Cat/Jumped/Over, CAT/juMped/OvEr/ and cat/jumped/over specify the same.
- Long directory names are supported, however the Directory Path NULL may not exceed 256 bytes..

**See Also:** [SD\\_FIND\\_FIRST](#) and [SD\\_FIND\\_NEXT](#)

### ezLCD+ Response

After receiving the SD\_FILE\_LIST command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:

3Ahex (58dec), followed by the NULL-terminated string containing the directory files and sub-directories list:

- Entries (files or sub-directories) are separated by the Line Feed character (0Ahex or 10dec)
- Entries are sent in no particular order
- Sub-directories have '/' as their last character

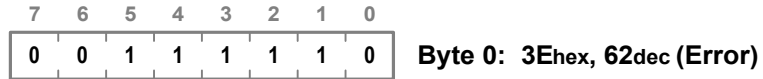
For example:

```
3Ahex      Start
whatever.txt file
Pictures/  directory
Cat.doc    file
```

---

ezLCD+.bin      *file*  
SOURCES/        *directory*  
**0**                *End (NULL)*

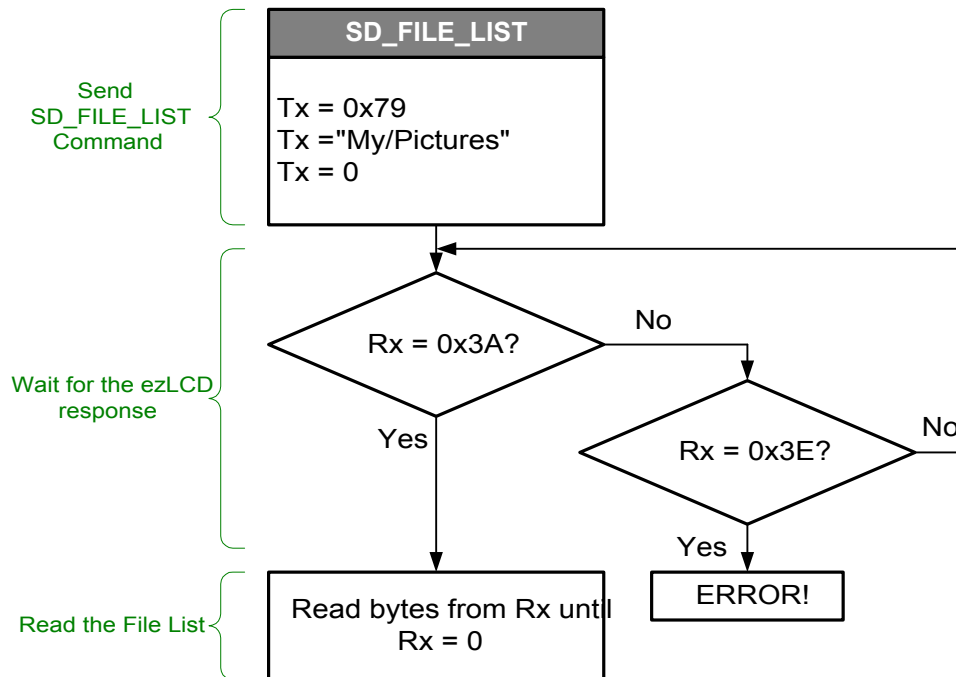
In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_LIST command

### Example:

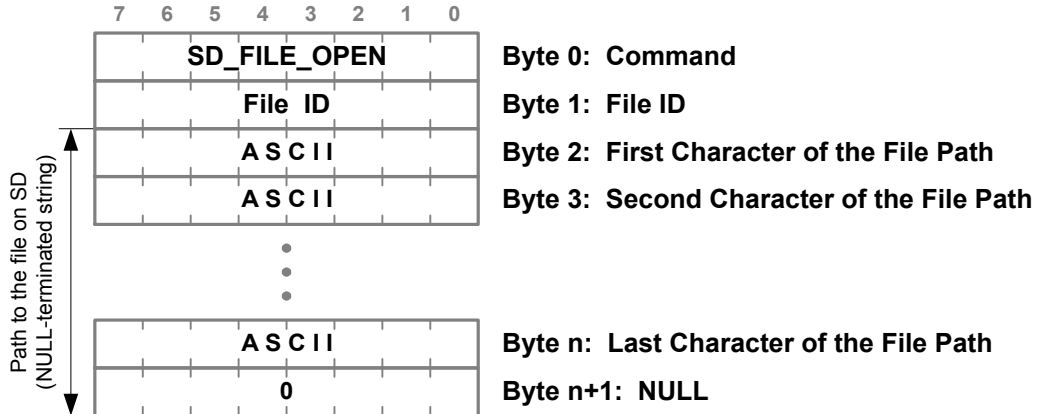
The following flow chart shows an example of reading the file list from the directory My/Pictures



### 8.57 SD\_FILE\_OPEN

**Description:** Opens an **existing** SD Flash file for reading or writing. File Position Index is set to 0. In order to open non-existing, new file, use the command [SD\\_FILE\\_CREATE](#)  
**Supported file systems:** FAT12, FAT16, FAT32

**Code:** 71hex, 113dec



**Notes:** SD card has to be formatted in the supported file system.

**See Also:** [SD\\_FILE\\_CREATE](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

**About the File ID:**

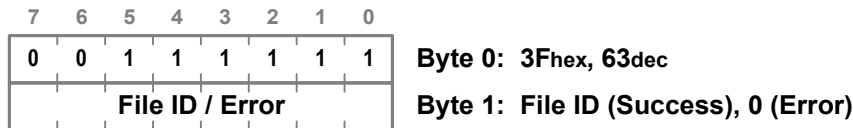
File ID identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

**About the File Path:**

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A: /Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

**ezLCD+ Response**

After receiving the SD\_FILE\_OPEN command, the ezLCD+ responds with the following sequence:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_OPEN command.

**Touch Screen Processing**

SD\_FILE\_OPEN command temporarily disables the touch screen. The touch screen will be automatically re-enabled when all files are closed. This can be done by issuing the [SD\\_FILE\\_CLOSE](#) or [SD\\_FILE\\_CLOSE\\_ALL](#) command.

**Note:** The touch screen is temporary disabled, even if due to error no file is opened. If this is the case,

issuing "dummy" [SD\\_FILE\\_CLOSE](#) or [SD\\_FILE\\_CLOSE\\_ALL](#) command will re-enable the touch screen.

**Example:**

The following sequence will open file MyFile.dat

```
SD_FILE_OPEN    71 hex
 1              1 dec (File ID)
'M'             4D hex
'y'            79 hex
'F'            46 hex
'i'            69 hex
'l'            6C hex
'e'            65 hex
'.'            2E hex
'd'            64 hex
'a'            63 hex
't'            74 hex
NULL           0 hex
```

If the file has successfully been opened, the ezLCD+ responds with the following sequence:

```
3F hex
 1 dec
```

In case of the failure, the following sequence will be sent by the ezLCD+:

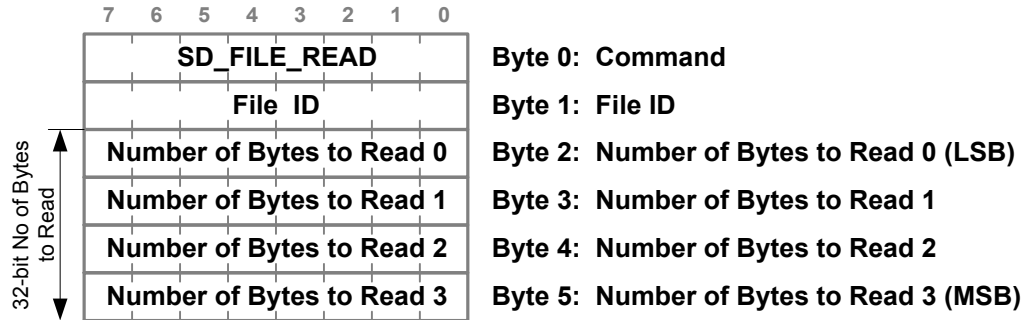
```
3F hex
 0 dec
```

## 8.58 SD\_FILE\_READ

**Description:** Reads the specified number of bytes from the opened SD Flash file, starting from File Position Index. File Position Index is incremented by the number of the bytes read, however it will not exceed file\_size - 1.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 75<sub>hex</sub>, 117<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.  
This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

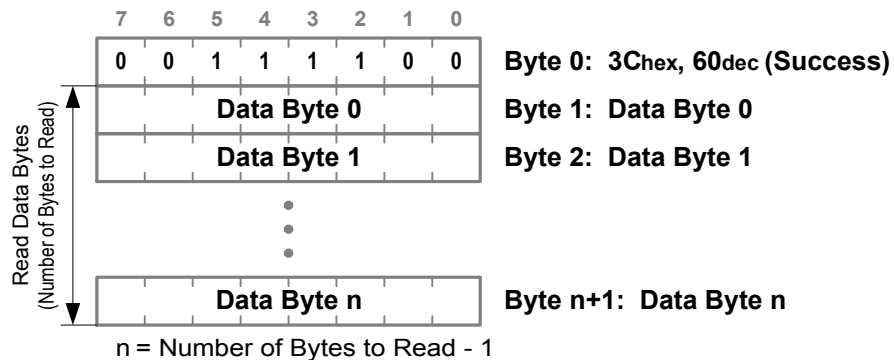
### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

### ezLCD+ Response

After receiving the SD\_FILE\_READ command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



**Note:** If the Number of Bytes to Read is greater than the number of bytes left in the file, all of the extra bytes will be preempted by 0.

In case of an **error**:

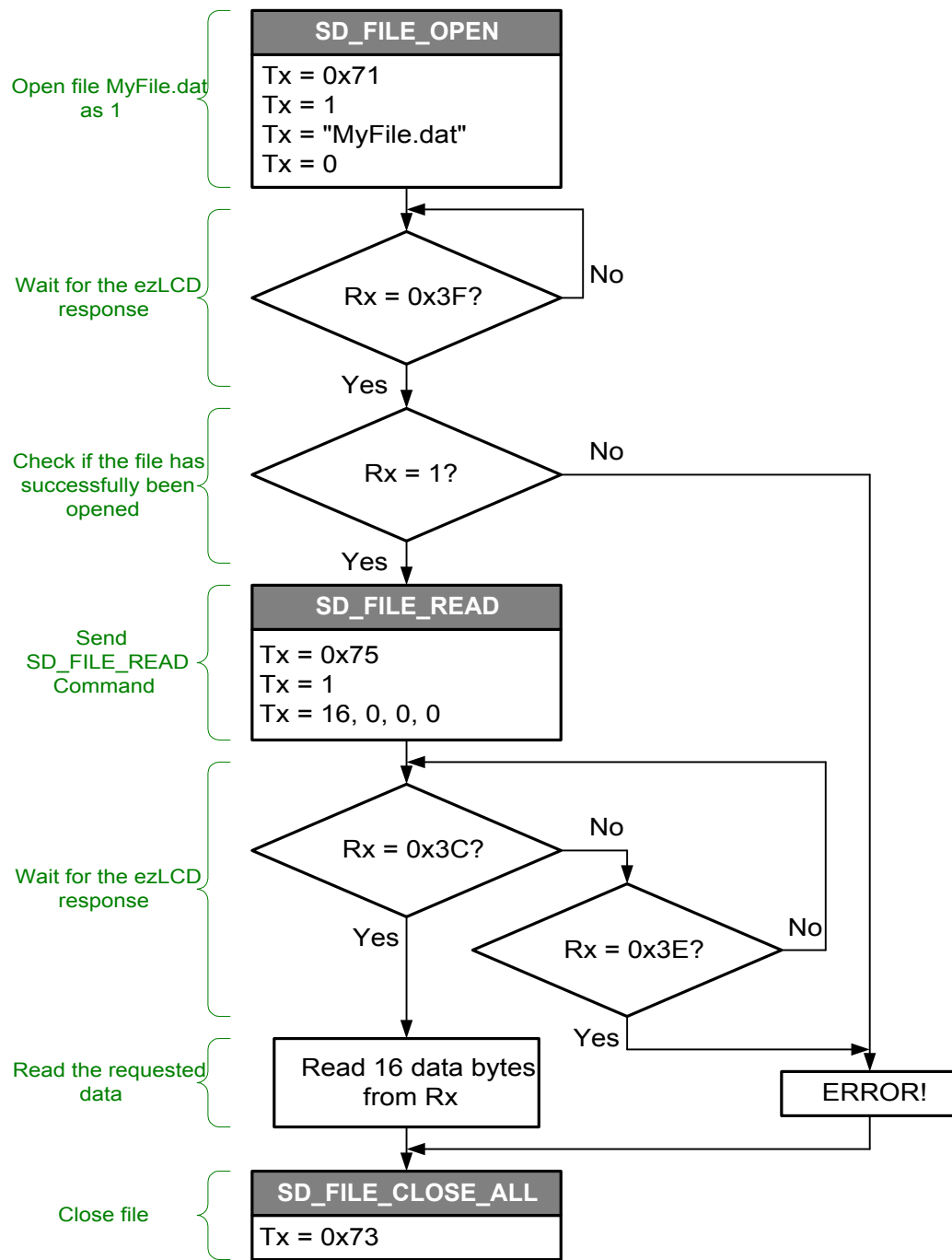
7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

**Byte 0: 3Ehex, 62dec (Error)**

The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_READ command.

**Example:**

The following flow chart shows an example of reading first 16 bytes from the file MyFile.dat

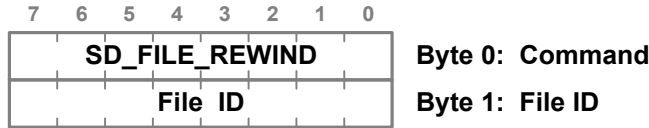


## 8.59 SD\_FILE\_REWIND

**Description:** Moves the File Position Index to the beginning of the opened SD Flash file.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 7Ah<sub>hex</sub>, 122<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system. This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command, or created and opened by the [SD\\_FILE\\_CREATE](#) command.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_SEEK](#), [SD\\_FILE\\_READ](#), [SD\\_FILE\\_WRITE](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

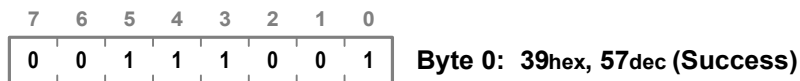
### About the File Position Index

The File Position Index specifies the Read/Write position offset (in bytes) from the beginning of the file. Upon opening of the file, the File Position Index is set to 0. The File Position Index is incremented by the subsequent read or write operations on the opened file.

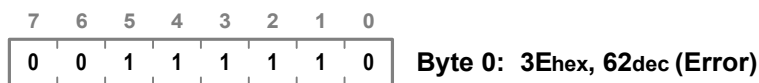
### ezLCD+ Response

After receiving the SD\_FILE\_REWIND command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_REWIND command.

### Example:

The following sequence will set the File Position Index at the beginning of the file.

```
SD_FILE_REWIND 7A hex
1 1 dec (File ID)
```

If the File Position Index has successfully been moved, the ezLCD+ responds with the following sequence:

39 hex

In case of the failure, the following sequence will be sent by the ezLCD+:

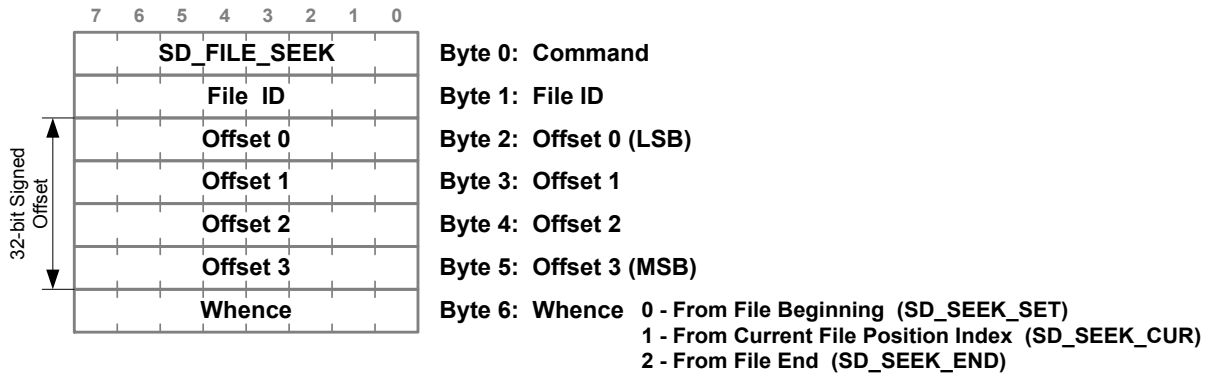
3E hex

## 8.60 SD\_FILE\_SEEK

**Description:** Moves the File Position Index of the opened SD Flash file by the specified number of bytes, from the position specified by the 'Whence' parameter.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 7Chex, 124dec



**Notes:** SD card has to be formatted in the supported file system.

This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command, or created and opened by the [SD\\_FILE\\_CREATE](#) command.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_REWIND](#), [SD\\_FILE\\_READ](#), [SD\\_FILE\\_WRITE](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

### About the Offset:

Offset is specified by the 32-bit signed integer. When the offset is negative, the File Position Index is moved backwards.

### About the File Position Index

The File Position Index specifies the Read/Write position offset (in bytes) from the beginning of the file. Upon opening of the file, the File Position Index is set to 0. The File Position Index is incremented by the subsequent read or write operations on the opened file.

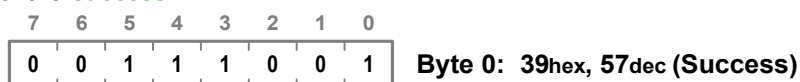
### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

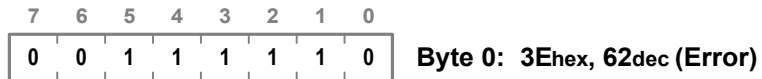
### ezLCD+ Response

After receiving the SD\_FILE\_SEEK command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_SEEK command.

### Example:

The following sequence will advance the File Position Index by 23 bytes.

<b>SD_FILE_SEEK</b>	<b>7C hex</b>	
1	1 dec	(File ID)
23	23 dec	(Offset LSB)
0	0 dec	
0	0 dec	
0	0 dec	(Offset MSB)
<b>Whence</b>	1 dec	(from the current File Position Index)

If the File Position Index has successfully been moved, the ezLCD+ responds with the following sequence:

39 hex

In case of the failure, the following sequence will be sent by the ezLCD+:

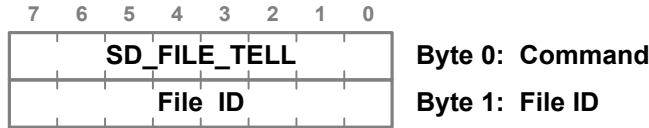
3E hex

## 8.61 SD\_FILE\_TELL

**Description:** Gets the File Position Index of the opened SD Flash file.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 7B<sub>hex</sub>, 123<sub>dec</sub>



**Notes:** SD card has to be formatted in the supported file system.

This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command, or created and opened by the [SD\\_FILE\\_CREATE](#) command.

