ezLCD+

External Commands Manual

Supported Firmware Ver: 2.2

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 Lerusalimschy**

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 seperately.

Table of Contents

ezLCD+ External Commands Manual

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[1]
- Embedded 32bit processor (Atmel AT32AP7000) with LCD Controller
- 4 Mega Bytes of embedded flash for storing custom fonts and bitmaps
- SD/MicroSD[1] 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[2]
- RS232 TTL
- USB
- I2C
- SPI
- SD/MMC
- Ethernet[2] (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.

[1] Refer to H/W Reference Manual of your ezLCD+ product for more details.

[2] 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.

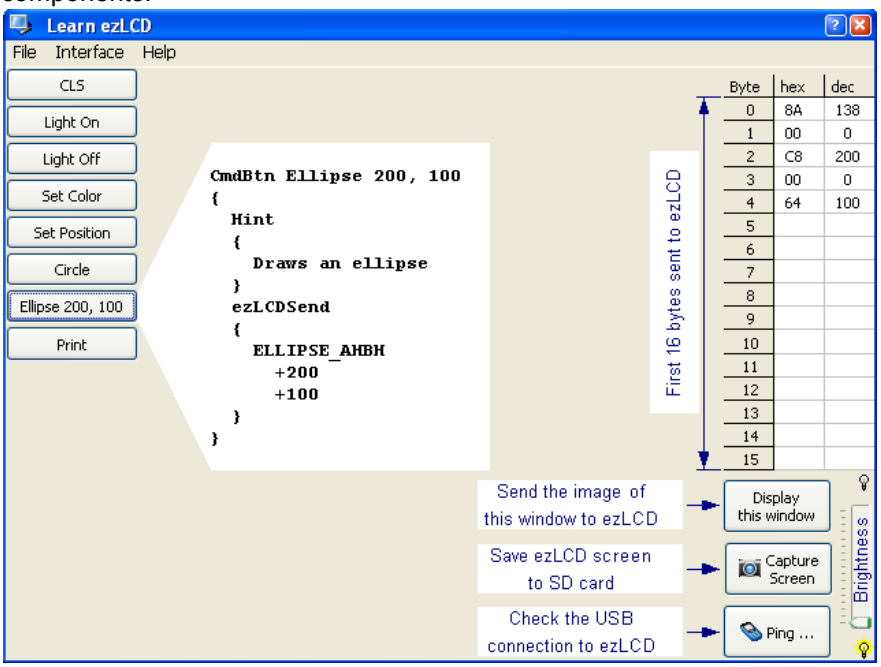
4. Go to chapter: "Quick Start: External Commands".

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

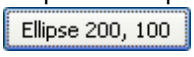
About Learn_ezLCD

"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: 

is shown to the right of it.

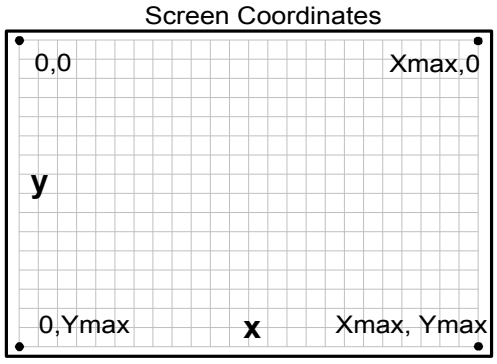
Permanent components of "Learn_ezLCD" are located in the right part of it's window: They include:

- 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

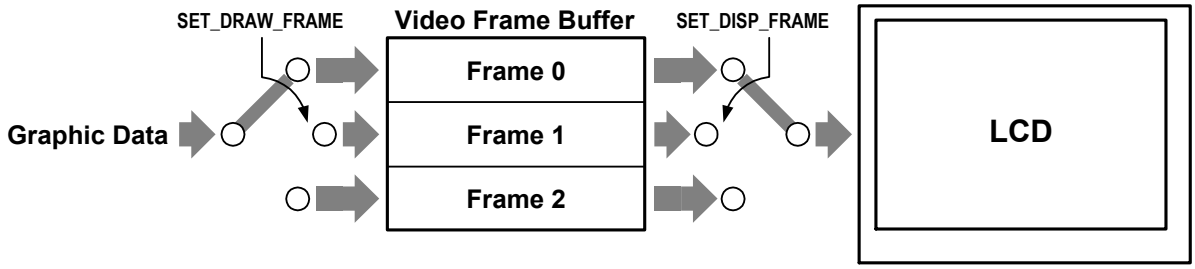
Additionally, ezLCD+ supports direct pixel drawing on the display. The following commands can be used for that:
PLOT, PLOT_XHYH, PUT_BITMAP_RGB

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.

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

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

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

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

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:

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

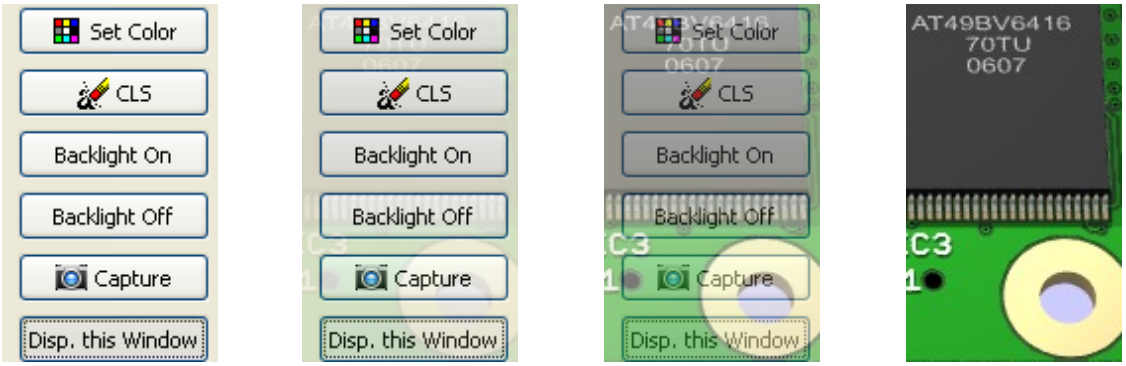
- Transparent Color.
  - Set by commands: SET_TR_COLOR_RGB, TR_COLOR_NONE
  - Specifies the color, which is ignored during Bitmap drawing
  - Only Bitmaps are affected (direct pixel drawing is not affected).

- 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.



Alpha = 0 Alpha = 32 Alpha = 128 Alpha = 255

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

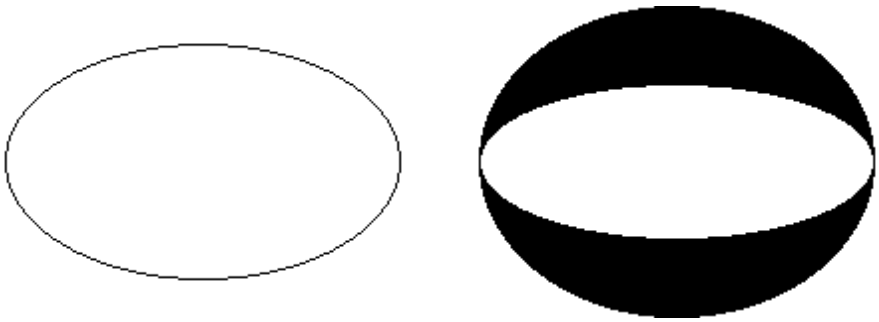
- Pen Height
  - Specifies the vertical dimension of Pen (in pixels), when drawing curves
  - Ignored when drawing straight lines
  - Set by commands: SET_PEN_SIZE and SET_PEN_HEIGHT

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

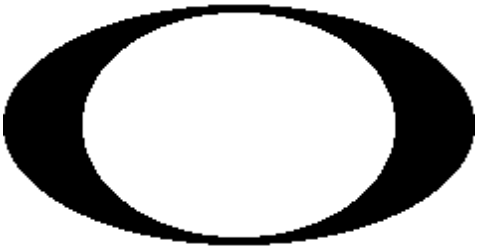
Pen Size = 1 Pen Size = 4 Pen Size = 20

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+.

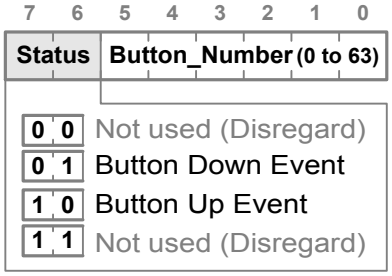
4. In your code, define the buttons using BUTTON_DEF or BUTTON_DEF_LONG command.
Send the above command for each of the buttons that you want to use. The button definition commands specify:
- Button Number (ID)
- Initial state of the button
- Bitmaps for each of the button states
- The position of the 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:



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)

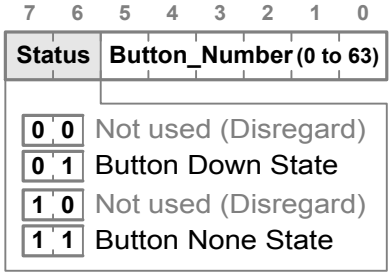
At this point the ezLCD+, broadcasts 5 to 20 times per second:
Button Down state, if the particular button is pressed.
Button None state, if no button is pressed.

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.

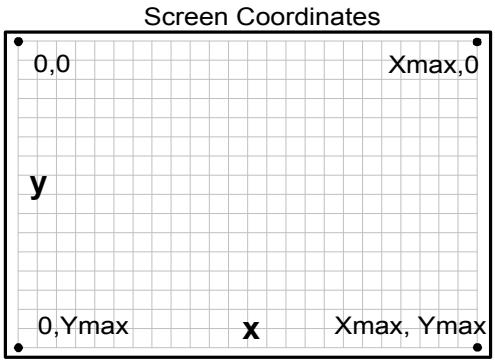
1. Touch Screen: Not pressed
   ezLCD+ Sends: FF$_{hex}$ continuously, 5 to 20 times per second
2. Touch Screen: Pressed in the Button 4 touch zone
   ezLCD+ Sends: 44$_{hex}$ continuously, 5 to 20 times per second
3. Touch Screen: Finger is removed from the touch screen
   ezLCD+ Sends: FF$_{hex}$ continuously,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

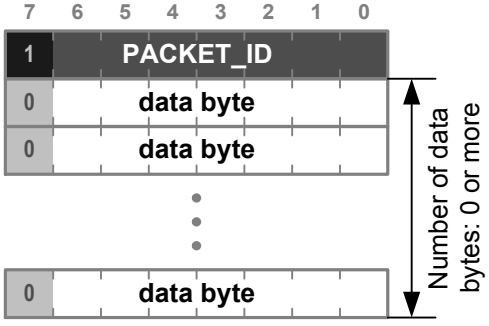
In order to select the CalibratedXY protocol, send TOUCH_PROTOCOL(64) to the ezLCD+.

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 pressed.
Length: 3 bytes, including the Packet ID
Code: **81**hex, **129**dec

7	6	5	4	3	2	1	0	
1	TOUCH_X							**Byte 0: PACKET_ID (81 hex)**
0	x6	x5	x4	x3	x2	x1	x0	**Byte 1: bits 0 to 6 of X**
0	x13	x12	x11	x10	x9	x8	x7	**Byte 2: bits 7 to 14 of X**

TOUCH_Y Packet

Description: The TOUCH_Y packet represents the touch screen X coordinate. It is sent only if the touch screen pressed.
Length: 3 bytes, including the Packet ID
Code: **82**hex, **130**dec

7	6	5	4	3	2	1	0	
1	TOUCH_Y							**Byte 0: PACKET_ID (82 hex)**
0	y6	y5	y4	y3	y2	y1	y0	**Byte 1: bits 0 to 6 of Y**
0	y13	y12	y11	y10	y9	y8	y7	**Byte 2: bits 7 to 14 of Y**

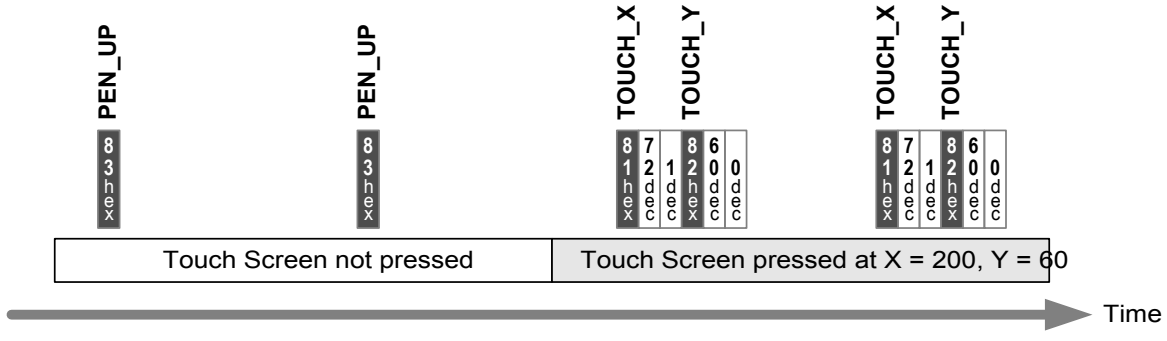
PEN_UP Packet

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: **83**hex, **131**dec

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.

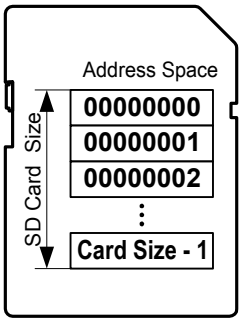


5 SD Card Operations

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

Summary of the SD File Operations commands.

Command	Description
SD_FORMAT	Formats the SD in the specified file system
SD_FILE_LIST	Obtains the list of files and sub-directories which reside in the specified SD Directory.
SD_FIND_FIRST and SD_FIND_NEXT	Obtain the list of SD files and sub-directories (one by one), which match the specified search pattern.
SD_FILE_OPEN	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 SD_FILE_CREATE
SD_FILE_CREATE	Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.
SD_FILE_CLOSE	Closes SD Flash file.
SD_FILE_CLOSE_ALL	Closes all opened SD Flash files.
SD_FILE_GET_SIZE	Gets the size (in bytes) of the opened SD Flash file.
SD_FILE_READ	Reads the specified number of bytes from the opened SD Flash file, starting from File Position Index.
SD_FILE_WRITE	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.
SD_FILE_SEEK	Moves the File Position Index of the opened SD Flash file by the specified number of bytes, from the position specified by the parameter.
SD_FILE_REWIND	Moves the File Position Index to the beginning of the opened SD Flash file.
SD_FILE_TELL	Gets the File Position Index of the opened SD Flash file.
SD_FILE_DELETE	Deletes the SD file.
SD_FOLDER_CREATE	Creates a new folder (directory) on the SD.
SD_FOLDER_DELETE	Deletes an empty folder (directory) on the SD.
SD_SPACE_INFO	Gets the information about the space usage (in bytes) of the formatted SD Card.
SD_PUT_ICON	Reads and displays the bitmap file.
SD_SCREEN_CAPTURE	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:

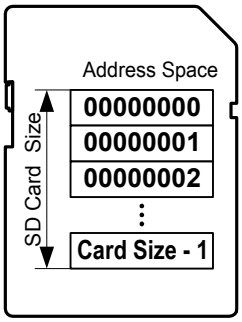
- Directory Path specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: `'*'` and `'?'` are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Directory Path is not case-sensitive. The drive and root directory do not 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
SD_SIZE	Gets the physical size (in bytes) of the SD Card.
SD_RAW_READ	Reads the data from SD starting from the specified SD address.
SD_RAW_WRITE	Writes the data on SD starting from the specified SD address.
SD_INSERTED	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 functions.
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(fptr_t refresh);
```

Arguments: `refresh` - Display Refresh Function (NULL for no refresh).
```
fptr_t is defined as:
typedef VOID (CALLBACK *fptr_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 Favr32WaitForByte(DWORD dev_no, UCHAR d, DWORD timeout);
```

Arguments:
```
dev_no   - Favr-32 device no (0 - 126)
d        - byte to wait for
timeout  - timeout in milliseconds
```

Returns:
```
1        - Success (byte received)
0        - Failure (timeout)
```

Favr32WaitForByteOrError

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 Favr32WaitForByteOrError(DWORD dev_no, UCHAR d, UCHAR e, DWORD timeout);
```

Arguments:
```
dev_no   - Favr-32 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
```

Favr32FlushRead

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

Prototype:
```
VOID __stdcall Favr32FlushRead(DWORD dev_no);
```

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

Favr32GetSN

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+10x Manual"*.

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

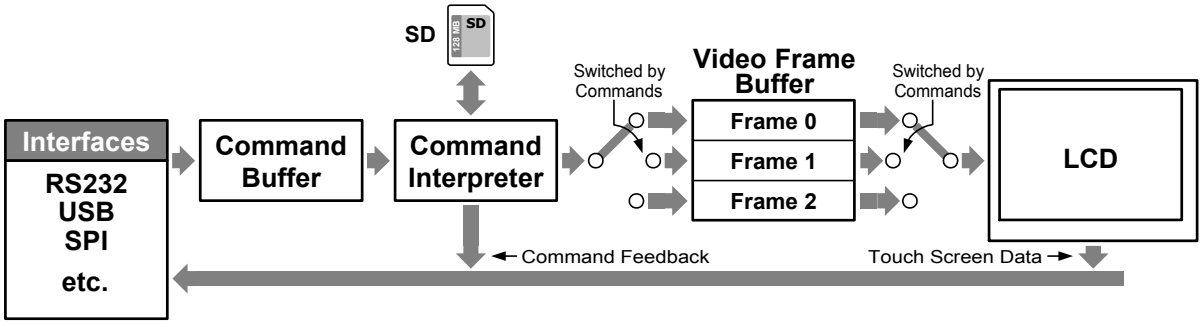
SDfdel - Deletes the file from the SD Card
Sources: fdel.c
Headers: CmdCodes.h, favr-32.h

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
SET_COLOR_RGB	31	hex	Set the drawing color to:
Red	0	hex	green
Green	FF	hex	
Blue	0	hex	
SET_XHYH	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	
CIRCLE_RH	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
SET_XHYH	Sets the Current Position
SET_XH	Sets only the X-coordinate of the Current Position
SET_YH	Sets only the Y-coordinate of the Current Position
SAVE_POSITION	Stores the Current Position to the Position ID
RESTORE_POSITION	Restores the Current Position saved by the SAVE_POSITION command

Drawing Colors

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

Drawing Pen

Command	Description
SET_PEN_SIZE	Sets the size of drawing Pen (both width and height)
SET_PEN_HEIGHT	Sets height of the drawing Pen. Affects only drawing of curves

Plotting Single Pixels

Command	Description
PLOT	Plots a point at Current Position in Current Color.
PLOT_XHYH	Plots a point in Current Color at the specified position.

Lines

Command	Description
LINE_TO_XHYH	Draws a line in Current Color, from Current Position to the specified position
H_LINEH	Quickly draws a horizontal line from the Current Position to the column specified by the parameter
V_LINEH	Quickly draws a vertical line from Current Position, to the row specified by the parameter

Curves

Command	Description
CIRCLE_RH	Draws a circle in Current Color centered at Current Position

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

Polygons

Command	Description
BOXHH	Draws a rectangle
BOXHH_FILL	Draws a rectangle filled with Current Color
POLYGON	Draws a polygon filled with the Current Color. The first vertex is located at the Current Position

Filling

Command	Description
CLS	Clears the screen by filling it with the Current Color
FILL	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.
FILL_BOUND	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
SET_EDIT_RECT	Sets the rectangle for editing (replacing colors, etc).
REPLACE_COLOR	Replaces color in the Edit Rectangle set by SET_EDIT_RECT command.

Frames

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

Command	Description
	according to the value of transparency Alpha. The resulting image is held in the destination Frame.
MERGE_RECT	Merges the rectangular portion of one frame with the other according to the value of transparency Alpha.

Displaying Bitmap Images

Command	Description
PUT_BITMAP_RGB	Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP
PUT_PICT_NO	Displays a bitmap (icon) with its upper-left corner positioned at the Current Position.The bitmap is read from the User ROM
SD_PUT_ICON	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
SELECT_FONT	Selects the Bitmap Font from User ROM
SET_FT_FONT	Selects and configures True Type Font from User ROM.
SD_LOAD_FONT	Loads and selects font file from SD card. Both, Bitmap Fonts and True Type Fonts are supported.
SET_FT_UNIBASE	Sets the base code for the Unicode characters, so they can be printed using 8-bit codes. Only the True Type Fonts are affected.

Printing Text

Command	Description
PRINT_CHAR	Prints an ASCII-coded character at the Current Position
PRINT_FT_UNICHAR	Prints a 16-bit Unicode-coded True Type character at the Current Position
PRINT_STRING	Prints a null-terminated ASCII-coded string starting at the Current Position
PRINT_FT_UNISTRING	Prints a null-terminated 16-bit Unicode-coded True Type string starting at the Current Position
PRINT_CHAR_BG	Prints a bitmap character at Current Position on the background specified by the SET_BG_COLORH command
PRINT_STRING_BG	Prints null-terminated string of bitmap characters starting at Current Position on the background specified by SET_BG_COLORH command
CACHE_FT_CHARS	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.
CACHE_FT_UNICHARS	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.

Command	Description
SET_FT_ANGLE	Sets the angle, in which the True Type Font characters and strings are printed
TEXT_NORTH, TEXT_SOUTH, TEXT_EAST, TEXT_WEST	Set the orientation of the text.

