

1 ezLCD-004

1.1 Overview

Congratulations on your purchase of ezLCD-004!

The ezLCD-004 is an all-in-one advanced color TFT LCD panel which includes:

- 320x234 pixel, 65536 color, 5.6" TFT LCD
- LCD controller (Solomon SSD1906)
- Embedded 32bit ARM processor (Atmel AT91SAM7A3)
- 1 Mega Byte embedded serial flash for storing frequently used bitmaps
- Power supply, which generates all the voltages needed by the logic and the display itself
- Touch screen
- Interface drivers and other circuitry

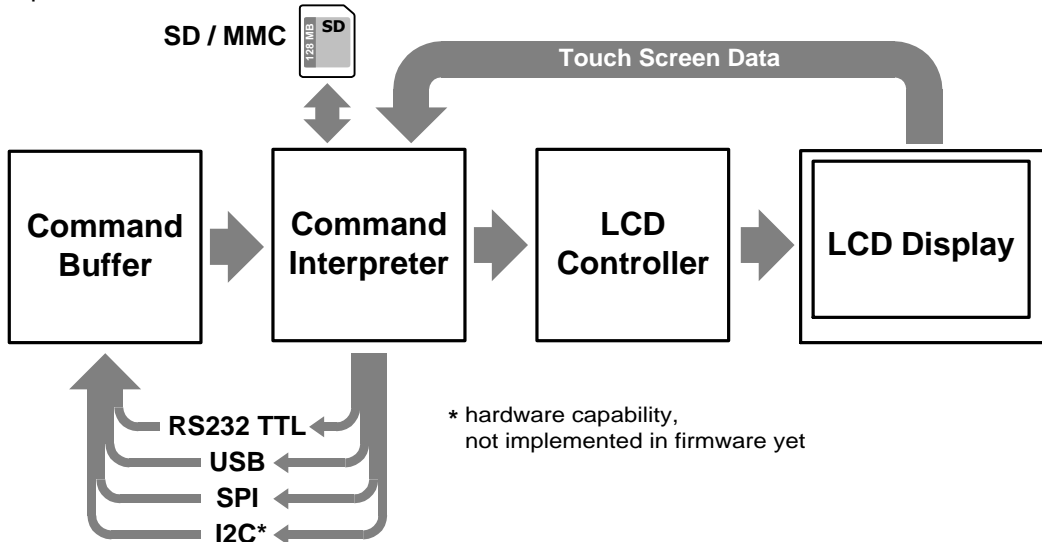
The ezLCD-004 communicates with the outside world through several interfaces:

- RS232 TTL
- USB
- I2C (not implemented in firmware yet)
- SPI
- SD/MMC

The ezLCD-004 is driven by a set of [commands](#), which can be fed through any of the implemented interfaces. The device may be used as an "intelligent" display, or as a stand alone device. There is enough of flash memory left in AT91SAM7A3 to incorporate additional graphical instructions, or to customize the software for particular tasks. Possible applications include automotive, avionics, nautical, industrial control, hobby, etc.

1.2 Operation

The ezLCD-004 is driven by a set of 8 bit [commands](#), which can be received by any of the implemented interfaces.



Each of the implemented interfaces uses the same set of [ezLCD Commands](#).

Upon arrival, the [ezLCD Commands](#) are stored into the 14336 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 drives the **LCD Controller** with the corresponding set of signals and instructions. 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 a centered position at $x = 160, y = 100$.

Pseudo-Code (ANSI C format):

```

SetColorh(GREEN); /* Set the drawing color to green */
SetXhy(160, 100); /* Set the position to x = 160, y = 100 */
CircleRh(60); /* Draw the circle with the radius of 60 pixels */
  
```

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

Mnemonic	Value	Format	Comment
SET_COLORH	84	hex	Set the drawing color to:
Green LSB	11100000	bin	
Green MSB	00000111	bin	green
SET_XHY	85	hex	Set the drawing position to:
x MSB	0	dec	
x LSB	160	dec	x (column) = 160
y	100	dec	y (row) = 100
CIRCLE_RH	89	hex	Draw the circle with the radius of:
r MSB	0	dec	
r LSB	60	dec	60 pixels

1.3 Hardware & Interfaces

1.3.1 Specifications

Electrical Characteristics

Parameter	Symbol	Min.	Typ.	Max.	Unit	Remark
Power Supply Voltage	V _{cc}	4.75	5.00	5.25	V	
Power Supply Current	I _{cc}	0.9	1.0	1.3	A	V _{cc} = 5V
Hi Level Logical Input Voltage	V _{IH}	2.2	3.3	3.6	V	Some pins are +5V - tolerant. Please refer to the Chapter: Pin Configuration
Lo Level Logical Input Voltage	V _{IL}	-0.3	0	1	V	
Operating Temperature	T _{opa}	0		60	°C	
Storage Temperature	T _{stg}	-25		80	°C	

Display Specifications

Parameter	Specification	Unit
Display Resolution	320(W) x 234(H)	dot
Active Area	113.28(W) x 84.708(H)	mm
Screen Size	5.6(Diagonal)	inch
Dot Pitch	0.354(W) x 0.362(H)	mm
Surface Treatment	Anti-glare(Haze=6% typical)	
View Angle Direction	6 o'clock	