**See Also:** [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_SEEK](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

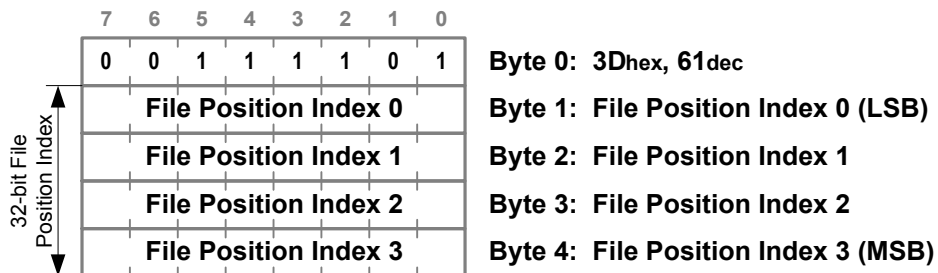
### About the File Position Index

The File Position Index specifies the Read/Write position offset (in bytes) from the beginning of the file. Upon opening of the file, the File Position Index is set to 0. The File Position Index is incremented by the subsequent read or write operations on the opened file.

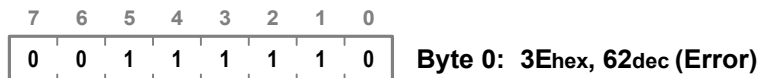
### ezLCD+ Response

After receiving the SD\_FILE\_TELL command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:

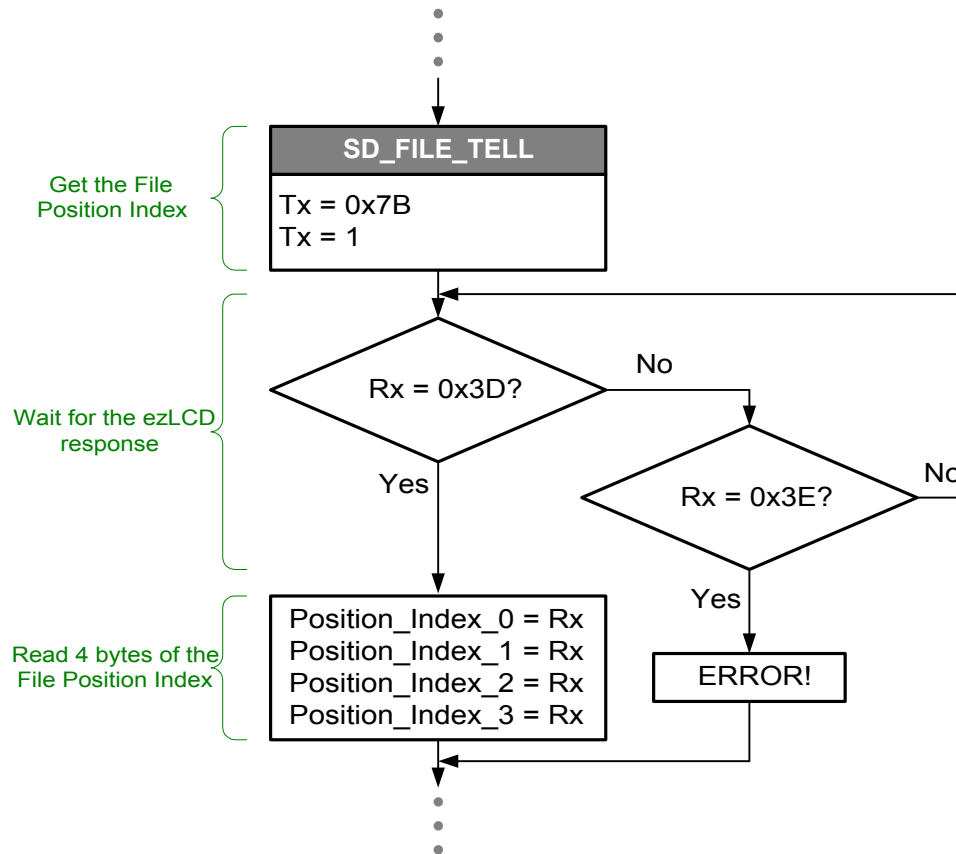


The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_TELL command.

*SD\_FILE\_TELL example is shown on the next page.*

**Example:**

The following flow chart shows an example of getting the File Position Index of the opened file with File Id = 1.

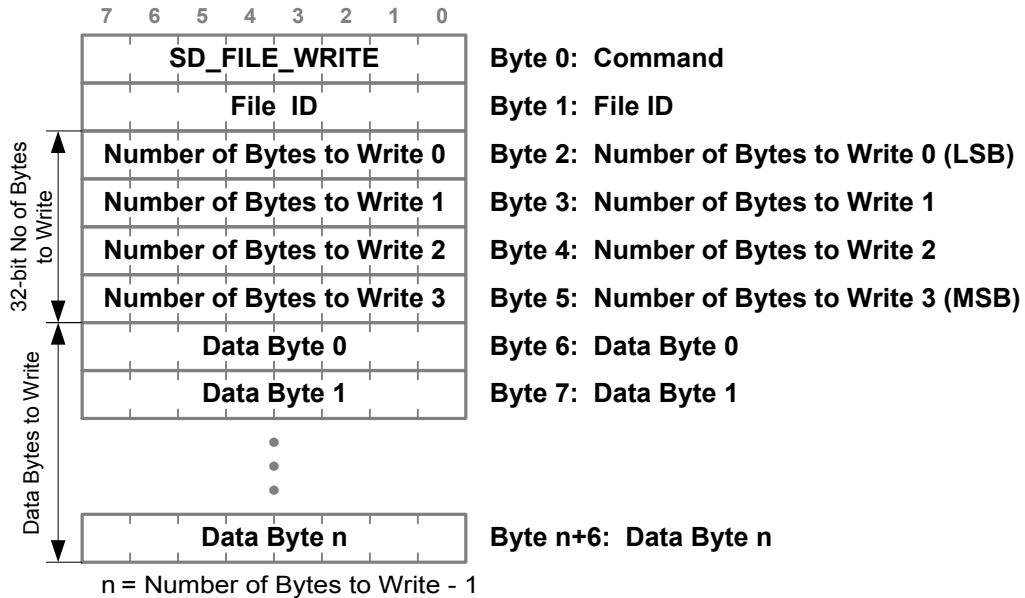


## 8.62 SD\_FILE\_WRITE

**Description:** Writes the specified number of bytes to the opened SD Flash file, starting from File Position Index. File Position Index is incremented by the number of the bytes written.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 77hex, 119dec



**Notes:** SD card has to be formatted in the supported file system. This command works only if the file is already opened by the [SD\\_FILE\\_OPEN](#) command, or created and opened by the [SD\\_FILE\\_CREATE](#) command.

**See Also:** [SD\\_FILE\\_CREATE](#), [SD\\_FILE\\_OPEN](#), [SD\\_FILE\\_CLOSE](#), [SD\\_FILE\\_CLOSE\\_ALL](#)

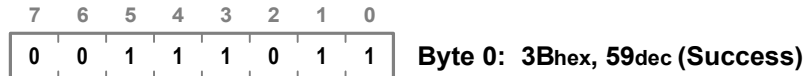
### About the File ID:

File ID is returned in the response to the [SD\\_FILE\\_CREATE](#) or [SD\\_FILE\\_OPEN](#) command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

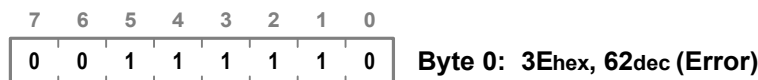
### ezLCD+ Response

After receiving the SD\_FILE\_WRITE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



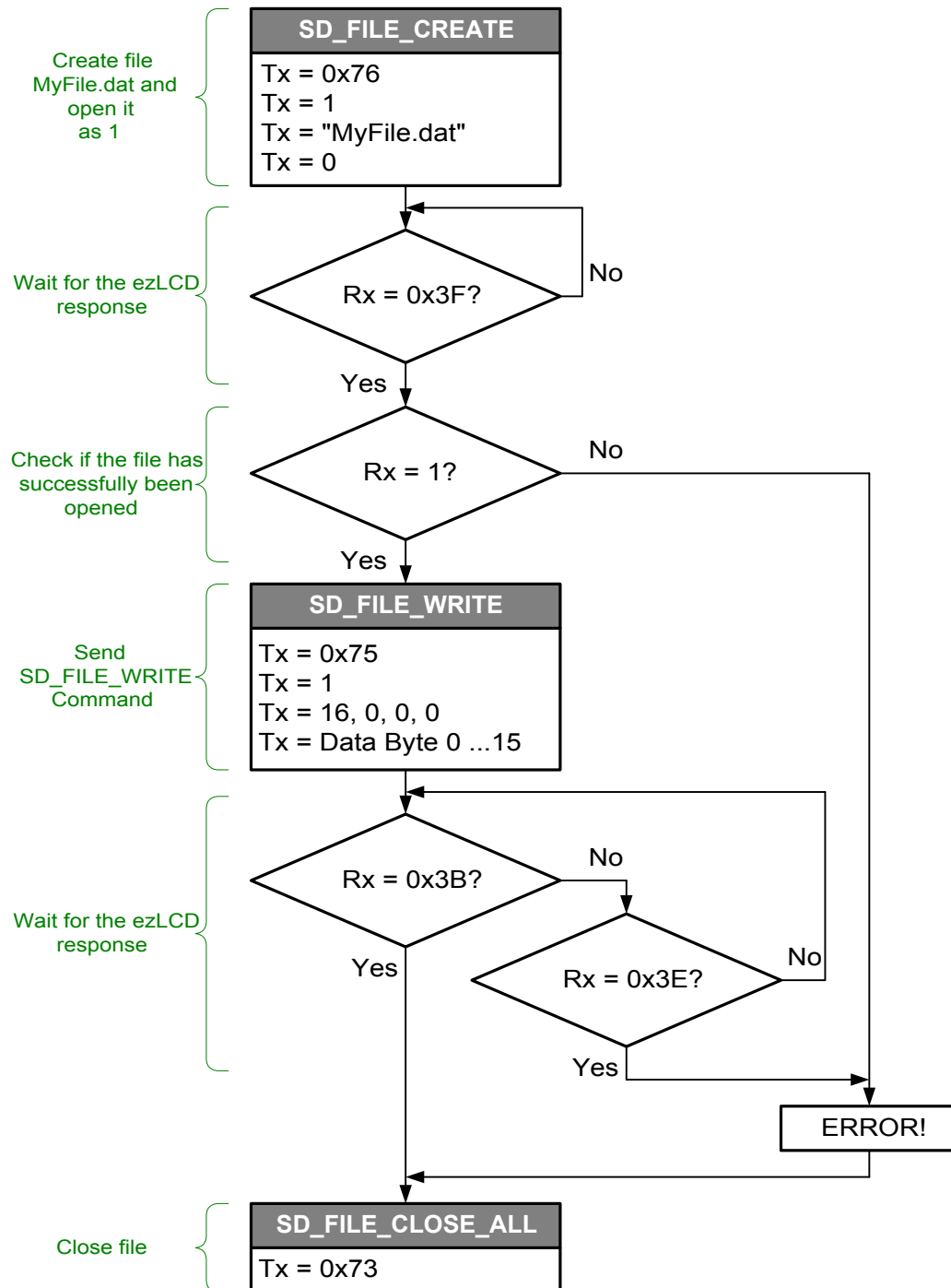
In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FILE\_WRITE command.

**Example:**

The following flow chart shows an example of writing 16 bytes into the created file MyFile.dat

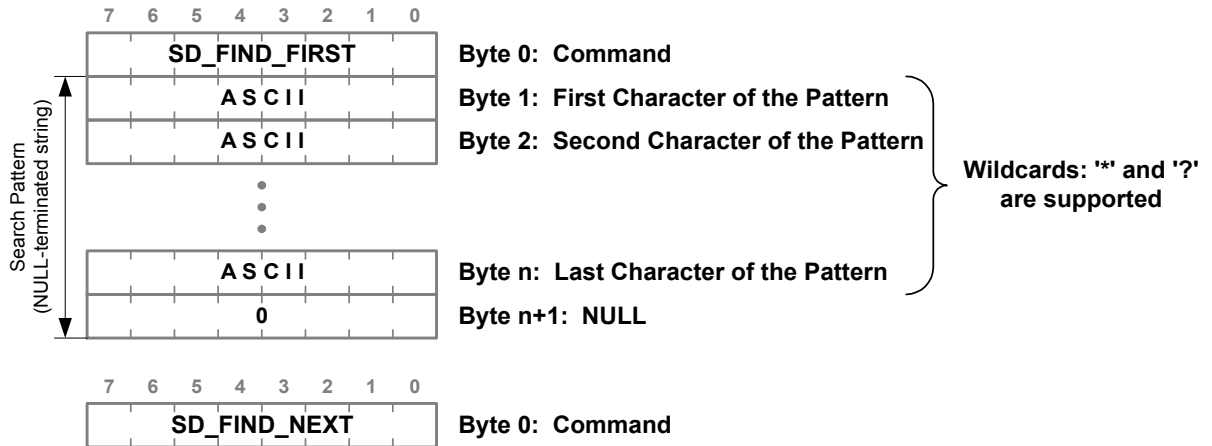


### 8.63 SD\_FIND\_FIRST and SD\_FIND\_NEXT

**Description:** Obtain the list of SD files and sub-directories (one by one), which match the specified search pattern.

Supported file systems: FAT12, FAT16, FAT32

**Codes:** SD\_FIND\_FIRST: **4A**hex, **74**dec  
 SD\_FIND\_NEXT : **4B**hex, **75**dec



SD\_FIND\_FIRST gets only the first found file or directory, which matches the search pattern. Each time, the SD\_FIND\_NEXT is issued, it finds the next file or directory, which matches the search pattern specified in the last SD\_FIND\_FIRST command. The difference between the above described mechanism and SD\_FILE\_LIST command is that it obtains the files and directories one by one, while SD\_FILE\_LIST obtains them all at once.

**Note:** SD card has to be formatted in the supported file system.

**About the Search Pattern:**

- Specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: '\*' and '?' are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Search Pattern is not case-sensitive. The drive and root directory do not have to be indicated, for example: A:/Cat/Jumped/Over, CAT/juMped/OvEr/ and cat/jumped/over specify the same.
- Long directory and file names are supported, however the Search Pattern + NULL may not exceed 256 bytes..

**See Also:** [SD\\_FILE\\_LIST](#)

**ezLCD+ Response**

After receiving any of the described commands, the ezLCD+ responds with either of the following sequences:

In case of the **success**:

**3A**hex (**58**dec), followed by the NULL-terminated string containing file or directory name. Directories have '/' as their last character

Examples:

3Ah<sub>hex</sub>            *Start*  
 whatever.txt      *file*  
 0                    *End (NULL)*

or

3Ah<sub>hex</sub>            *Start*  
 Pictures/          *directory*  
 0                    *End (NULL)*

In case no files were found or in case of an **error**:

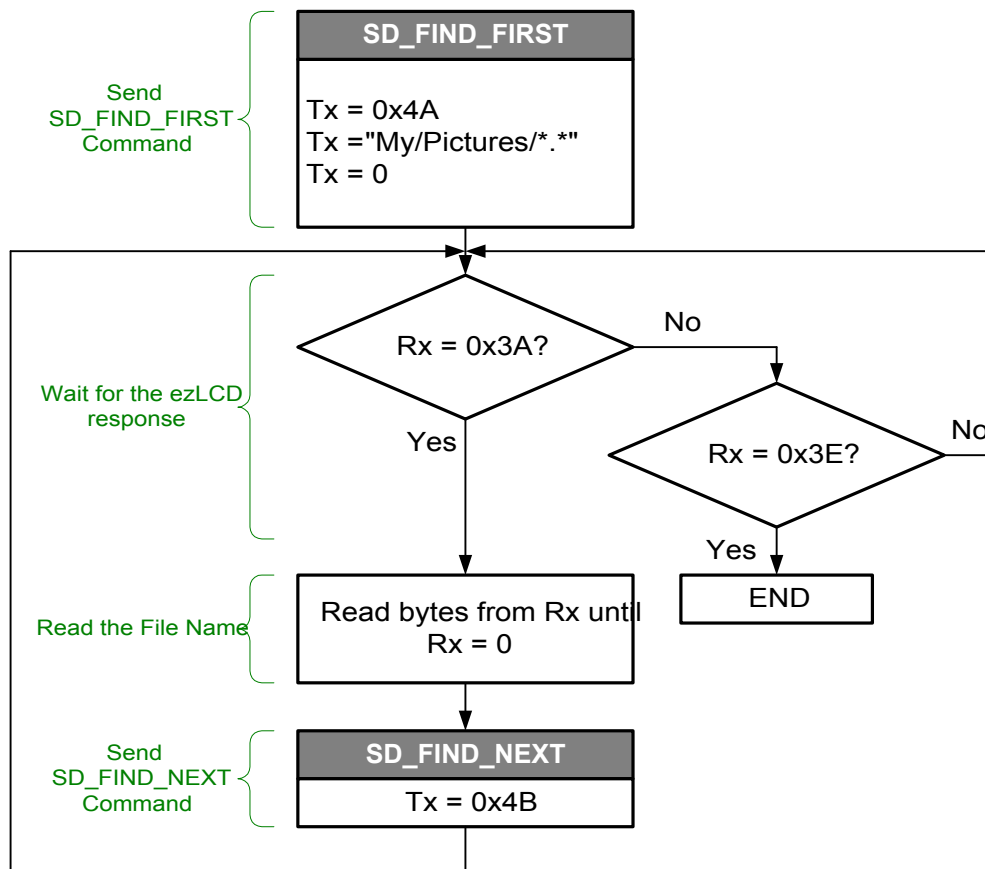
7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

Byte 0: 3E<sub>hex</sub>, 62<sub>dec</sub> (Error)

The ezLCD+ response is sent through the same interface, which received the command

### Example:

The following flow chart shows an example of reading the file list from the directory My/Pictures



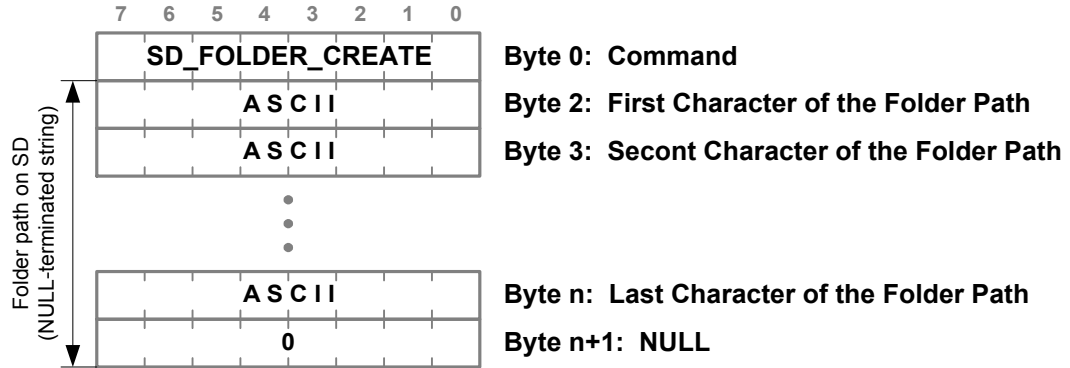


## 8.64 SD\_FOLDER\_CREATE

**Description:** Creates a new folder (directory) on the SD.  
This command is similar to the DOS "mkdir" command.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 46hex, 70dec



**Notes:** SD card has to be formatted in the supported file system.  
Parent directory (folder) has to exist.

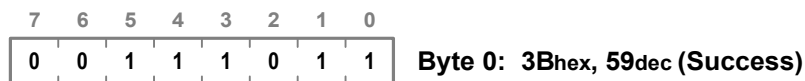
### About the Folder Path:

- Folder Path specifies the full path to the directory on the SD.
- Directories (folders) should be separated by: / (**not by:** \ like in Windows and DOS).
- Long names are supported, however the Folder Path (+ NULL) may not exceed 256 bytes.