Backlight

Command	Description
LIGHT_ON	Turns on the screen backlight
LIGHT_OFF	Turns off the screen backlight
LIGHT_BRIGHT	Sets the brightness of the screen backlight

Touch Screen

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

SD I/O

Command	Description
SD_FORMAT	Formats the SD in the specified file system
SD_FILE_LIST	Obtains the list of files and sub-directories which reside in the specified SD Directory.
SD_FIND_FIRST and SD_FIND_NEXT	Obtain the list of SD files and sub-directories (one by one), which match the specified search pattern.
SD_FILE_OPEN	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 SD_FILE_CREATE
SD_FILE_CREATE	Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.
SD_FILE_CLOSE	Closes SD Flash file.
SD_FILE_CLOSE_ALL	Closes all opened SD Flash files.
SD_FILE_GET_SIZE	Gets the size (in bytes) of the opened SD Flash file.
SD_FILE_READ	Reads the specified number of bytes from the opened SD Flash file, starting from File Position Index.
SD_FILE_WRITE	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
SD_FILE_SEEK	Moves the File Position Index of the opened SD Flash file by the specified number of bytes, from the position specified by the parameter.
SD_FILE_REWIND	Moves the File Position Index to the beginning of the opened SD Flash file.
SD_FILE_TELL	Gets the File Position Index of the opened SD Flash file.
SD_FILE_DELETE	Deletes the SD file.
SD_FOLDER_CREATE	Creates a new folder (directory) on the SD.
SD_FOLDER_DELETE	Deletes an empty folder (directory) on the SD.
SD_SPACE_INFO	Gets the information about the space usage (in bytes) of the formatted SD Card.
SD_PUT_ICON	Reads and displays the bitmap file.
SD_SCREEN_CAPTURE	Saves an image of the displayed screen to the SD as .bmp file.
SD_SIZE	Gets the physical size (in bytes) of the SD Card.
SD_RAW_READ	Reads the data from SD starting from the specified SD address.
SD_RAW_WRITE	Writes the data on SD starting from the specified SD address.
SD_INSERTED	Checks if the SD card is inserted

Lua Scripts

Command	Description
RUN_LUA	Runs a Lua script
RUN_LUA_ROM	Runs a Lua script from the User ROM

CRC Protection

Command	Description
CRC_PKG	Sends CRC-protected data package to the ezLCD+

System

Command	Description
PING	Checks if the ezLCD+ is connected and ready to receive commands
GET_SERIAL_NO	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

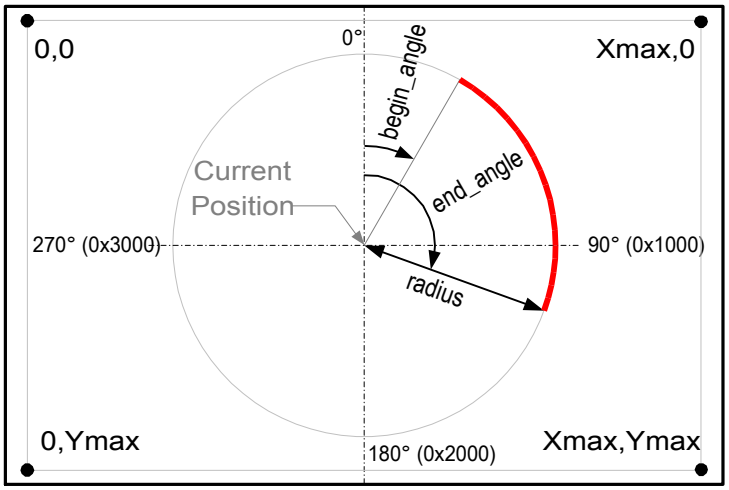
7	6	5	4	3	2	1	0	
ARCH								**Byte 0 (Command)**
radius_MSB								**Byte 1 (Radius MSB)**
radius_LSB								**Byte 2 (Radius LSB)**
begin_angle_MSB								**Byte 3 (Arc Begin Angle MSB)**
begin_angle_MSB								**Byte 4 (Arc Begin Angle LSB)**
end_angle_MSB								**Byte 5 (Arc End Angle MSB)**
end_angle_LSB								**Byte 6 (Arc End Angle LSB)**

See Also: PIEH, CIRCLE_RH, ELLIPSE_ARCH

Angle Coding: The full angle (360°) is equal to 4000hex (16384dec).
To transform degrees to ARC angle units:
Angle_lcd = Angle_deg x 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:

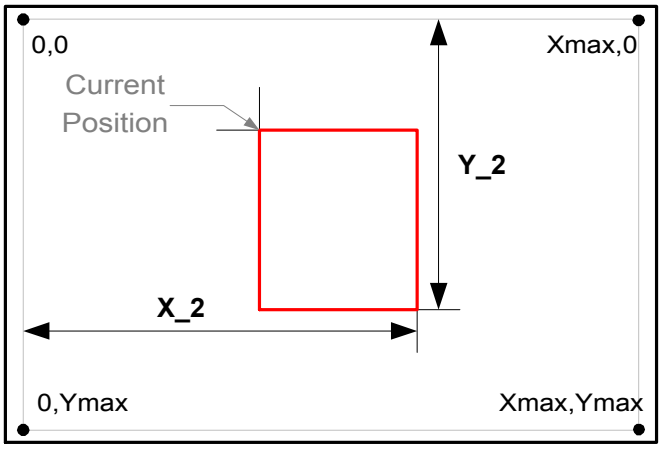
The following sequence will draw a green arc from 45 to 225 degrees with the center positioned at (160, 117) and a radius of 80.
225 x 2048 / 45 = 10240 (2800hex)

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



Ref: Screen Coordinates

See Also: SET_XHYH, BOXHH_FILL

Example:

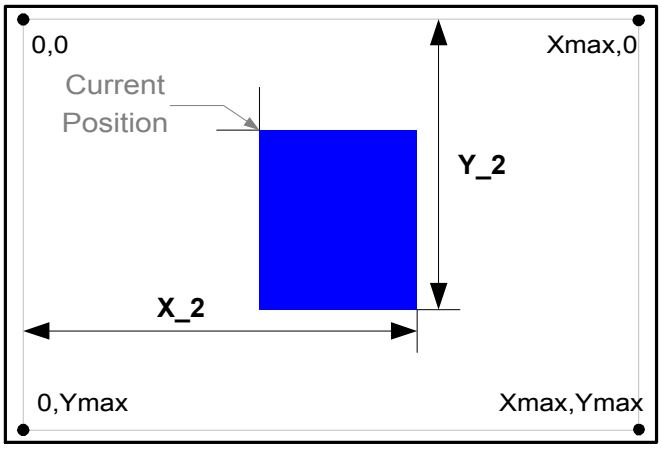
The following sequence will draw a rectangle with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

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



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**hex, **181**dec

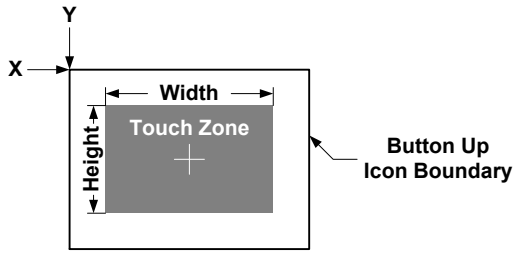
7	6	5	4	3	2	1	0		
BUTTON_DEF_LONG								Byte 0:	Command
button_no								Byte 1:	Button No (0 to 63)
state								Byte 2:	Initial State (1: Up, 2: Down, 3: Disabled, 4: Non-Visible)
buton Up icon MSB								Byte 3:	Icon No in User ROM for button Up (FFFFhex= none)
buton Up icon LSB								Byte 4:	
buton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

About the Touch Zone:

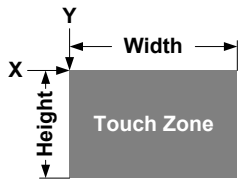
Touch Zone is the active touch response area of the button. It is specified by **With** (Byte 13) and **Height** (Byte 14).

- 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:

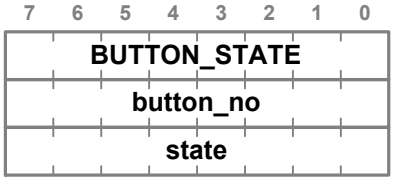
- **Button Up Icon in User ROM: 8**
- **Button Down Icon in User ROM: 9**
- **No Icon for Button Disabled state**

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

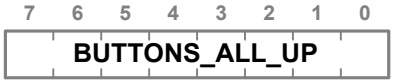
Example:

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**hex, **179**dec



Byte 0: Command

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

Byte 0: Command

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:

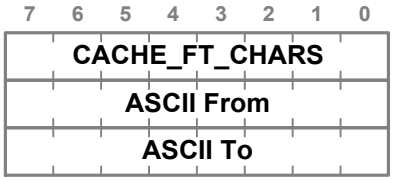
The following sequence will delete all Buttons.

```
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**hex, **149**dec



Byte 0: Command
Byte 1: Character From
Byte 2: Character To

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:

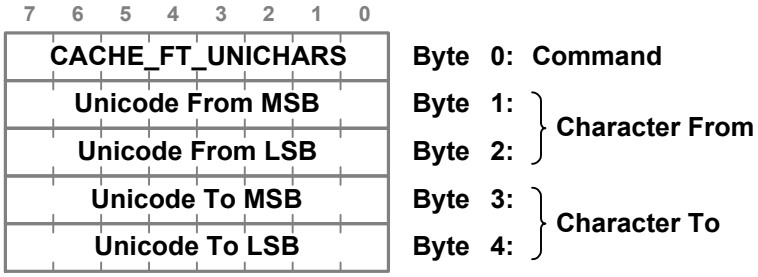
The following sequence will cache the characters from 'a' to 'z'.

```
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

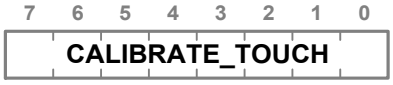
Example:

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

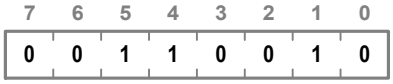
Description: Starts the touch screen calibration procedure
Code: **B6**hex, **182**dec



Byte 0: Command

ezLCD+ Response

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



Byte 0: 32hex, 50dec (Finished)

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

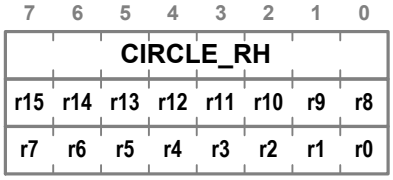
Example:

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



Byte 0 (Command)
Byte 1 (radius MSB)
Byte 2 (radius LSB)

See Also: SET_XHYH, SET_COLOR_RGB, ELLIPSE_AHBH

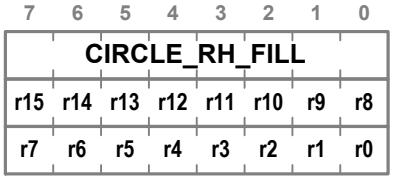
Example:

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: **99**hex, **153**dec



Byte 0 (Command)
Byte 1 (radius MSB)
Byte 2 (radius LSB)

Note: Since this is a filled figure, Pen parameters are ignored.

See Also: SET_XHY, SET_COLORH, ELLIPSE_AHBH_FILL

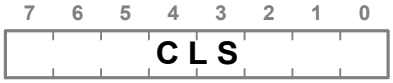
Example:

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: **21**hex, **33**dec



Byte 0 (Command)

Note: CLS is not affected by the transparency Alpha

See Also: SET_COLOR_RGB

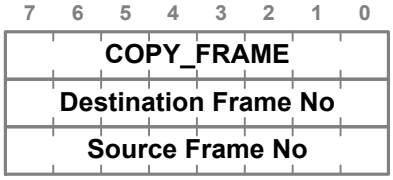
Example:

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**hex, **83**dec



Byte 0: Command
Byte 1: Destination Frame No (0 to 2)
Byte 2: Source Frame No (0 to 2)

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:

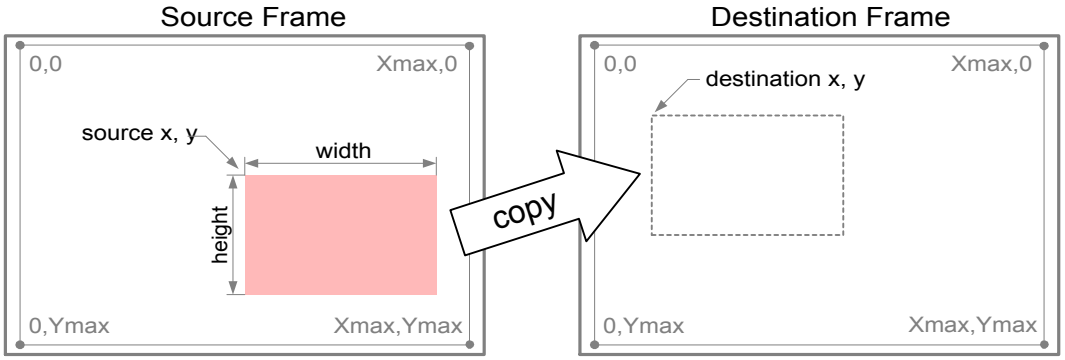
The following sequence will copy the contents of Frame 1 to Frame 0

```
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**hex, **85**dec

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



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:

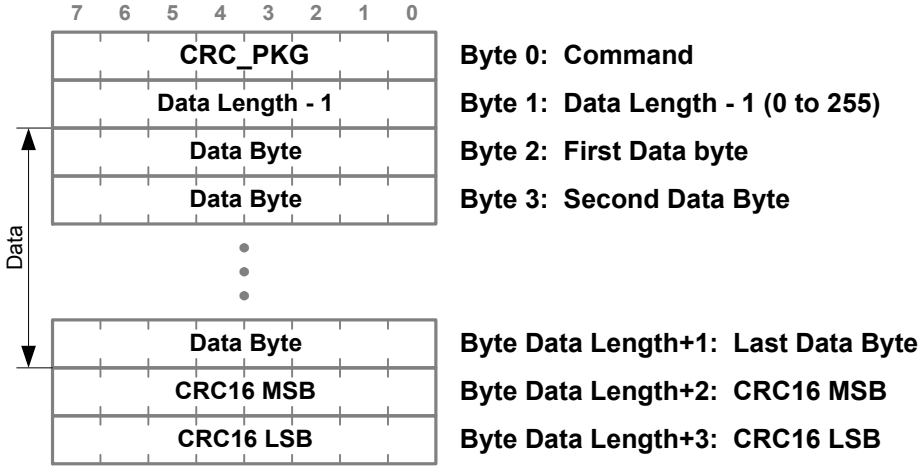
The following sequence will copy the rectangle size 100x50 from Frame 1 (320, 240) to Frame 0 (40, 60)

```
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**hex, **204**dec



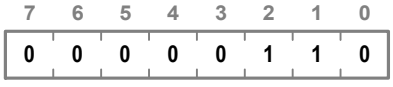
CRC16:
- Calculated over Data Bytes
- CCITT standard CRC-16:
  16 bit, non-reflected CRC
  Polynomial: 1021hex
  Seed: FFFFhex
  Final xor: 0

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

ezLCD+ Response

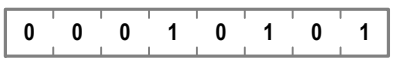
After receiving the CRC_PKG command, the ezLCD+ calculates CRC and compares it to the received CRC.

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



Byte 0: 06hex, 6dec (CRC OK)

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



Byte 0: 15hex, 21dec (CRC FAIL)

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

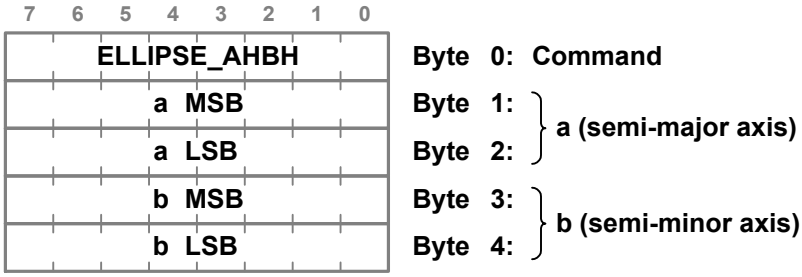
Note: The 'C' source code example of calculating CCITT standard CRC-16 is shown on the next page.

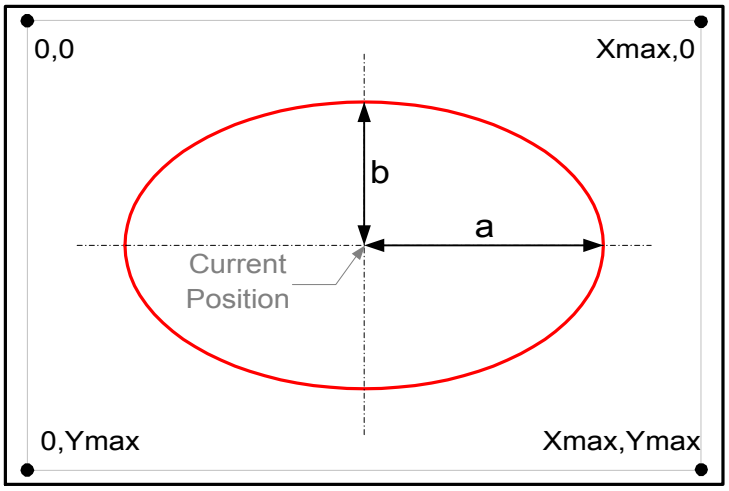

```
// CCITT standard CRC-16
//  ====================
// 16 bit, non-reflected CRC using:
// polynomial: 0x1021
// seed: 0xFFFF
// final xor: 0
// Endian: independent
#define CRC16_SEED         0xFFFF
#define CRC16_FINAL_XOR    0x0000
#define CRC16_START    CRC16_SEED
#define CRC16_NEXT(crc, next_byte) (((crc) << 8) ^ Crc16Table[((crc) >> 8) ^ (next_byte)])
#define CRC16_END(crc) ((crc) ^ CRC16_FINAL_XOR)
const unsigned short  Crc16Table[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, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
  0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
  0xedae, 0xfd8f, 0xcdec, 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  Crc16(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

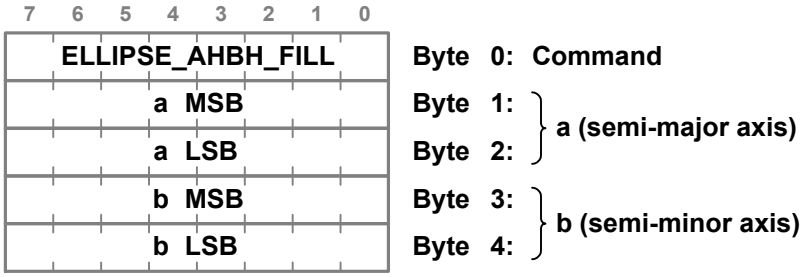
Example:

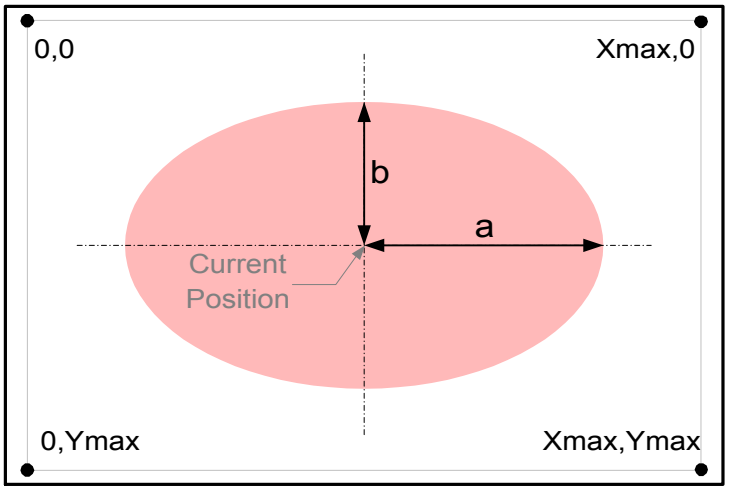
The following sequence will draw a red ellipse with the center positioned at (160, 117).

```
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

7	6	5	4	3	2	1	0		
ELLIPSE_ARCH								Byte 0:	Command
a MSB								Byte 1:	a (semi-major axis)
a LSB								Byte 2:	
b MSB								Byte 3:	b (semi-minor axis)
b LSB								Byte 4:	
begin angle LSB								Byte 1:	begin angle
begin angle MSB								Byte 2:	
end angle MSB								Byte 3:	end angle
end angle LSB								Byte 4:	

See Also: ARCH, ELLIPSE_PIEH

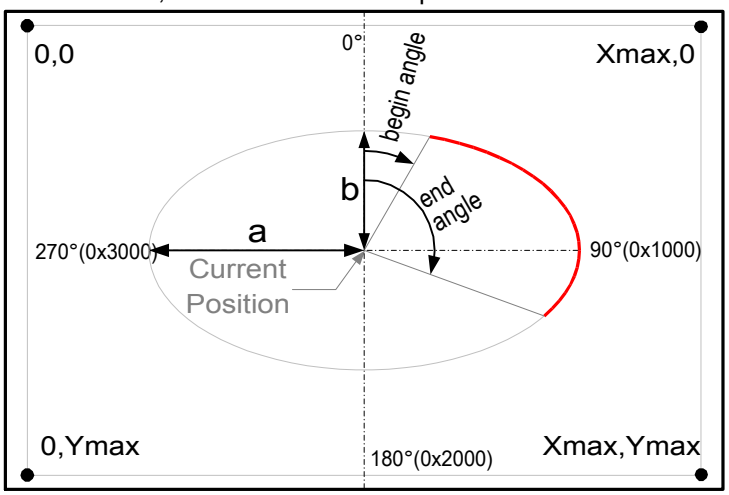
Angle Coding: The full angle (360°) is equal to 4000hex (16384dec).
To transform degrees to ARC angle units:

```
Angle_lcd = Angle_deg x 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 arc from 22.5 to 225 degrees and a = 80, b = 40.
22.5 x 2048 / 45 = 1024 (400hex)
225 x 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: **8E**hex, **142**dec

7	6	5	4	3	2	1	0		
ELLIPSE_PIEH								Byte 0: Command	
a MSB								Byte 1:	a (semi-major axis)
a LSB								Byte 2:	
b MSB								Byte 3:	b (semi-minor axis)
b LSB								Byte 4:	
begin angle MSB								Byte 1:	begin angle
begin angle LSB								Byte 2:	
end angle MSB								Byte 3:	end angle
end angle LSB								Byte 4:	

See Also: PIEH, ELLIPSE_ARCH

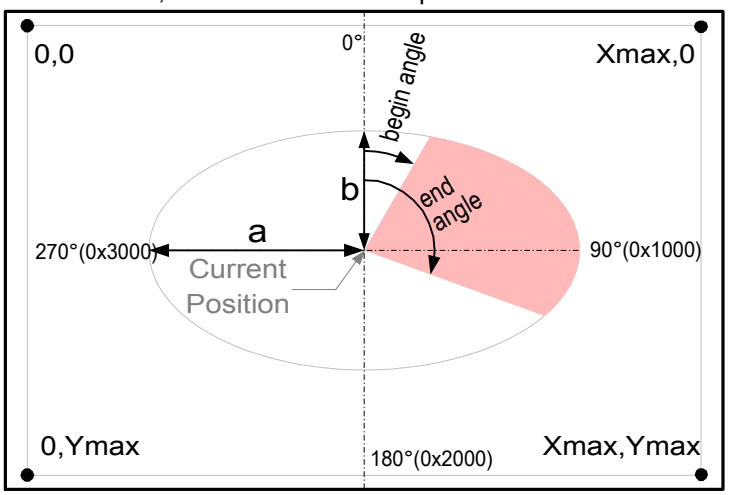
Angle Coding: The full angle (360°) is equal to 4000hex (16384dec).
To transform degrees to ARC angle units:

```
Angle_lcd = Angle_deg x 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:

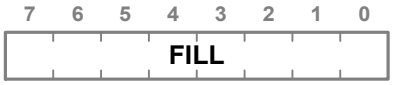
The following sequence will draw an ellipse pie from 22.5 to 225 degrees and a = 80, b = 40.
22.5 x 2048 / 45 = 1024 (400hex)
225 x 2048 / 45 = 10240 (2800hex)

```
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)
```

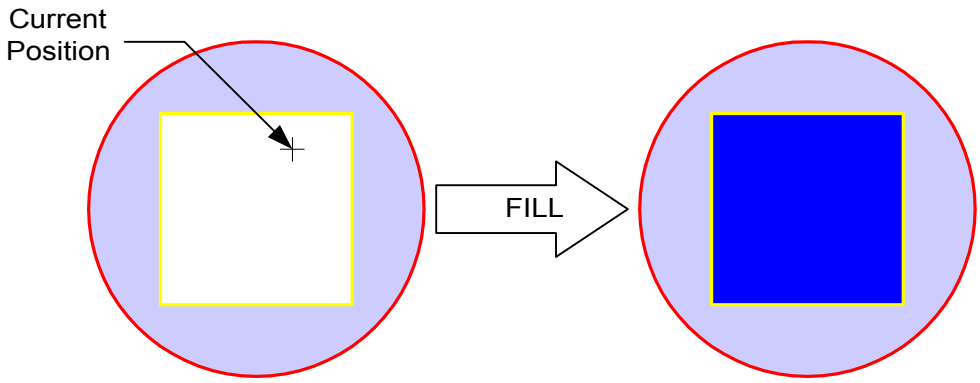
8.22 FILL

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



Byte 0 (Command)



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:

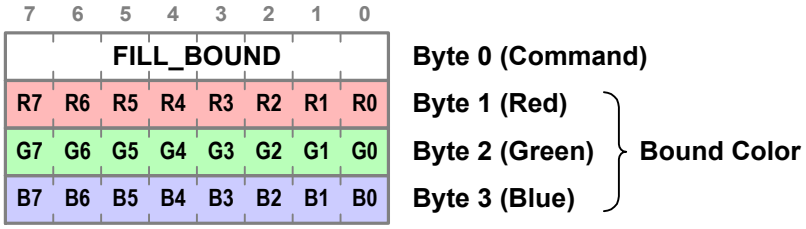
The following sequence will clear the area surrounding point at (160, 117) with blue (ref: drawing above).