Backlight

Backlight lamp life time with T_a = 25°C: Min = 20000 hrs, Typ = 30000 hrs

1.3.3 Pin Configuration

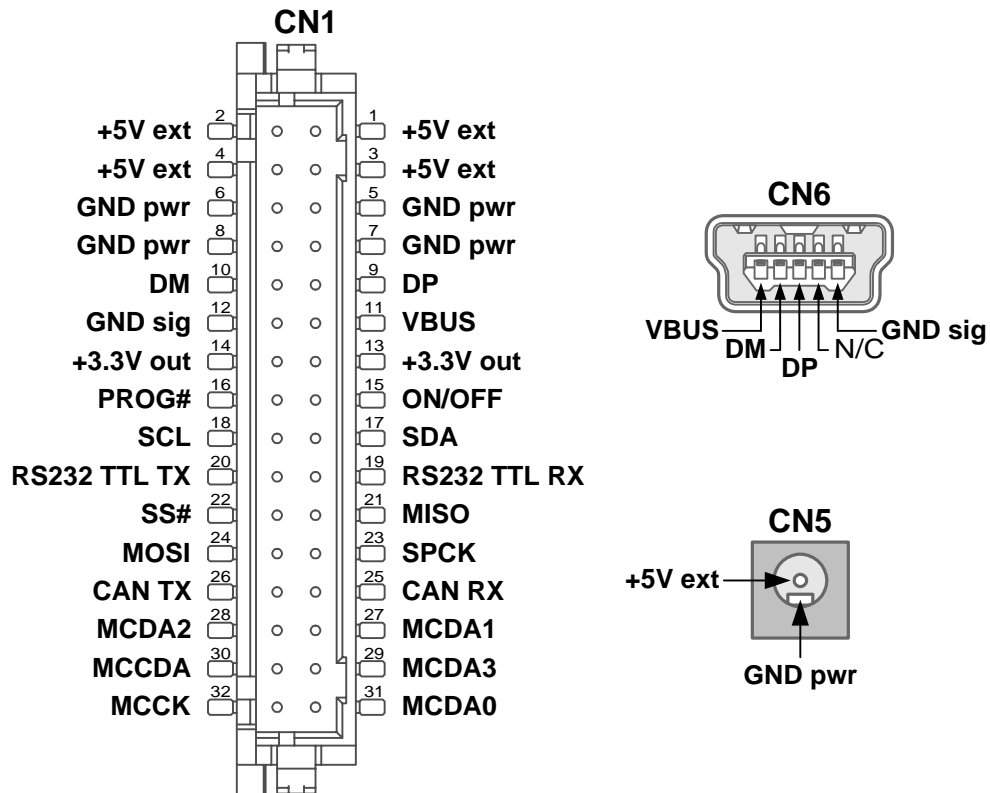


Figure 1. LCD-004 Connectors

The table below describes the pins and signals of the ezLCD-004.

Pin Name	Connector	Type	Description
+5V ext	CN1, CN5	Ext. Pwr	External power voltage. 5V/2A Min = +4.75V Max = +5.25V
GND Pwr	CN1, CN5	Gnd	Power GND. Use as +5V ext return.
GND sig	CN1, CN6	Gnd	Signal GND. Use as a gnd for all interfaces
DM	CN1, CN6	I/O	USB Data Minus
DP	CN1, CN6	I/O	USB Data Plus
VBUS	CN1, CN6	Pwr/Input	USB VBUS +5V
+3.3V out	CN1	Pwr/Out	+3.3V/0.5A regulated voltage output. May be used as a power supply for external devices.
ON/OFF	CN1	Input	ezLCD-004 ON/OFF signal. The same function as SW1 (ON) switch. +3.6 to +7V turns ON the ezLCD-004 power. 0 to +1V turns OFF the ezLCD-004 power. Rin = 10 kOhm

This table is continued on the next page

Pin Name	Connector	Type	Description
PROG#	CN1	Input	Firmware download signal. The same

			function as SW2 (PROG) pushbutton. The ezLCD-004 enters the firmware bootloader state, if this pin is connected to GND during the power up.
SCL	CN1	I/O	I2C interface SCL Min = 0V Max = +3.3V (Not +5V tolerant) Recommended: OPEN/GND signal with pull-up resistors connected to the +3.3.
SDA	CN1	I/O	I2C interface SDA Min = 0V Max = +3.3V (Not +5V tolerant) Recommended: OPEN/GND signal with pull-up resistors connected to the +3.3.
RS232 TTL TX	CN1	Output	RS232 TTL Output Min = 0V Max = +3.3V
RS232 TTL RX	CN1	Input	RS232 TTL Input Min = 0V Max = +3.3V (+5V tolerant)
SS#	CN1	Input	SPI interface SS signal Min = 0V Max = +3.3V (Not +5V tolerant)
MISO	CN1	Output	SPI Master Input Slave Output signal Min = 0V Max = +3.3V
MOSI	CN1	Input	SPI Master Output Slave Input signal Min = 0V Max = +3.3V (Not +5V tolerant)
SPCK	CN1	Input	SPI Clock Input Min = 0V Max = +3.3V (Not +5V tolerant)
CAN TX	CN1	Output	CAN Interface Transmit
CAN RX	CN1	Input	CAN Interface Receive
MCDA0	CN1	I/O	SD/MMC DAT0 (Data Line 0) Min = 0V Max = +3.3V
MCDA1	CN1	I/O	SD/MMC DAT1 (Data Line 1) Min = 0V Max = +3.3V
MCDA2	CN1	I/O	SD/MMC DAT2 (Data Line 2) Min = 0V Max = +3.3V
MCDA3	CN1	I/O	SD/MMC DAT3 (Data Line 3) Min = 0V Max = +3.3V
MCCDA	CN1	I/O	SD/MMC CMD (Command/Response) Min = 0V Max = +3.3V
MCKK	CN1	Output	SD/MMC CLK (Clock) Min = 0V Max = +3.3V

1.3.4 High Current USB Connection

The following describes how to connect High-Current USB Cable so it will supply both power and USB signals to the ezLCD-004.

Connect **+5V** from the USB cable to the following ezLCD pins:

+5V ext
VBUS

Connect **GND** from the USB cable to the following ezLCD pins:

GND pwr
GND sig

Connect **DM** from the USB cable to the **DM** pin of the ezLCD.

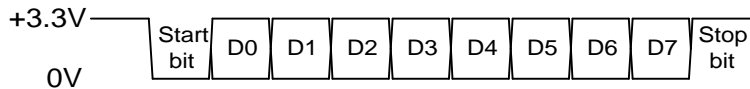
Connect **DP** from the USB cable to the **DP** pin of the ezLCD.

1.3.5 RS232 TTL

Default Communication Parameters

Baudrate: 115200 bps
No of Data Bits: 8
No of Stop Bits: 1
Parity: Off
Handshake: None

Pin Name	Connector	Type	Description
RS232 TTL TX	CN1	Output	RS232 TTL Output Min = 0V Max = +3.3V
RS232 TTL RX	CN1	Input	RS232 TTL Input Min = 0V Max = +3.3V (+5V tolerant)



Warning: RS232 TTL uses logic level signals: Min = 0V, Max = +3.3V (+5V tolerant). Connecting RS232 TTL to "standard" RS232 interface with the signal levels of ± 3 V, ± 5 V, etc. may damage the ezLCD-004 and void the warranty.

ezLCD-004 Power-Up Ready Transmission

If the RS232 TTL is set as the "Default Transmitter", the ezLCD-004 sends EZLCD_READY byte (**EA** hex, **234**dec). The EZLCD_READY byte is sent one time only, upon the power-up when the ezLCD-004 RS232 TTL interface is ready to receive the commands.