### ezLCD+ Response

After receiving the SD\_FOLDER\_CREATE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FOLDER\_CREATE command.

*SD\_FOLDER\_CREATE example is shown on the next page*

**Example:**

The following sequence will create folder MyDir in the root directory

```
SD_FOLDER_CREATE 46 hex
'M'              4D hex
'y'              79 hex
'D'              44 hex
'i'              69 hex
'r'              72 hex
NULL             0 hex
```

If the folder has successfully been created, the ezLCD+ responds with the following sequence:

```
3B hex
```

In case of the failure, the following sequence will be sent by the ezLCD+:

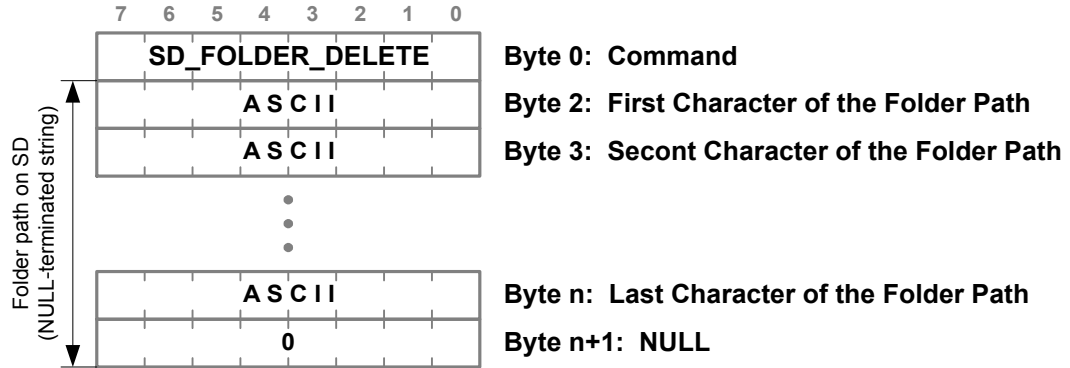
```
3E hex
```

## 8.65 SD\_FOLDER\_DELETE

**Description:** Deletes an empty folder (directory) on the SD  
This command is similar to the DOS "rmdir" command.

Supported file systems: FAT12, FAT16, FAT32

**Code:** 4Dhex, 77dec



**Notes:** SD card has to be formatted in the supported file system.  
Folder (directory) has to be empty

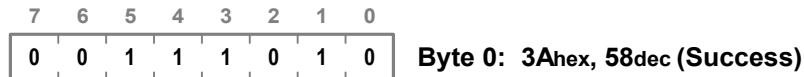
### About the Folder Path:

- Folder Path specifies the full path to the directory on the SD.
- Directories (folders) should be separated by: / (**not by:** \ like in Windows and DOS).
- Long names are supported, however the Folder Path (+ NULL) may not exceed 256 bytes.
- Wildcards are not allowed.

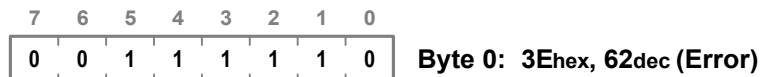
### ezLCD+ Response

After receiving the SD\_FOLDER\_DELETE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_FOLDER\_DELETE command.

*SD\_FOLDER\_DELETE example is shown on the next page*

**Example:**

The following sequence will delete folder MyDir from the root directory

SD_FOLDER_DELETE	4D hex
'M'	4D hex
'y'	79 hex
'D'	44 hex
'i'	69 hex
'r'	72 hex
NULL	0 hex

If the folder has successfully been deleted, the ezLCD+ responds with the following sequence:

3A hex

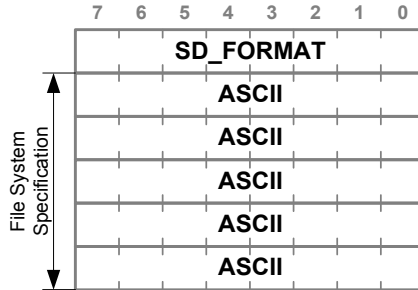
In case of the failure, the following sequence will be sent by the ezLCD+:

3E hex

## 8.66 SD\_FORMAT

**Description:** Formats the SD in the specified file system.  
Supported file systems: FAT12, FAT16, FAT32

**Code:** 4F<sub>hex</sub>, 79<sub>dec</sub>



**Byte 0: Command**

**Byte 1: The first character of the File System Specification**

**Byte 2: The second character of the File System Specification**

**Byte 3: The third character of the File System Specification**

**Byte 4: The fourth character of the File System Specification**

**Byte 5: The fifth character of the File System Specification**

**Warning:** This command will erase all files on the SD

### About the File System Specification:

- Sets the file system in which the SD will be formatted.
- 5 ASCII characters
- ASCII characters only. For example: the code of '1' is 31<sub>hex</sub>.
- Supported file systems: FAT12, FAT16, FAT32

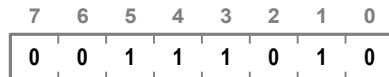
### About the supported file systems

	FAT12	FAT16	FAT32
<b>Full Name</b>	File Allocation Table		
	12-bit version	16-bit version	32-bit version
<b>Introduced</b>	1977	July 1988	August 1996
<b>Max file size</b>	32 MB	2 GB	4 GB
<b>Max number of files</b>	4,077	65,517	268,435,437
<b>Max volume size</b>	32 MB	2 GB	8 TB

### ezLCD+ Response

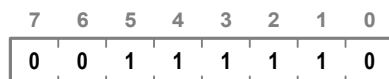
After receiving the SD\_FORMAT command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



**Byte 0: 3A<sub>hex</sub>, 58<sub>dec</sub> (Success)**

In case of an **error**:



**Byte 0: 3E<sub>hex</sub>, 62<sub>dec</sub> (Error)**

The ezLCD+ response is sent through the same interface, which received the SD\_FORMAT command.

**Example:**

The following sequence will format the SD in FAT16

```
SD_FORMAT    4F hex
'F'          46 hex
'A'          41 hex
'T'          54 hex
'1'          31 hex
'6'          36 hex
```

If the folder has successfully been deleted, the ezLCD+ responds with the following sequence:

```
3A hex
```

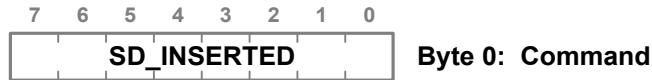
In case of the failure, the following sequence will be sent by the ezLCD+:

```
3E hex
```

## 8.67 SD\_INSERTED

**Description:** Checks if the SD card is inserted

**Code:** 49hex, 73dec



ezLCD+ Response

After receiving the SD\_INSERTED command, the ezLCD+ responds with either of the following sequences:

If an SD card is inserted in the SD slot:



If there is no card inserted in the SD slot:



The ezLCD+ response is sent through the same interface, which received the SD\_INSERTED command.

### Example:

The following sequence will check if the SD card is present in the SD slot.

SD\_INSERTED 49 hex

### ezLCD+ Response

If an SD card is present in the SD slot:

3D hex

If there is no card inserted in the SD slot:

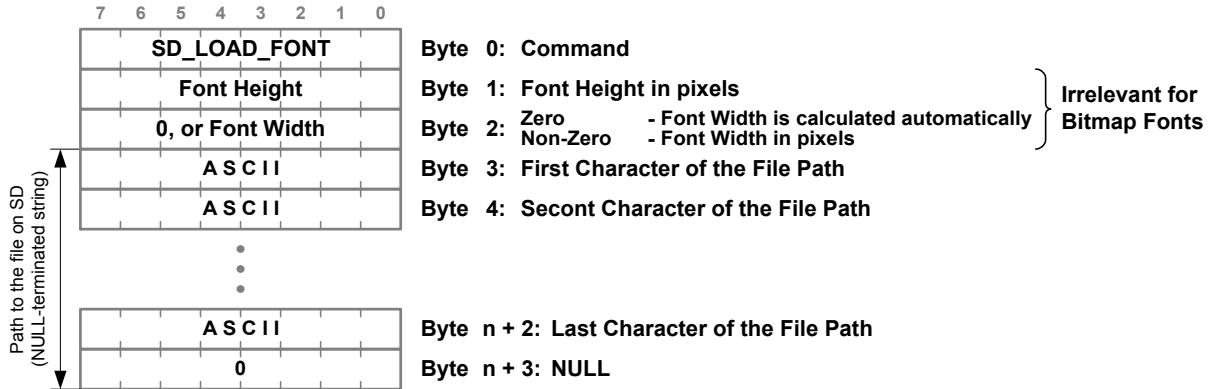
3E hex

### 8.68 SD\_LOAD\_FONT

**Description:** Loads and selects font file from SD card. Both, [Bitmap Fonts](#) and [True Type Fonts](#) are supported.

[Supported file systems:](#) FAT12, FAT16, FAT32

**Code:** 92hex, 146dec



The successfully loaded font is automatically selected as the Current Font for printing commands. The contents of bytes 1 and 2 is irrelevant in case of the Bitmap Font file (.ezf). Since True Type (Free Type) fonts (.ttf, .otf) are scalable, their size has to be specified as shown above. If Byte 2 is set to 0, the width of the loaded True Type font will be calculated automatically (recommended). Each time the True Type font is loaded from SD, the font cache is cleared.

Selection of the font type (Bitmap or True Type) depends of the filename extension:

- Bitmap Font: .ezf
- True Type Font: .ttf, .otf

**See Also:** [SELECT\\_FONT](#), [SET\\_FT\\_FONT](#)

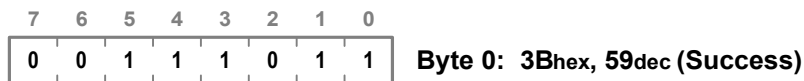
**About the File Path:**

- File Path specifies the full path to the file on SD including directory, filename, and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

**ezLCD+ Response**

After execution of the SD\_LOAD\_FONT command, the ezLCD+ responds with either of the following sequences:

In case of **success**:



In case of an **error**:

7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

**Byte 0: 3Ehex, 62dec (Error)**

The ezLCD+ response is sent through the same interface, which received the SD\_LOAD\_FONT command.

**Example:**

The following sequence will load the True Type font "Arial.ttf" from SD.

```
SD_LOAD_FONT 92 hex
 16          16 dec (Height = 16)
  0          0      (Width = automatic)
"Arial.ttf"
  NULL      0 (NULL)
```

If the font has been loaded successfully, the ezLCD+ responds with:

```
3B hex
```

In case of failure, the following byte will be sent by the ezLCD+:

```
3E hex
```

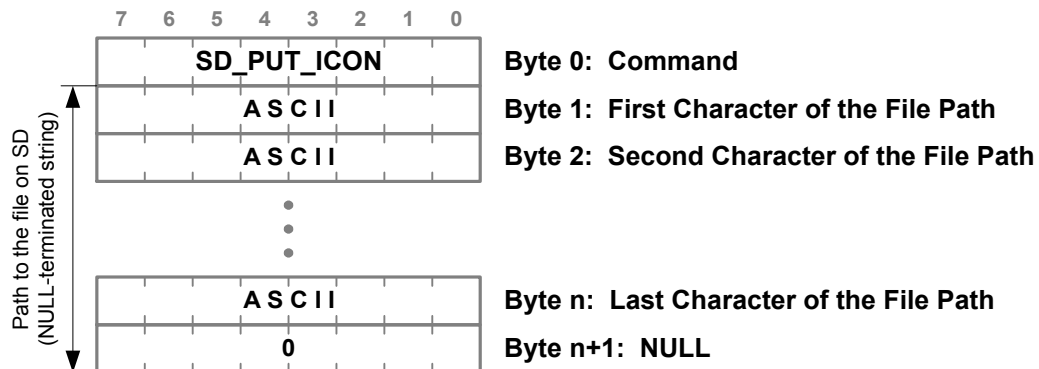
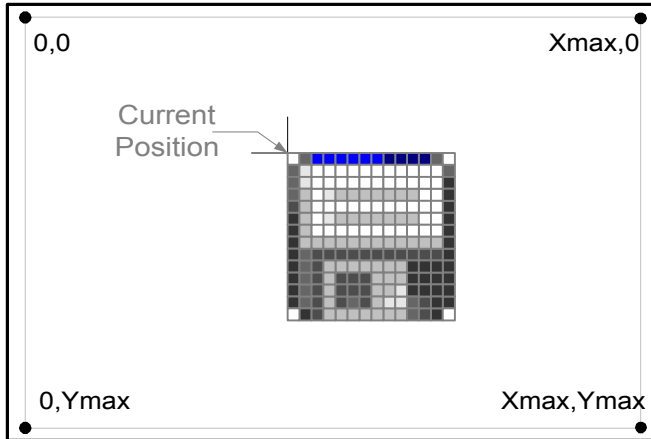
## 8.69 SD\_PUT\_ICON

**Description:** Displays an icon with its upper-left corner positioned at the Current Position. The icon is read from the file on the SD card attached to the SD/MMC interface.

**Supported file systems:** FAT12, FAT16, FAT32

**Supported formats:** .jpg, 24-bit .bmp and .ezp

**Code:** 70hex, 112dec



**Note:** SD card has to be formatted in the supported file system.

**See Also:** [PUT\\_PICT\\_NO](#)

### About the File Path:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 256 bytes.

### Example:

The following sequence will display the bitmap from file ezLCD.jpg with its upper-left corner positioned at (60, 43).

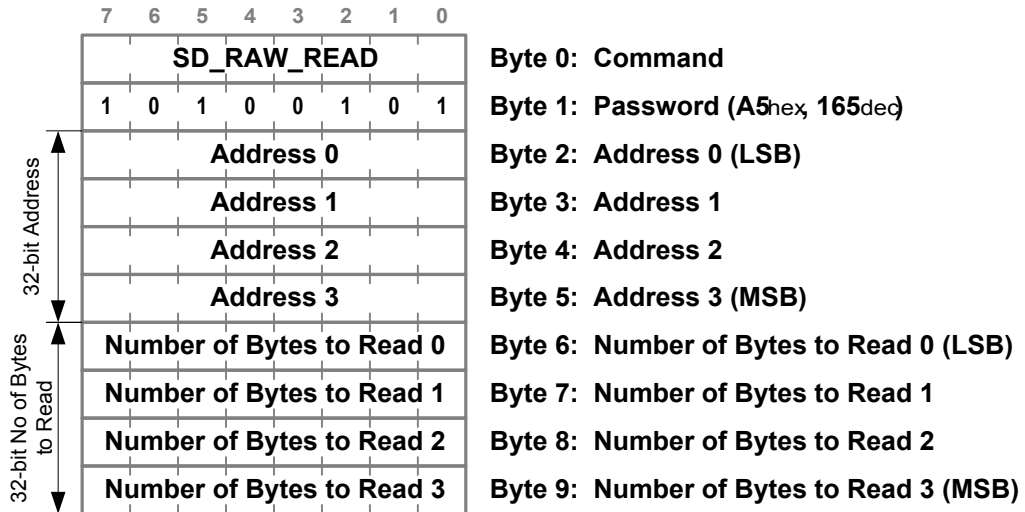
```
SET_XHYH    33 hex
```

0	0 dec	(x MSB)
60	60 dec	(x LSB)
0	0 dec	(y MSB)
43	43 dec	(y LSB)
<b>SD_PUT_ICON</b>	<b>70 hex</b>	
<b>'e'</b>	<b>65 hex</b>	
<b>'z'</b>	<b>7A hex</b>	
<b>'L'</b>	<b>4C hex</b>	
<b>'C'</b>	<b>43 hex</b>	
<b>'D'</b>	<b>44 hex</b>	
<b>'.'</b>	<b>2E hex</b>	
<b>'j'</b>	<b>6A hex</b>	
<b>'p'</b>	<b>70 hex</b>	
<b>'g'</b>	<b>67 hex</b>	
<b>NULL</b>	<b>0 hex</b>	

## 8.70 SD\_RAW\_READ

**Description:** Reads the data from SD starting from the specified SD address. SD is treated as a memory with the starting address 0.

**Code:** 7E<sub>hex</sub>, 126<sub>dec</sub>

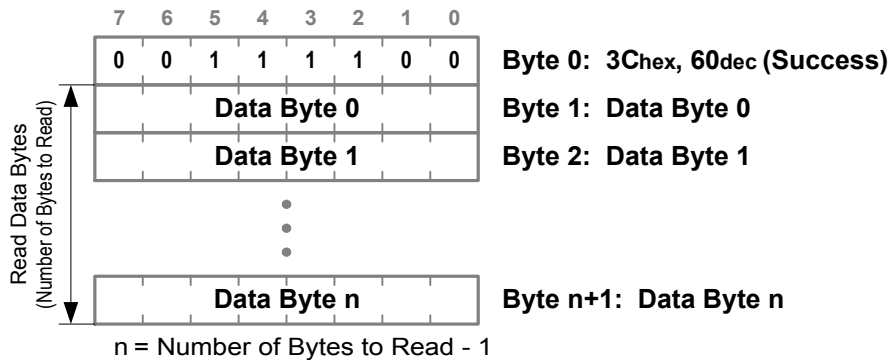


**See Also:** [SD\\_RAW\\_WRITE](#)

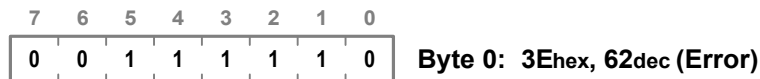
### ezLCD+ Response

After receiving the SD\_RAW\_READ command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



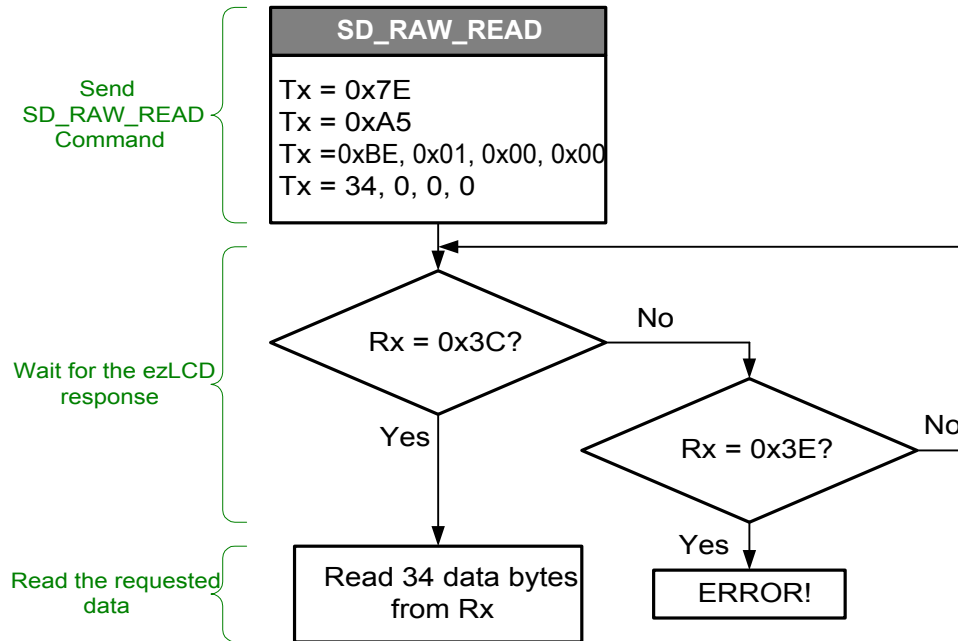
In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_RAW\_READ command.