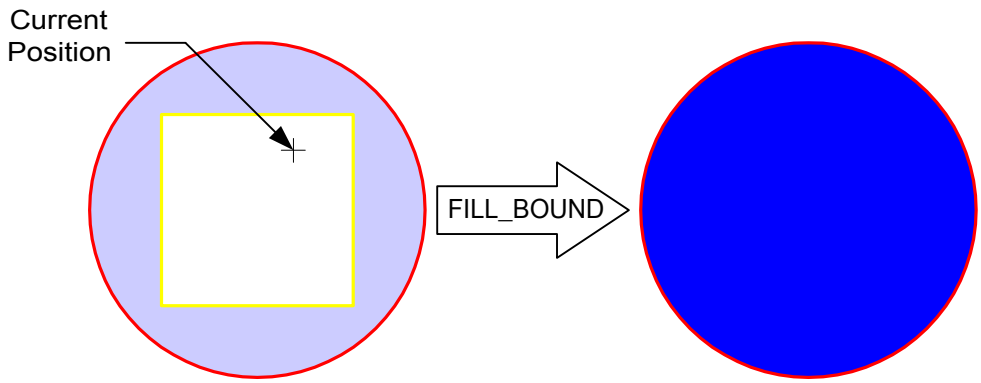
```
SET_COLOR_RGB  31 hex
 Red            0
 Green          0
 Blue          FF hex
SET_XHYH       33 hex
   0            0 dec (x MSB)
 160          160 dec (x LSB)
   0            0 dec (y MSB)
 117          117 dec (y LSB)
FILL           9B hex
```


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

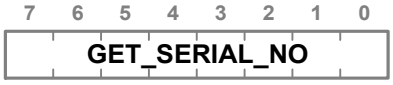
Example:

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

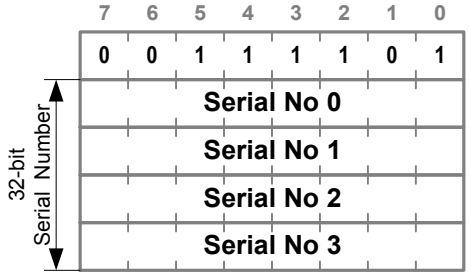
Description: Gets the value of the User Device Serial Number (Configuration Key: SerialNo).
Code: **9D**hex, **157**dec



Byte 0: Command

ezLCD+ Response

After receiving the GET_SERIAL_NO command, the ezLCD+ responds with:



Byte 0: 3Dhex, 61dec
Byte 1: Serial No 0 (MSB)
Byte 2: Serial No 1
Byte 3: Serial No 2
Byte 4: Serial No 3 (LSB)

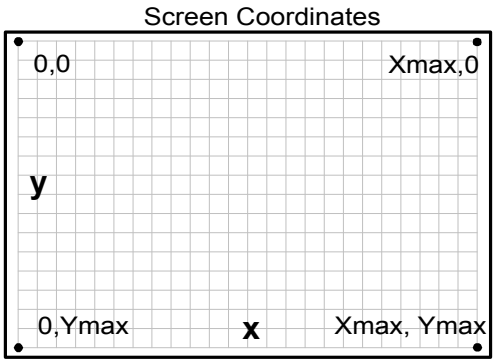
The ezLCD+ response is sent through the same interface, which received the GET_SERIAL_NO command.
Configuration Keys are described in *"ezLCD+10x Manual"*.

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

7	6	5	4	3	2	1	0	
H_LINEH								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x LSB)**



See Also: V_LINE, SET_XHY

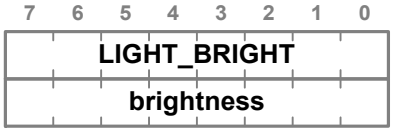
Example:

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: **80**hex, **128**dec



Byte 0 (Command)

Byte 1 (brightness 0 to 255)

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**hex, **35**dec

Byte 0 (Command)

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**hex, **34**dec

Byte 0 (Command)

See Also: LIGHT_OFF, LIGHT_BRIGHT

Example:

The following sequence will turn on the screen backlight.

```
LIGHT_ON     22 hex
```


8.29 LINE_TO_XHYH

Description: Draws a line in Current Color, from Current Position to the specified position.
Code: **3F**hex, **63**dec

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

Ref: Screen Coordinates

See Also: SET_XHYH, SET_COLOR_RGB

Example:

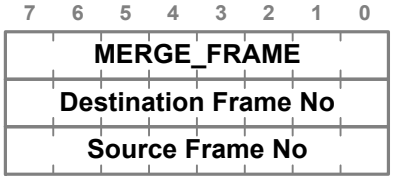
The following sequence will draw a red line across the screen.

```
SET_COLOR_RGB  31 hex
 Red           FF hex
 Green          0
 Blue           0
SET_XHYH       33 hex
  0             0 dec (x MSB)
  0             0 dec (x LSB)
  0             0 dec (y MSB)
  0             0 dec (y LSB)
LINE_TO_XHYH   3F hex
  2             2 dec (x MSB)
127           127 dec (x LSB 2*256+127=639)
  1             1 dec (y MSB)
223           223 dec (y LSB 1*256+223=479)
```

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: **54**hex, **84**dec



Byte 0: Command

Byte 1: Destination Frame No (0 to 2)

Byte 2: Source Frame No (0 to 2)

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

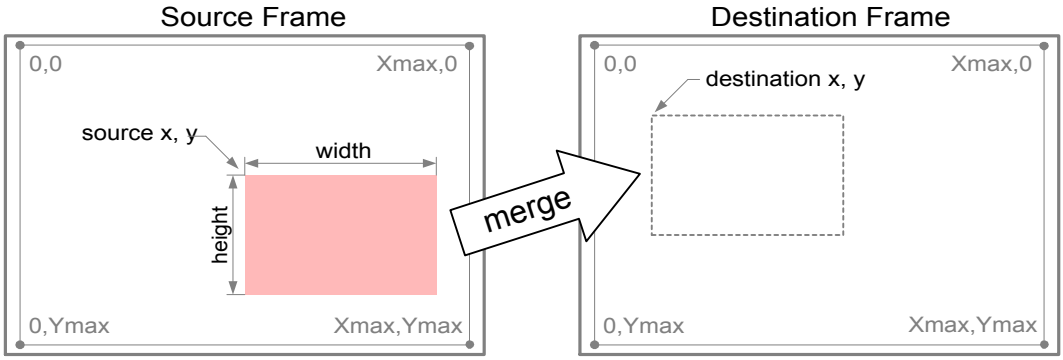
```
COPY_FRAME   53 hex
   0          0 (Destination Frame No)
   1          1 (Source Frame No)
```


8.31 MERGE_RECT

Description: Merges the rectangular portion of one frame with the other according to the value of transparency Alpha.

Code: **56**hex, **86**dec

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



When Alpha = 255, both commands (COPY_RECT and MERGE_RECT) yield identical results.

Note: Merging a rectangle wth 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**hex, **144**dec

7	6	5	4	3	2	1	0	
PIEH								**Byte 0 (Command)**
radius_MSB								**Byte 1 (Radius MSB)**
radius_LSB								**Byte 2 (Radius LSB)**
begin_angle_MSB								**Byte 3 (Pie Begin Angle MSB)**
begin_angle_MSB								**Byte 4 (Pie Begin Angle LSB)**
end_angle_MSB								**Byte 5 (Pie End Angle MSB)**
end_angle_LSB								**Byte 6 (Pie End Angle LSB)**

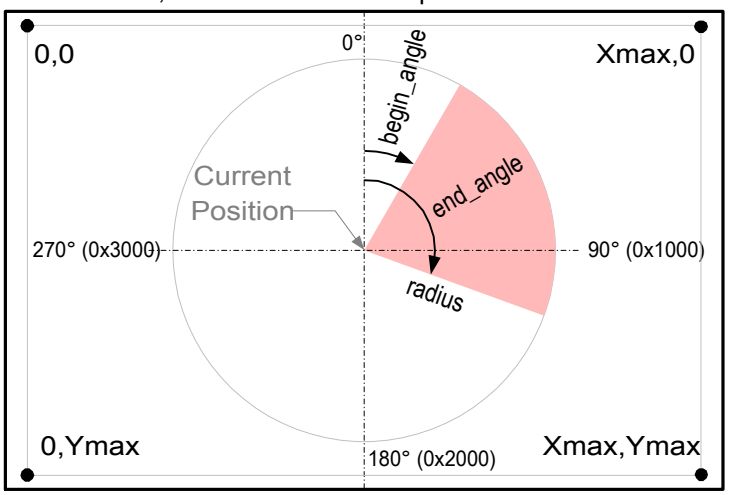
Note: Since this is a filled figure, Pen parameters are ignored.

See Also: ARCH, SET_XHYH, SET_COLOR_RGB, CIRCLE_RH, ELLIPSE_PIEH

Angle Coding: The full angle (360°) is equal to 4000hex (16384dec).
To transform degrees to ARC angle units:
Angle_lcd = Angle_deg x 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



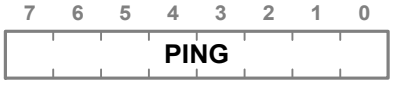
Example:

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

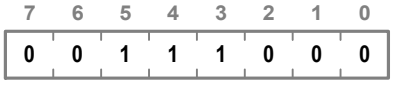
Description: Checks if the ezLCD+ is connected and ready to receive commands.
Code: **83**hex, **131**dec



Byte 0: Command

ezLCD+ Response

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



Byte 0: 38hex**, 56**dec**(PONG)**

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:

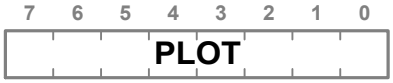
```
PING     83 hex
```

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



Byte 0 (Command)

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: **3E**hex, **62**dec

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

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)
```

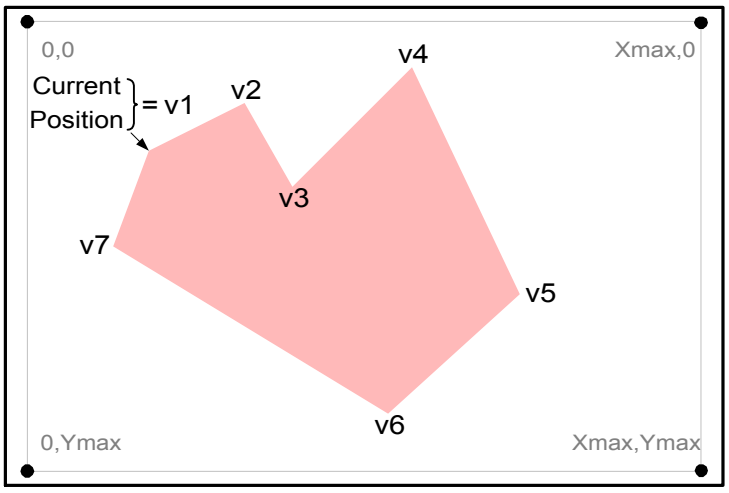
8.36 POLYGON

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

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

7	6	5	4	3	2	1	0		
POLYGON								Byte 0:	Command
n = No of Vertices - 1								Byte 1:	n = No Of Vertices - 1 (2 to 255)
x15	x14	x13	x12	x11	x10	x9	x8	Byte 2:	Vertex 2 X-coordinate
x7	x6	x5	x4	x3	x2	x1	x0	Byte 3:	
y15	y14	y13	y12	y11	y10	y9	y8	Byte 4:	Vertex 2 Y-coordinate
y7	y6	y5	y4	y3	y2	y1	y0	Byte 5:	
x15	x14	x13	x12	x11	x10	x9	x8	Byte 6:	Vertex 3 X-coordinate
x7	x6	x5	x4	x3	x2	x1	x0	Byte 7:	
y15	y14	y13	y12	y11	y10	y9	y8	Byte 8:	Vertex 3 Y-coordinate
y7	y6	y5	y4	y3	y2	y1	y0	Byte 9:	
⋮									
x15	x14	x13	x12	x11	x10	x9	x8	Byte n x 4 - 2:	Vertex n+1 X-coordinate
x7	x6	x5	x4	x3	x2	x1	x0	Byte n x 4 - 1:	
y15	y14	y13	y12	y11	y10	y9	y8	Byte n x 4:	Vertex n+1 Y-coordinate
y7	y6	y5	y4	y3	y2	y1	y0	Byte n x 4 + 1:	

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

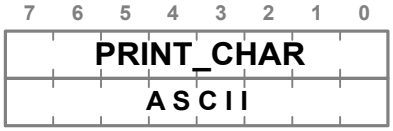
Example:

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: **2C**hex, **44**dec



Byte 0 (Command)

Byte 1 (ASCII Character)

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:

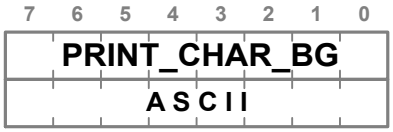
The following sequence will print character 'M'.

```
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



Byte 0 (Command)

Byte 1 (ASCII Character)

Note: This command cannot print the True Type Fonts.

See Also: SELECT_FONT, SET_BG_COLORH, PRINT_STRING_BG

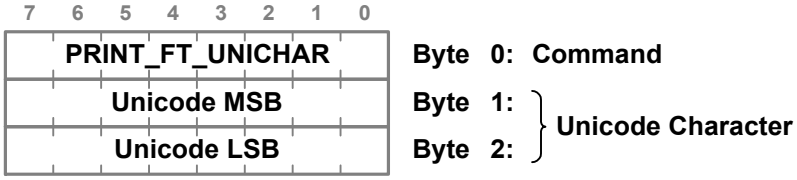
Example:

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: **97**hex, **151**dec



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:

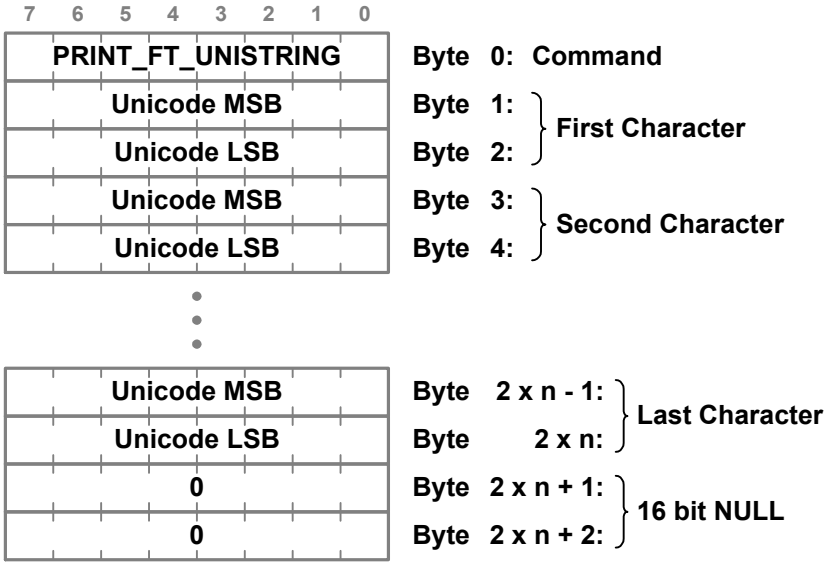
The following sequence will print character 'M'.

```
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: **98**hex, **152**dec



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

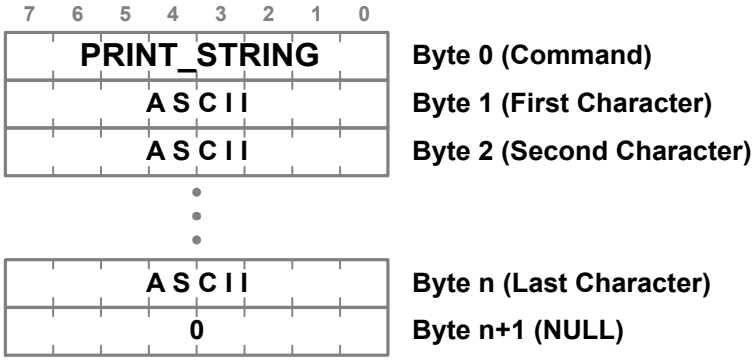
Example:

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:

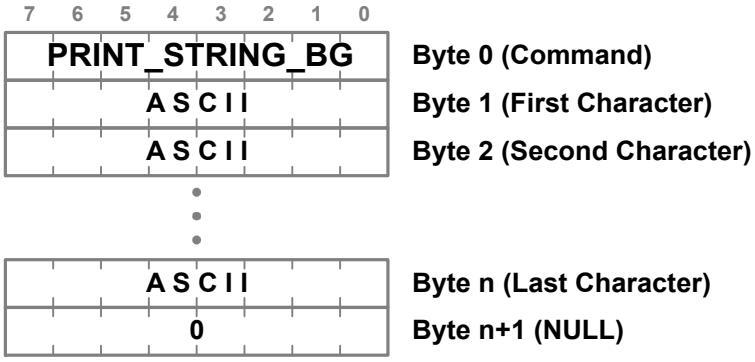
The following sequence will print "LCD" in purple.

```
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

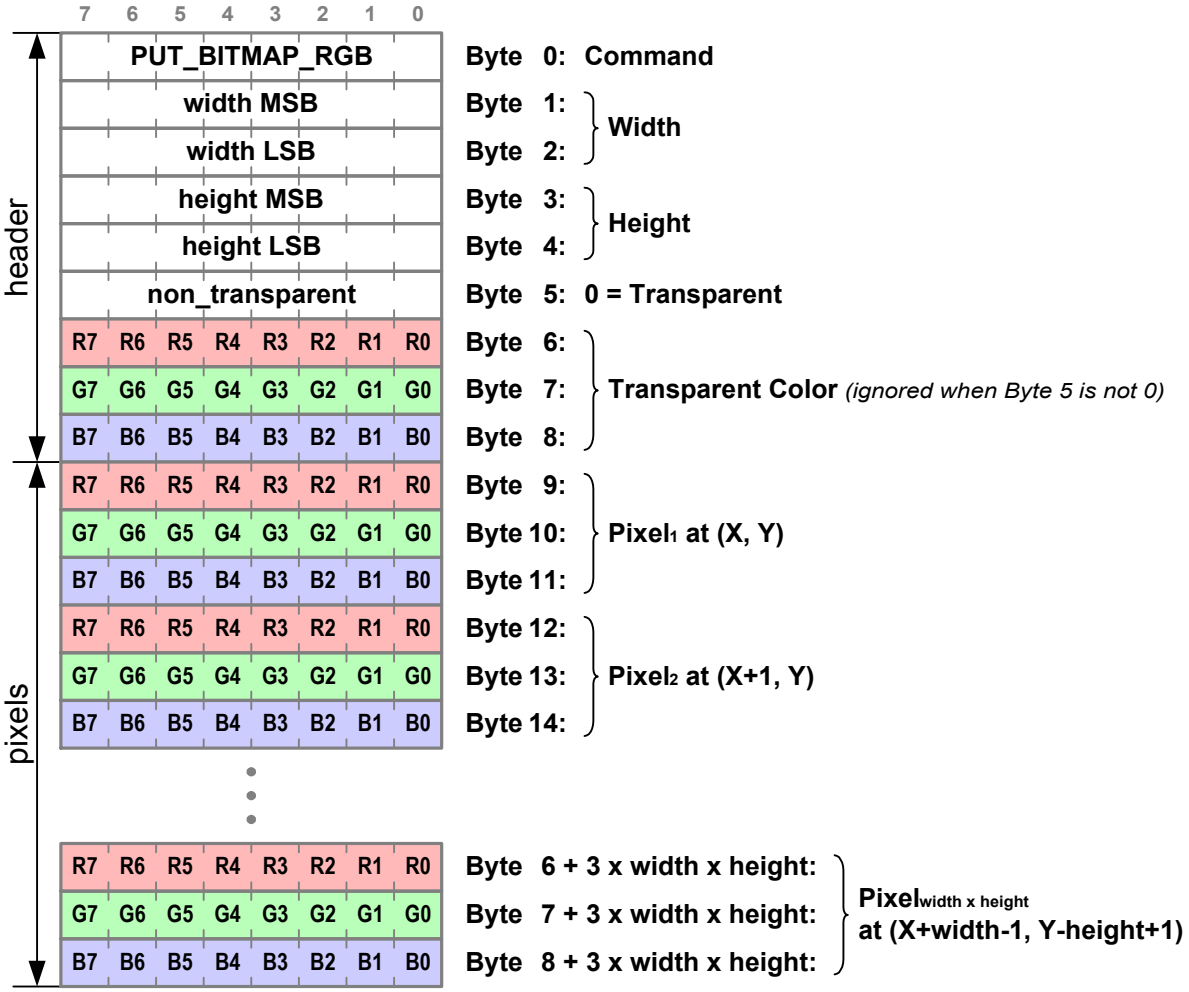
Example:

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
```

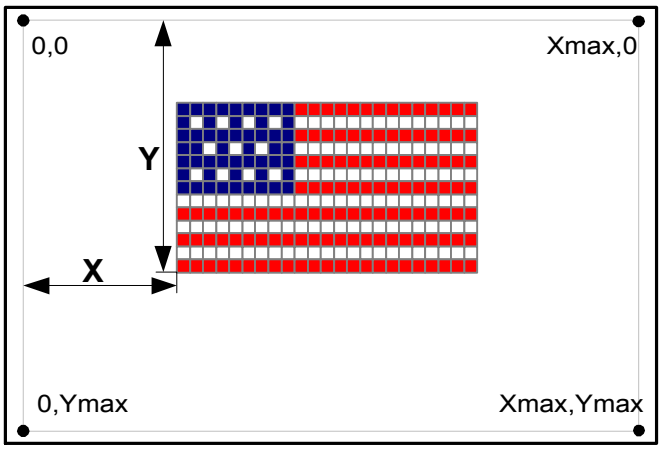

8.43 PUT_BITMAP_RGB

Description: Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP
Code: **9F**hex, **159**dec



Notes: 1. The total number of bytes is: 3 x width x 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

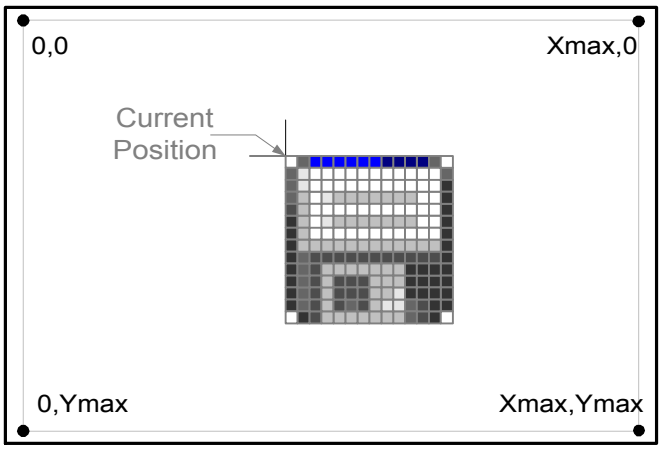
Pixel 1

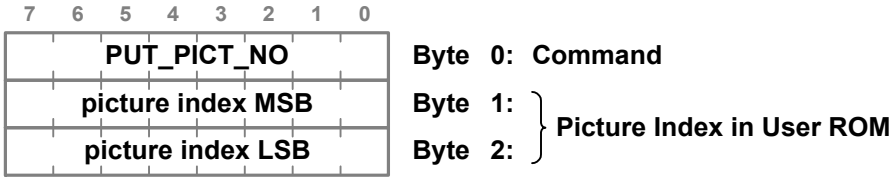
See Also: SET_XHYH, SET_COLOR_RGB

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: **59**hex, **89**dec





See Also: SD_PUT_ICON

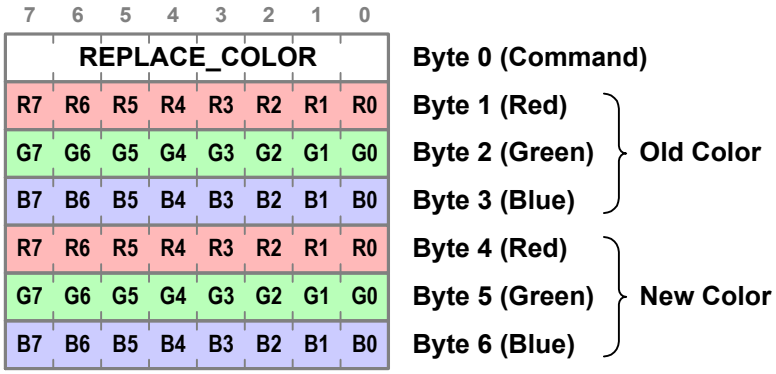
Example:

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

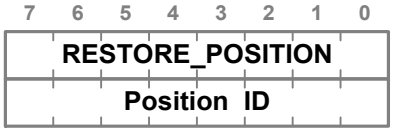
Example:

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: **36**hex, **54**dec



Byte 0 (Command)

Byte 1 (Position ID: 0 to 255)

See Also: SAVE_POSITION, SET_XHYH

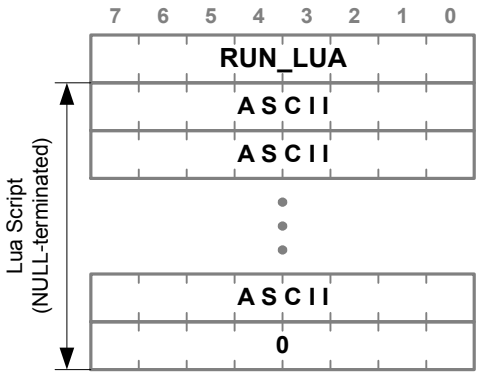
Example:

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



Byte 0: Command
Byte 1: First Character of the Lua Script
Byte 2: Second Character of the Lua Script
Byte n: Last Character of the Lua Script
Byte n+1: NULL

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.

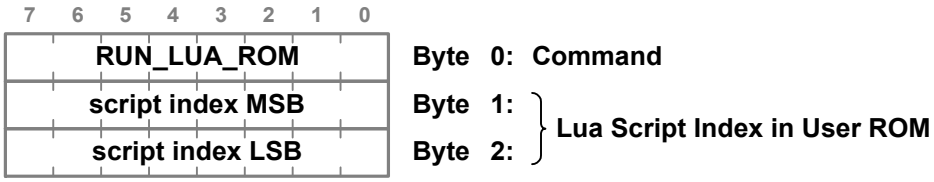
Example:

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

```
RUN_LUA    A7 hex
           ez.Cls(0xFFFFFF)
           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:

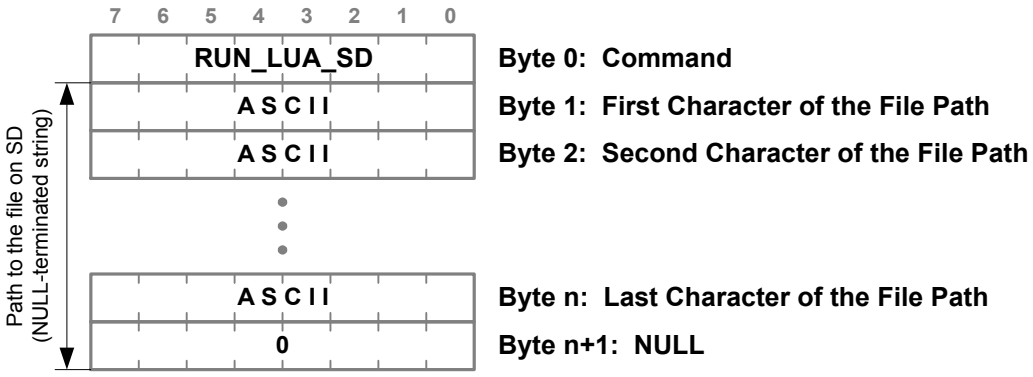
The following sequence will run Lua script no 2

```
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**hex, **169**dec



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:

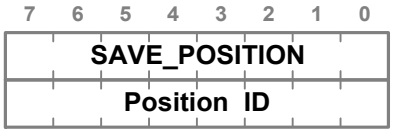
The following sequence will run Lua script from SD file: ezLCD.lua

```
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



Byte 0 (Command)

Byte 1 (Position ID: 0 to 255)

See Also: RESTORE_POSITION, SET_XHYH

Example:

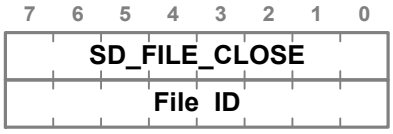
The following sequence will draw 3 lines with the common starting point: (160, 117)

```
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**hex, **114**dec



Byte 0: Command

Byte 1: File ID

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:

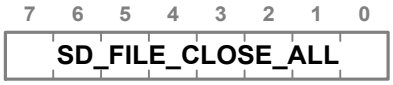
The following sequence will close SD file 1.