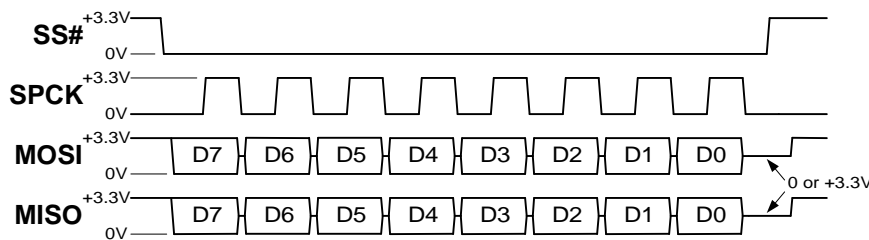
The "Default Transmitter" can be set by the ezLCDconfig utility. See Chapter: [Firmware Customization](#).

1.3.6 SPI

Communication Parameters

Max f_{SPCK} : 4MHz
SPCK Idle: Low
No of Data Bits: 8
Bit Order: MSB goes first
Data sampled on: Leading edge of the SPCK (rising edge)
ezLCD-004 is: SPI Slave

Pin Name	Connector	Type	Description
SS#	CN1	Input	SPI interface SS signal Min = 0V Max = +3.3V (Not +5V tolerant)
MISO	CN1	Output	SPI Master Input Slave Output signal Min = 0V Max = +3.3V
MOSI	CN1	Input	SPI Master Output Slave Input signal Min = 0V Max = +3.3V (Not +5V tolerant)
SPCK	CN1	Input	SPI Clock Input Min = 0V Max = +3.3V (Not +5V tolerant)



Warning: SPI inputs **are not** +5V tolerant. Driving the inputs with voltages out of the range specified in the table above, may damage the ezLCD-004 and void the warranty.

Receiving the data from the ezLCD-004

Since:

- The ezLCD-004 is configured as an SPI Slave and
- All transmissions through the SPI interface have to be initiated by the Master, it is the user responsibility to query the ezLCD for any new data, like for example: touch screen coordinates.

Each time, the byte is send through the SPI interface to the ezLCD, the unit responds on the MISO pin. if you want to query the ezLCD without sending any command: send 0 to the ezLCD.

ezLCD-004 Power-Up Ready Transmission

If the SPI is set as the "Default Transmitter", the ezLCD-004 sends EZLCD_READY byte (**EA**_{hex}, **234** dec). The EZLCD_READY byte is sent one time only, upon the power-up when the ezLCD-004 SPI interface is ready to receive the commands.

The "Default Transmitter" can be set by the ezLCDconfig utility. See Chapter: [Firmware Customization](#).

1.3.7 SD/MMC

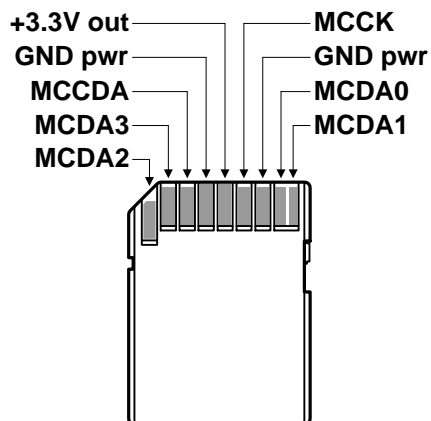
The SD/MMC Interface hardware supports the MultiMediaCard (MMC) Specification V2.2 and the SD Memory Card Specification V1.0. SD Memory Card operations are supported by the firmware. For now, the MultiMediaCard is not supported.

The ezLCD-004 firmware automatically adjusts the baudrate and other timing parameters by reading the SD Card parameters in the slow clock mode.

Max Available Baudrate: 96 Mbps (read from the SD Card)
24 Mbps (write to the SD Card)

Pin Name	Connector	Type	Description
MCDA0	CN1	I/O	SD/MMC DAT0 (Data Line 0) Min = 0V Max = +3.3V
MCDA1	CN1	I/O	SD/MMC DAT1 (Data Line 1) Min = 0V Max = +3.3V
MCDA2	CN1	I/O	SD/MMC DAT2 (Data Line 2) Min = 0V Max = +3.3V
MCDA3	CN1	I/O	SD/MMC DAT3 (Data Line 3) Min = 0V Max = +3.3V
MCCDA	CN1	I/O	SD/MMC CMD (Command/Response) Min = 0V Max = +3.3V
MCCK	CN1	Output	SD/MMC CLK (Clock) Min = 0V Max = +3.3V

The above signals are repeated in the ezLCD-004 embedded MMC/SD slot. If, from any reason the ezLCD-004 embedded MMC/SD slot cannot be used, the drawing below shows how to connect SD/MMC signals to the external SD Card.



For more information, please refer to the Chapter: [SD Card Operations](#)

1.4 Firmware

1.4.1 Firmware Upgrade

The firmware upload to the ezLCD-004 is performed through the USB interface. EzLCD-004 has to be running the Bootloader software. The Bootloader software resides in the ezLCD-004 ROM. The program will jump into the Bootloader when the PROG# pin is connected to ground during power-up (this can also be accomplished by holding down the PROG button on the back of the ezLCD-004). Once ezLCD-004 software enters the Bootloader, it stays there until the next power-up/reset.

The ezLCD-004 firmwares are distributed as Windows executable files. In order to upgrade the firmware, the ezLCD-004 USB driver has to be installed.

To upgrade ezLCD-004 firmware:

1. Make sure that the ezLCD-004 USB driver is installed
2. Power off the ezLCD-004
3. Connect PC to the ezLCD-004 via USB
4. Connect the PROG# pin to the ground (or hold down PROG button on back of ezLCD-004)
5. Power on the ezLCD-004. Make sure that ezLCD runs the bootloader software (It should display "ezLCD Bootloader" in the top of the screen).
6. On your PC run ezLCD-004 firmware
7. Follow the displayed instructions
8. Recycle the ezLCD-004 power when finished

The above procedure will load the new firmware into the ezLCD-004 leaving the bootloader software unchanged. This way, the operation can always be repeated in case something unexpected happens during firmware upgrade (power or USB connection failure, PC hang up, etc.).

Upgrading the bootloader (not recommended)

It is neither necessary nor recommended to upgrade the bootloader. The bootloader upgrade operation is not safe and as such is not covered by the warranty. The ezLCD-004 may fail, if something unexpected happens during the bootloader upgrade (power or USB connection failure, PC hang up, etc.). If your ezLCD-004 fails and cannot be rebooted, you may always send the unit to the manufacturer, where the firmware+bootloader can be restored for a small fee.

Warning: Do the following at your own risk!

The firmware+bootloader upgrade procedure is almost identical to the one used to upgrade the firmware only. The only difference is that the firmware executable should be called with 'loadall' parameter.