**Example:**

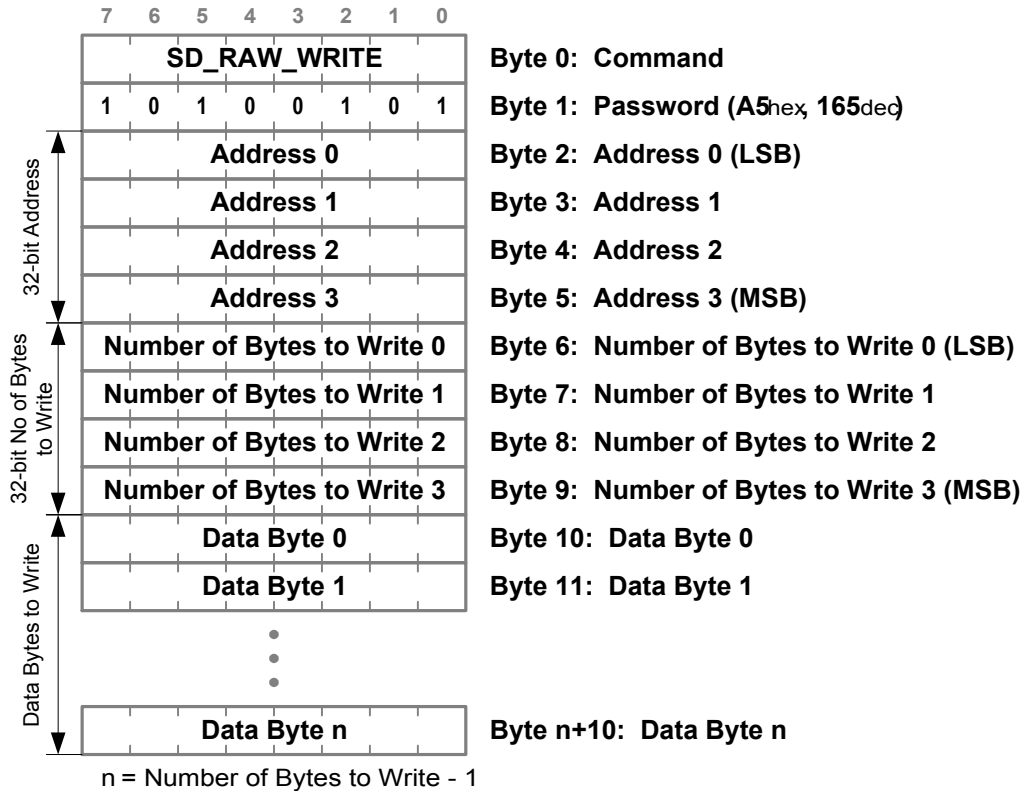
The following flow chart shows an example of reading 34 bytes starting from the SD address 000001BE<sub>hex</sub>.



## 8.71 SD\_RAW\_WRITE

**Description:** Writes the data on SD starting from the specified SD address. SD is treated as a memory with the starting address 0.

**Code:** **7F**<sub>hex</sub>, **127**<sub>dec</sub>



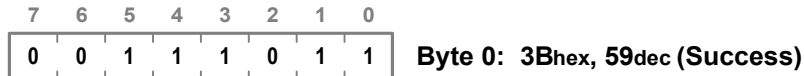
**Warning!** This command performs raw write on SD. Use it with caution. It may overwrite the existing SD files or corrupt the SD file formatting.

**See Also:** [SD\\_RAW\\_READ](#)

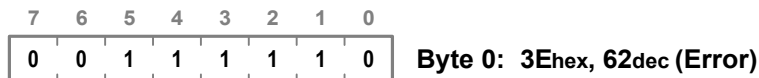
### ezLCD+ Response

After receiving the SD\_RAW\_WRITE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:

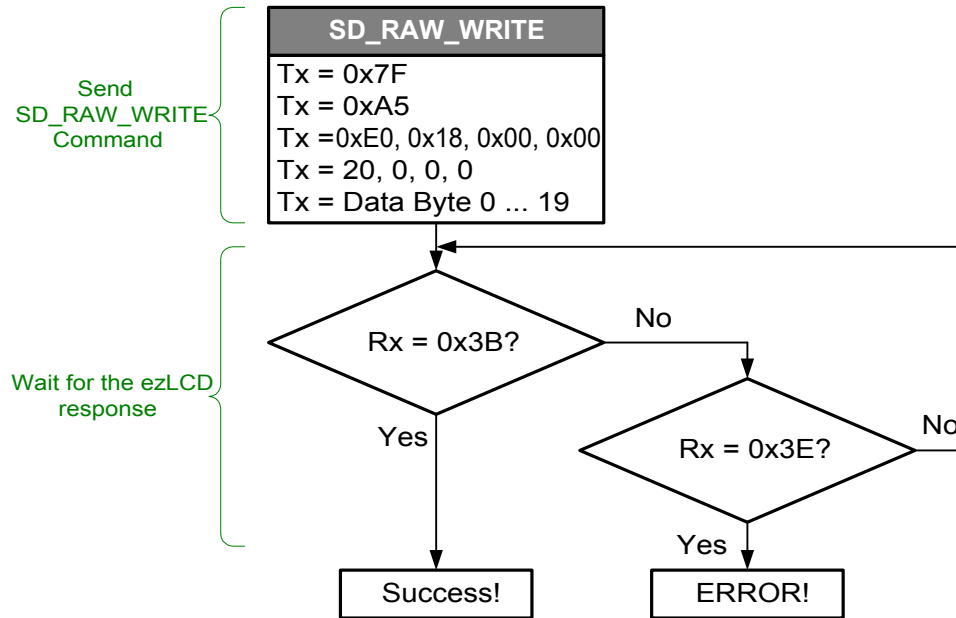


The ezLCD+ response is sent through the same interface, which received the SD\_RAW\_WRITE command.



**Example:**

The following flow chart shows an example of writing 20 bytes starting from the SD address 000018E0<sub>hex</sub>.

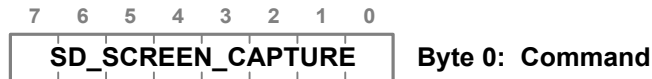


## 8.72 SD\_SCREEN\_CAPTURE

**Description:** Saves an image of the displayed screen to the SD as .bmp file.

[Supported file systems:](#) FAT12, FAT16, FAT32

**Code:** 44hex, 68dec



This command is helpful when writing the documentation of your ezLCD+ project, because the captured screen images may be used as examples.

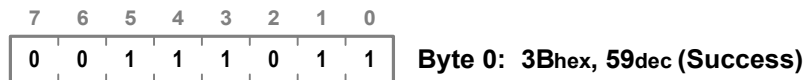
Screen capture files have names "Scr\_xxxx.bmp", where xxxx is a consecutive number. For example: Scr\_0001.bmp, Scr\_0002.bmp, etc. The files are created in the "Scr\_Cap" SD folder. If the SD does not have the "Scr\_Cap" folder, it will be created automatically.

**Notes:** SD card has to be formatted in the supported file system.  
This command may take up to 2 seconds to execute.

### ezLCD+ Response

After execution of the SD\_SCREEN\_CAPTURE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_SCREEN\_CAPTURE command.

### Example:

The following sequence will save the image of the displayed screen to the SD file.

SD\_SCREEN\_CAPTURE 44 hex

If the screen image has been written to the .bmp file, the ezLCD+ responds with:

3B hex

In case of the failure, the following byte will be sent by the ezLCD+:

3E hex

## 8.73 SD\_SIZE

**Description:** Gets the physical size (in bytes) of the SD Card.

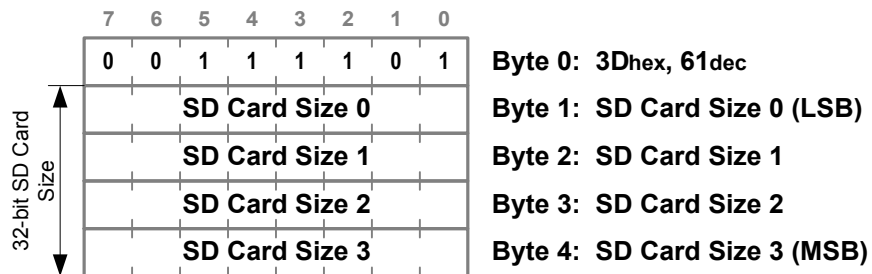
**Code:** 78<sub>hex</sub>, 120<sub>dec</sub>



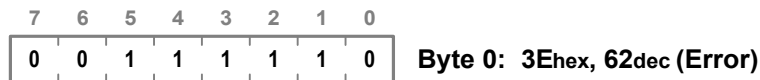
### ezLCD+ Response

After receiving the SD\_SIZE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



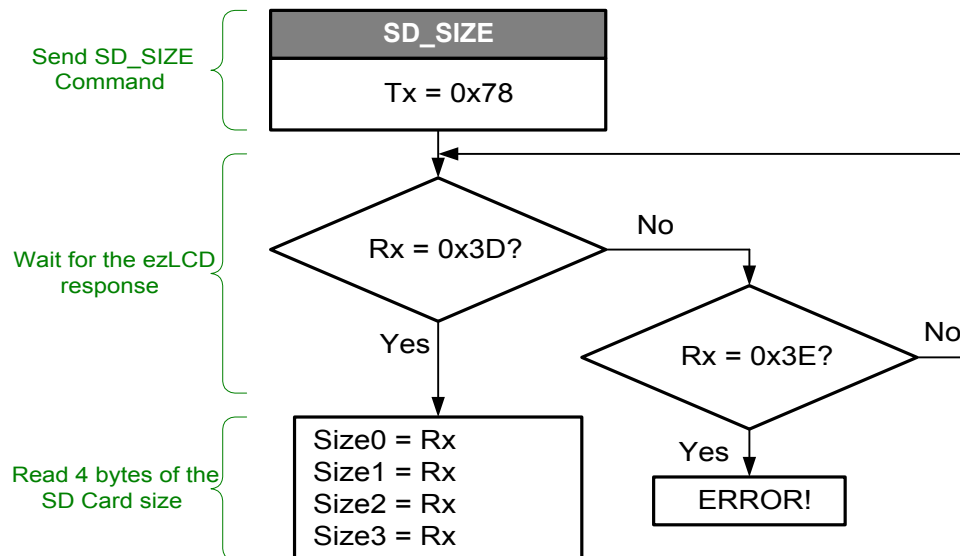
In case of an **error**:



The ezLCD+ response is sent through the same interface, which received the SD\_SIZE command.

### Example:

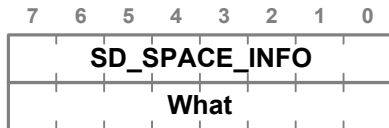
The following flow chart shows an example of getting the size of the SD Card.



## 8.74 SD\_SPACE\_INFO

**Description:** Gets the information about the space usage (in bytes) of the formatted SD Card.  
[Supported file systems:](#) FAT12, FAT16, FAT32

**Code:** 48hex, 72dec



**Byte 0: Command**

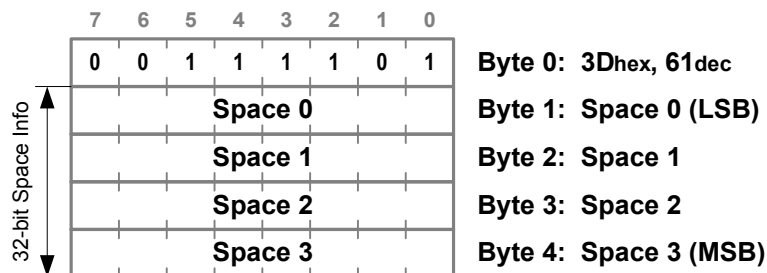
**Byte 1: What** 1 - Get Free Space  
 2 - Get Used Space  
 3 - Get Bad Space  
 Any Other Number - Get Total Formatted Space

**Notes:** SD card has to be formatted in the supported file system.

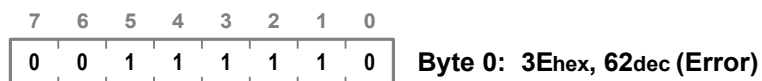
### ezLCD+ Response

After receiving the SD\_SPACE\_INFO command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



In case of an **error**:

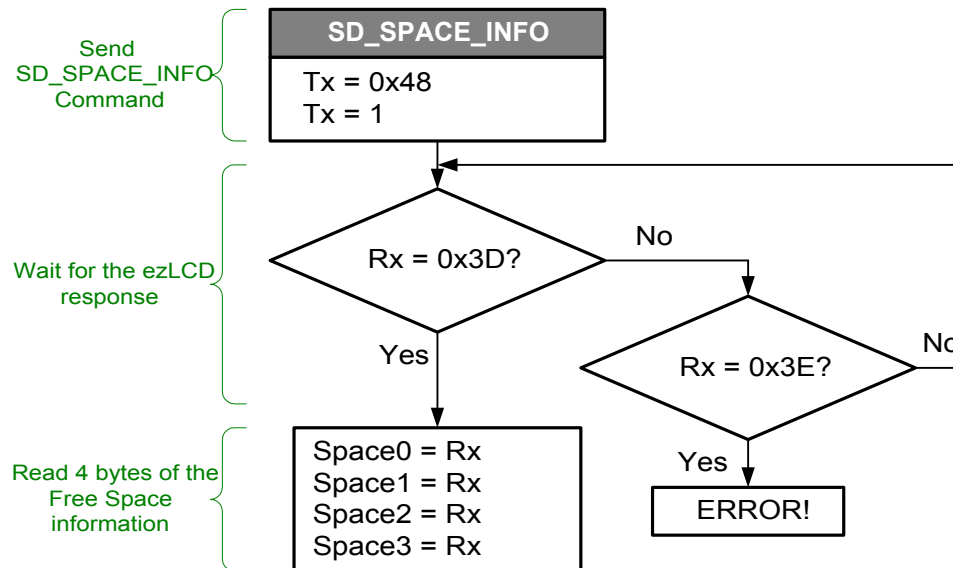


The ezLCD+ response is sent through the same interface, which received the SD\_SPACE\_INFO command.

*SD\_SPACE\_INFO example is shown on the next page.*

**Example:**

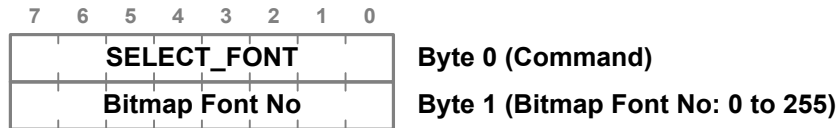
The following flow chart shows an example of getting the number of the available bytes on the formatted SD card.



## 8.75 SELECT\_FONT

**Description:** Selects the [Bitmap Font](#) from User ROM.

**Code:** 2Bhex, 43dec



**Note:** 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

See Also: [PRINT\\_STRING](#), [PRINT\\_CHAR](#)

### Example:

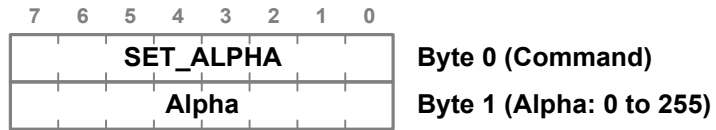
The following sequence will print a black bitmap character 'M' using Font 2.

```
SELECT_FONT  2B hex
2            2 dec
SET_COLOR_RGB 31 hex
Red         0
Green       0
Blue        0
PRINT_CHAR   2C hex
'M'         4D hex
```

## 8.76 SET\_ALPHA

**Description:** Sets value of the transparency [Alpha](#).

**Code:** 20hex, 32dec



**Note:** The drawings are rendered almost 3 time slower, when Alpha is set to any value other than 255 or 0.

**Ref:** [Transparency](#)

**See Also:** [SET\\_COLOR\\_RGB](#)

### Example:

The following sequence will draw a semi-transparent color filled circle

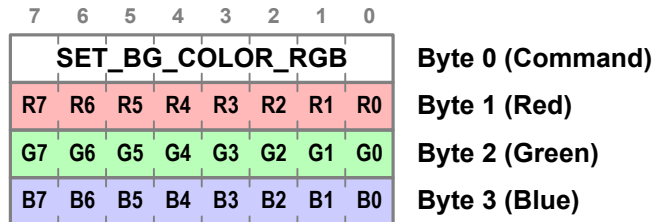
```
SET_ALPHA      20 hex
               128 dec (Alpha)
CIRCLE_RH_FILL 99 hex
               0 dec (radius MSB)
               80 dec (radius LSB)
```

## 8.77 SET\_BG\_COLOR\_RGB

**Description:** Sets the Background Color for the following instructions:

[PRINT\\_CHAR\\_BG](#)  
[PRINT\\_STRING\\_BG](#)

**Code:** 32hex, 50dec



See Also: [PRINT\\_CHAR\\_BG](#), [PRINT\\_STRING\\_BG](#)

### Example:

The following sequence will print "LCD" in yellow on a navy background, using Bitmap Font 0.

```

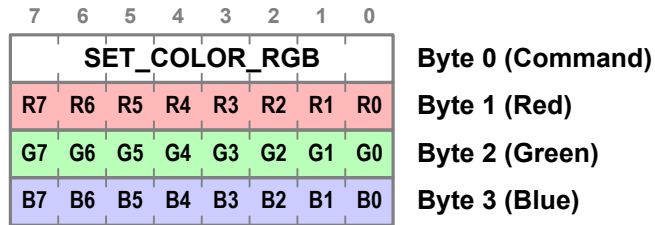
SET_BG_COLOR_RGB 32 hex
  Red              0
  Green            0
  Blue            80 hex
SET_COLOR_RGB     31 hex
  Red             FF hex
  Green           FF hex
  Blue            0
SELECT_FONT       2B hex
  0               0 dec
PRINT_STRING_BG   3D hex
'L'              4C hex
'C'              43 hex
'D'              44 hex
NULL             0 hex

```

## 8.78 SET\_COLOR\_RGB

**Description:** Sets the Current Color.

**Code:** 31hex, 49dec



See Also: [CLS](#), [PLOT](#)

### Example:

The following sequence will fill the whole screen with green.