```
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**hex, **115**dec



Byte 0: Command

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

See Also: SD_FILE_OPEN, SD_FILE_CREATE, SD_FILE_CLOSE

Example:

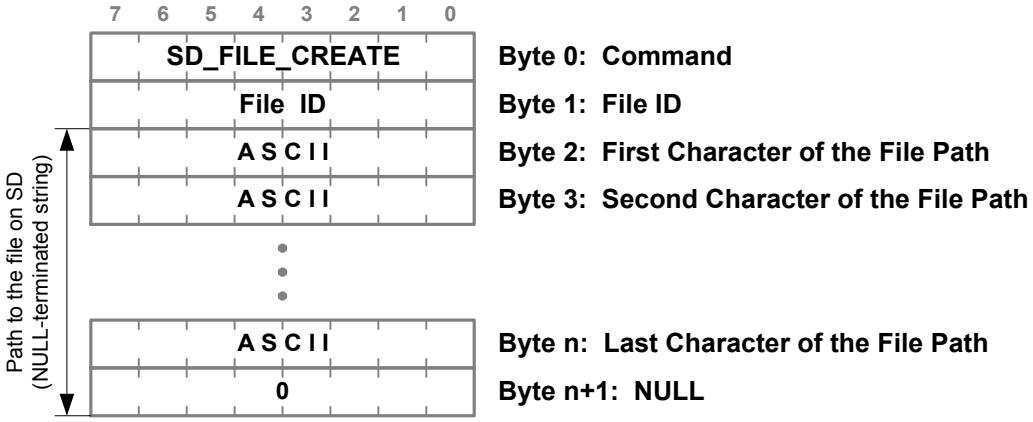
The following sequence will close all opened SD files and re-enable the touch screen.

```
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: **76**hex, **118**dec



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

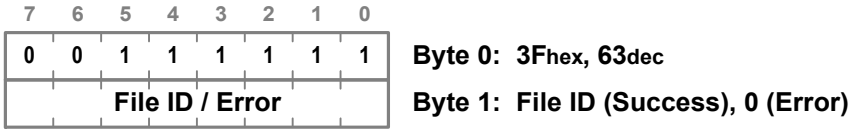
See Also: SD_FILE_OPEN, SD_FILE_CLOSE, SD_FILE_CLOSE_ALL

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 temporary 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:

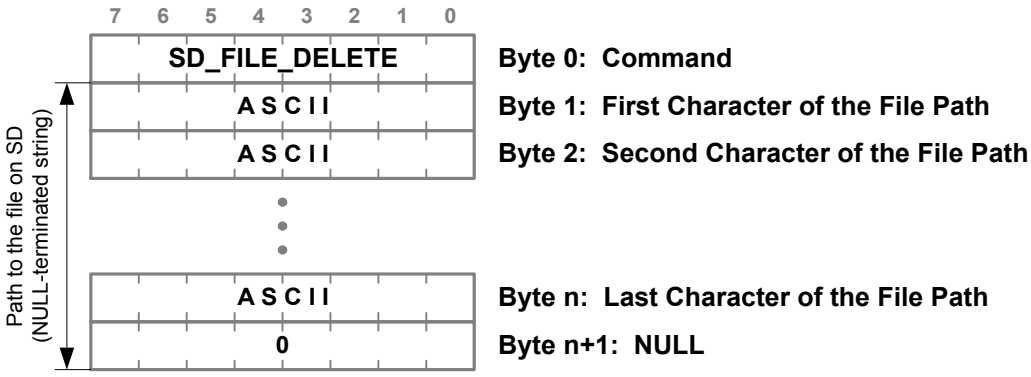
```
3F hex
 1 dec
```

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**hex, **125**dec



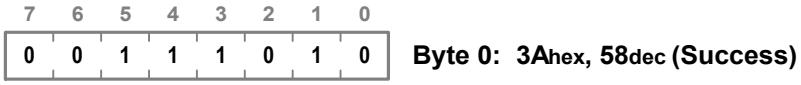
Notes: SD card has to be formatted in the supported file system.
A read-only or opened file cannot be deleted.

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

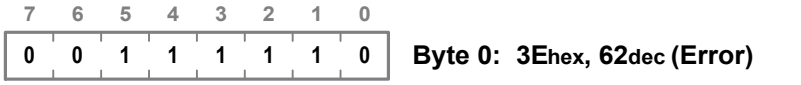
After receiving the SD_FILE_DELETE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Ahex, 58dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (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
```

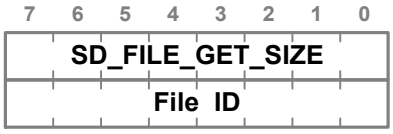
In case of the failure, the following sequence will be sent by the ezLCD+:

```
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**hex, **116**dec



Byte 0: Command
Byte 1: File ID

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

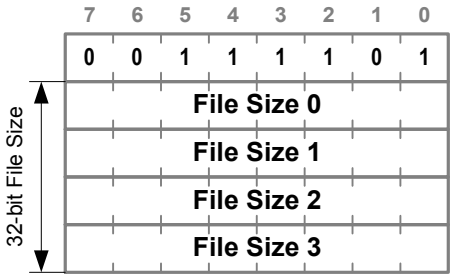
See Also: SD_FILE_OPEN, SD_FILE_CLOSE, SD_FILE_CLOSE_ALL

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

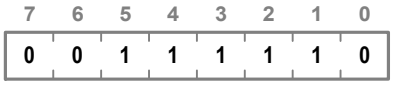
After receiving the SD_FILE_GET_SIZE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Dhex, 61dec
Byte 1: File Size 0 (LSB)
Byte 2: File Size 1
Byte 3: File Size 2
Byte 4: File Size 3 (MSB)

In case of an **error**:



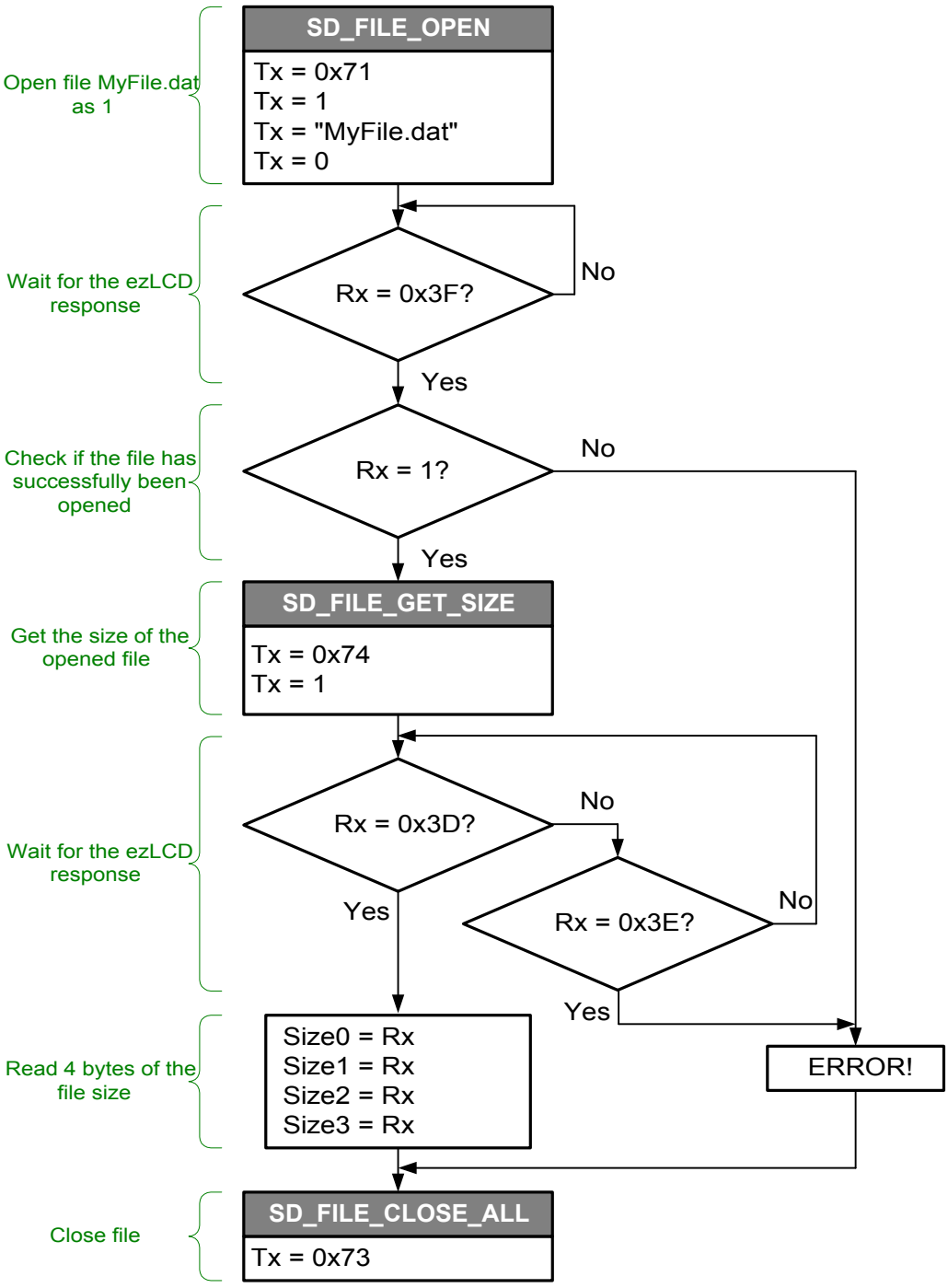
Byte 0: 3Ehex, 62dec (Error)

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:

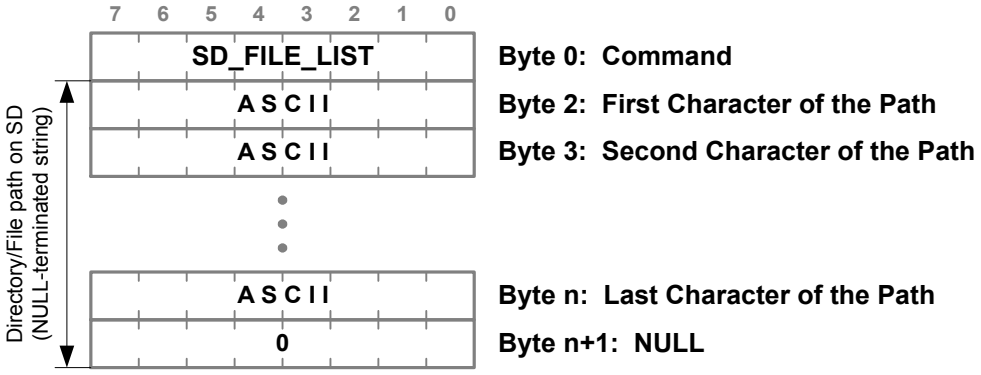
The following flow chart shows an example of getting the size of the file MyFile.dat



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: **79**hex, **121**dec



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 (**58**dec), 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 (**0A**hex or **10**dec)
- 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**:

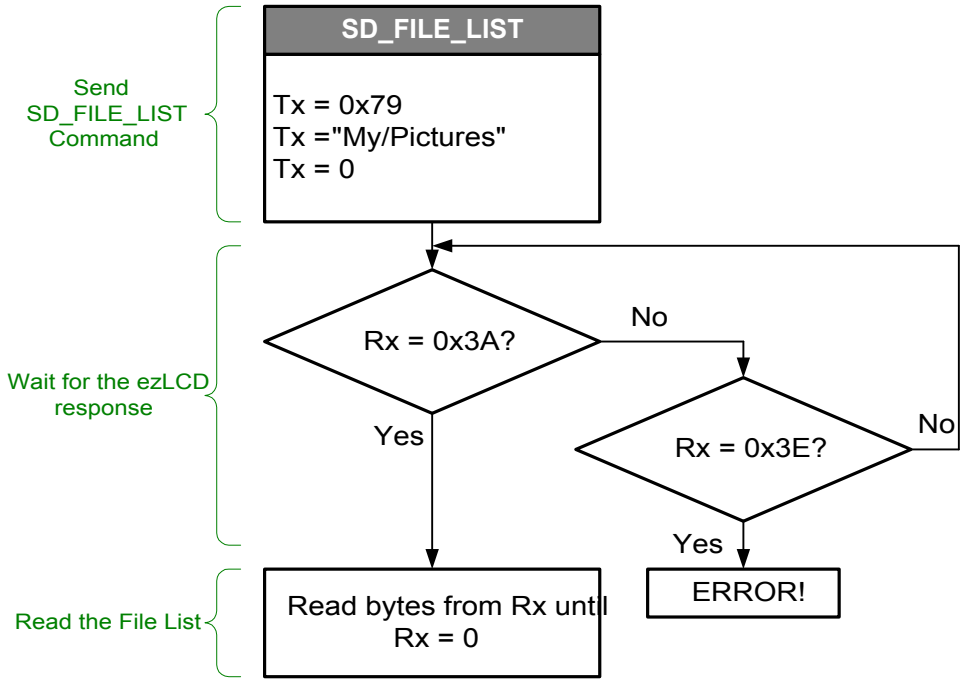
7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

Byte 0: 3Ehex, 62dec (Error)

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

Example:

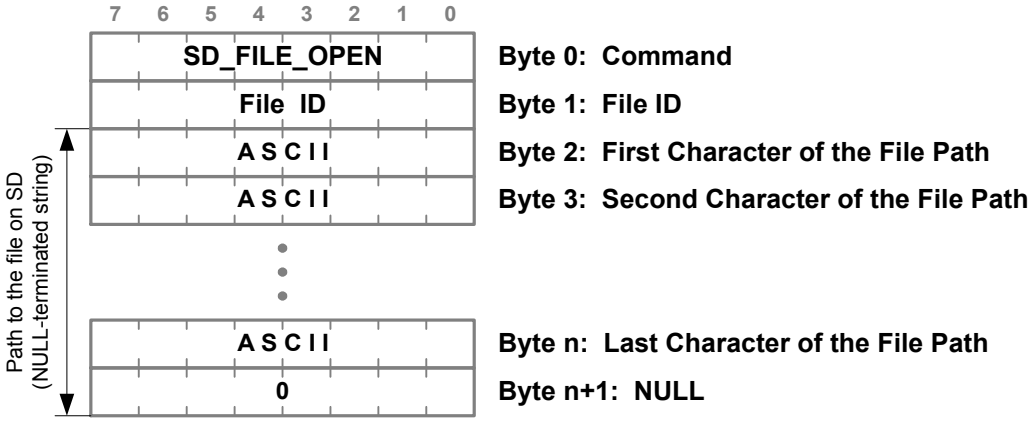
The following flow chart shows an example of reading the file list from the directory My/Pictures



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: **71**hex, **113**dec



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

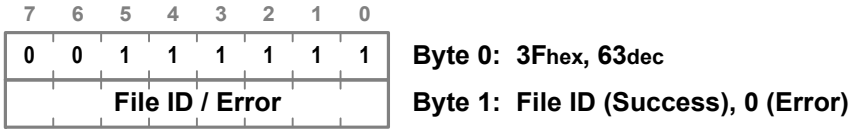
See Also: SD_FILE_CREATE, SD_FILE_CLOSE, SD_FILE_CLOSE_ALL

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 temporary 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
```