For example, if the firmware file is ezLCD004_24.exe:

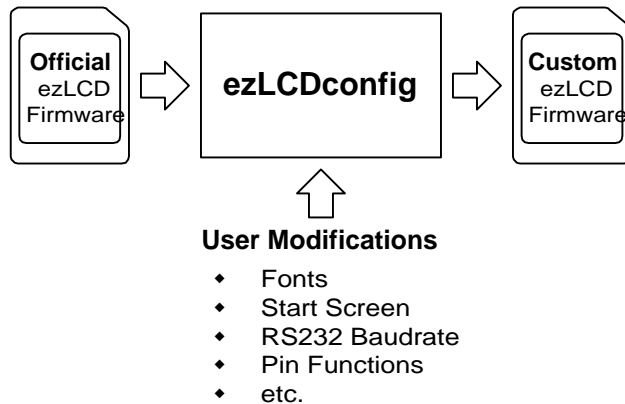
1. Open command window in the directory, where ezLCD004_24.exe resides
2. Type: ezLCD004_24 loadall and press Enter

A few remarks, which may help you to avoid sending the unit back to the manufacturer.

The bootloader software is executed from RAM. It stays there until the next power-off. Do not recycle the ezLCD power if your PC or USB connections fails during the upgrade. Once the ezLCD-004 runs the bootloader software, there is no need to power it off in order to repeat the upgrade procedure. In another words: skip steps 2, 4 and 5 of the upgrade procedure.

1.4.2 Firmware Customization

Many aspects of the ezLCD firmware can be customized in order to fit the particular needs. A special utility, which allows for such modifications, is currently being developed. That utility has a working name: ezLCDconfig, however this name may change in the future. The latest pre-release version of the ezLCDconfig utility is available for download at <http://www.ezlcd.com>.



As it is shown on the picture above, the ezLCDconfig allows the user to:

1. Read the official ezLCD firmware
2. Modify it
3. Save the modified firmware

The ezLCDconfig utility should run on any PC with the Microsoft .NET Ver: 2.0 installed. Currently, the ezLCDconfig allows for customization of:

- Fonts
 - Rearranging
 - Converting of Windows fonts into the ezLCD fonts
 - Adding new fonts
 - Removing fonts
- Backlight
 - Startup backlight On or Off
 - Startup backlight brightness
- Start Screen
 - Background color
 - Startup bitmap
- Touch Screen
 - Default protocol
- RS232
 - Baudrate
- Transmitter
 - Default transmit interface
- Pin Functions
 - Disabling and enabling interfaces
 - Alternate CN1 pin functions:
 - Command Execution In Progress

When Command Execution In Progress (Exec Cmd) is selected for the particular pin, that pin outputs Hi (+2.3 to +3.3V) when the command is executed and Lo (0 to +1V) between commands. The pin switches to Hi on the first byte of the command. The pin switches to Lo only when all bytes of the particular command are processed and the command execution is finished.

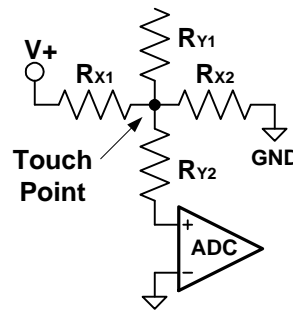
- ezNow Buzzer
Buzzer On/Off output for the ezNow board
- ezNow AVR Prog
Enables the programming of the ezNow board embedded AVR processor through the ezLCD-004 USB connector.

1.5 Touch Screen

1.5.1 Introduction

The ezLCD-004 has a 4-wire resistive analog touch screen. This touch screen consists of two layers of transparent resistive material with silver ink for electrodes. These two layers are stacked on an insulating layer of glass, separated by tiny spacer dots. They are interfaced electrically to the dedicated ezLCD-004 A/D converter. During measurement of a given coordinate, one of the resistive planes is powered along its axis and the other plane is used to sense the location of the coordinate on the powered plane.

For example, in case of the X coordinate measurement, the X plane is powered, as shown on the drawing below. The Y plane is used to sense where the pen is located on the powered plane as follows: At the location where the pen depresses the touch screen, the planes are shorted. The voltage measured on the sensed plane is proportional to the location of the touch on the powered plane. This voltage is then converted by the dedicated ezLCD-004 ADC as shown on the drawing below.



Note:

The ezLCD-004 touch screen is of the industrial type. It requires bigger activation pressure than the ones used on PDAs, Cell Phones, etc. Since the distance between X and Y planes is bigger, this touch screen is more sensitive to an uneven pressure, particularly close to the edges.

1.5.2 Calibration

The touch screen calibration starts when the `PROG#` pin is connected to ground while the unit power is already on (this can also be accomplished by holding down the PROG button on the back of the ezLCD-004).

To calibrate the touch screen:

1. Power on the ezLCD-004.
2. Connect the `PROG#` pin to the ground for up to 2 seconds (or hold down PROG button on back of ezLCD-002).
3. Follow the instructions displayed on the ezLCD screen

1.5.3 Data Protocols

Currently, the ezLCD-004 can broadcast the touch screen data using the following protocols:

1. [ezButton](#)
 - Touch screen buttons can be defined [BUTTON_DEF](#) command.
 - ezLCD sends Button Down and Button Up events for the buttons defined by the [BUTTON_DEF](#) 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 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](#).

1.5.3.1 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-004 Serial Flash.

Use `ezLCD004flash.exe` or other utility to store the bitmaps in the ezLCD-004 Serial Flash. 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` command.

Send `BUTTON_DEF` command for each of the buttons that you want to use. The `BUTTON_DEF` command specifies:

- 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-004 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.

1.5.3.1.1 ezButton Events

The ezButton events are coded in one byte:

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

Where:

Button_Number: The number (ID) of the button, which has caused the event. The button number is specified by the [BUTTON_DEF](#) 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 the [BUTTON_DEF](#) command.

1. Touch Screen: Not pressed
ezLCD-004 Sends: Nothing
2. Touch Screen: Pressed in the Button 4 touch zone
ezLCD-004 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-004 Sends: 84hex only 1 time, in the moment when the finger is removed from the touch screen
4. Touch Screen: Pressed outside any of the buttons
ezLCD-004 Sends: Nothing
5. Touch Screen: Finger slides into the Button 4 touch zone
ezLCD-004 Sends: Nothing
6. Touch Screen: Finger is removed from the touch screen
ezLCD-004 Sends: Nothing (because no button has been pressed)
7. Touch Screen: Pressed in the Button 4 touch zone
ezLCD-004 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-004 Sends: Nothing (the Button 4 is still considered pressed)
9. Touch Screen: Finger is removed from the touch screen
ezLCD-004 Sends: 84hex only 1 time, in the moment when the finger is removed from the touch screen

1.5.3.2 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-004 Serial Flash.