SET\_COLOR\_RGB 31 hex

Red 0

Green FF hex

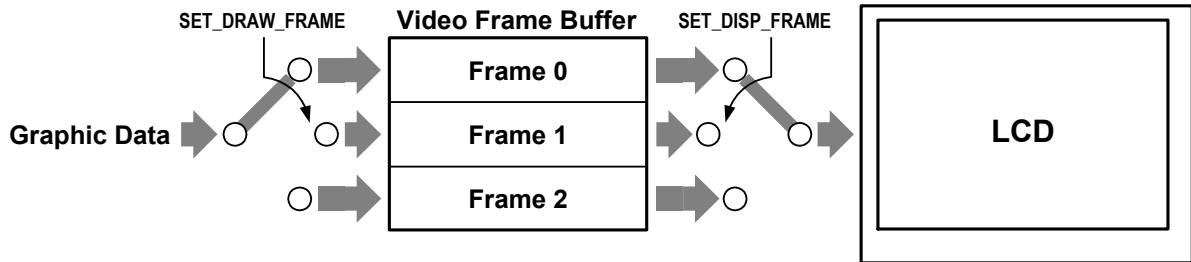
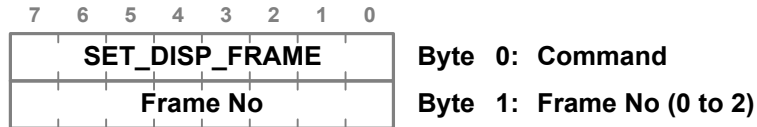
Blue 0

CLS 21 hex

## 8.79 SET\_DISP\_FRAME

**Description:** Sets the [Frame](#) to be displayed on the screen.

**Code:** 52<sub>hex</sub>, 82<sub>dec</sub>



Ref: [Frames](#)

See Also: [SET\\_DRAW\\_FRAME](#)

### Example:

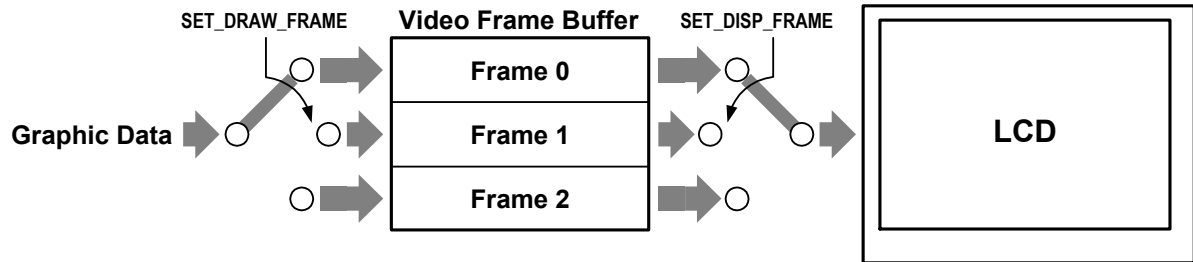
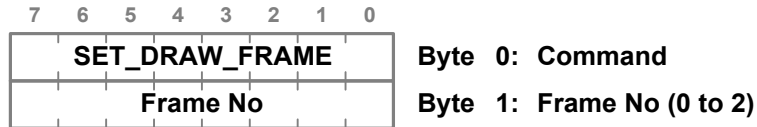
The following sequence will set the display to Frame 0.

```
SET_DISP_FRAME 52 hex
0              0 (Frame No)
```

## 8.80 SET\_DRAW\_FRAME

**Description:** Sets the [Frame](#) that ezLCD+ commands draw on.

**Code:** 51hex, 81dec



Ref: [Frames](#)

See Also: [SET\\_DISP\\_FRAME](#)

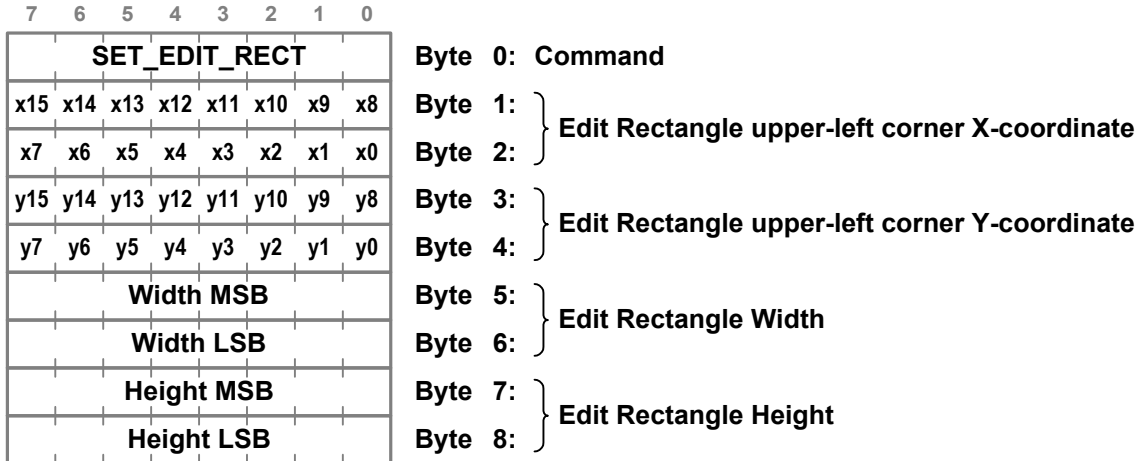
### Example:

The following sequence will set the drawing to Frame 1.

```
SET_DRAW_FRAME 51 hex
1              1 (Frame No)
```

### 8.81 SET\_EDIT\_RECT

**Description:** Sets the rectangle for editing (replacing colors, etc).  
**Code:** 5Chex, 92dec



See Also: [REPLACE\\_COLOR](#)

#### Example:

The following sequence will replace color red with green inside the rectangle size of 100x50 and positioned at (320, 240).

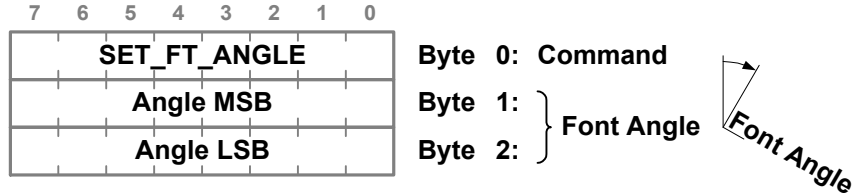
```

SET_EDIT_RECT 5C hex
  1          1 dec (X MSB)
 64         64 dec (X LSB)
  0          0 dec (Y MSB)
240        240 dec (Y LSB)
  0          0 dec (Width MSB)
100        100 dec (Width LSB)
  0          0 dec (Height MSB)
 50         50 dec (Height LSB)
REPLACE_COLOR 5D hex
Red        FF hex (Old color red component)
Green      0      (Old color green component)
Blue       0      (Old color blue component)
Red        0      (New color red component)
Green     FF hex (New color green component)
Blue       0      (New color blue component)
    
```

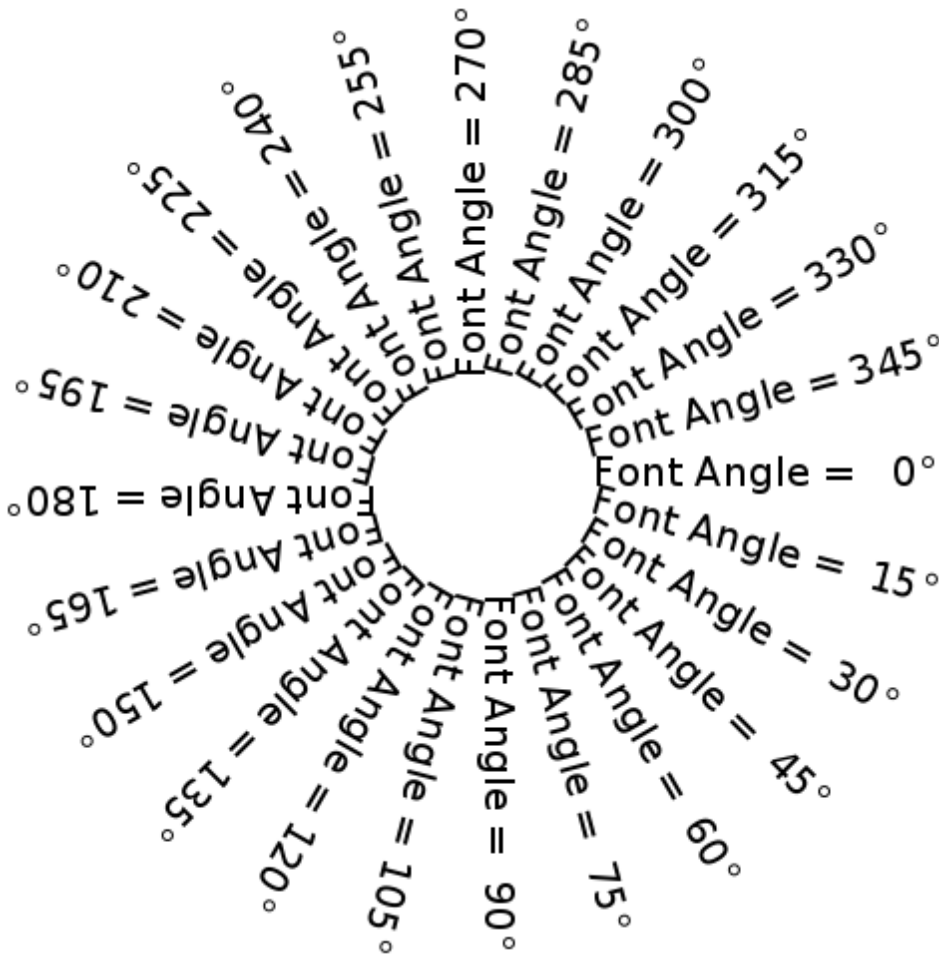
## 8.82 SET\_FT\_ANGLE

**Description:** Sets the angle, in which the [True Type Font](#) characters and strings are printed.

**Code:** **9A**hex, **154**dec



**See Also:** [TEXT\\_NORTH](#), [TEXT\\_SOUTH](#), [TEXT\\_WEST](#), [TEXT\\_EAST](#)



**Angle Coding:** The full angle (360°) is equal to 4000hex (16384dec).

To transform degrees to font angle units:

**Font\_Angle = Angle\_deg x 2048 / 45**

For example:

2048dec = 800hex = 45°

4096dec = 1000hex = 90°

8192dec = 2000hex = 180°

12288dec = 3000hex = 270°

---

$16384_{\text{dec}} = 4000_{\text{hex}} = 360^\circ = 0^\circ$

**Note:** SET\_FT\_ANGLE clears the font glyph cache. No cache is used after this command is issued. In order to re-enable cache use one of the following commands: [TEXT\\_NORTH](#), [TEXT\\_SOUTH](#), [TEXT\\_WEST](#), [TEXT\\_EAST](#)

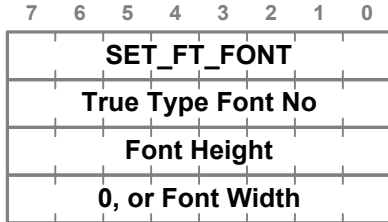
### Example:

The following sequence will print a black True Type string "Font Angle = 45°" with the angle of 45°.

```
SET_FT_FONT    91 hex
  0             0 dec (Font No = 0)
  24           24 dec (Height = 24)
  0             0      (Width = automatic)
SET_COLOR_RGB  31 hex
  Red          0
  Green        0
  Blue         0
SET_FT_FONT    9A hex
  08           08 hex (Font Angle MSB)
  00           00 hex (Font Angle LSB)
PRINT_STRING   2D hex
Font Angle = 45°
                0 (NULL)
```

### 8.83 SET\_FT\_FONT

**Description:** Selects and configures [True Type Font](#) from User ROM.  
**Code:** 91hex, 145dec



- Byte 0: Command**
- Byte 1: True Type Font No in User ROM (0 to 255)**
- Byte 2: Font Height in pixels**
- Byte 3: Zero** - Font Width is calculated automatically  
**Non-Zero** - Font Width in pixels

Since True Type (Free Type) fonts are scalable, their size has to be specified as shown above. If the Byte 3 is set to 0, the width of the selected font will be calculated automatically (recommended). Each time the True Type font (or it's size) is changed, the font cache is cleared.

**Note:** The following True Type fonts are installed in the User ROM, when ezLCD+ 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*

Ref: [Fonts](#), [True Type Fonts](#)

See Also: [SET\\_FT\\_UNIBASE](#), [PRINT\\_STRING](#), [PRINT\\_CHAR](#), [PRINT\\_FT\\_UNISTRING](#), [PRINT\\_FT\\_UNICHAR](#)

#### Example:

The following sequence will print a black True Type character 'M' using 24-pixel-height True Type Font 2.

```

SET_FT_FONT 91 hex
  2          2 dec (Font No = 2)
 24         24 dec (Height = 24)
  0          0   (Width = automatic)
SET_COLOR_RGB 31 hex
  Red        0
  Green      0
  Blue       0
PRINT_CHAR 2C hex
'M'        4D hex
    
```

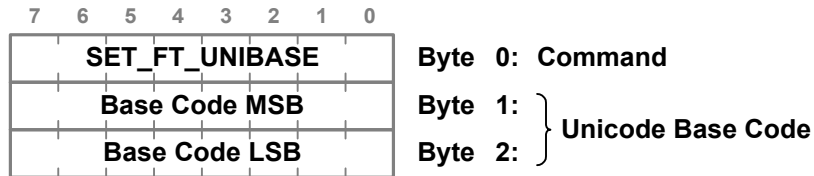


## 8.84 SET\_FT\_UNIBASE

**Description:** Sets the base code for the Unicode characters, so they can be printed using 8-bit codes.

Only the [True Type Fonts](#) are affected.

**Code:** 93hex, 147dec



The ezLCD+ uses 16 bits to access the Unicode characters. The value set by the SET\_FT\_UNIBASE command is added to the character code, so the Unicode characters can be printed using ASCII print commands, like [PRINT\\_STRING](#) and [PRINT\\_CHAR](#), or cached by [CACHE\\_FT\\_CHARS](#).

Unicode Character = Unicode Base + ASCII character.

For example, the pictures below show string "The quick brown fox jumps over a lazy dog", printed by the [PRINT\\_STRING](#) command, with True Type Font 0, using different Unicode Bases:

Unicode Base = 0

The quick brown fox jumps over a lazy dog

Unicode Base = 03CE hex

Тжгтпузбйтарнхмтднциулоствнфгртяткяшчтвне

The ezLCD+ can also print characters and strings using full 16 bit character coding. Please, refer to the commands: [PRINT\\_FT\\_UNICHAR](#) and [PRINT\\_FT\\_UNISTRING](#).

**Note:** Upon power-up, the Unicode Base is set to 0.

**See Also:** [SET\\_FT\\_FONT](#), [PRINT\\_STRING](#), [PRINT\\_CHAR](#), [CACHE\\_FT\\_CHARS](#), [PRINT\\_FT\\_UNISTRING](#), [PRINT\\_FT\\_UNICHAR](#)

### Example:

The picture above (Unicode Base = 03CE hex), has been generated using the following sequence:

```

SET_FT_FONT      91 hex
  0              0 dec (Font No = 0)
 24             24 dec (Height = 24)
  0             0      (Width = automatic)
SET_FT_UNIBASE  93 hex
  03            03 hex (Unicode Base MSB)
  CE           CE hex (Unicode Base LSB)
PRINT_STRING    2D hex
The quick brown fox jumps over a lazy dog
0 (NULL)

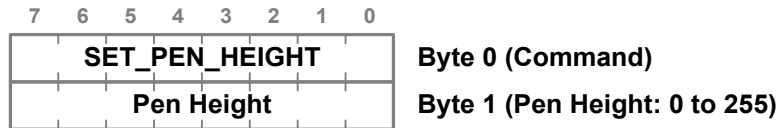
```



## 8.85 SET\_PEN\_HEIGHT

**Description:** Sets height of the drawing [Pen](#)  
This command affects only the drawing of curves

**Code:** 82hex, 130dec



- Pen Width
  - Specifies the horizontal dimension the drawing line (in pixels)
  - Set by command: SET\_PEN\_SIZE
- Pen Height
  - Specifies the vertical dimension of Pen (in pixels), when drawing curves
  - Ignored when drawing straight lines
  - Set by commands: SET\_PEN\_SIZE and SET\_PEN\_HEIGHT

**Notes:** Straight lines are not drawn when Pen Width is set to 0.  
Curves are not drawn when either Pen Width or Height is set to 0.  
SET\_PEN\_SIZE sets both: width and height of the Pen  
SET\_PEN\_HEIGHT sets only the height of the Pen

**See Also:** [SET\\_PEN\\_SIZE](#)

### Example:

The figure below:



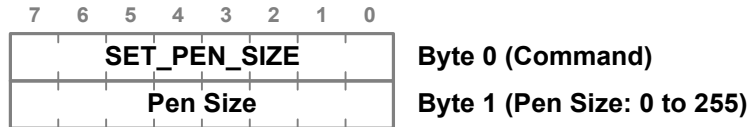
will be drawn by the following sequence:

```
SET_PEN_SIZE      81 hex
  size           40 dec
SET_PEN_HEIGHT   82 hex
  height         4 dec
ELLIPSE_AHBH     8A hex
  0              0 dec (a MSB)
100              100 dec (a LSB)
  0              0 dec (b MSB)
  60             60 dec (b LSB)
```

## 8.86 SET\_PEN\_SIZE

**Description:** Sets the size of drawing [Pen](#) (both width and height)

**Code:** 81hex, 129dec



- Pen Width
  - Specifies the horizontal dimension the drawing line (in pixels)
  - Set by command: SET\_PEN\_SIZE
- Pen Height
  - Specifies the vertical dimension of Pen (in pixels), when drawing curves
  - Ignored when drawing straight lines
  - Set by commands: SET\_PEN\_SIZE and SET\_PEN\_HEIGHT

**Notes:** Straight lines are not drawn when Pen Width is set to 0.  
 Curves are not drawn when either Pen Width or Height is set to 0.  
 SET\_PEN\_SIZE sets both: width and height of the Pen  
 SET\_PEN\_HEIGHT sets only the height of the Pen

**See Also:** [SET\\_PEN\\_HEIGHT](#)

### Example:

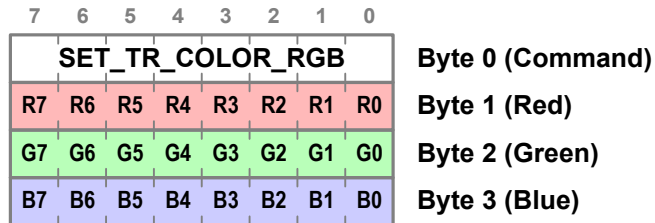
The following sequence will draw 8 pixel thick horizontal line:

```
SET_PEN_SIZE  81 hex
size          8 dec
H_LINEH      A0 hex
0             0 dec (x MSB)
125          125 dec (x LSB)
```

## 8.87 SET\_TR\_COLOR\_RGB

**Description:** Specifies the color, which is ignored during [Bitmap](#) drawing. Only [Bitmaps](#) are affected.

**Code:** **5A**hex, **90**dec



Use this command to define the transparent color of bitmaps. The transparent color can be set to none by TR\_COLOR\_NONE command. After the power-up the transparent color is set to none.