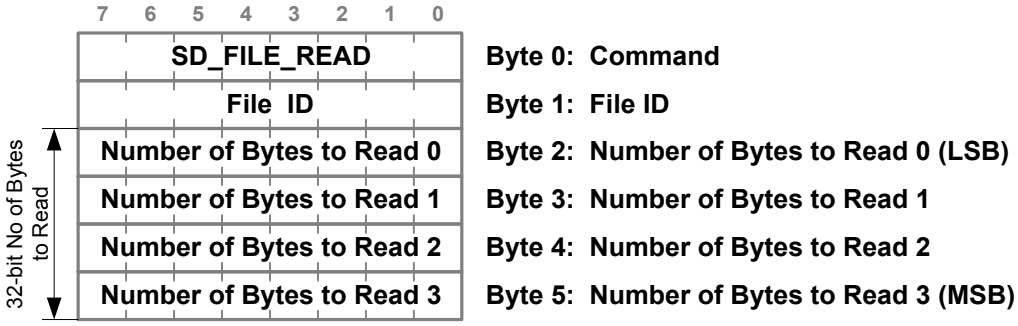
In case of the failure, the following sequence will be sent by the ezLCD+:

```
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**hex, **117**dec



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

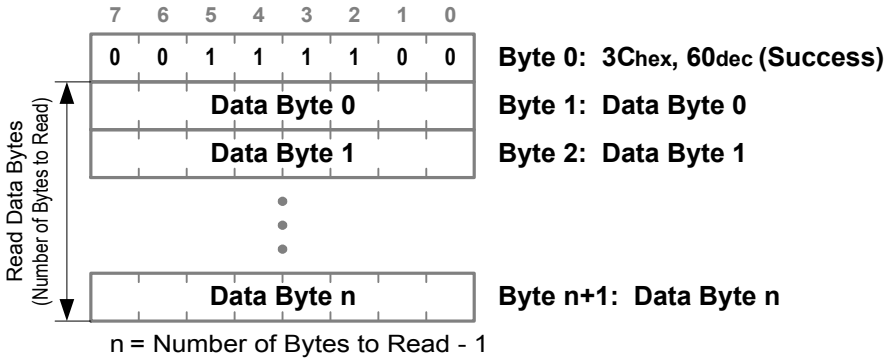
See Also: SD_FILE_OPEN, SD_FILE_CLOSE, SD_FILE_CLOSE_ALL

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

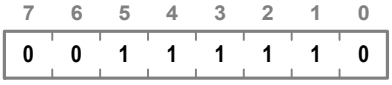
After receiving the SD_FILE_READ command, the ezLCD+ responds with either of the following sequences:

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**:

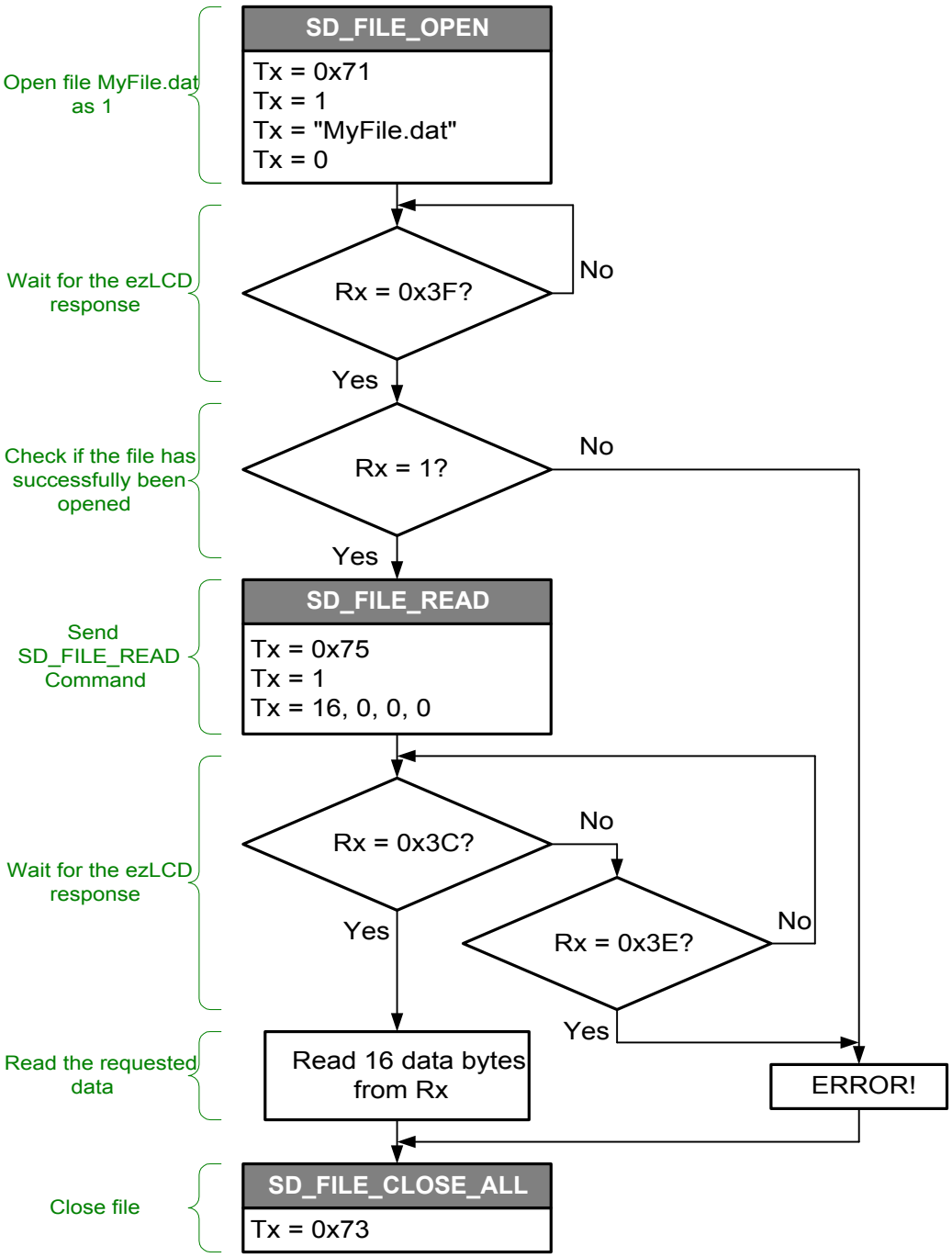


Byte 0: 3E$_{hex}$, 62$_{dec}$ (Error)

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

Example:

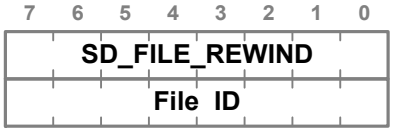
The following flow chart shows an example of reading first 16 bytes from the file MyFile.dat



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: **7A**hex, **122**dec



Byte 0: Command

Byte 1: File ID

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

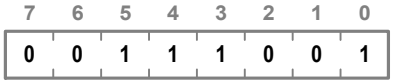
About the File ID:
File ID is returned in the response to the SD_FILE_OPEN command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

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

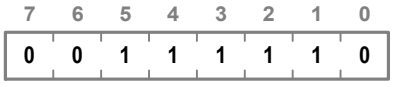
After receiving the SD_FILE_REWIND command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 39hex, 57dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (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:

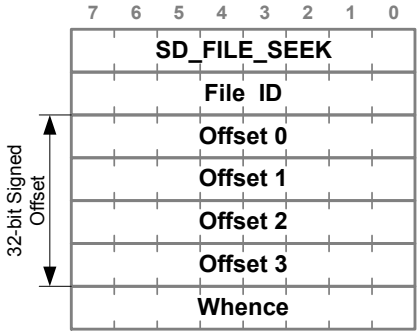
39 hex

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: **7C**hex, **124**dec



Byte 0: Command
Byte 1: File ID
Byte 2: Offset 0 (LSB)
Byte 3: Offset 1
Byte 4: Offset 2
Byte 5: Offset 3 (MSB)
Byte 6: Whence **0 - From File Beginning (SD_SEEK_SET)**
1 - From Current File Position Index (SD_SEEK_CUR)
2 - From File End (SD_SEEK_END)

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.

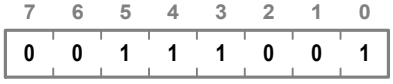
About the File Position Index
The File Position Index specifies the Read/Write position offset (in bytes) from the beginning of the file. Upon opening of the file, the File Position Index is set to 0. The File Position Index is incremented by the subsequent read or write operations on the opened file.

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

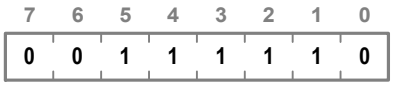
After receiving the SD_FILE_SEEK command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 39hex, 57dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (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.

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

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

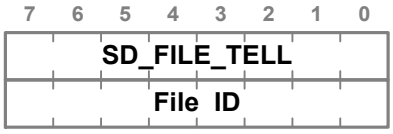
39 hex

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**hex, **123**dec



Byte 0: Command
Byte 1: File ID

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

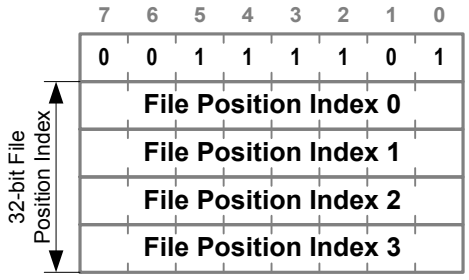
About the File ID:
File ID is returned in the response to the SD_FILE_OPEN command. It identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

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

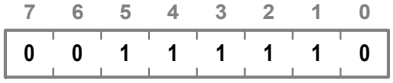
After receiving the SD_FILE_TELL command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Dhex, 61dec
Byte 1: File Position Index 0 (LSB)
Byte 2: File Position Index 1
Byte 3: File Position Index 2
Byte 4: File Position Index 3 (MSB)

In case of an **error**:



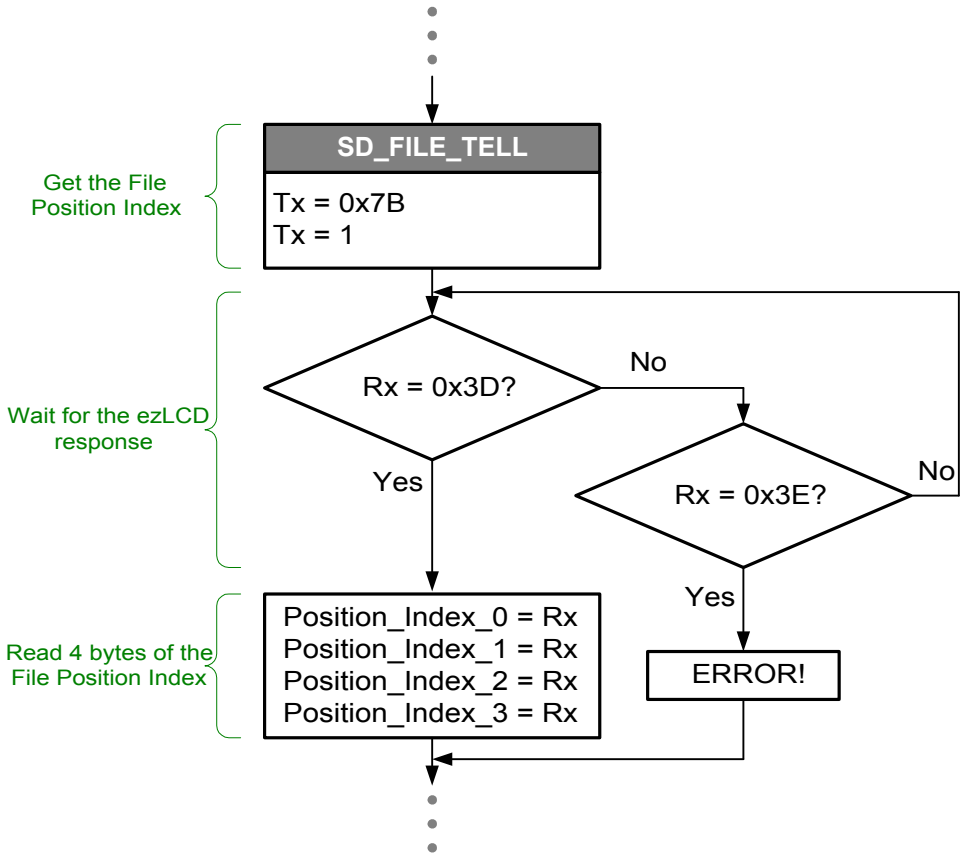
Byte 0: 3Ehex, 62dec (Error)

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:

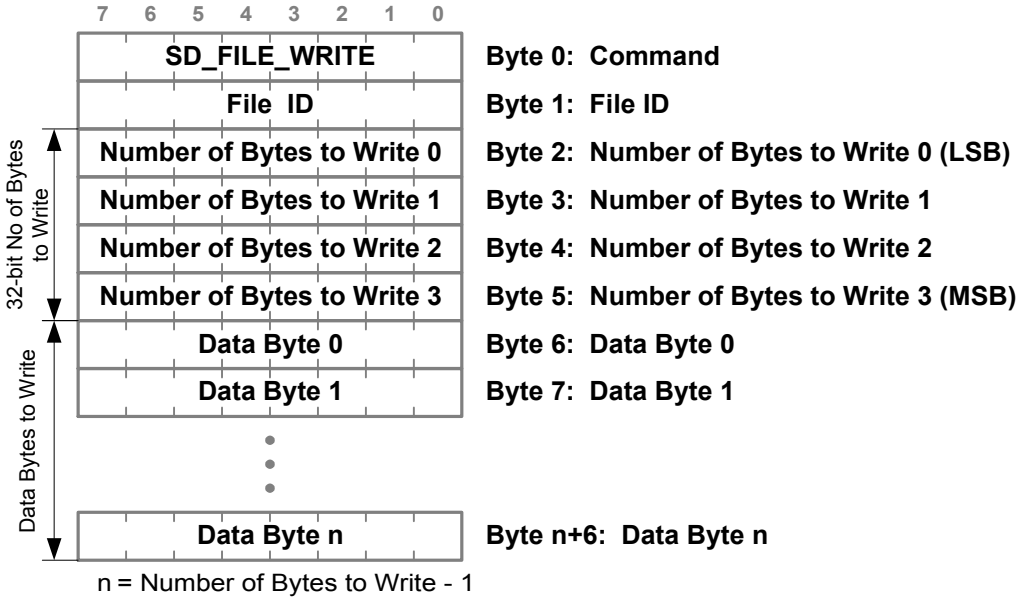
The following flow chart shows an example of getting the File Position Index of the opened file with File Id = 1.



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: **77**hex, **119**dec



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.

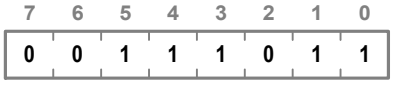
See Also: SD_FILE_CREATE, SD_FILE_OPEN, SD_FILE_CLOSE, SD_FILE_CLOSE_ALL

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

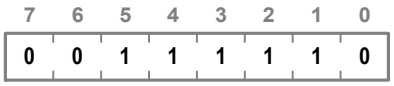
After receiving the SD_FILE_WRITE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Bhex, 59dec (Success)

In case of an **error**:

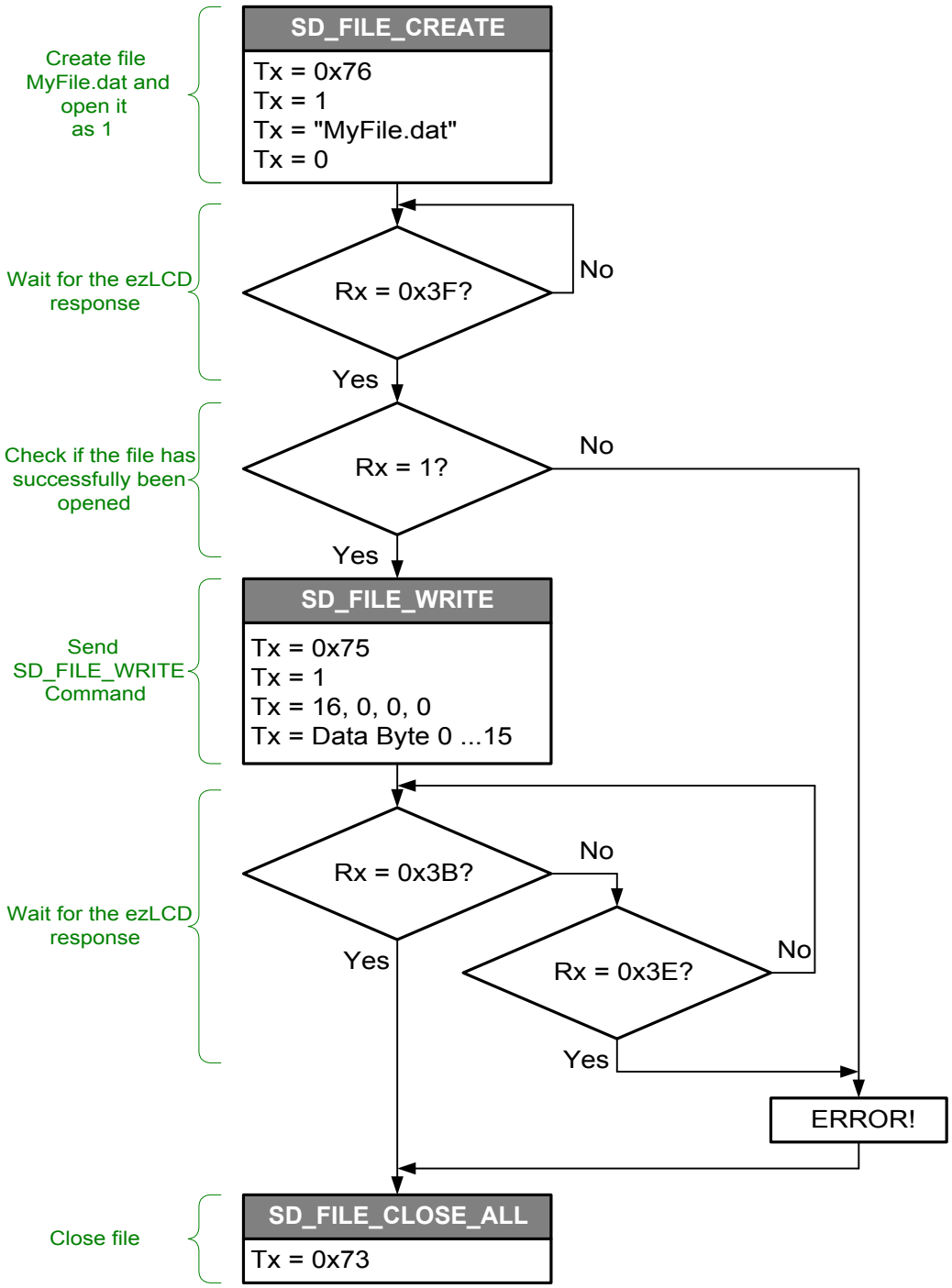


Byte 0: 3Ehex, 62dec (Error)

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

Example:

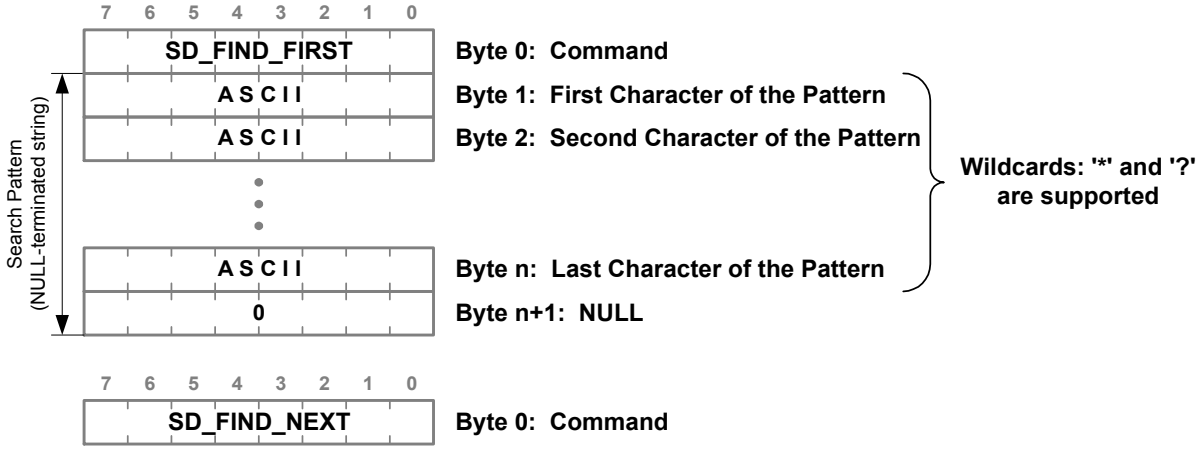
The following flow chart shows an example of writing 16 bytes into the created file MyFile.dat



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 : 4Bhex, **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**:

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

Examples:

3Ahex *Start*
`whatever.txt` *file*
0 *End (NULL)*

or

3Ahex *Start*
`Pictures/` *directory*
0 *End (NULL)*

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

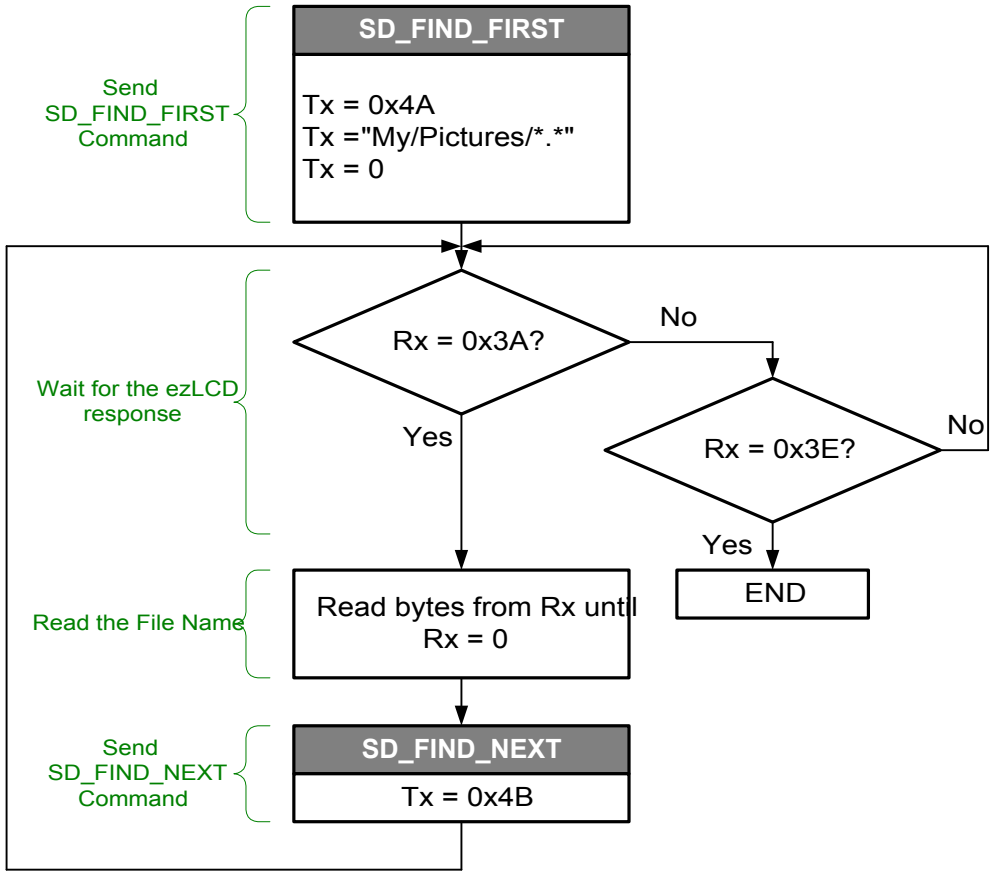
7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

Byte 0: 3Ehex, 62dec (Error)

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

Example:

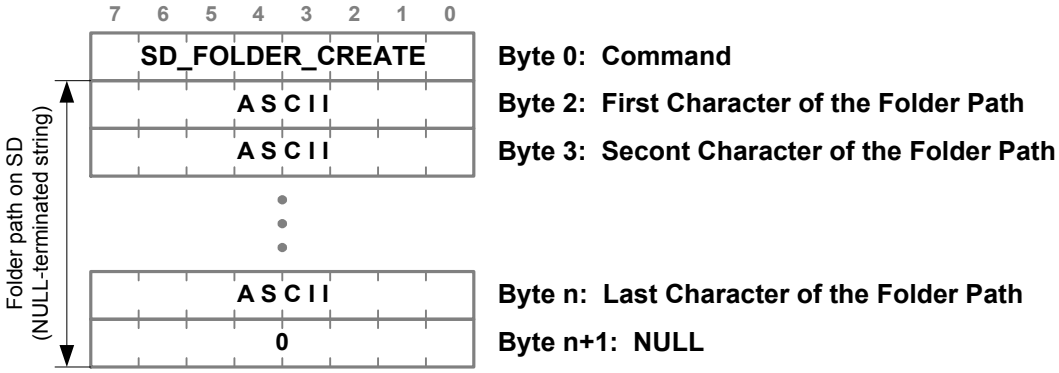
The following flow chart shows an example of reading the file list from the directory My/Pictures



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: **46**hex, **70**dec



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

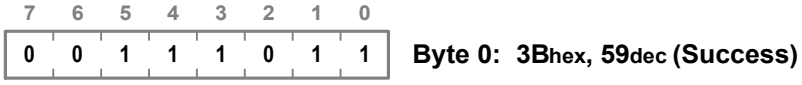
About the Folder Path:

- 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.

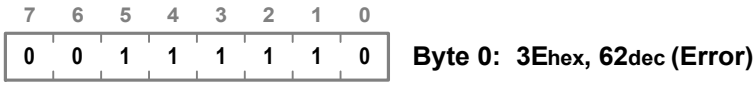
ezLCD+ Response

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
```

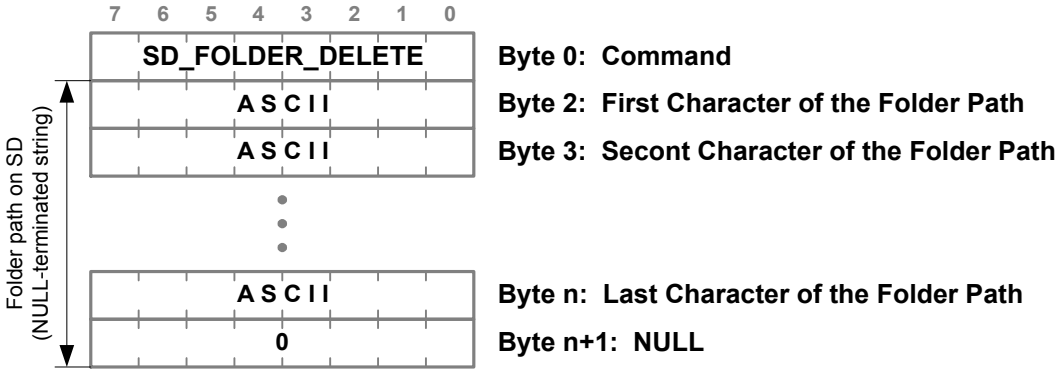
In case of the failure, the following sequence will be sent by the ezLCD+:

```
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: **4D**hex, **77**dec



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

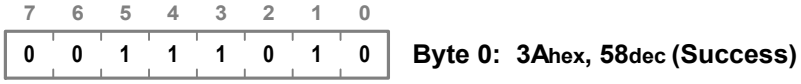
About the Folder Path:

- 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

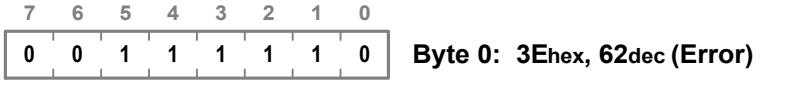
After receiving the SD_FOLDER_DELETE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Ahex, 58dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (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
```

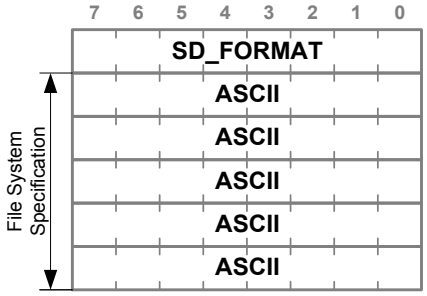
In case of the failure, the following sequence will be sent by the ezLCD+:

```
3E hex
```

8.66 SD_FORMAT

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

Code: **4F**hex, **79**dec



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 31hex.
- Supported file systems: FAT12, FAT16, FAT32

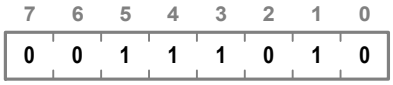
About the supported file systems

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

ezLCD+ Response

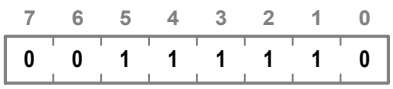
After receiving the SD_FORMAT command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Ahex, 58dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (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:

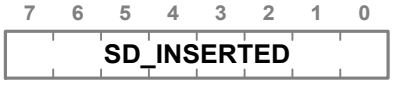
3A hex

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

3E hex

8.67 SD_INSERTED

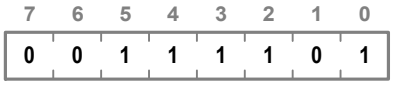
Description: Checks if the SD card is inserted
Code: **49**hex, **73**dec



Byte 0: Command

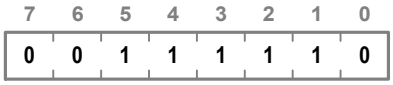
ezLCD+ Response
After receiving the SD_INSERTED command, the ezLCD+ responds with either of the following sequences:

If an SD card is inserted in the SD slot:



Byte 0: 3Dhex, 61dec (SD is inserted)

If there is no card inserted in the SD slot:



Byte 0: 3Ehex, 62dec (SD not inserted / Error)