Use `ezLCD004flash.exe` or other utility to store the bitmaps in the ezLCD-004 Serial Flash. Note which bitmap ID should be assigned to each state of the particular button.

3. In your code, select ezButton 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 command.

Send BUTTON_DEF command for each of the buttons that you want to use. The BUTTON_DEF command specifies:

- 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.

1.5.3.2.1 cuButton States

The ezButton events are coded in one byte:

		7	6	5	4	3	2	1	0
Status		Button_Number (0 to 63)							
0	0	Not used (Disregard)							
0	1	Button Down State							
1	0	Not used (Disregard)							
1	1	Button None State							

Where:

Button_Number: The number (ID) of the button, which has caused the event. The button number is specified by the [BUTTON_DEF](#) 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 the [BUTTON_DEF](#) command.

1. Touch Screen: Not pressed
ezLCD-004 Sends: FF_{hex} continuously, 5 to 20 times per second
2. Touch Screen: Pressed in the Button 4 touch zone
ezLCD-004 Sends: 44_{hex} continuously, 5 to 20 times per second
3. Touch Screen: Finger is removed from the touch screen
ezLCD-004 Sends: FF_{hex} continuously, 5 to 20 times per second
4. Touch Screen: Pressed outside any of the buttons
ezLCD-004 Sends: FF_{hex} continuously, 5 to 20 times per second
5. Touch Screen: Finger slides into the Button 4 touch zone
ezLCD-004 Sends: FF_{hex} continuously, 5 to 20 times per second
6. Touch Screen: Finger is removed from the touch screen
ezLCD-004 Sends: FF_{hex} continuously, 5 to 20 times per second
7. Touch Screen: Pressed in the Button 4 touch zone
ezLCD-004 Sends: 44_{hex} continuously, 5 to 20 times per second
8. Touch Screen: Finger slides out of the Button 4 touch zone
ezLCD-004 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-004 Sends: FF_{hex} continuously, 5 to 20 times per second

1.5.3.3 CalibratedXY

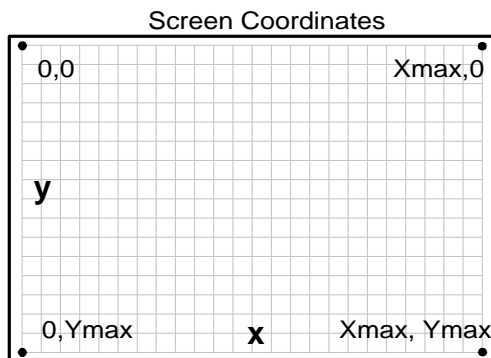
When the CalibratedXY protocol is selected, the ezLCD:

- 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.

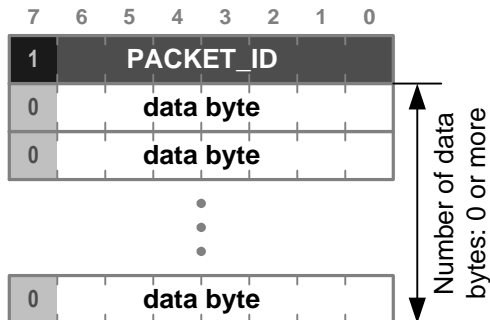


For ezLCD-004:

- 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



1.5.3.3.1 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: 81hex, 129dec

7 6 5 4 3 2 1 0

1	TOUCH_X						
0	x6	x5	x4	x3	x2	x1	x0
0	x13	x12	x11	x10	x9	x8	x7

Byte 0: PACKET_ID (81 hex)

Byte 1: bits 0 to 6 of X

Byte 2: bits 7 to 14 of X

TOUCH_Y Packet

Description: The TOUCH_Y packet represents the touch screen X coordinate. It is sent only if the touch screen pressed.

Length: 3 bytes, including the Packet ID

Code: 82hex, 130dec

7 6 5 4 3 2 1 0

1	TOUCH_Y						
0	y6	y5	y4	y3	y2	y1	y0
0	y13	y12	y11	y10	y9	y8	y7

Byte 0: PACKET_ID (82 hex)

Byte 1: bits 0 to 6 of Y

Byte 2: bits 7 to 14 of Y

PEN_UP Packet

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

Length: 1 byte, including the Packet ID

Code: 83hex, 131dec

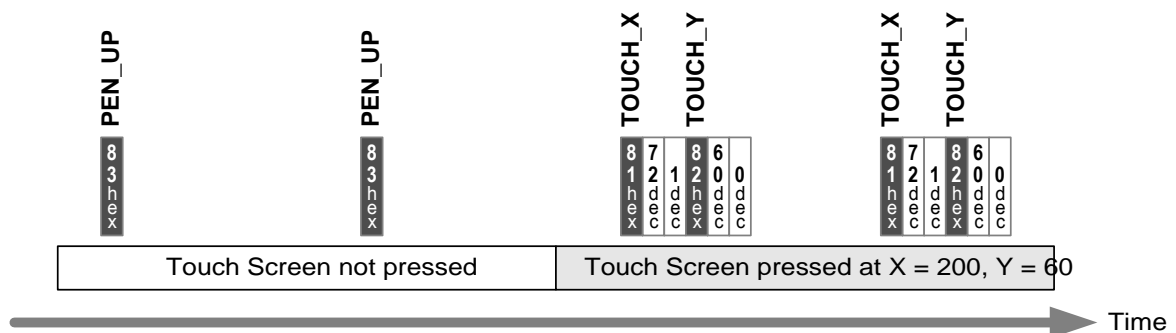
7 6 5 4 3 2 1 0

1	PEN_UP						
---	--------	--	--	--	--	--	--

Byte 0: PACKET_ID (83 hex)

Example:

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

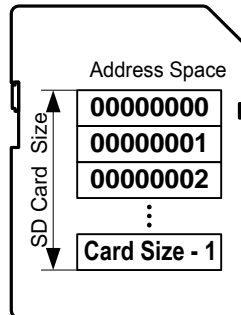


1.6 SD Card Operations

1.6.1 Introduction

The ezLCD-004 has a set of [commands](#), which perform write and read operation on the SD (Secure Digital) memory card, connected through the [SD/MMC](#) interface. The SD Memory Card Specification V1.0 is supported.

The SD Memory Card is "seen" by the ezLCD-004 as an external memory (8bit x Card Size), as it is shown on the drawing below



Note: The ezLCD firmware ignores the position of the SD Card Write Protect Switch.

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](#).

1.6.2 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 [SD 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.

[SD_FORMAT](#)

Formats the SD in the specified file system.

[SD_FILE_LIST](#)

Gets the list of files and sub-directories which reside in the specified SD Directory.