**See Also:** [TR\\_COLOR\\_NONE](#)

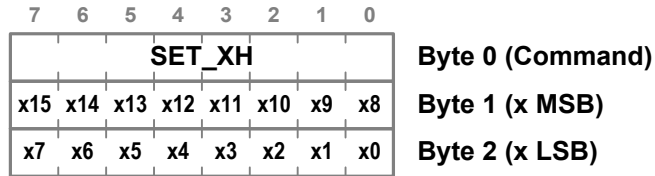
### Example:

The following sequence will set the bitmap transparent color to white.

```
SET_TR_COLOR_RGB 5A hex
Red              FF hex
Green            FF hex
Blue             FF hex
```

## 8.88 SET\_XH

**Description:** Sets only the X-coordinate of the Current Position. Y coordinate remains unchanged  
**Code:** 6Ehex, 110dec



Ref: [Screen Coordinates](#)

See Also: [SET\\_YH](#), [SET\\_XHYH](#)

### Example:

The following sequence will put a 2 blue points in the same row.

```
SET_COLOR_RGB 31 hex
Red           0
Green        0
Blue        FF hex
SET_XH       6E hex
0            0 dec (x MSB)
160         160 dec (x LSB)
PLOT        26 hex
SET_XH       6E hex
0            0 dec (x MSB)
170         170 dec (x LSB)
PLOT        26 hex
```

## 8.89 SET\_XHYH

**Description:** Sets the Current Position.

**Code:** 33hex, 51dec

7	6	5	4	3	2	1	0	
<b>SET_XHYH</b>								<b>Byte 0 (Command)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x MSB)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x LSB)</b>
y15	y14	y13	y12	y11	y10	y9	y8	<b>Byte 3 (y MSB)</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 4 (y LSB)</b>

Ref: [Screen Coordinates](#)

See Also: [PLOT](#), [LINE\\_TO\\_XHYH](#), [CIRCLE\\_RH](#)

### Example:

The following sequence will draw a red vertical line from (95, 10) to (95, 110).

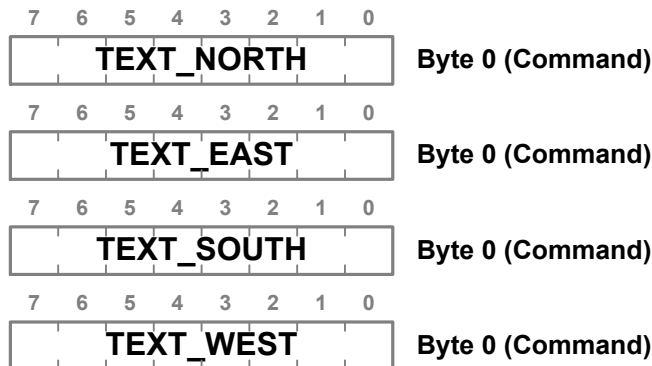
```
SET_COLOR_RGB 31 hex
Red           FF hex
Green        0
Blue         0
SET_XHYH     33 hex
0            0 dec (x MSB)
95          95 dec (x LSB)
0            0 dec (y MSB)
10          10 dec (y LSB)
V_LINEH     A1 hex
0            0 dec (y MSB)
110         110 dec (y LSB)
```



## 8.91 TEXT\_NORTH, SOUTH, EAST, WEST

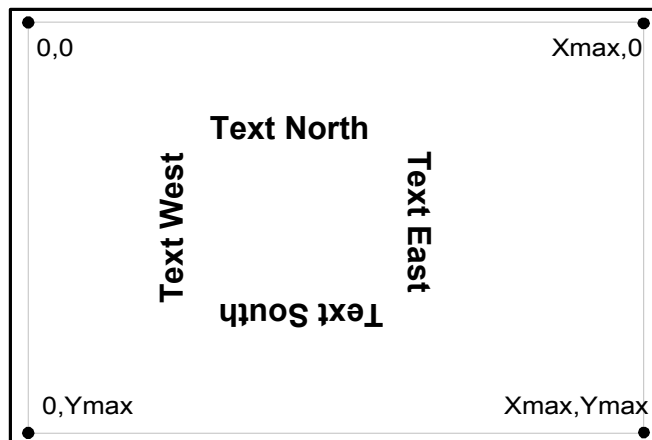
**Description:** Set the orientation of the text, as shown on the picture below.

**Code:**  
**TEXT\_NORTH:** 60hex, 96dec  
**TEXT\_EAST :** 61hex, 97dec  
**TEXT\_SOUTH:** 62hex, 98dec  
**TEXT\_WEST :** 63hex, 99dec



For [TrueType](#) fonts the difference between these commands and the [SET\\_FT\\_ANGLE](#) command, is that these commands take advantage of the font cache, while the SET\_FT\_ANGLE command does not. In fact, the SET\_FT\_ANGLE command disables the cache, which than can be re-enabled by issuing one of the commands described on this page.

**Note:** TEXT\_NORTH is the default text orientation



See Also: [PRINT\\_CHAR](#), [PRINT\\_STRING](#), [SET\\_FT\\_ANGLE](#)

### Example:

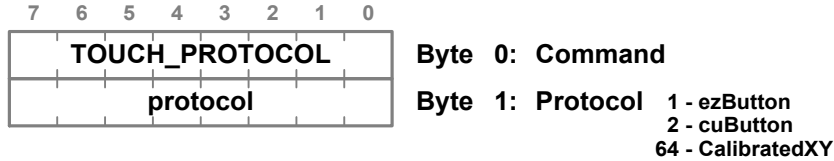
The following sequence will print a text pattern similar to the one shown on the picture above.

```
SET_XHYH      33 hex
0             0 dec (x MSB)
60           60 dec (x LSB)
0             0 dec (y MSB)
```

```
10          10 dec (y LSB)
SELECT_FONT 2B hex
0           0 dec
TEXT_NORTH 60 hex
PRINT_STRING 2D hex
"Text North "
NULL        0 hex (NULL)
TEXT_EAST 61 hex
PRINT_STRING 2D hex
" Text East "
NULL        0 hex (NULL)
TEXT_SOUTH 62 hex
PRINT_STRING 2D hex
" Text South "
NULL        0 hex (NULL)
TEXT_WEST 63 hex
PRINT_STRING 2D hex
" Text West "
NULL        0 hex (NULL)
```

## 8.92 TOUCH\_PROTOCOL

**Description:** Changes the default behavior of the ezLCD+ touch control function  
**Code:** **B2**<sub>hex</sub>, **178**<sub>dec</sub>



### About the Touch Protocols:

Currently, the following touch protocols are implemented:

1. [ezButton](#)
  - Touch screen buttons can be defined [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.
  - ezLCD+ sends Button Down and Button Up events for the buttons defined by the [BUTTON\\_DEF](#) or [BUTTON\\_DEF\\_LONG](#) command.
  - Easy protocol. Button IDs and events are coded in 1 byte.
  - Events are sent only once per button state change.
2. [cuButton](#)
  - Similar to the ezButton, however the button states are sent continuously, 5 to 20 times per second.
3. [CalibratedXY](#)
  - ezLCD+ sends [TOUCH\\_X](#) and [TOUCH\\_Y](#) packets (X and Y coordinates), when the screen is pressed
  - ezLCD+ sends [PEN\\_UP](#) packets when the touch screen is not pressed.
  - Multi-byte packed oriented protocol.
  - Packets are sent continuously, 5 to 50 times per second.

**Note:** Upon the Power-Up the ezLCD+ does not send any touch screen data until the proper protocol is selected.

**See Also:** [BUTTON\\_DEF](#), [BUTTON\\_DEF\\_LONG](#), [BUTTON\\_STATE](#), [BUTTONS\\_ALL\\_UP](#), [BUTTONS\\_DELETE\\_ALL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Screen Operations](#)
  - [ezButton](#)
  - [cuButton](#)
  - [CalibratedXY](#)

### Example:

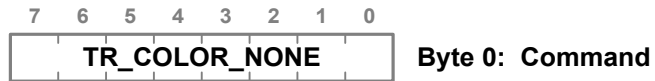
The following sequence will change the Touch Protocol to ezButton.

```
TOUCH_PROTOCOL B2 hex (Command)
                1      1 dec (ezButton Protocol)
```

### 8.93 TR\_COLOR\_NONE

**Description:** Sets the [transparent color](#) of bitmaps to none.  
Only [Bitmaps](#) are affected.

**Code:** **5B**hex, **91**dec



Use this command to define the transparent color of bitmaps. The transparent color can be set to none by TR\_COLOR\_NONE command. After the power-up the transparent color is set to none.

**See Also:** [SET\\_TR\\_COLOR\\_RGB](#)

#### **Example:**

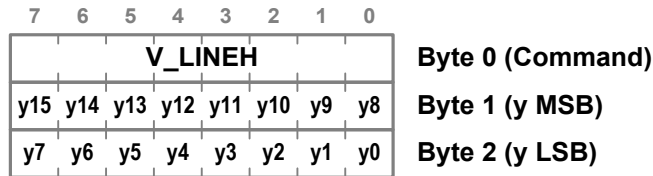
The following sequence will set the bitmap transparent color to none.

TR\_COLOR\_NONE 5B hex

## 8.94 V\_LINEH

**Description:** Quickly draws a vertical line from Current Position, to the row specified by the parameter.

**Code:** **A1**hex, **161**dec



Ref: [Screen Coordinates](#)

See Also: [H\\_LINEH](#), [SET\\_XHYH](#)

### Example:

The following sequence will draw a blue vertical line from (95, 10) to (95, 110).

```
SET_COLOR_RGB 31 hex
Red           0
Green        0
Blue        FF hex
SET_XHYH     33 hex
0             0 dec (x MSB)
95           95 dec (x LSB)
0             0 dec (y MSB)
10           10 dec (y LSB)
V_LINEH      A1 hex
0             0 dec (y MSB)
110          110 dec (y LSB)
```

## 8.95 Legacy Commands

The ezLCD+ is backwards compatible with the command sets used by the previous ezLCD products:

- ezLCD-001 - 240x160 pixels, 256 colors
- ezLCD-002 - 240x160 pixels, 256 colors, 1MB User Flash, Touch Screen
- ezLCD-003 - 240x160 pixels, 256 colors, 1MB User Flash
- ezLCD-004 - 320x240 pixels 65536 colors, 1MB User Flash, SD Card, Touch Screen

The previous ezLCD products, have a smaller resolution and color number than ezLCD+. The table below shows the number of bytes needed to specify pixel color and coordinates for particular ezLCD products:

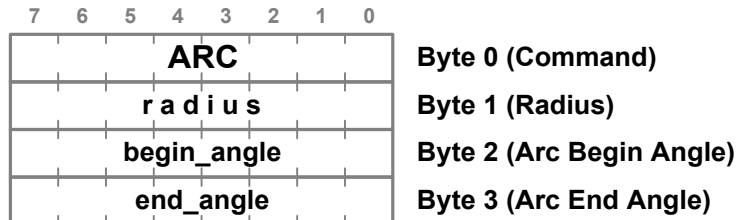
	ezLCD+	ezLCD-004	ezLCD-003	ezLCD-002	ezLCD-001
<b>No of Bytes per Color</b>	3	2	1	1	1
<b>No of Bytes per Column (X)</b>	2	2	1	1	1
<b>No of Bytes per Row (Y)</b>	2	1	1	1	1

The commands described in this chapter have the smaller number of color and coordinate bytes than the number needed for the ezLCD+. In order to accept these commands, ezLCD+ emulates the previous ezLCD products.

## 8.95.1 ARC

**Description:** Draws an Arc in Current Color, with the center at Current Position, starting on Begin Angle and ending on the End Angle.

**Code:** 2Fhex, 47dec



**See Also:** [SET\\_XY](#), [SET\\_COLOR](#), [CIRCLE\\_R](#)

**Angle Coding:** The angle range is from 0 to 255.

To transform degrees to ARC angle units:

$\text{Angle\_lcd} = \text{Angle\_deg} \times 32 / 45$

For example:

32 = 45°

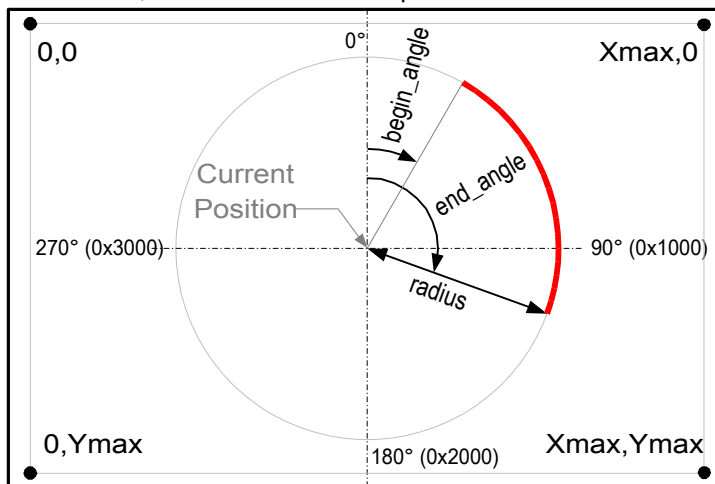
64 = 90°

128 = 180°

192 = 270°

0 = 0° = 360°

The angle is drawn clockwise with the zero positioned at the top of a screen, as it is shown on the picture below

**Example:**

The following sequence will draw a green arc from 45 to 225 degrees with the center positioned in the middle of a screen.

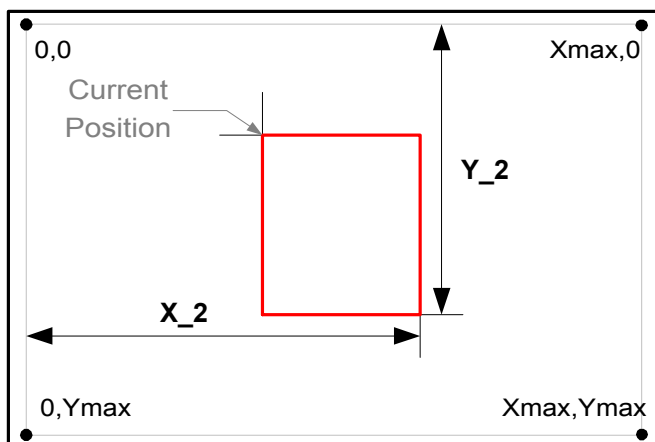
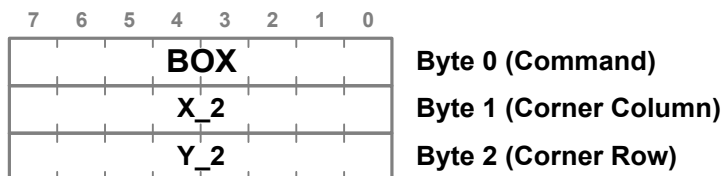
```
SET_COLOR      24 hex
GREEN          00111000 bin
SET_XY        25 hex
```

```
120          120 dec
  80          80 dec
ARC        2F hex
  60          60 dec (radius)
  32          32 dec (begin_angle = 45 degrees)
 160         160 dec (end_angle = 225 degrees)
\
```

## 8.95.2 BOX

**Description:** Draws a rectangle.

**Code:** 42hex, 66dec



See Also: [SET\\_XY](#), [BOX\\_FILL](#)

### Example:

The following sequence will draw the red rectangle

```

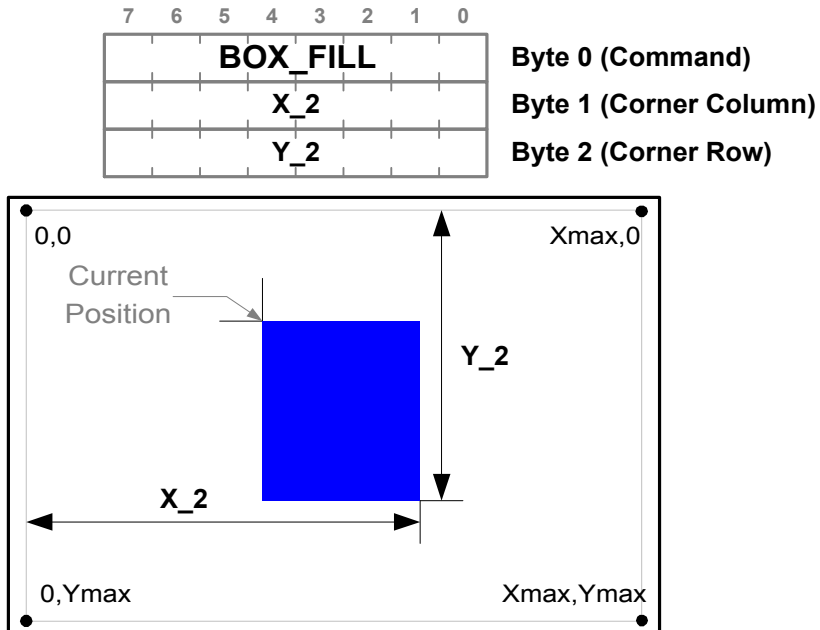
SET_COLOR  24 hex
RED        0000111 bin
SET_XY     25 hex
           95      95 dec
           40      10 dec
BOX        42 hex
180        180 dec (X_2)
120        120 dec (Y_2)

```

## 8.95.3 BOX\_FILL

**Description:** Draws a rectangle filled with Current Color

**Code:** 43hex, 67dec



See Also: [SET\\_XY](#), [BOX](#)

### Example:

The following sequence will draw the rectangle filled with blue color

```

SET_COLOR    24 hex
RED          11000000 bin
SET_XY       25 hex
             95      95 dec
             40      10 dec
BOX_FILL     43 hex
180          180 dec (X_2)
120          120 dec (Y_2)

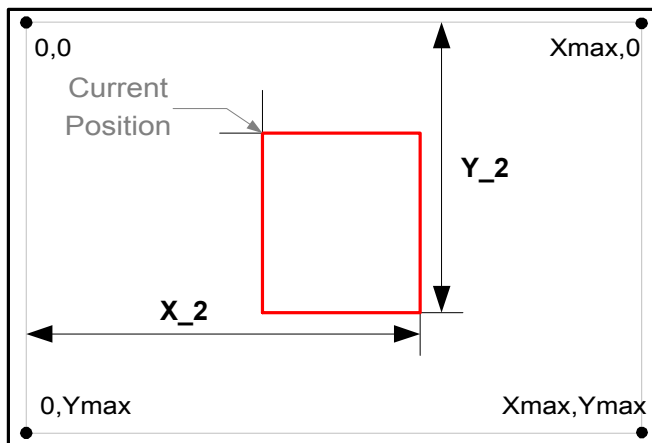
```

## 8.95.4 BOXH

**Description:** Draws a rectangle.

**Code:** **A2**hex, **162**dec

7	6	5	4	3	2	1	0	
<b>BOXH</b>								<b>Byte 0 (Command)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x2 MSB)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x2 LSB)</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 3 (y2)</b>



See Also: [SET\\_XHY](#), [BOXH\\_FILL](#)