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

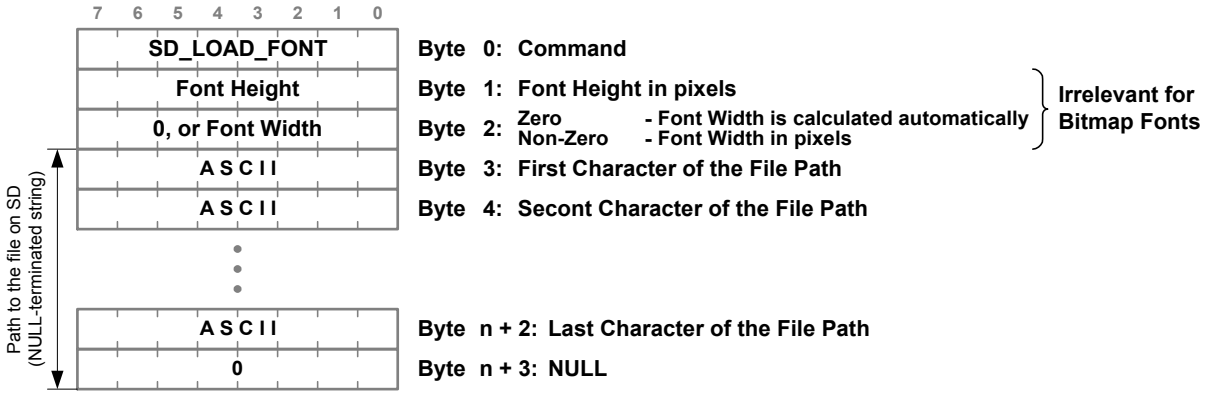
If an SD card is present in the SD slot:
```
3D hex
```

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: **92**hex, **146**dec



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

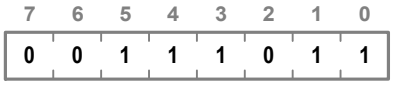
See Also: SELECT_FONT, SET_FT_FONT

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

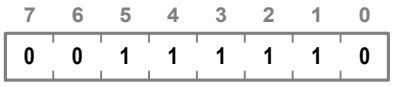
After execution of the SD_LOAD_FONT command, the ezLCD+ responds with either of the following sequences:

In case of **success**:



Byte 0: 3Bhex, 59dec (Success)

In case of an **error**:



Byte 0: 3E$_{hex}$, 62$_{dec}$ (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)
```

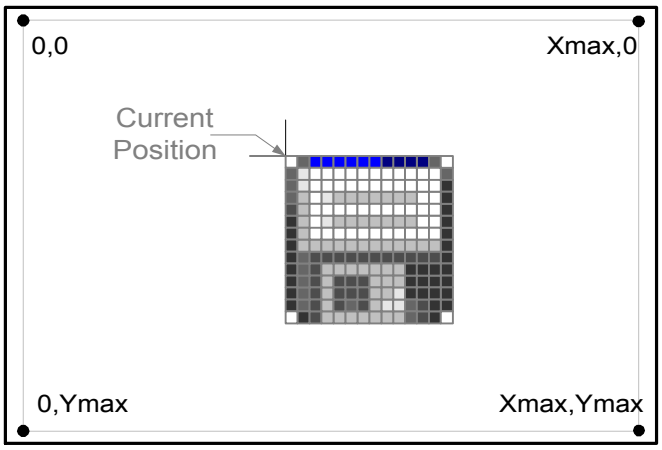
If the font has been loaded successfully, the ezLCD+ responds with:
3B hex

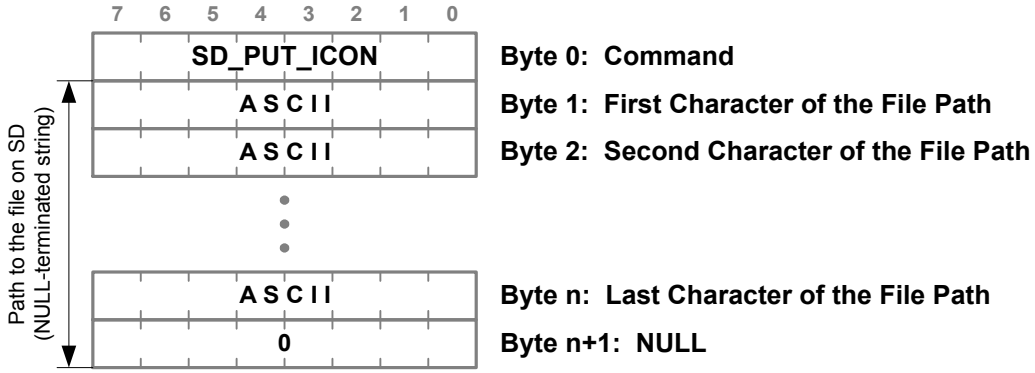
In case of failure, the following byte will be sent by the ezLCD+:
3E hex

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: **70**hex, **112**dec





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
```

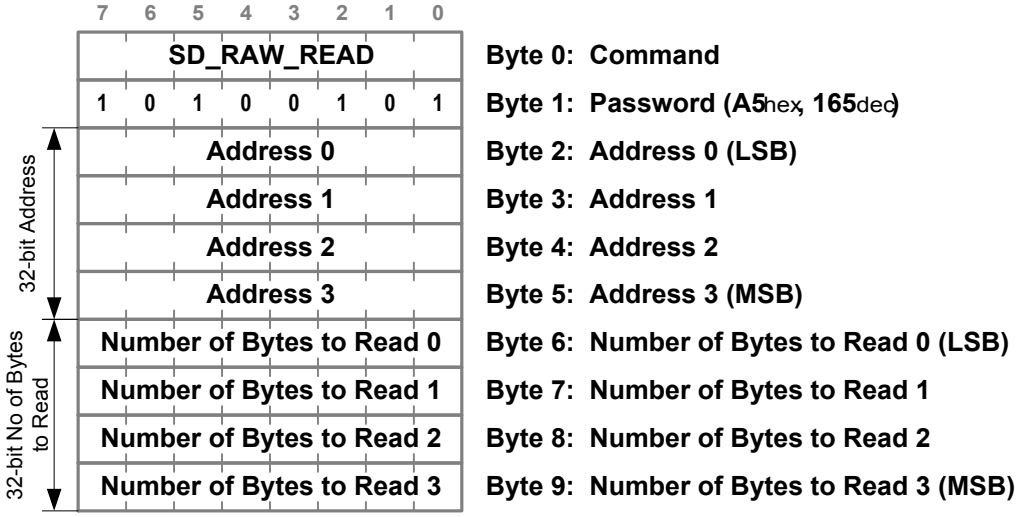


```
   0          0 dec (x MSB)
  60         60 dec (x LSB)
   0          0 dec (y MSB)
  43         43 dec (y LSB)
SD_PUT_ICON  70 hex
 'e'         65 hex
 'z'         7A hex
 'L'         4C hex
 'C'         43 hex
 'D'         44 hex
 '.'         2E hex
 'j'         6A hex
 'p'         70 hex
 'g'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


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**hex, **126**dec

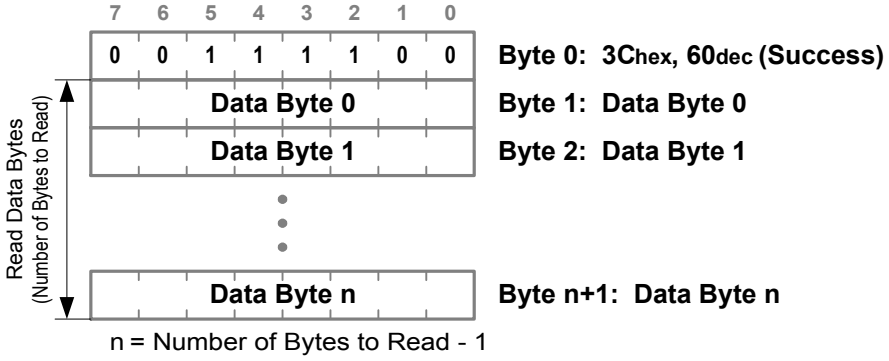


See Also: SD_RAW_WRITE

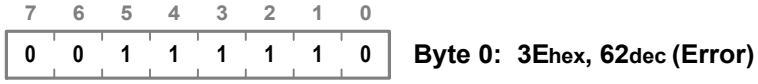
ezLCD+ Response

After receiving the SD_RAW_READ command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



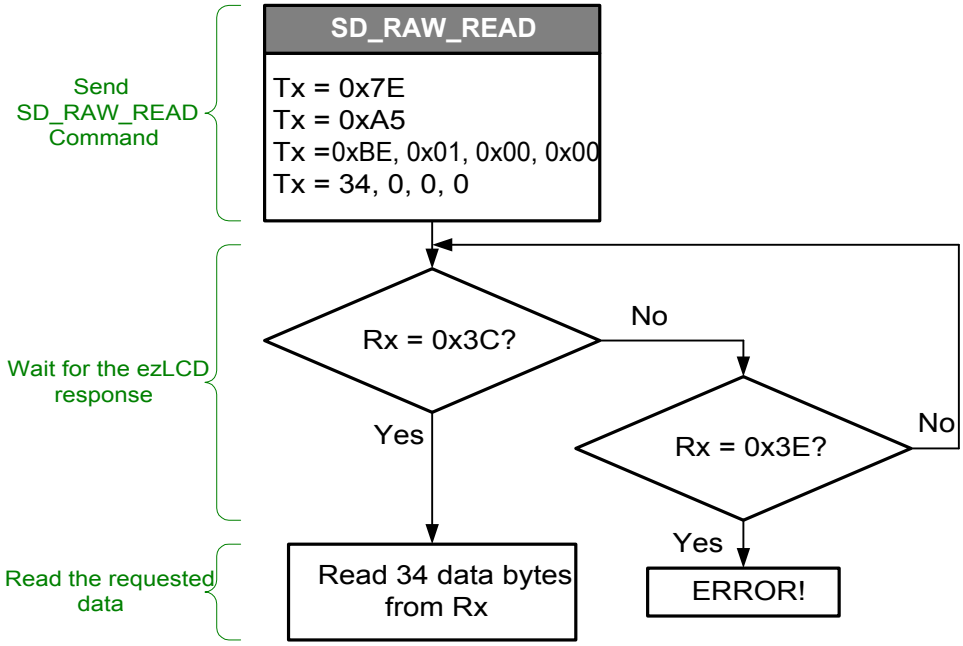
In case of an **error**:



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

Example:

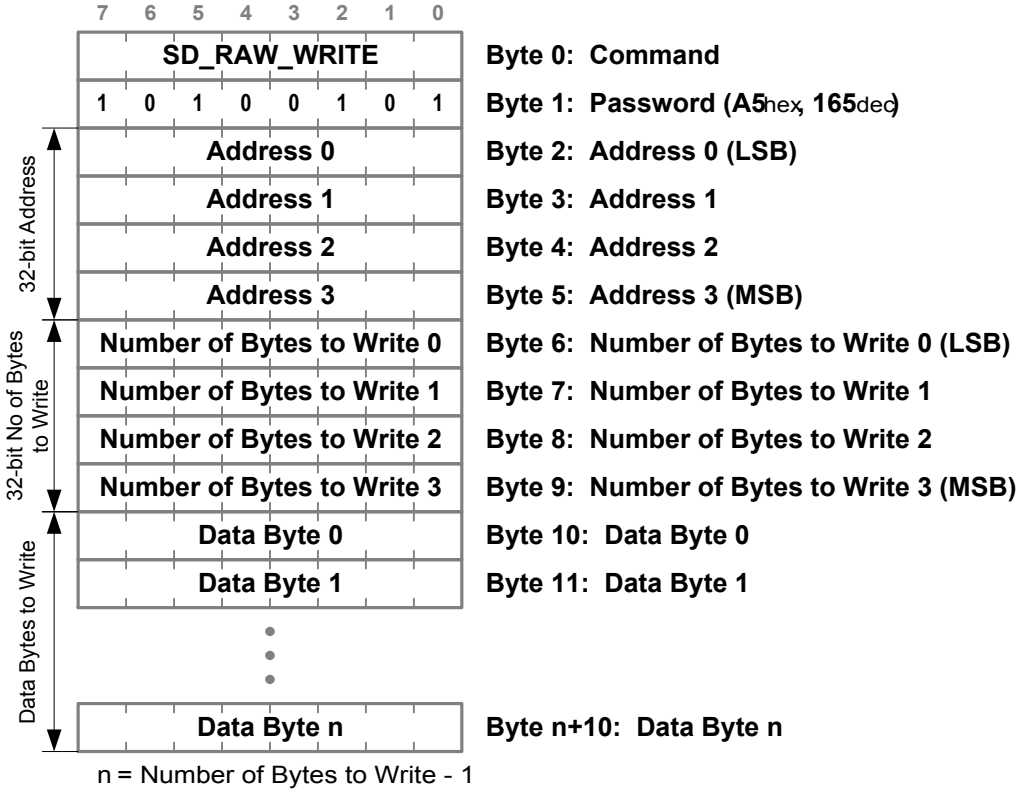
The following flow chart shows an example of reading 34 bytes starting from the SD address 000001BEhex**.**



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**hex, **127**dec



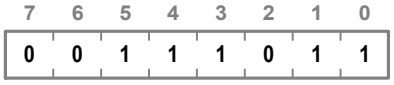
Warning! This command performs raw write on SD. Use it with caution. It may overwrite the existing SD files or corrupt the SD file formatting.

See Also: SD_RAW_READ

ezLCD+ Response

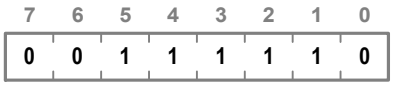
After receiving the SD_RAW_WRITE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Bhex, 59dec (Success)

In case of an **error**:

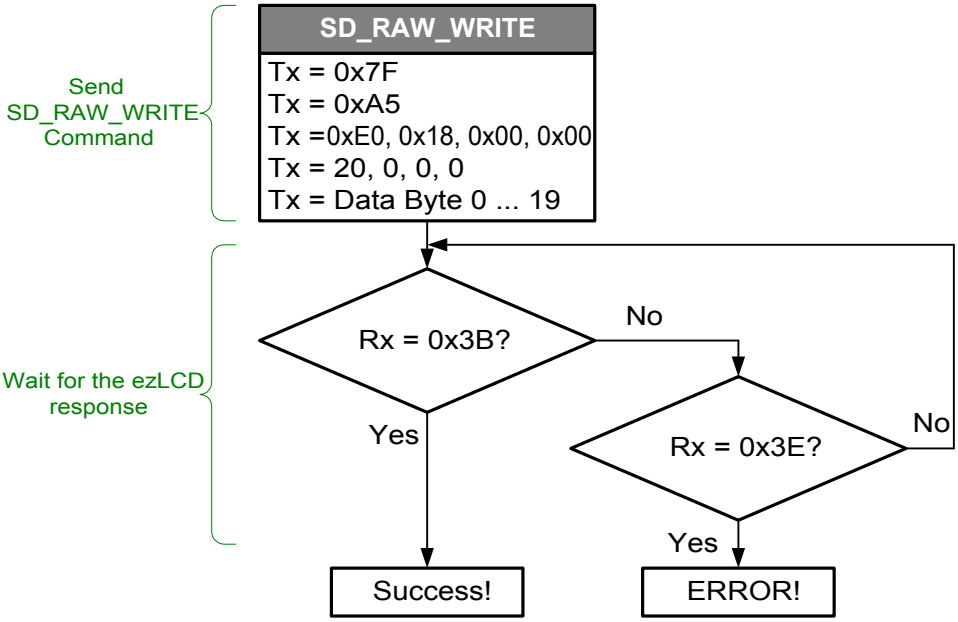


Byte 0: 3Ehex, 62dec (Error)

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

Example:

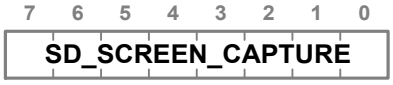
The following flow chart shows an example of writing 20 bytes starting from the SD address 000018E0hex**.**



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: **44**hex, **68**dec



Byte 0: Command

This command is helpful when writing the documentation of your ezLCD+ project, because the captured screen images may be used as examples.

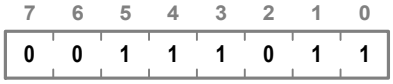
Screen capture files have names "Scr_xxxx.bmp", where xxxx is a consecutive number. For example: Scr_0001.bmp, Scr_0002.bmp, etc. The files are created in the "Scr_Cap" SD folder. If the SD does not have the "Scr_Cap" folder, it will be created automatically.

Notes: SD card has to be formatted in the supported file system.
This command may take up to 2 seconds to execute.

ezLCD+ Response

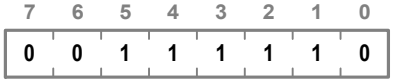
After execution of the SD_SCREEN_CAPTURE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Bhex, 59dec (Success)

In case of an **error**:



Byte 0: 3Ehex, 62dec (Error)

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

Example:

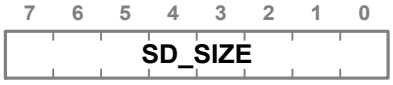
The following sequence will save the image of the displayed screen to the SD file.
SD_SCREEN_CAPTURE 44 hex

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**hex, **120**dec

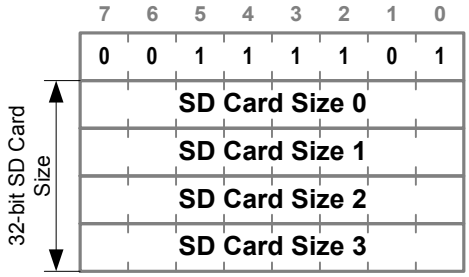


Byte 0: Command

ezLCD+ Response

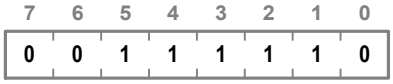
After receiving the SD_SIZE command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Dhex, 61dec
Byte 1: SD Card Size 0 (LSB)
Byte 2: SD Card Size 1
Byte 3: SD Card Size 2
Byte 4: SD Card Size 3 (MSB)

In case of an **error**:

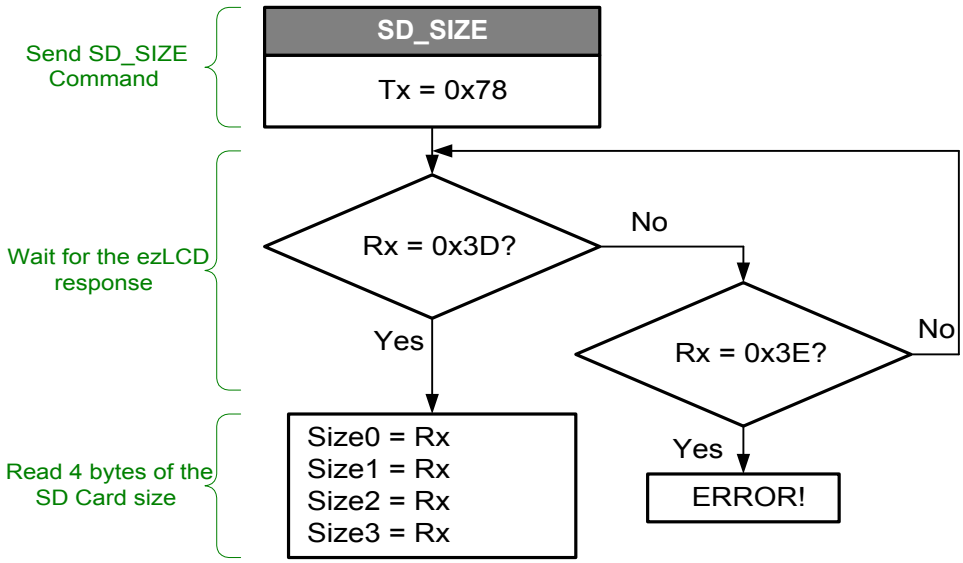


Byte 0: 3Ehex, 62dec (Error)

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

Example:

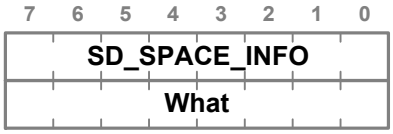
The following flow chart shows an example of getting the size of the SD Card.



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: **48**hex, **72**dec



Byte 0: Command

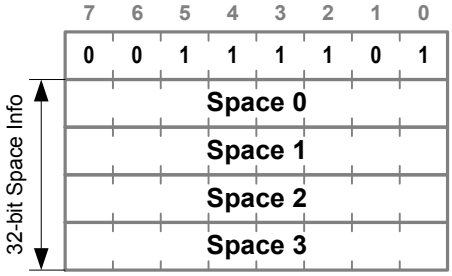
Byte 1: What
1 - Get Free Space
2 - Get Used Space
3 - Get Bad Space
Any Other Number - Get Total Formatted Space

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

ezLCD+ Response

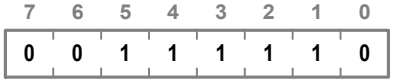
After receiving the SD_SPACE_INFO command, the ezLCD+ responds with either of the following sequences:

In case of the **success**:



Byte 0: 3Dhex, 61dec
Byte 1: Space 0 (LSB)
Byte 2: Space 1
Byte 3: Space 2
Byte 4: Space 3 (MSB)

In case of an **error**:



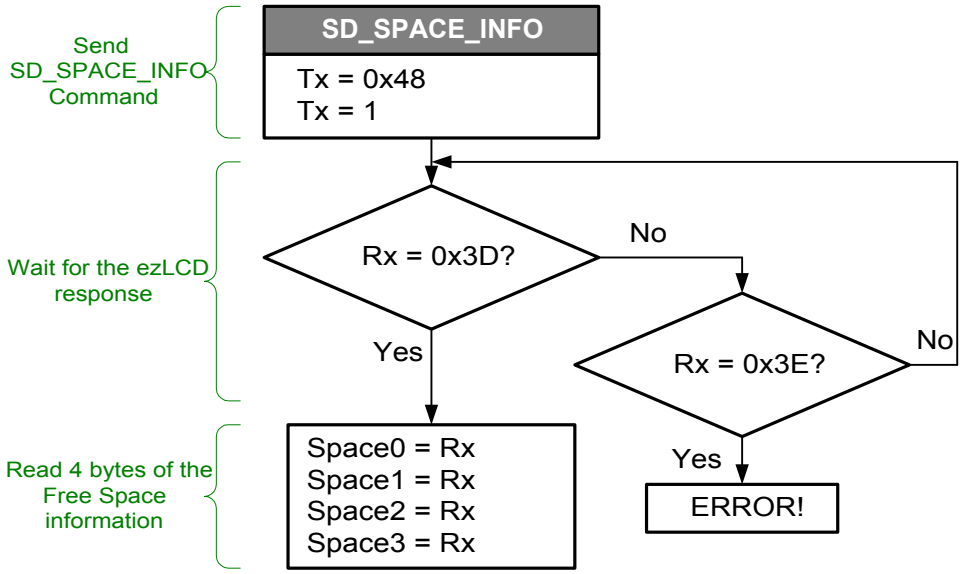
Byte 0: 3Ehex, 62dec (Error)

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.

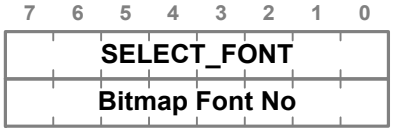
Example:

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: **2B**hex, **43**dec



Byte 0 (Command)

Byte 1 (Bitmap Font No: 0 to 255)

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

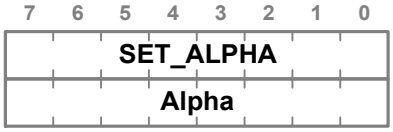
Example:

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: **20**hex, **32**dec



Byte 0 (Command)

Byte 1 (Alpha: 0 to 255)

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:

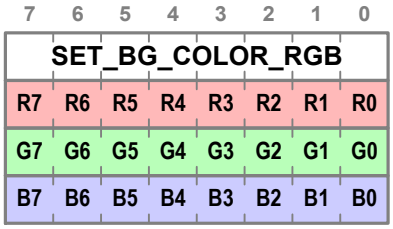
The following sequence will draw a semi-transparent color filled circle

```
SET_ALPHA       20 hex
  128          128 dec (Alpha)
CIRCLE_RH_FILL  99 hex
  0              0 dec (radius MSB)
 80             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: **32**hex, **50**dec



Byte 0 (Command)
Byte 1 (Red)
Byte 2 (Green)
Byte 3 (Blue)

See Also: PRINT_CHAR_BG, PRINT_STRING_BG

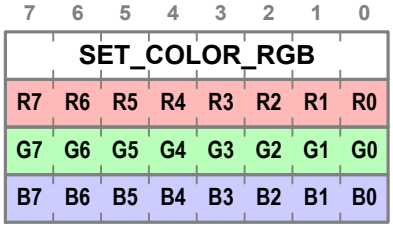
Example:

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: **31**hex, **49**dec



Byte 0 (Command)
Byte 1 (Red)
Byte 2 (Green)
Byte 3 (Blue)

See Also: CLS, PLOT

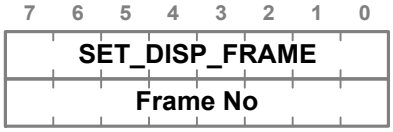
Example:

The following sequence will fill the whole screen with green.