[SD_FILE_OPEN](#)

Opens an existing SD Flash file for reading or writing. File Position Index is set to 0. Temporary disables the Touch Screen. The Touch Screen will be automatically re-enabled when all files are closed. 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. Temporary disables the Touch Screen. The Touch Screen will be automatically re-enabled when all files are closed.

[SD_FILE_CLOSE](#)

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

[SD_FILE_CLOSE_ALL](#)

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

[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. File Position Index is incremented by the number of the bytes read, however it will not exceed file_size - 1.

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.

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.

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 64 bytes.

About the SD Directory/File Path used in the SD_FILE_LIST command:

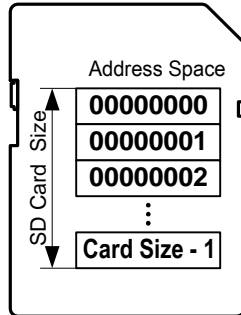
- Directory Path specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: '*' and '?' are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Directory Path is not case-sensitive. The drive and root directory do not have 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 64 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 64 bytes.

1.6.3 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.

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

1.6.4 SD Source Code Examples

SD Source Code examples are provided to illustrate how to use SD access commands. They are provided on "as is" bases. There is no warranty whatsoever.

ezLCD-004 Interface

All examples use USB interface.

Development Environment

All examples are written in 'C'.

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

Note: '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>

Directories

Exe - build executables
Sources - 'C' sources
VisualC - 'MS Visual C++ 6.0' projects
Borland - 'Borland C++ Builder 6.0' projects

Projects

SDicon - Displays a bitmap from the SD card on the ezLCD-004 screen.
Sources: **icon.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDsize - Reads and displays the size of the SD card.
Sources: **size.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDflist - Reads and prints the list of the files on the SD directory.
Sources: **flist.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDfsize - Reads and displays the size of the file from formatted SD card.
Sources: **fsize.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDfget - Copies the file from SD Card to the PC.
Sources: **fget.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDfput - Copies the file from the PC to SD Card.
Sources: **fput.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDfdel - Deletes the file from the SD Card.
Sources: **fdel.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDrawrd - Reads the raw data from the SD Card and displays it on the PC screen.
Sources: **rawrd.c**, ezLCDio.c, ezLCDdll.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h

SDformat - Formats the SD card in FAT16, displaying the progress on the ezLCD.
Sources: **format.c**, ezLCDio.c, ezLCDdll.c, fat16.c
Headers: mytypes.h, ezLCDio.h, ezLCDdll.h, CmdCodes.h, fat16.h

Common Files

- ezLCDdll.c - This file is used to dynamically load the ezLCD.dll and initialize it's functions.
The ezLCD.dll is a user-mode USB driver supplied with the ezLCD-004.
- ezLCDio.c - Contains functions, which handle the USB communication between ezLCD-004
and the PC using the ezLCD.dll routines.

Project-specific Files

- icon.c - SDicon project main file.
- size.c - SDsize project main file.
- flist.c - SDflist project main file
- fsize.c - SDfsize project main file.
- fget.c - SDfget project main file.
- fput.c - SDfput project main file.
- fdel.c - SDfdel project main file.
- rawrd.c - SDrawrd project main file.
- format.c - SDformat project main file.
- fat16.c - Contains functions, which map FAT16 structures.
This file is used in the SDformat project only.

1.7 ezLCD Commands

General

CLS
SET_COLORH

Drawing Position

RESTORE_POSITION
SAVE_POSITION
SET_XH
SET_XHY
SET_Y

Backlight

LIGHT_BRIGHT
LIGHT_ON
LIGHT_OFF

Points

PLOT
PLOT_XHY

Lines

H_LINEH
V_LINE
LINE_TO_XHY

Figures

ARCH
PIEH
CIRCLE_RH
CIRCLE_RH_FILL
BOXH
BOXH_FILL

Bitmaps

PUT_BITMAP
PUT_SF_ICON
SD_PUT_ICON

Text and Fonts

SELECT_FONT
SET_BG_COLOR
TEXT_NORTH
TEXT_EAST
TEXT_SOUTH
TEXT_WEST
PRINT_CHAR
PRINT_CHAR_BG
PRINT_STRING
PRINT_STRING_BG

Touch Screen

BUTTON_DEF
BUTTON_STATE
BUTTONS_ALL_UP
BUTTONS_DELETE_ALL

TOUCH_PROTOCOL

SD Flash Card

SD_FILE_CLOSE
SD_FILE_CLOSE_ALL
SD_FILE_CREATE
SD_FILE_DELETE
SD_FILE_GET_SIZE
SD_FILE_LIST
SD_FILE_OPEN
SD_FILE_READ
SD_FILE_REWIND
SD_FILE_SEEK
SD_FILE_TELL
SD_FILE_WRITE
SD_FIND_FIRST and SD_FIND_NEXT
SD_FOLDER_CREATE
SD_FOLDER_DELETE
SD_FORMAT
SD_INSERTED
SD_PUT_ICON
SD_SCREEN_CAPTURE
SD_RAW_READ
SD_RAW_WRITE
SD_SIZE

ezNow Buzzer

EZNOW_BUZZER_BEEP
EZNOW_BUZZER_OFF
EZNOW_BUZZER_ON

System

PING

Legacy Commands

ARC
BOX
BOX_FILL
CIRCLE_R
CIRCLE_R_FILL
LINE_TO_XY
PLOT_XY
PUT_BITMAP
SET_BG_COLOR
SET_COLOR
SET_XY

1.7.1 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							
radius_MSB							
radius_LSB							
begin_angle_MSB							
begin_angle_LSB							
end_angle_MSB							
end_angle_LSB							

Byte 0 (Command)
Byte 1 (Radius MSB)
Byte 2 (Radius LSB)
Byte 3 (Arc Begin Angle MSB)
Byte 4 (Arc Begin Angle LSB)
Byte 5 (Arc End Angle MSB)
Byte 6 (Arc End Angle LSB)

See Also: [PIEH](#), [SET_XHY](#), [SET_COLORH](#), [CIRCLE_RH](#)

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

$$\text{Angle_lcd} = \text{Angle_deg} \times 2048 / 45$$

For example:

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

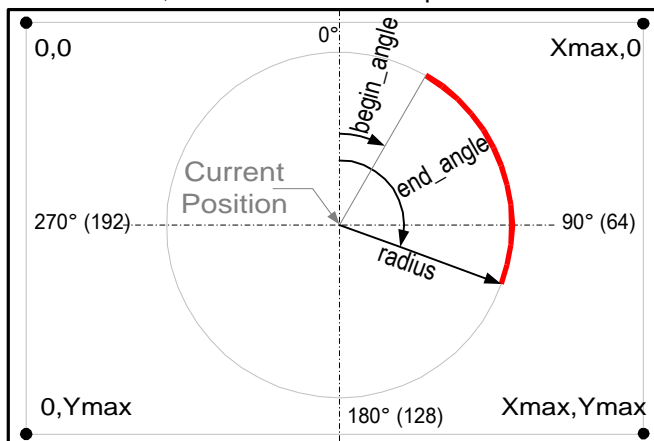
$$4096_{\text{dec}} = 1000_{\text{hex}} = 90^\circ$$

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

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

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

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



Example:

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

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

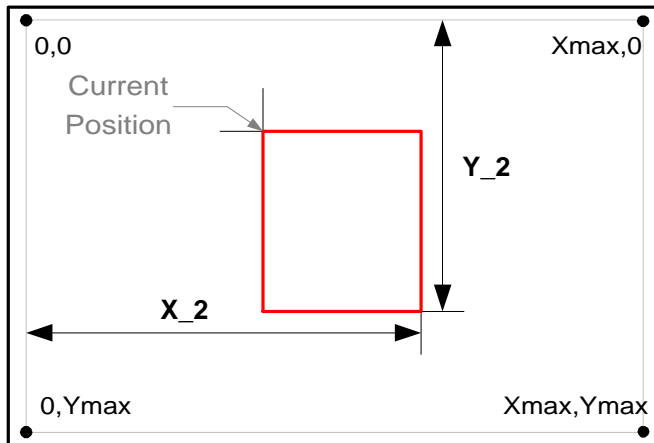
```
SET_COLORH 84 hex
GREEN_LSB  11100000 bin
GREEN_MSB  00000111 bin
SET_XHY    85 hex
  0        0 dec (x MSB)
 160      160 dec (x LSB)
 117      117 dec (y)
ARCH      8F hex
  0        0 dec (radius MSB)
  80      80 dec (radius LSB)
  08      08 hex (begin_angle MSB)
  00      00 hex (begin_angle LSB)
  28      28 hex (end_angle MSB)
  00      00 hex (end_angle LSB)
```

1.7.2 BOXH

Description: Draws a rectangle.

Code: **A2**_{hex}, **162**_{dec}

	7	6	5	4	3	2	1	0	
	BOXH								
	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
           180    0 dec (X_2 MSB)
           180   180 dec (X_2 LSB)
           120   120 dec (Y_2)

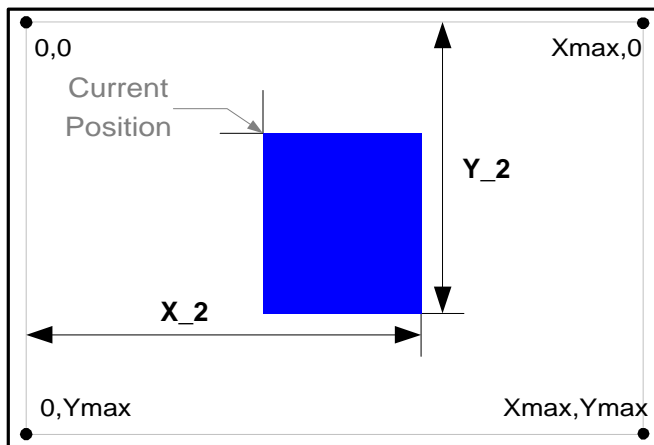
```

1.7.3 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								
	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
           180    0 dec (X_2 MSB)
           180    180 dec (X_2 LSB)
           120    120 dec (Y_2)

```

1.7.4 BUTTON_DEF

Description: Defines and draws a touch button

Code: B0_{hex}, 176_{dec}

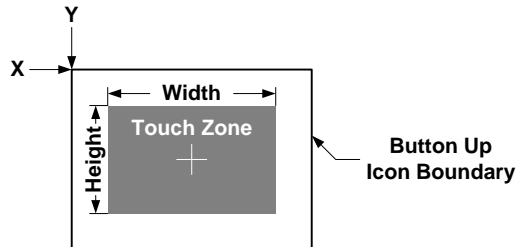
	7	6	5	4	3	2	1	0	
BUTTON_DEF									Byte 0: Command
button_no									Byte 1: Button No (0 to 63)
state									Byte 2: Initial State (1: Up, 2: Down, 3: Disabled, 4: Non-Visible)
button_up_icon									Byte 3: Icon No in Ser. Flash for button Up (255 = none)
button_down_icon									Byte 4: Icon No in Ser. Flash for button Down (255 = none)
button_disabled_icon									Byte 5: Icon No in Ser. Flash 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:

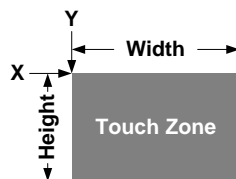
Touch Zone is the active touch response area of the button. It is specified by **With** (Byte 9) and **Height** (Byte 10).

- 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](#)
- [ezButton](#)

- [cuButton](#)

Example:

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

- Button Up Icon in serial flash: 8
- Button Down Icon in serial flash: 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.

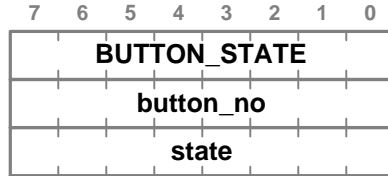
The button will be initially drawn using Button Up icon.

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 the serial flash)
9	9	dec	(Button Down Icon No. in the serial flash)
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)

1.7.5 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](#) command, if we do not wish the button to be initially drawn.

See Also: [BUTTON_DEF](#), [BUTTONS_ALL_UP](#), [BUTTONS_DELETE_ALL](#), [TOUCH_PROTOCOL](#)

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

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

Example:

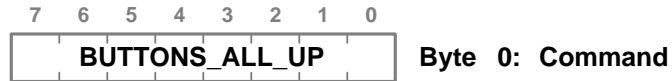
The following sequence will change the state of the Button No. 4 to the Button Down. The button will be redrawn using Button Down Icon.

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

1.7.6 BUTTONS_ALL_UP

Description: Changes the state of all defined touch buttons to Button Up

Code: B3_{hex}, 179_{dec}



Note:

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

See Also: [BUTTON_DEF](#), [BUTTON_STATE](#), [BUTTONS_DELETE_ALL](#), [TOUCH_PROTOCOL](#)

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

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

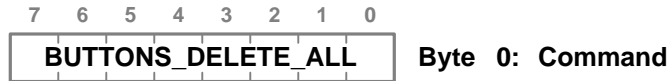
Example:

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

`BUTTONS_ALL_UP B3 hex` (Command)

1.7.7 BUTTONS_DELETE_ALL

Description: Deletes all touch buttons
Code: **B4**_{hex}, **180**_{dec}



Note:

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

See Also: [BUTTON_DEF](#), [BUTTON_STATE](#), [BUTTONS_ALL_UP](#), [TOUCH_PROTOCOL](#)

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

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

Example:

The following sequence will delete all Buttons.