### Example:

The following sequence will draw a red rectangle with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

```

SET_COLORH 84 hex
RED_LSB    00000000 bin
RED_MSB    11111000 bin
SET_XHY    85 hex
           0      0 dec (x MSB)
           95    95 dec (x LSB)
           10    10 dec (y)
BOXH       A2 hex
           0      0 dec (X_2 MSB)
          180    180 dec (X_2 LSB)
          120    120 dec (Y_2)

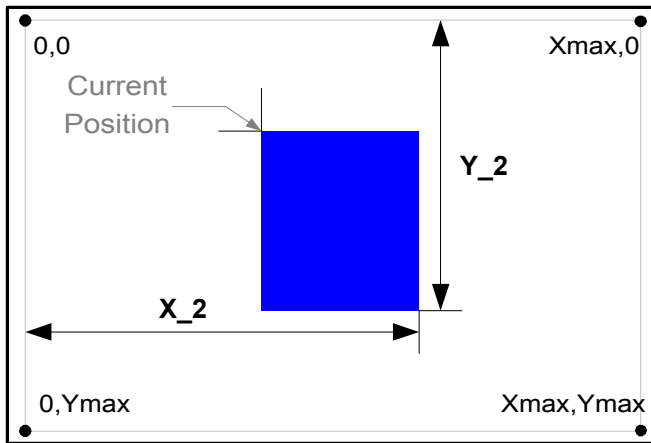
```

## 8.95.5 BOXH\_FILL

**Description:** Draws a rectangle filled with Current Color.

**Code:** **A3**hex, **163**dec

	7	6	5	4	3	2	1	0	
	<b>BOXH_FILL</b>								
	x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x2 MSB)</b>
	x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x2 LSB)</b>
	y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 3 (y2)</b>



See Also: [SET\\_XHY](#), [BOXH](#)

### Example:

The following sequence will draw a blue filled rectangle, with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

```

SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_XHY    85 hex
           0      0 dec (x MSB)
           95     95 dec (x LSB)
           10     10 dec (y)
BOXH_FILL  A3 hex
           0      0 dec (X_2 MSB)
          180    180 dec (X_2 LSB)
          120    120 dec (Y_2)

```

8.95.6 **BUTTON\_DEF**

**Description:** Defines and draws a touch button

**Code:** **B0**hex, **176**dec

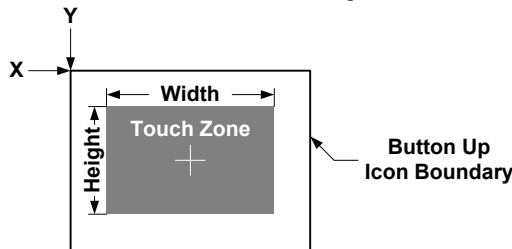
7 6 5 4 3 2 1 0	<b>BUTTON_DEF</b>	<b>Byte 0: Command</b>
	button_no	<b>Byte 1: Button No (0 to 63)</b>
	state	<b>Byte 2: Initial State (1: Up, 2: Down, 3: Disabled, 4: Non-Visible)</b>
	button_up_icon	<b>Byte 3: Icon No in User ROM for button Up (255 = none)</b>
	button_down_icon	<b>Byte 4: Icon No in User ROM for button Down (255 = none)</b>
	button_disabled_icon	<b>Byte 5: Icon No in User ROM for button Disabled (255 = none)</b>
x15 x14 x13 x12 x11 x10 x9 x8		<b>Byte 6: Button upper-left corner X-coordinate MSB</b>
x7 x6 x5 x4 x3 x2 x1 x0		<b>Byte 7: Button upper-left corner X-coordinate LSB</b>
y7 y6 y5 y4 y3 y2 y1 y0		<b>Byte 8: Button upper-left corner Y-coordinate</b>
	touch_zone_width	<b>Byte 9: Touch Zone width</b>
	touch_zone_height	<b>Byte 10: Touch Zone height</b>

**About the Touch Zone:**

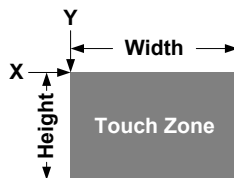
Touch Zone is the active touch response area of the button. It is specified by **With** (Byte 9) and **Height** (Byte 10).

- If the Button Up Icon is defined (Byte 3 is not 255), the Touch Zone is centered on it.
- If the Button Up Icon is none (Byte 3 = 255), the position of the upper-left corner of the Touch Zone is specified by **X** (Bytes: 6 and 7) and **Y** (Byte 8).

Both cases are shown on the drawings below:



Button Up Icon is defined (Byte 3 is not 255)



Button Up Icon is none (Byte 3 = 255)

**See Also:** [BUTTON\\_STATE](#), [BUTTONS\\_ALL\\_UP](#), [BUTTONS\\_DELETE\\_ALL](#), [TOUCH\\_PROTOCOL](#)

**Important:** Before using this command, please read the following chapters:

- [Touch Screen Operations](#)
- [ezButton](#)

- [cuButton](#)

### Example:

The following sequence will define the Button No. 4 with the following bitmaps:

- Button Up Icon in User ROM: 8
- Button Down Icon in User ROM: 9
- No Icon for Button Disabled state

The button will be positioned at X = 260 and Y = 170.

It's Touch Zone will have the width of 40 and the height of 30.

The button will be initially drawn using Button Up icon.

```

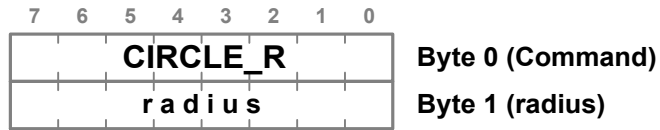
BUTTON_DEF  B0 hex (Command)
  4          4 dec (Button No)
  1          1 dec (Initial State: Button Up)
  8          8 dec (Button Up Icon No. in User ROM)
  9          9 dec (Button Down Icon No. in User ROM)
 255        255 dec (No Icon for Button Disabled)
  1          1 dec (Upper-left corner X MSB)
  4          4 dec (Upper-left corner X LSB)
 170        170 dec (Upper-left corner Y)
  40         40 dec (Width of the Touch Zone)
  30         30 dec (Height of the Touch Zone)

```

## 8.95.7 CIRCLE\_R

**Description:** Draws a circle in Current Color at Current Position

**Code:** 29hex, 41dec



See Also: [SET\\_XY](#), [SET\\_COLOR](#)

### Example:

The following sequence will draw a green circle in the middle of the screen.

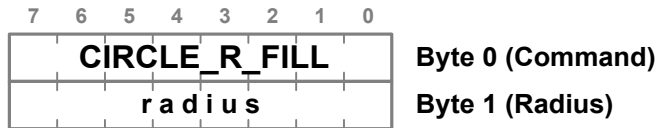
```

SET_COLOR  24 hex
GREEN      00111000 bin
SET_XY     25 hex
120        120 dec
80         80 dec
CIRCLE_R   29 hex
60         60 dec

```

## 8.95.8 CIRCLE\_R\_FILL

**Description:** Draws a circle in Current Color at Current Position, filled with Current Color  
**Code:** 39hex, 57dec



See Also: [SET\\_XY](#), [SET\\_COLOR](#)

### Example:

The following sequence will draw a red filled circle in the middle of the screen.

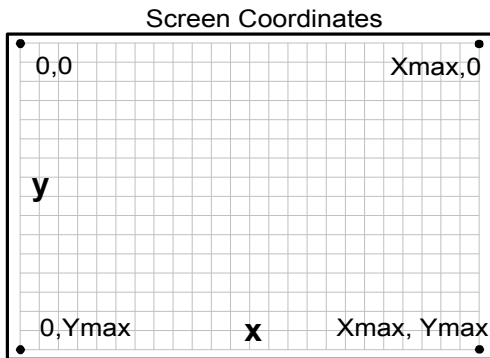
```
\SET_COLOR      24 hex
RED             00000111 bin
SET_XY         25 hex
120            120 dec
80             80 dec
CIRCLE_R_FILL  39 hex
60            60 dec
```

8.95.9 LINE\_TO\_XHY

**Description:** Draws a line in Current Color, from Current Position to the specified position.

**Code:** 88hex, 136dec

	7	6	5	4	3	2	1	0	
	<b>LINE_TO_XHY</b>								
	x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 0 (Command)</b>
	x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 1 (x MSB)</b>
	y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 2 (x LSB)</b>
									<b>Byte 3 (y)</b>



See Also: [SET\\_XHY](#), [SET\\_COLORH](#), [PLOT](#)

**Example:**

The following sequence will draw a red line across the screen.

```

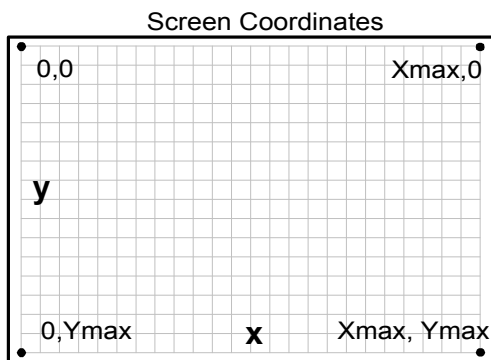
SET_COLORH      84 hex
RED_LSB         00000000 bin
RED_MSB         11111000 bin
SET_XHY         85 hex
0               0 dec (x0 MSB)
0               0 dec (x0 LSB)
0               0 dec (y0)
LINE_TO_XHY     88 hex
1               1 dec (x1 MSB)
63              63 dec (x1 LSB)
233             233 dec (y1)
    
```

## 8.95.10 LINE\_TO\_XY

**Description:** Draws a line in Current Color, from the Current Position to the to specified position

**Code:** 28hex, 40dec

	7	6	5	4	3	2	1	0	
	<b>LINE_TO_XY</b>								
	x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 1 (x)</b>
	y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 2 (y)</b>



**See Also:** [SET\\_XY](#), [SET\\_COLOR](#), [PLOT](#)

### Example:

The following sequence will draw a red line across the screen.

```

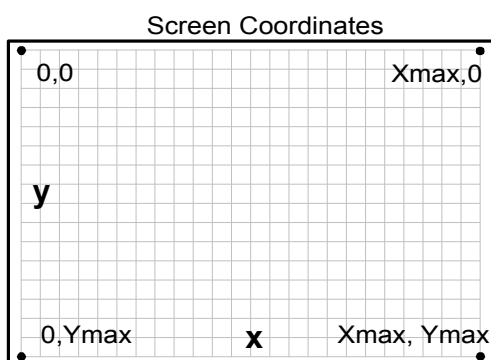
SET_COLOR      24 hex
RED            00000111 bin
SET_XY        25 hex
0              0 dec
0              0 dec
LINE_TO_XY    28 hex
239           239 dec
159           159 dec

```

## 8.95.11 PLOT\_XHY

**Description:** Plots a point in Current Color at the specified position.  
**Code:** 87hex, 135dec

7	6	5	4	3	2	1	0	
<b>PLOT_XHY</b>								<b>Byte 0 (Command)</b>
x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 1 (x MSB)</b>
x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 2 (x LSB)</b>
y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 3 (y)</b>



See Also: [SET\\_XHY](#), [SET\\_COLORH](#), [PLOT](#)

### Example:

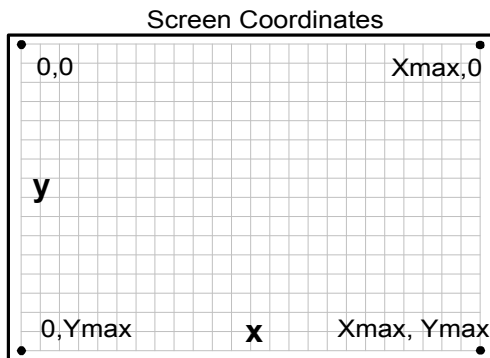
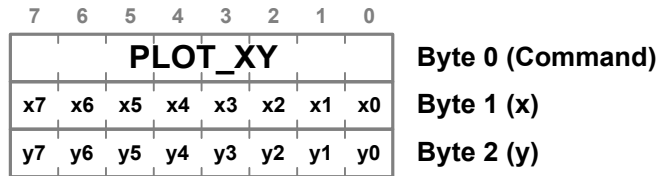
The following sequence will put a red point at (310, 117).

```
SET_COLORH 84 hex
RED_LSB    00000000 bin
RED_MSB    11111000 bin
PLOT_XHY   87 hex
  1         1 dec (x MSB)
  60        60 dec (x LSB 1*256+54=310)
 117       117 dec (y)
```

## 8.95.12 PLOT\_XY

**Description:** Plots a point in Current Color, at specified position.

**Code:** 27hex, 39dec



See Also: [SET\\_XY](#), [SET\\_COLOR](#), [PLOT](#)

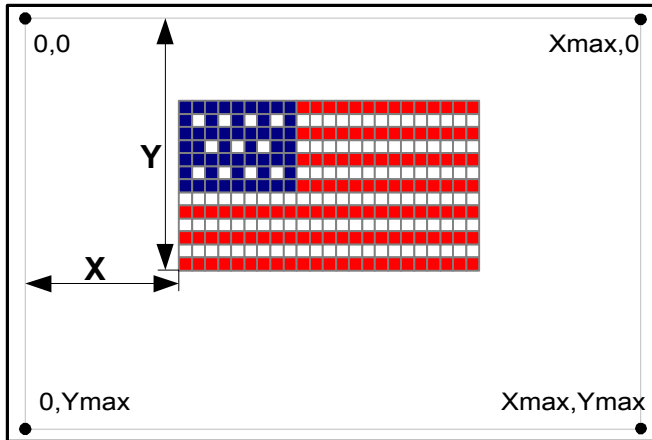
### Example:

The following sequence will put the red point in the middle of the screen.

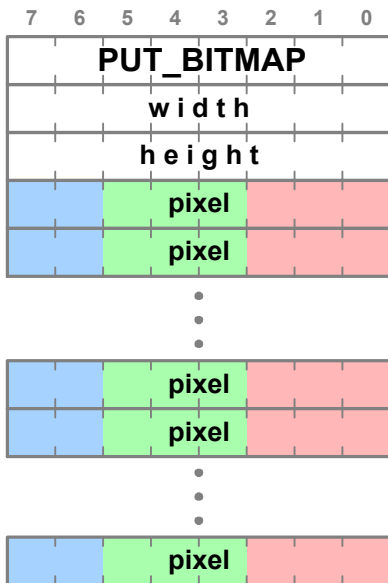
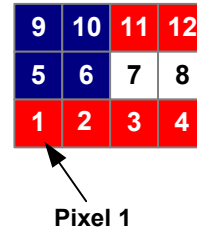
```
SET_COLOR 24 hex
RED 0000111 bin
PLOT_XY 27 hex
120 120 dec
80 80 dec
```

8.95.13 PUT\_BITMAP

**Description:** Puts Bitmap on the screen starting at Current Position, then UP and RIGHT  
**Code:** 2Ehex, 46dec



Example of the order of the pixels in case of the 4x3 bitmap.



Byte 0 (Command)  
 Byte 1 (Bitmap Width)  
 Byte 2 (Bitmap Height)  
 Byte 3 (pixel at: X, Y)  
 Byte 4 (pixel at: X+1, Y)  
 ...  
 Byte width+2 (pixel at: X+width-1, Y)  
 Byte width+3 (pixel at: X, Y-1)  
 ...  
 Byte height x width + 2 (pixel at: X+width-1, Y-height+1)

**Note:** The total number of bytes is: width x height + 3

See Also: [SET\\_XY](#), [SET\\_COLOR](#)

**Example:**

The following sequence will put 4x3 bitmap at x = 60, y = 80

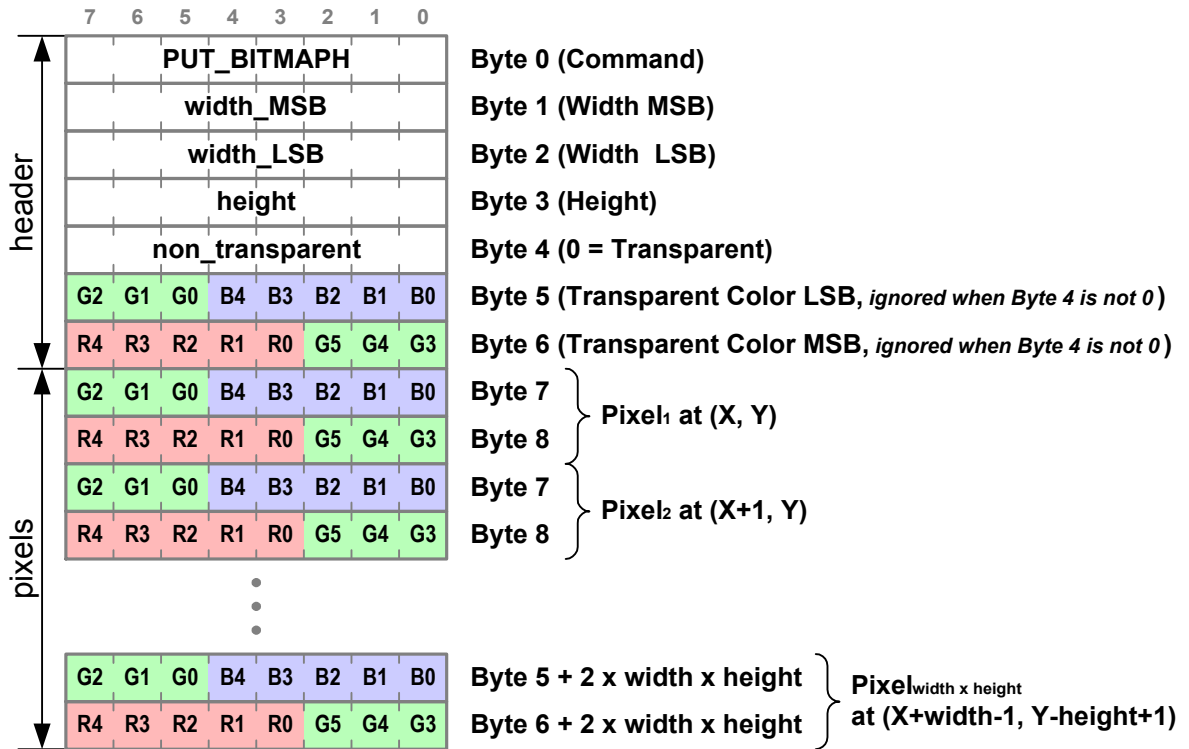
```
SET_XY      25 hex
x           60 dec
y           80 dec
PUT_BITMAP  2E hex  -----+--
width      4 dec   |
height     3 dec   |
```

```
pixel (x = 60, y = 80)
pixel (x = 61, y = 80)
pixel (x = 62, y = 80)
pixel (x = 63, y = 80)
pixel (x = 60, y = 79)
pixel (x = 61, y = 79)
pixel (x = 62, y = 79)
pixel (x = 63, y = 79)
pixel (x = 60, y = 78)
pixel (x = 61, y = 78)
pixel (x = 62, y = 78)
pixel (x = 63, y = 78)
-----+
```

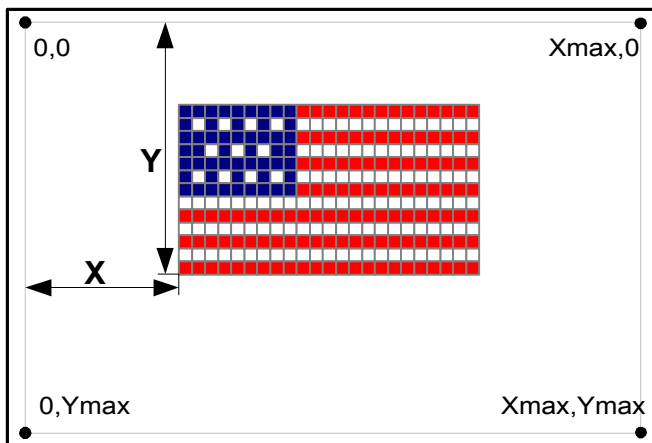
TOTAL:  
4 x 3 + 3 = 15 bytes

8.95.14 PUT\_BITMAP

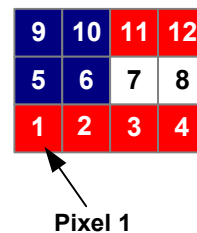
**Description:** Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP  
**Code:** 9Ehex, 158dec



- Notes:**
1. The total number of bytes is:  $2 \times \text{width} \times \text{height} + 7$
  2. When Byte 4 = 0, Bytes 5 and 6 specify the Transparent Color. Pixels equal to the Transparent Color are ignored during bitmap drawing. All pixels are drawn when Byte 4 is not 0.