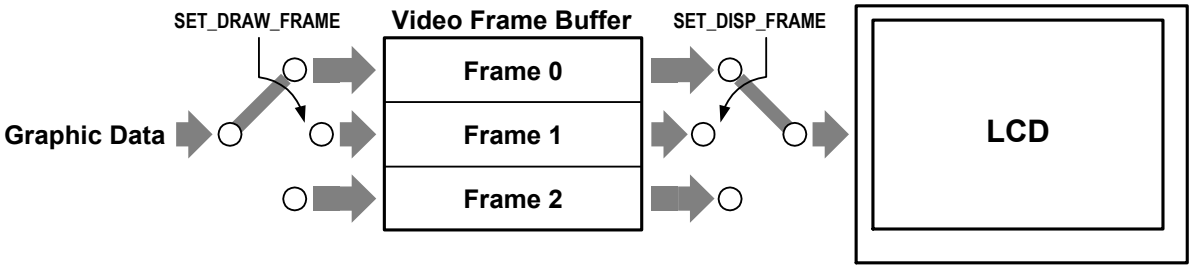
```
SET_COLOR_RGB  31 hex
 Red            0
 Green         FF hex
 Blue           0
CLS            21 hex
```

8.79 SET_DISP_FRAME

Description: Sets the Frame to be displayed on the screen.
Code: **52**hex, **82**dec



Byte 0: Command
Byte 1: Frame No (0 to 2)



Ref: Frames

See Also: SET_DRAW_FRAME

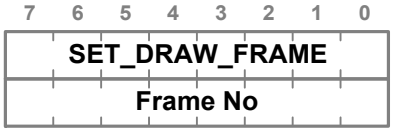
Example:

The following sequence will set the display to Frame 0.

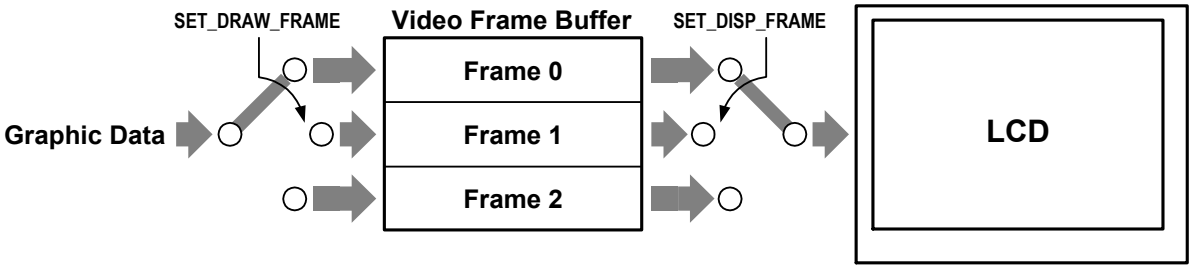
```
SET_DISP_FRAME   52 hex
   0              0 (Frame No)
```


8.80 SET_DRAW_FRAME

Description: Sets the Frame that ezLCD+ commands draw on.
Code: **51**hex, **81**dec



Byte 0: Command
Byte 1: Frame No (0 to 2)



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: **5C**hex, **92**dec

7	6	5	4	3	2	1	0		
SET_EDIT_RECT								Byte 0:	Command
x15	x14	x13	x12	x11	x10	x9	x8	Byte 1:	Edit Rectangle upper-left corner X-coordinate
x7	x6	x5	x4	x3	x2	x1	x0	Byte 2:	
y15	y14	y13	y12	y11	y10	y9	y8	Byte 3:	Edit Rectangle upper-left corner Y-coordinate
y7	y6	y5	y4	y3	y2	y1	y0	Byte 4:	
Width MSB								Byte 5:	Edit Rectangle Width
Width LSB								Byte 6:	
Height MSB								Byte 7:	Edit Rectangle Height
Height LSB								Byte 8:	

See Also: REPLACE_COLOR

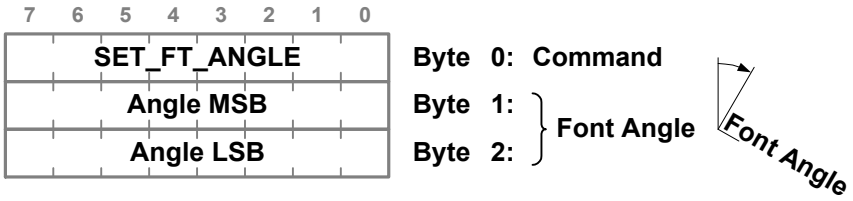
Example:

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

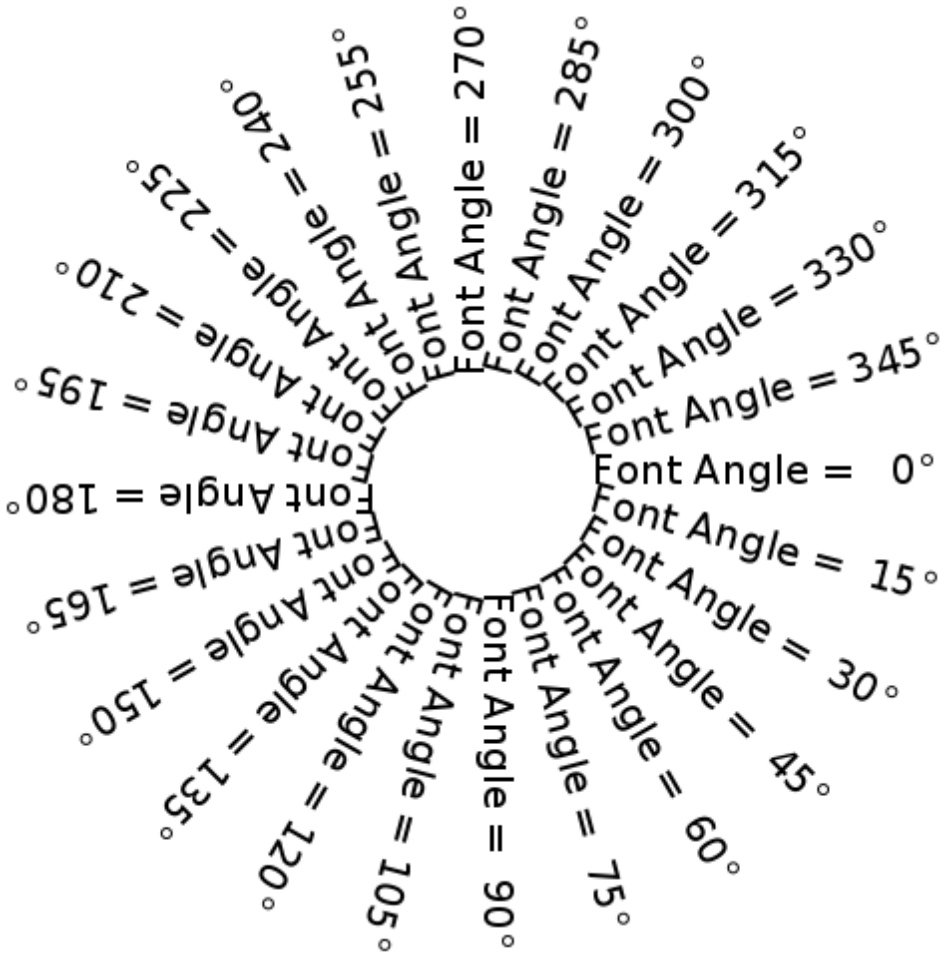
```
SET_EDIT_RECT  5C  hex
   1            1 dec (X MSB)
  64           64 dec (X LSB)
   0            0 dec (Y MSB)
 240          240 dec (Y LSB)
   0            0 dec (Width MSB)
 100          100 dec (Width LSB)
   0            0 dec (Height MSB)
  50           50 dec (Height LSB)
REPLACE_COLOR  5D  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°
```


16384dec = 4000hex = 360° = 0°

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

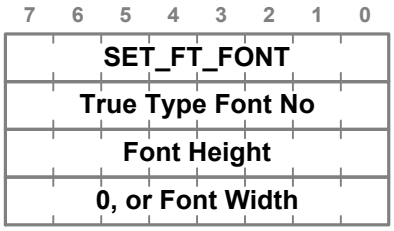
Example:

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: **91**hex, **145**dec



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:

Ref: Fonts, True Type Fonts

See Also: SET_FT_UNIBASE, PRINT_STRING, PRINT_CHAR, PRINT_FT_UNISTRING, PRINT_FT_UNICHAR

Example:

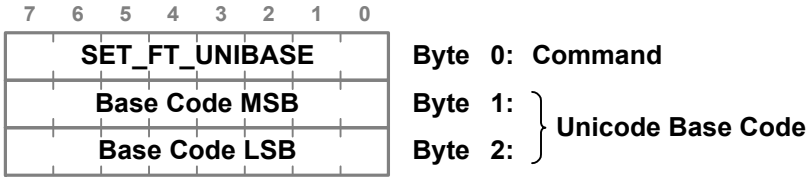
The following sequence will print a black True Type character 'M' using 24-pixel-height True Type Font 2.

```
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: **93**hex, **147**dec



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

ТжгтпузбйтарнхмтднцтиулостнфгртЯткЯшчтвне

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:

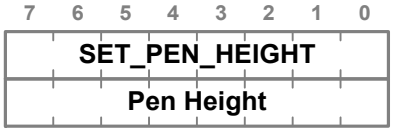
The picture above (Unicode Base = 03CE hex)**, has been generated using the following sequence:**

```
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: **82**hex, **130**dec



Byte 0 (Command)

Byte 1 (Pen Height: 0 to 255)

- Pen Width
  - Specifies the horizontal dimension the drawing line (in pixels)
  - Set by command: SET_PEN_SIZE

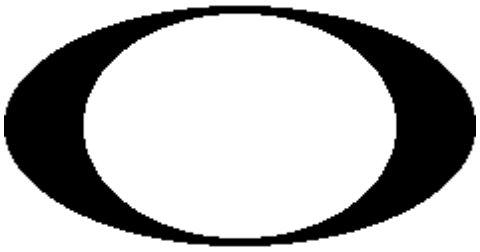
- Pen Height
  - Specifies the vertical dimension of Pen (in pixels), when drawing curves
  - Ignored when drawing straight lines
  - Set by commands: SET_PEN_SIZE and SET_PEN_HEIGHT

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:

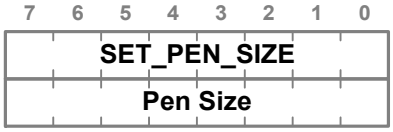


will be drawn by the following sequence:

```
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: **81**hex, **129**dec



Byte 0 (Command)

Byte 1 (Pen Size: 0 to 255)

- 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:

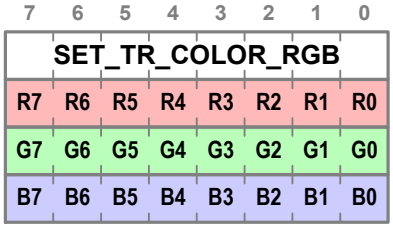
The following sequence will draw 8 pixel thick horizontal line:

```
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



Byte 0 (Command)
Byte 1 (Red)
Byte 2 (Green)
Byte 3 (Blue)

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

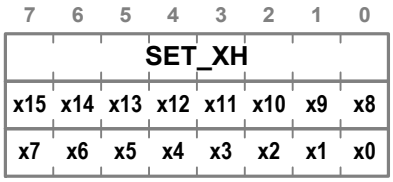
Example:

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: **6E**hex, **110**dec



Byte 0 (Command)
Byte 1 (x MSB)
Byte 2 (x LSB)

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: **33**hex, **51**dec

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

Ref: Screen Coordinates

See Also: PLOT, LINE_TO_XHYH, CIRCLE_RH

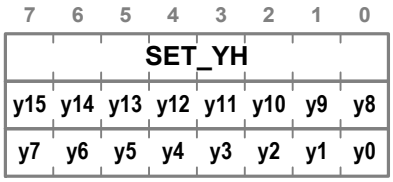
Example:

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)
```

8.90 SET_YH

Description: Sets only the Y-coordinate of the Current Position. X coordinate remains unchanged
Code: **6F**hex, **111**dec



Byte 0 (Command)
Byte 1 (y MSB)
Byte 2 (y LSB)

Ref: Screen Coordinates

See Also: SET_XH, SET_XHYH

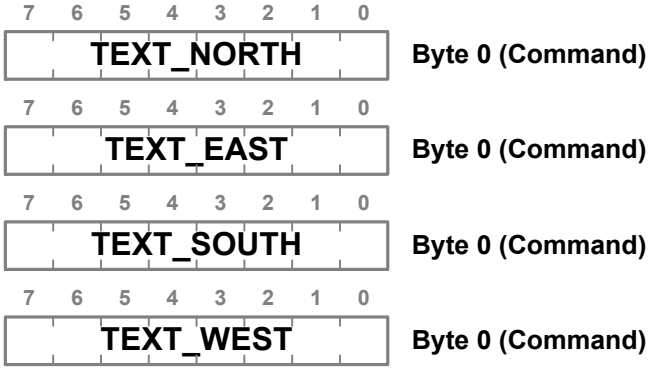
Example:

The following sequence will put a 2 blue points in the same column.

```
SET_COLOR_RGB  31 hex
 Red            0
 Green          0
 Blue          FF hex
SET_YH         6F hex
  0             0 dec (y MSB)
160           160 dec (y LSB)
PLOT           26 hex
SET_YH         6F hex
  0             0 dec (y MSB)
170           170 dec (y LSB)
PLOT           26 hex
```

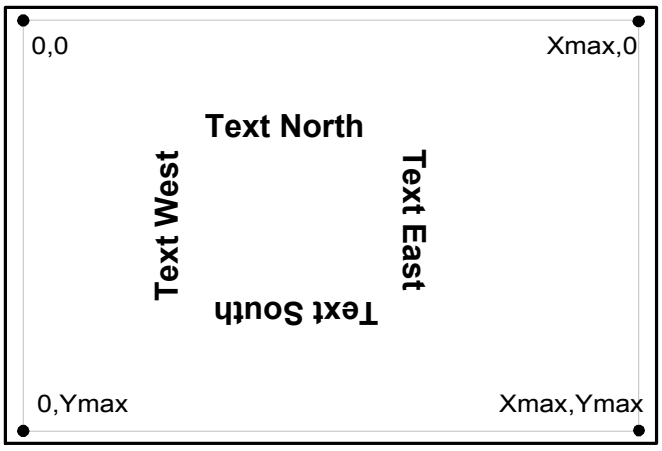

8.91 TEXT_NORTH, SOUTH, EAST, WEST

Description: Set the orientation of the text, as shown on the picture below.
Code: **TEXT_NORTH : 60**hex, **96**dec
TEXT_EAST : 61hex, **97**dec
TEXT_SOUTH : 62hex, **98**dec
TEXT_WEST : 63hex, **99**dec



For True Type 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.

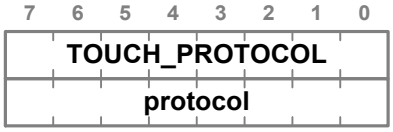
```
SET_XHYH       33 hex
   0            0 dec (x MSB)
  60           60 dec (x LSB)
   0            0 dec (y MSB)
```



```
  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**hex, **178**dec



Byte 0: Command

Byte 1: Protocol **1 - ezButton**
2 - cuButton
64 - CalibratedXY

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:

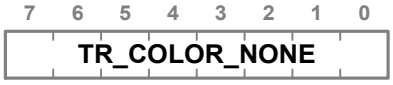
The following sequence will change the Touch Protocol to ezButton.

```
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



Byte 0: Command

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:

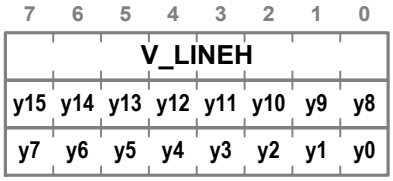
The following sequence will set the bitmap transparent color to nonr.

```
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
No of Bytes per Color	3	2	1	1	1
No of Bytes per Column (X)	2	2	1	1	1
No of Bytes per Row (Y)	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: **2F**hex, **47**dec

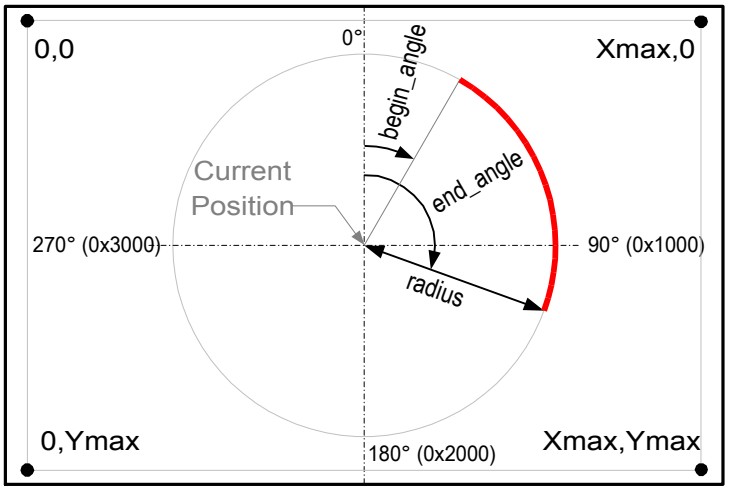
7	6	5	4	3	2	1	0	
ARC								**Byte 0 (Command)**
r a d i u s								**Byte 1 (Radius)**
begin_angle								**Byte 2 (Arc Begin Angle)**
end_angle								**Byte 3 (Arc End Angle)**

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:
`Angle_lcd = Angle_deg x 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.

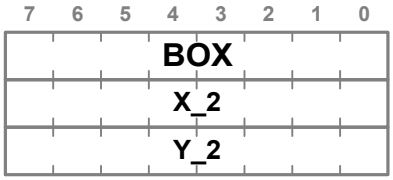
```
SET_COLOR       24 hex
GREEN           00111000 bin
SET_XY          25 hex
```



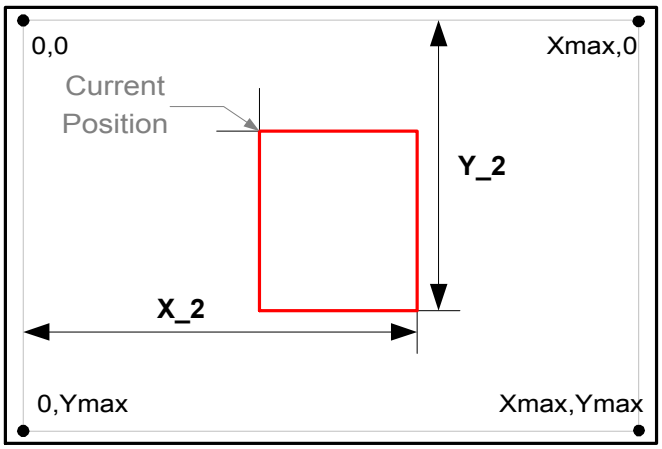
```
120           120 dec
 80            80 dec
ARC            2F hex
 60            60 dec (radius)
 32            32 dec (begin_angle = 45 degrees)
160           160 dec (end_angle = 225 degrees)
\
```


8.95.2 BOX

Description: Draws a rectangle.
Code: **42**hex, **66**dec



Byte 0 (Command)
Byte 1 (Corner Column)
Byte 2 (Corner Row)



See Also: SET_XY, BOX_FILL

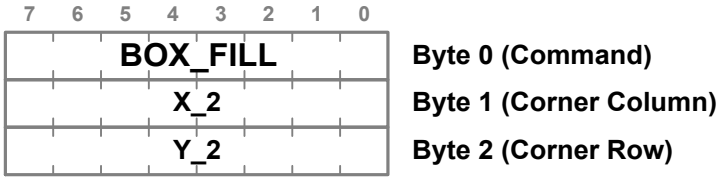
Example:

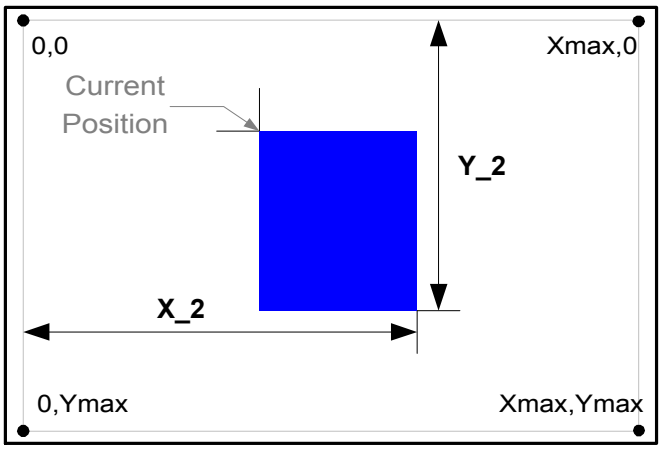
The following sequence will draw the red rectangle

```
SET_COLOR    24 hex
RED        00000111 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: **43**hex, **67**dec





See Also: SET_XY, BOX

Example:

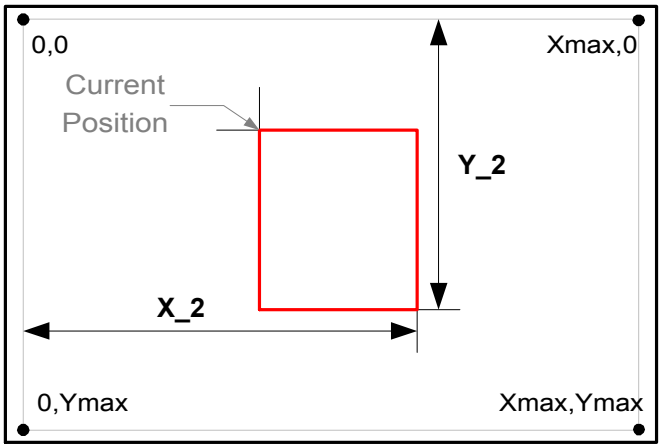
The following sequence will draw the rectangle filled with blue color

```
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	
BOXH								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x2 MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x2 LSB)**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 3 (y2)**



See Also: SET_XHY, BOXH_FILL

Example:

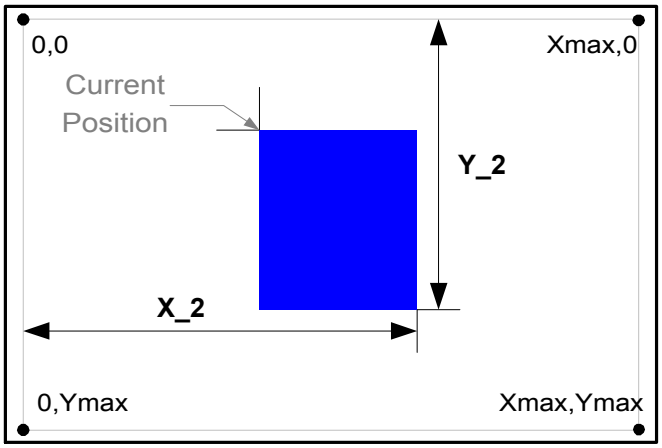
The following sequence will draw a red rectangle with the top left corner positioned at (95, 10) and the bottom right corner at (180, 120).

```
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	
BOXH_FILL								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x2 MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x2 LSB)**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 3 (y2)**



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

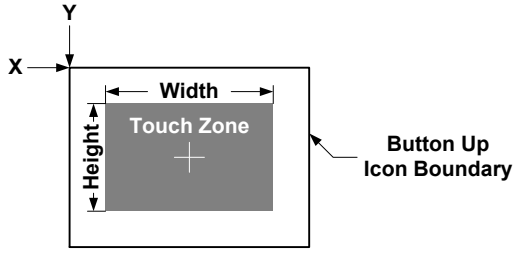
Description: Defines and draws a touch button
Code: **B0**hex, **176**dec

7	6	5	4	3	2	1	0	
BUTTON_DEF								**Byte 0: Command**
button_no								**Byte 1: Button No (0 to 63)**
state								**Byte 2: Initial State (1: Up, 2: Down, 3: Disabled, 4: Non-Visible)**
button_up_icon								**Byte 3: Icon No in User ROM for button Up (255 = none)**
button_down_icon								**Byte 4: Icon No in User ROM for button Down (255 = none)**
button_disabled_icon								**Byte 5: Icon No in User ROM for button Disabled (255 = none)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 6: Button upper-left corner X-coordinate MSB**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 7: Button upper-left corner X-coordinate LSB**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 8: Button upper-left corner Y-coordinate**
touch_zone_width								**Byte 9: Touch Zone width**
touch_zone_height								**Byte 10: Touch Zone height**

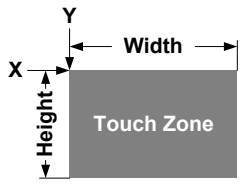
About the Touch Zone:
Touch Zone is the active touch response area of the button. It is specified by **With** (Byte 9) and **Height** (Byte 10).

- 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:

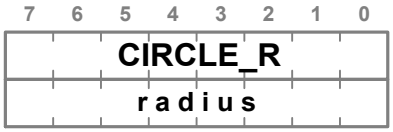
The following sequence will define the Button No. 4 with the following bitmaps:
- **Button Up Icon in User ROM: 8**
- **Button Down Icon in User ROM: 9**
- **No Icon for Button Disabled state**

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: **29**hex, **41**dec



Byte 0 (Command)

Byte 1 (radius)

See Also: SET_XY, SET_COLOR

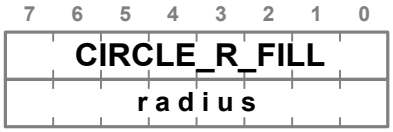
Example:

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: **39**hex, **57**dec



Byte 0 (Command)

Byte 1 (Radius)

See Also: SET_XY, SET_COLOR

Example:

The following sequence will draw a red filled circle in the middle of the screen.

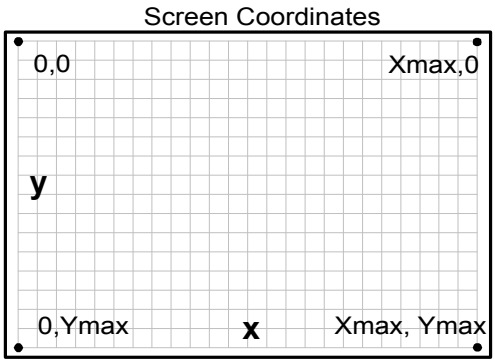
```
\SET_COLOR       24 hex
RED             00000111 bin
SET_XY           25 hex
120             120 dec
80               80 dec
CIRCLE_R_FILL    39 hex
60               60 dec
```


8.95.9 LINE_TO_XHY

Description: Draws a line in Current Color, from Current Position to the specified position.

Code: **88**hex, **136**dec

7	6	5	4	3	2	1	0	
LINE_TO_XHY								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x LSB)**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 3 (y)**



See Also: SET_XHY, SET_COLORH, PLOT

Example:

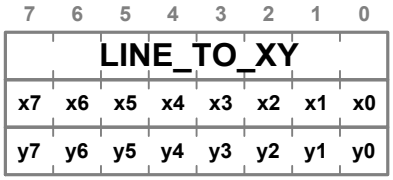
The following sequence will draw a red line across the screen.

```
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

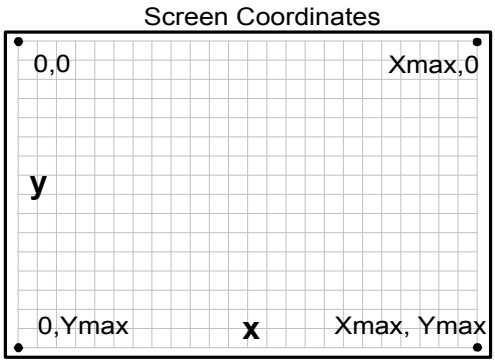
Code: **28**hex, **40**dec



Byte 0 (Command)

Byte 1 (x)

Byte 2 (y)



See Also: SET_XY, SET_COLOR, PLOT

Example:

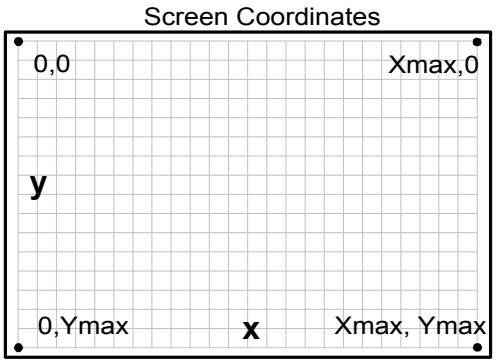
The following sequence will draw a red line across the screen.

```
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: **87**hex, **135**dec

7	6	5	4	3	2	1	0	
PLOT_XHY								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x LSB)**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 3 (y)**



See Also: SET_XHY, SET_COLORH, PLOT

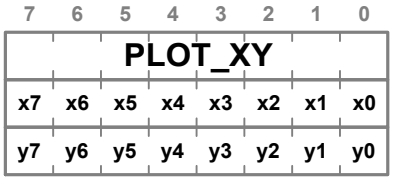
Example:

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

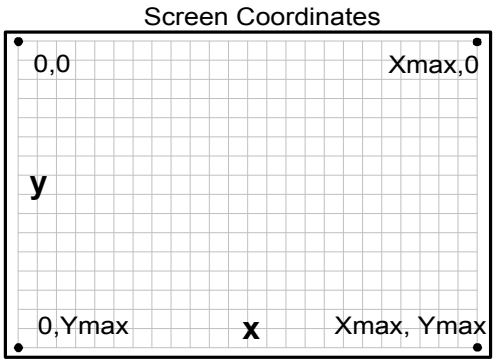
```
SET_COLORH   84 hex
RED_LSB      00000000 bin
RED_MSB      11111000 bin
PLOT_XHY     87 hex
  1           1 dec (x MSB)
 60          60 dec (x LSB 1*256+54=310)
117         117 dec (y)
```

8.95.12 PLOT_XY

Description: Plots a point in Current Color, at specified position.
Code: **27**hex, **39**dec



Byte 0 (Command)
Byte 1 (x)
Byte 2 (y)



See Also: SET_XY, SET_COLOR, PLOT

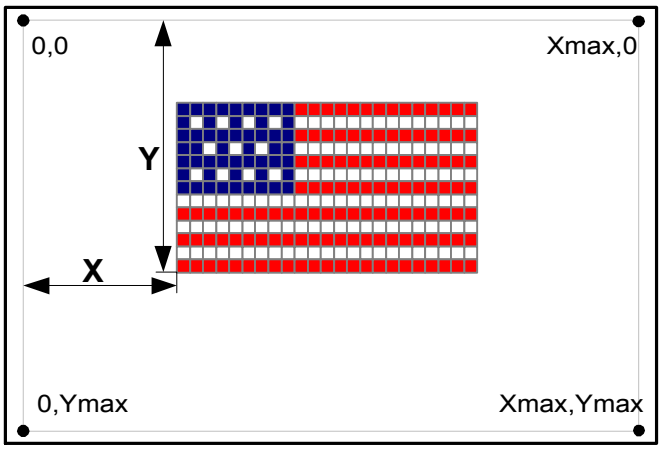
Example:

The following sequence will put the red point in the middle of the screen.

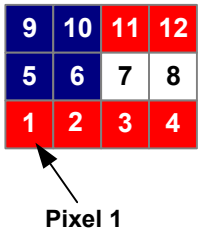
```
SET_COLOR    24 hex
RED          00000111 bin
PLOT_XY      27 hex
120          120 dec
80           80 dec
```

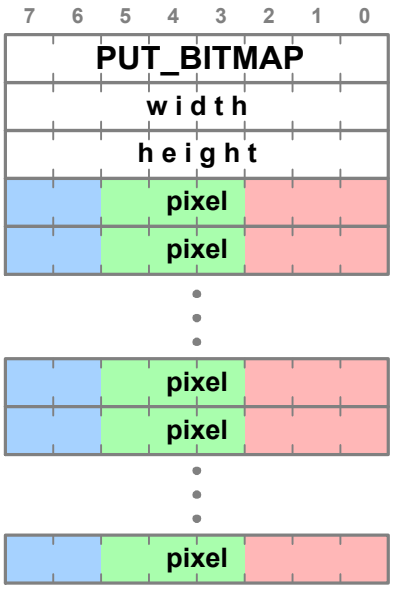

8.95.13 PUT_BITMAP

Description: Puts Bitmap on the screen starting at Current Position, then UP and RIGHT
Code: **2E**hex, **46**dec



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)

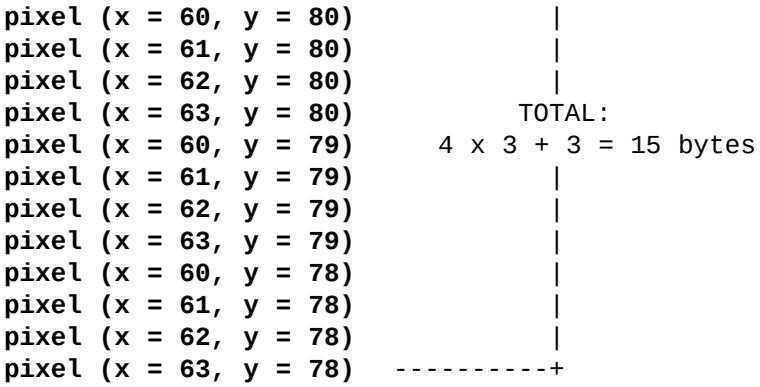
Note: The total number of bytes is: width x height + 3

See Also: SET_XY, SET_COLOR

Example:

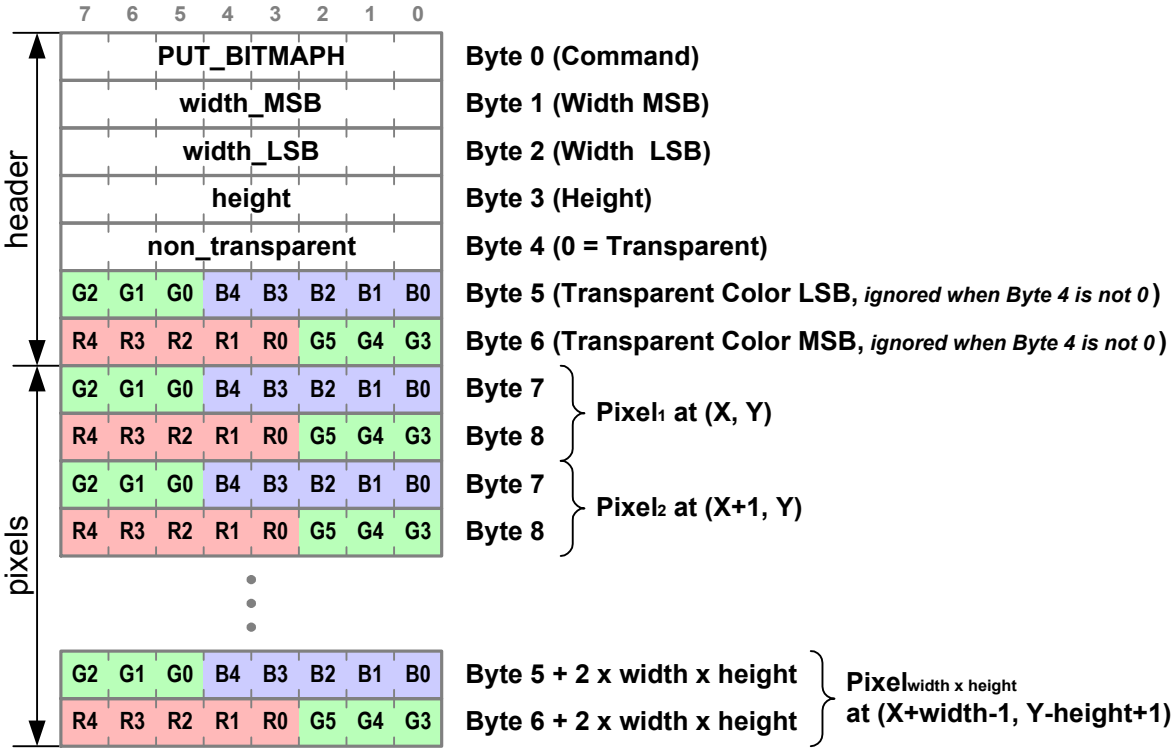
The following sequence will put 4x3 bitmap at x = 60, y = 80

```
SET_XY        25 hex
x             60 dec
y             80 dec
PUT_BITMAP    2E hex     ----------+--
width          4 dec               |
height         3 dec               |
```

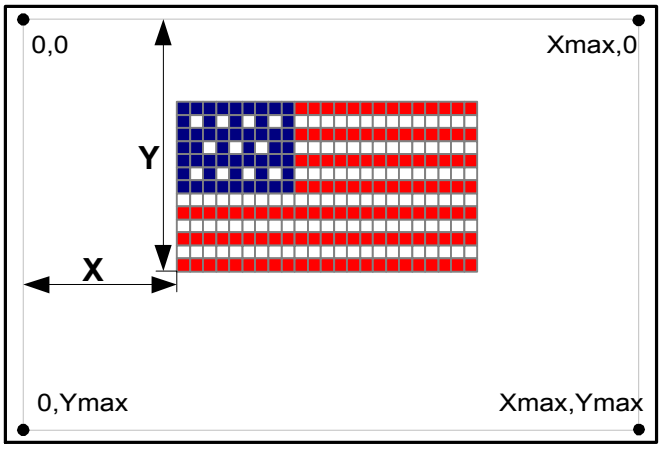



8.95.14 PUT_BITMAPH

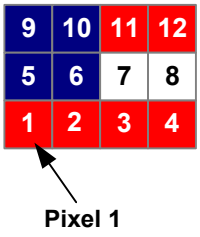
Description: Displays a Bitmap on the screen starting at Current Position, then RIGHT and UP
Code: **9E**hex, **158**dec



Notes: 1. The total number of bytes is: 2 x width x 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.

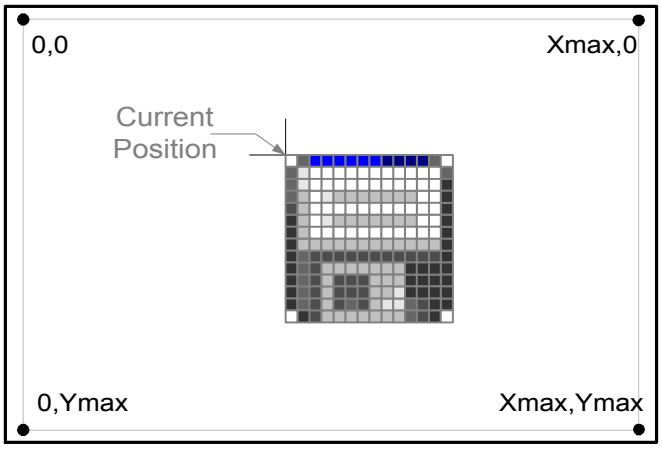


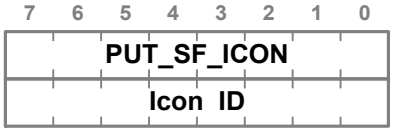
See Also: SET_XHY, SET_COLORH

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: **58**hex, **88**dec





Byte 0 (Command)

Byte 1 (Icon ID: 0 to 254)

Note: Maximum number of icons is 255 (IDs 0 to 254)

See Also: SET_XHY

Example:

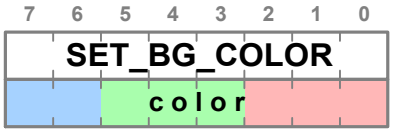
The following sequence will display Icon No. 3 with its upper-left corner positioned at (60, 43).

```
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

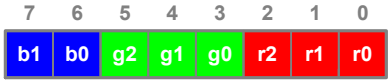
Code: **34**hex, **52**dec



Byte 0 (Command)

Byte 1 (Color Code)

Note: The 256 color palette has the following color coding:



See Also: PRINT_CHAR_BG, PRINT_STRING_BG

Example:

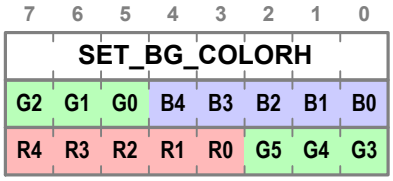
The following sequence print Yellow "LCD" on the Navy background, in the middle of a screen, using font no 0.

```
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: **94**hex, **148**dec



Byte 0 (Command)
Byte 1 (Color LSB)
Byte 2 (Color MSB)

See Also: PRINT_CHAR_BG, PRINT_STRING_BG

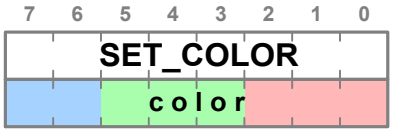
Example:

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

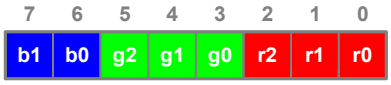
Description: Sets the Current Color
Code: **24**hex, **36**dec



Byte 0 (Command)

Byte 1 (Color Code)

Note: The 256 color palette has the following color coding:



See Also: CLS, PLOT

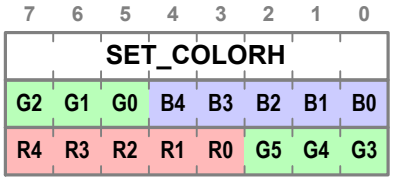
Example:

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: **84**hex, **132**dec



Byte 0 (Command)
Byte 1 (Color LSB)
Byte 2 (Color MSB)

See Also: CLS, PLOT

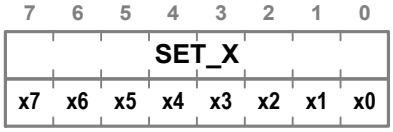
Example:

The following sequence will fill the whole screen with green.

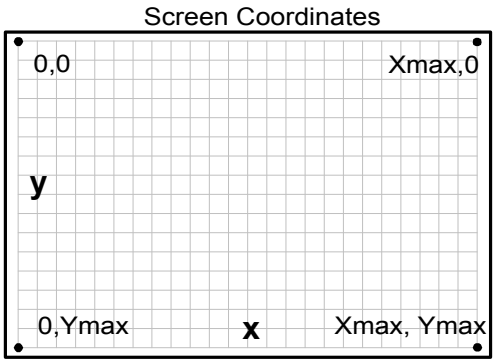
```
SET_COLORH   84 hex
GREEN_LSB    11100000 bin
GREEN_MSB    00000111 bin
CLS          21 hex
```


8.95.20 SET_X

Description: Sets only the X-coordinate of the Current Position. Y coordinate remains unchanged
Code: **5E**hex, **94**dec



Byte 0 (Command)
Byte 1 (x)



See Also: SET_Y, SET_XY

Example:

The following sequence will put a 2 blue points in the same row.

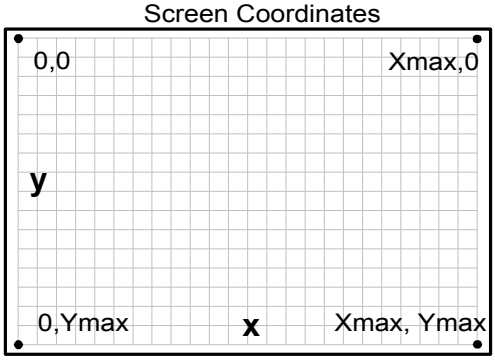
```
SET_COLORH   84 hex
BLUE_LSB     00011111 bin
BLUE_MSB     00000000 bin
SET_X         5E hex
 200         200 dec (x)
PLOT          26 hex
SET_X         5E hex
 208         208 dec (x)
PLOT          26 hex
```


8.95.21 SET_XY

Description: Sets the Current Position
Code: **25**hex, **37**dec

Byte 0 (Command)
Byte 1 (x)
Byte 2 (y)

Screen Coordinates



See Also: PLOT, LINE_TO_XY, CIRCLE_R

Example:

The following sequence will put the blue point in the middle of the screen.

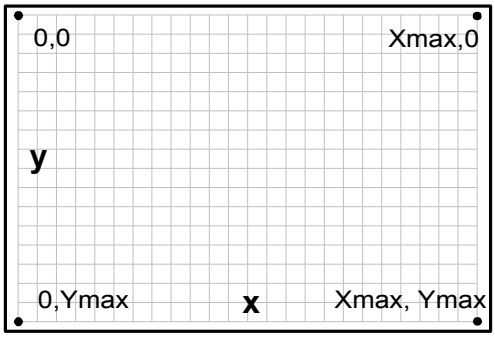
```
SET_COLOR    24 hex
BLUE         11000000 bin
SET_XY       25 hex
120         120 dec
80           80 dec
PLOT         26 hex
```


8.95.22 SET_XHY

Description: Sets the Current Position.
Code: **85**hex, **133**dec

7	6	5	4	3	2	1	0	
SET_XHY								**Byte 0 (Command)**
x15	x14	x13	x12	x11	x10	x9	x8	**Byte 1 (x MSB)**
x7	x6	x5	x4	x3	x2	x1	x0	**Byte 2 (x LSB)**
y7	y6	y5	y4	y3	y2	y1	y0	**Byte 3 (y)**

Screen Coordinates



See Also: PLOT, LINE_TO_XHY, CIRCLE_RH

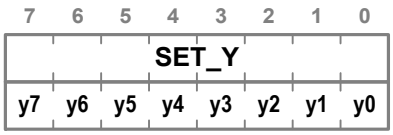
Example:

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

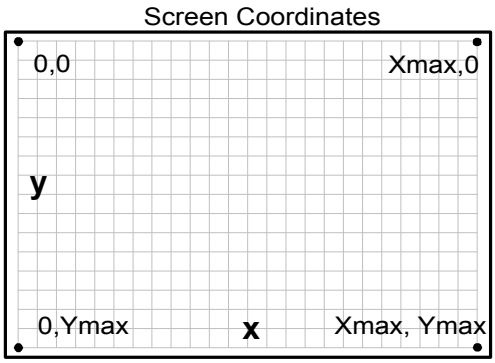
```
SET_COLORH   84 hex
BLUE_LSB     00011111 bin
BLUE_MSB     00000000 bin
SET_XHY      85 hex
  0           0 dec (x MSB)
160         160 dec (x LSB)
117         117 dec (y)
PLOT         26 hex
```

8.95.23 SET_Y

Description: Sets only the Y-coordinate of the Current Position. X coordinate remains unchanged
Code: **5F**hex, **95**dec



Byte 0 (Command)
Byte 1 (y)



See Also: SET_XH, SET_XHY

Example:

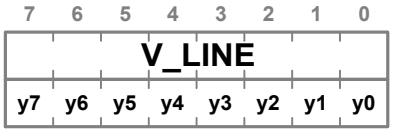
The following sequence will put a 2 blue points in the same column.

```
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.

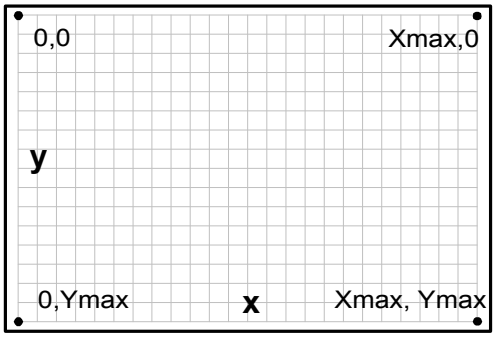
Code: **41**hex, **65**dec



Byte 0 (Command)

Byte 1 (Y)

Screen Coordinates



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

Configuration Keys Part of ezLCD+ Customization. Set of text words and values assigned to them. They are specifying the *User Configuration*. Similar, in concept, to the keys used in Windows .ini files.
Described in the *"ezLCD+10x Manual"*.

ezLCD+ Customization Modification of the default power-up parameters. Addition of custom fonts, bitmaps, Lua programs, etc.
Described in the *"ezLCD+10x Manual"*.

Firmware Operating software of the ezLCD+. Can be in-field upgraded.
Described in the *"ezLCD+10x Manual"*.

Lua Powerful, fast, light-weight, embeddable scripting language. By embedding Lua interpreter, the ezLCD+ become a true independent system (computer), which does not need any external host to drive it.
Described in the *"ezLCD+ Lua API Manual"*.

User Configuration Part of ezLCD+ Customization. Modifies some of the ezLCD+ default parameters like: communication parameters, start-up screen, etc. Upon the power-up the ezLCD+ CPU configures the ezLCD+ according to the data read from the User Configuration.
Described in the *"ezLCD+10x Manual"*.

User ROM Part of the ezLCD+ Customization. A place in the ezLCD+ flash, where user can store custom fonts, bitmaps, Lua programs, etc.
Described in the *"ezLCD+10x Manual"*.

Index

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- L -

- M -

- P -

- R -

- S -

- T -

- V -