Example of the order of the pixels in case of the 4x3 bitmap.

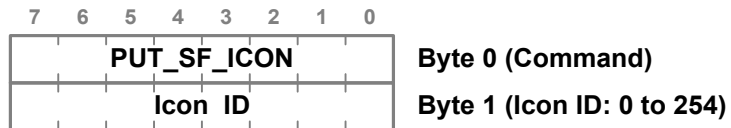
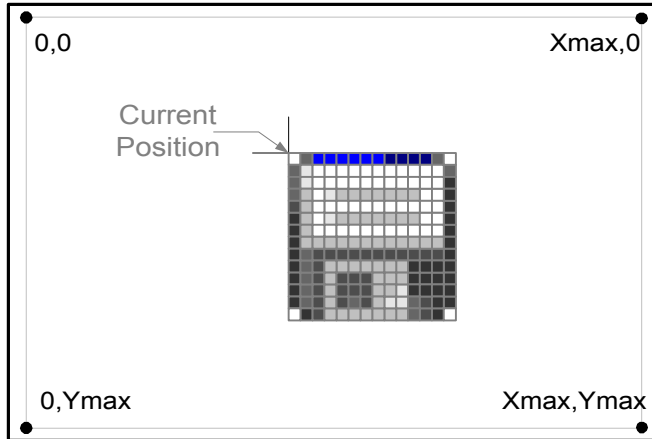


See Also: [SET\\_XHY](#), [SET\\_COLORH](#)

## 8.95.15 PUT\_SF\_ICON

**Description:** Displays an icon with its upper-left corner positioned at the Current Position. The icon is read from the ezLCD+ Serial Flash. Use the ezLCDflash utility to store icons in the ezLCD+ Serial Flash.

**Code:** 58hex, 88dec



**Note:** Maximum number of icons is 255 (IDs 0 to 254)

See Also: [SET\\_XHY](#)

### Example:

The following sequence will display Icon No. 3 with its upper-left corner positioned at (60, 43).

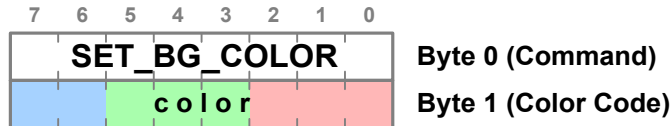
```
SET_XHY      85 hex
  0           0 dec (x MSB)
 60          60 dec (x LSB)
 43          43 dec (y)
PUT_SF_ICON  58 hex
  3           3 dec
```

## 8.95.16 SET\_BG\_COLOR

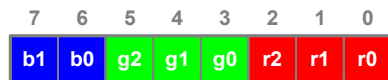
**Description:** Sets the Background Color for the following instructions:

[PRINT\\_CHAR\\_BG](#)  
[PRINT\\_STRING\\_BG](#)

**Code:** 34hex, 52dec



**Note:** The 256 color palette has the following color coding:



See Also: [PRINT\\_CHAR\\_BG](#), [PRINT\\_STRING\\_BG](#)

### Example:

The following sequence print Yellow "LCD" on the Navy background, in the middle of a screen, using font no 0.

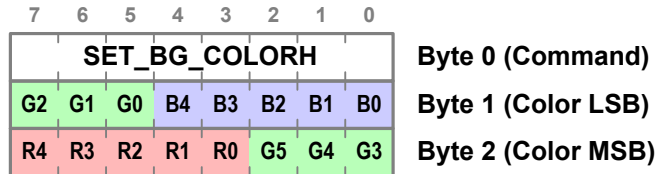
```
SET_BG_COLOR      34 hex
NAVY              10000000 bin
SET_COLOR        24 hex
YELLOW           00111111 bin
SET_XY           25 hex
120              120 dec
80               80 dec
SELECT_FONT      2B hex
0                0 dec
PRINT_STRING_BG  3D hex
'L'              4C hex
'C'              43 hex
'D'              44 hex
NULL             0 hex
```

## 8.95.17 SET\_BG\_COLORH

**Description:** Sets the Background Color for the following instructions:

[PRINT\\_CHAR\\_BG](#)  
[PRINT\\_STRING\\_BG](#)

**Code:** 94hex, 148dec



**See Also:** [PRINT\\_CHAR\\_BG](#), [PRINT\\_STRING\\_BG](#)

### Example:

The following sequence will print "LCD" in yellow on a navy background, using Font 0.

```

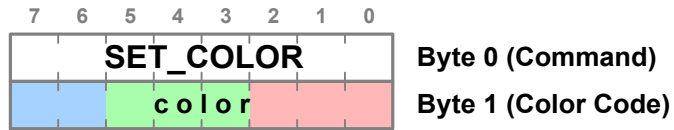
SET_BG_COLORH      94 hex
NAVY_LSB           00010000 bin
NAVY_MSB           00000000 bin
SET_COLORH        84 hex
YELLOW_LSB        11100000 bin
YELLOW_MSB        11111111 bin
SET_XHY           85 hex
0                  0 dec (x MSB)
160                160 dec (x LSB)
117                117 dec (y)
SELECT_FONT       2B hex
0                  0 dec
PRINT_STRING_BG   3D hex
'L'               4C hex
'C'               43 hex
'D'               44 hex
NULL              0 hex

```

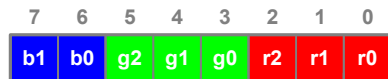
## 8.95.18 SET\_COLOR

**Description:** Sets the Current Color

**Code:** 24hex, 36dec



**Note:** The 256 color palette has the following color coding:



See Also: [CLS](#), [PLOT](#)

### Example:

The following sequence will fill the whole display with green

SET\_COLOR 24 hex

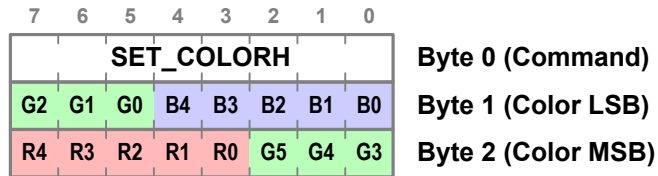
GREEN 00111000 bin

CLS 21 hex

## 8.95.19 SET\_COLORH

**Description:** Sets the Current Color.

**Code:** 84hex, 132dec



See Also: [CLS](#), [PLOT](#)

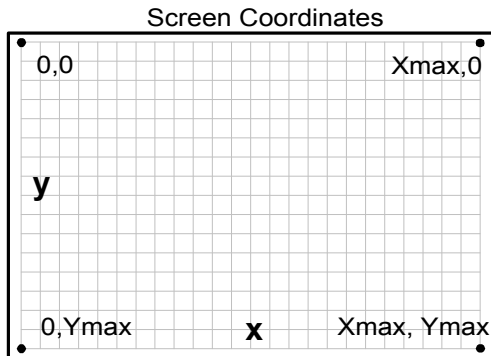
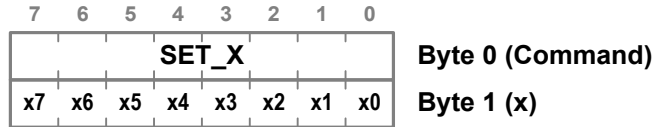
### Example:

The following sequence will fill the whole screen with green.

```
SET_COLORH 84 hex
GREEN_LSB 11100000 bin
GREEN_MSB 0000111 bin
CLS 21 hex
```

## 8.95.20 SET\_X

**Description:** Sets only the X-coordinate of the Current Position. Y coordinate remains unchanged  
**Code:** 5Ehex, 94dec



See Also: [SET\\_Y](#), [SET\\_XY](#)

**Example:**

The following sequence will put a 2 blue points in the same row.

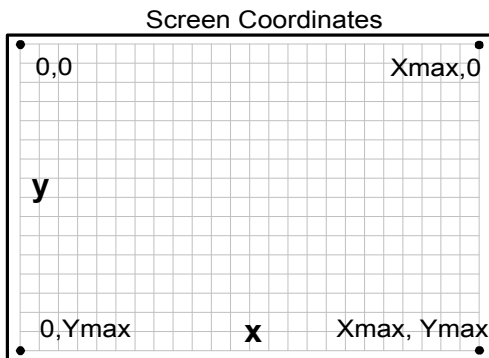
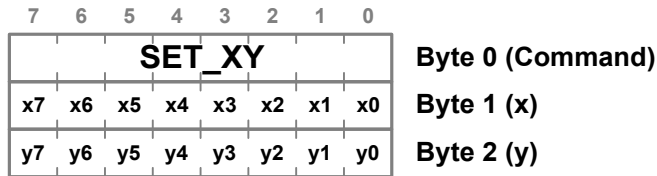
```

SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_X      5E hex
           200 dec (x)
PLOT       26 hex
SET_X      5E hex
           208 dec (x)
PLOT       26 hex
  
```

## 8.95.21 SET\_XY

**Description:** Sets the Current Position

**Code:** 25hex, 37dec



See Also: [PLOT](#), [LINE\\_TO\\_XY](#), [CIRCLE\\_R](#)

### Example:

The following sequence will put the blue point in the middle of the screen.

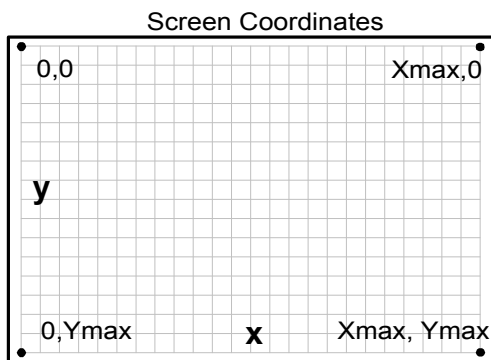
```
SET_COLOR 24 hex
BLUE      11000000 bin
SET_XY    25 hex
120      120 dec
80       80 dec
PLOT     26 hex
```

## 8.95.22 SET\_XHY

**Description:** Sets the Current Position.

**Code:** 85hex, 133dec

	7	6	5	4	3	2	1	0	
	<b>SET_XHY</b>								
	x15	x14	x13	x12	x11	x10	x9	x8	<b>Byte 0 (Command)</b>
	x7	x6	x5	x4	x3	x2	x1	x0	<b>Byte 1 (x MSB)</b>
	y7	y6	y5	y4	y3	y2	y1	y0	<b>Byte 2 (x LSB)</b>
									<b>Byte 3 (y)</b>



See Also: [PLOT](#), [LINE TO XHY](#), [CIRCLE RH](#)

### Example:

The following sequence will put a blue point at (160, 117).

```

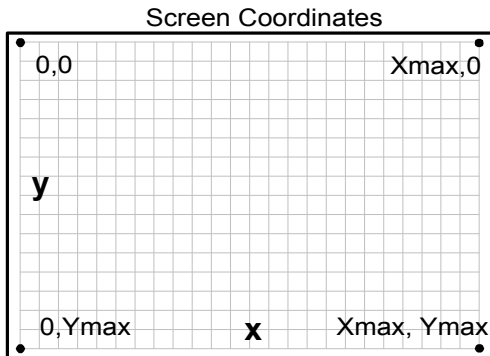
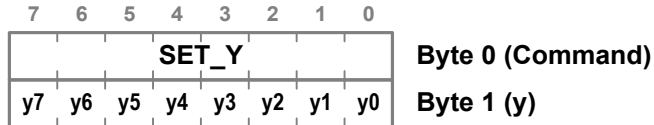
SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_XHY   85 hex
0          0 dec (x MSB)
160       160 dec (x LSB)
117       117 dec (y)
PLOT      26 hex

```

## 8.95.23 SET\_Y

**Description:** Sets only the Y-coordinate of the Current Position. X coordinate remains unchanged

**Code:** 5F<sub>hex</sub>, 95<sub>dec</sub>



See Also: [SET\\_XH](#), [SET\\_XHY](#)

### Example:

The following sequence will put a 2 blue points in the same column.

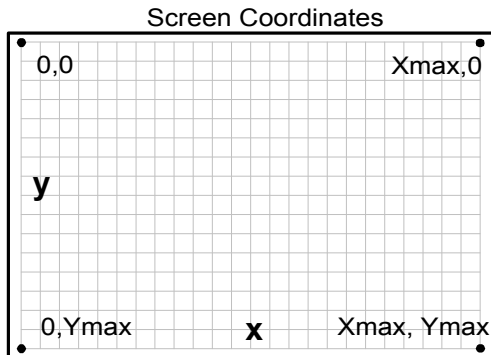
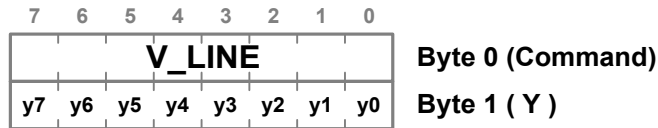
```

SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_Y      5F hex
70         70 dec (y)
PLOT       26 hex
SET_Y      5F hex
75         75 dec (y)
PLOT       26 hex
  
```

## 8.95.24 V\_LINE

**Description:** Quickly draws a vertical line from Current Position, to the row specified by the parameter.

**Code:** 41hex, 65dec



See Also: [H\\_LINEH](#), [SET\\_XHY](#)

### Example:

The following sequence will draw a blue vertical line from (95, 10) to (95, 110).

```

SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_XHY    85 hex
  0         0 dec (x MSB)
  95        95 dec (x LSB)
  10        10 dec (y)
V_LINE     41 hex
110        110 dec

```

## GLOSSARY

<b>Configuration Keys</b>	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> ".
<b>ezLCD+ Customization</b>	Modification of the default power-up parameters. Addition of custom fonts, bitmaps, Lua programs, etc. Described in the " <i>ezLCD+10x Manual</i> ".
<b>Firmware</b>	Operating software of the ezLCD+. Can be in-field upgraded. Described in the " <i>ezLCD+10x Manual</i> ".
<b>Lua</b>	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> ".
<b>User Configuration</b>	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> ".
<b>User ROM</b>	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> ".

# Index

## - A -

ARC 48, 189  
ARCH 48

## - B -

BOX 191, 193  
BOX\_FILL 192, 194  
BOXH 193  
BOXH\_FILL 194  
BOXHH 50  
BOXHH\_FILL 51  
BUTTON\_DEF 195  
BUTTON\_DEF\_LONG 53  
BUTTON\_STATE 55  
BUTTONS\_ALL\_UP 56  
BUTTONS\_DELETE\_ALL 57

## - C -

CACHE\_FT\_CHARS 58  
CACHE\_FT\_UNICHARS 60  
CalibratedXY 27  
CIRCLE\_R 62, 197  
CIRCLE\_R\_FILL 63, 198  
CIRCLE\_RH 62  
CIRCLE\_RH\_FILL 63  
CLS 64  
COPY\_FRAME 65  
cuButton 25

## - D -

Data Protocols 22

## - E -

ELLIPSE\_AHBH\_FILL 72  
ELLIPSE\_ARCH 74  
ELLIPSE\_PIEH 76  
ezButton 23

## - F -

FILL 78  
FILL\_BOUND 79

## - G -

GET\_SERIAL\_NO 80

## - H -

H\_LINE 81  
H\_LINEH 81

## - L -

LIGHT\_BRIGHT 82  
LIGHT\_OFF 83  
LIGHT\_ON 84  
LINE\_TO\_XHY 199  
LINE\_TO\_XHYH 85  
LINE\_TO\_XY 199, 200

## - M -

MERGE\_FRAME 86  
MERGE\_RECT 87

## - P -

Packets 28  
PIEH 89  
PING 91  
PLOT 92  
PLOT\_XHY 201  
PLOT\_XHYH 93  
PLOT\_XY 201, 202  
POLYGON 94  
PRINT\_CHAR 97  
PRINT\_CHAR\_BG 98  
PRINT\_FT\_UNICHAR 99  
PRINT\_FT\_UNISTRING 100  
PRINT\_STRING 101  
PRINT\_STRING\_BG 102  
PUT\_BITMAP 203, 205

PUT\_BITMAP\_RGB 103  
 PUT\_BITMAPH 205  
 PUT\_PICT\_NO 105  
 PUT\_SF\_ICON 206

## - R -

REPLACE\_COLOR 106  
 RESTORE\_POSITION 107  
 RUN\_LUA 108

## - S -

SAVE\_POSITION 112  
 SD\_FILE\_CLOSE 113  
 SD\_FILE\_CLOSE\_ALL 114  
 SD\_FILE\_CREATE 115  
 SD\_FILE\_DELETE 118  
 SD\_FILE\_GET\_SIZE 120  
 SD\_FILE\_LIST 122  
 SD\_FILE\_OPEN 125  
 SD\_FILE\_READ 128  
 SD\_FILE\_REWIND 131  
 SD\_FILE\_SEEK 133  
 SD\_FILE\_TELL 135  
 SD\_FILE\_WRITE 137  
 SD\_FIND\_FIRST 139  
 SD\_FIND\_NEXT 139  
 SD\_FOLDER\_CREATE 142  
 SD\_FOLDER\_DELETE 144  
 SD\_FORMAT 146  
 SD\_INSERTED 148  
 SD\_LOAD\_FONT 149  
 SD\_PUT\_ICON 152  
 SD\_RAW\_READ 154  
 SD\_RAW\_WRITE 156  
 SD\_SCREEN\_CAPTURE 159  
 SD\_SIZE 160  
 SD\_SPACE\_INFO 161  
 SELECT\_FONT 163  
 SET\_ALPHA 164  
 SET\_BG\_COLOR 207, 208  
 SET\_BG\_COLOR\_RGB 165  
 SET\_BG\_COLORH 208  
 SET\_COLOR 209, 210  
 SET\_COLOR\_RGB 166  
 SET\_COLORH 210

SET\_DISP\_FRAME 167  
 SET\_DRAW\_FRAME 168  
 SET\_EDIT\_RECT 169  
 SET\_FT\_ANGLE 170  
 SET\_FT\_FONT 173  
 SET\_FT\_UNIBASE 175  
 SET\_PEN\_HEIGHT 177  
 SET\_PEN\_SIZE 178  
 SET\_TR\_COLOR\_RGB 179  
 SET\_X 211  
 SET\_XH 180  
 SET\_XHY 213  
 SET\_XHYH 181  
 SET\_XY 212, 213  
 SET\_Y 214  
 SET\_YH 182

## - T -

TEXT 183  
 TEXT\_EAST 183  
 TEXT\_NORTH 183  
 TEXT\_SOUTH 183  
 TEXT\_WEST 183  
 TOUCH\_PROTOCOL 185  
 TR\_COLOR\_NONE 186

## - V -

V\_LINE 215  
 V\_LINEH 187