`BUTTONS_DELETE_ALL B4 hex` (Command)

1.7.8 CIRCLE_RH

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

Code: 89hex, 137dec

									CIRCLE_RH
r15	r14	r13	r12	r11	r10	r9	r8		Byte 0 (Command)
r7	r6	r5	r4	r3	r2	r1	r0		Byte 1 (radius MSB)
									Byte 2 (radius LSB)

See Also: [SET_XHY](#), [SET_COLORH](#)

Example:

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

```

SET_COLORH 84 hex
GREEN_LSB  11100000 bin
GREEN_MSB  00000111 bin
SET_XHY    85 hex
0          0 dec (x MSB)
160       160 dec (x LSB)
117       117 dec (y)
CIRCLE_RH 89 hex
0          0 dec (radius MSB)
80        80 dec (radius LSB)

```

1.7.9 CIRCLE_RH_FILL

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

Code: 99_{hex}, 153_{dec}

									Byte 0 (Command)
r15	r14	r13	r12	r11	r10	r9	r8		Byte 1 (radius MSB)
r7	r6	r5	r4	r3	r2	r1	r0		Byte 2 (radius LSB)

See Also: [SET_XHY](#), [SET_COLORH](#)

Example:

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

```

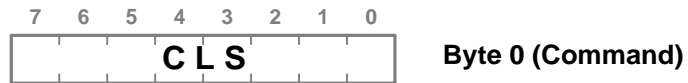
SET_COLORH      84 hex
RED_LSB         00000000 bin
RED_MSB         11111000 bin
SET_XHY        85 hex
0               0 dec (x MSB)
160            160 dec (x LSB)
117            117 dec (y)
CIRCLE_RH_FILL 99 hex
0              0 dec (radius MSB)
80            80 dec (radius LSB)

```

1.7.10 CLS

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

Code: 21hex, 33dec



See Also: [SET_COLORH](#)

Example:

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

```
SET_COLORH 84 hex
WHITE_LSB 11111111 bin
WHITE_MSB 11111111 bin
CLS 21 hex
```

1.7.11 EZNOW_BUZZER_BEEP

Description: Makes the buzzer on the ezNow board beep for the specified time

Code: D2_{hex}, 210_{dec}

7	6	5	4	3	2	1	0	
EZNOW_BUZZER_BEEP								Byte 0: Command
Beep Time [0.01s]								Byte 1: Beep Time (0 to 255 = 0 to 2.55s)

Prerequisites:

1. The ezNow board with a buzzer is attached to the connector CN1
2. CN1 pin 17 (SDA) function is changed to "Buzzer". This can be done by the ezLCDconfig utility. See Chapter: [Firmware Customization](#).

Notes:

1. This command does not delay the execution of the other commands.
2. This command is ineffective when the buzzer has already been turned on by the [EZNOW_BUZZER_ON](#) command

About the ezNow Board:

The ezNow is a bare printed circuit board, which expands the capabilities of the ezLCD-004. It is user-configurable. When assembled, it can be attached to the back of the ezLCD-004 and used for the development purposes or as a finished product. The ezNow board is available from the Earth Computer Tech. Inc. For more information please, refer to the ezNow manual.

See Also: [EZNOW_BUZZER_ON](#), [EZNOW_BUZZER_OFF](#)

Example:

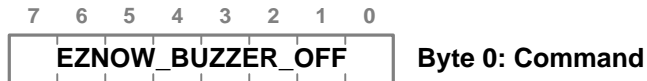
The following sequence will make the buzzer on the ezNow board beep for 200ms.

```
EZNOW_BUZZER_BEEP D2 hex (Command)
                   20      20 dec (Beep time = 20*0.01s = 0.2s = 200ms)
```

1.7.12 EZNOW_BUZZER_OFF

Description: Turns Off the buzzer on the ezNow board

Code: D0_{hex}, 208_{dec}



About the ezNow Board:

The ezNow is a bare printed circuit board, which expands the capabilities of the ezLCD-004. It is user-configurable. When assembled, it can be attached to the back of the ezLCD-004 and used for the development purposes or as a finished product. The ezNow board is available from the Earth Computer Tech. Inc. For more information please, refer to the ezNow manual.

See Also: [EZNOW_BUZZER_ON](#), [EZNOW_BUZZER_BEEP](#)

Example:

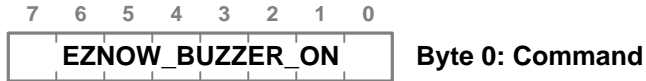
The following sequence will turn off the buzzer on the ezNow board.

`EZNOW_BUZZER_OFF D0 hex` (Command)

1.7.13 EZNOW_BUZZER_ON

Description: Turns On the buzzer on the ezNow board

Code: D1_{hex}, 209_{dec}



Prerequisites:

1. The ezNow board with a buzzer is attached to the connector CN1
2. CN1 pin 17 (SDA) function is changed to "Buzzer". This can be done by the ezLCDconfig utility. See Chapter: [Firmware Customization](#).

About the ezNow Board:

The ezNow is a bare printed circuit board, which expands the capabilities of the ezLCD-004. It is user-configurable. When assembled, it can be attached to the back of the ezLCD-004 and used for the development purposes or as a finished product. The ezNow board is available from the Earth Computer Tech. Inc. For more information please, refer to the ezNow manual.

See Also: [EZNOW_BUZZER_OFF](#), [EZNOW_BUZZER_BEEP](#)

Example:

The following sequence will turn on the buzzer on the ezNow board.

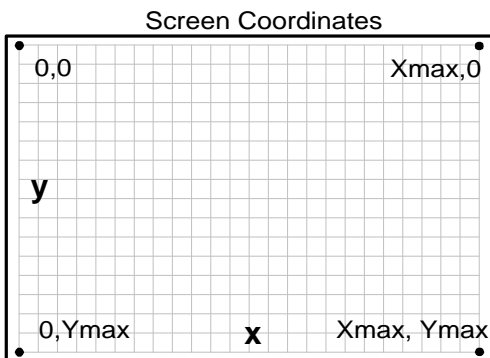
```
EZNOW_BUZZER_OFF D1 hex (Command)
```

1.7.14 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								
	x15	x14	x13	x12	x11	x10	x9	x8	Byte 0 (Command)
	x7	x6	x5	x4	x3	x2	x1	x0	Byte 1 (x MSB)
									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).

```

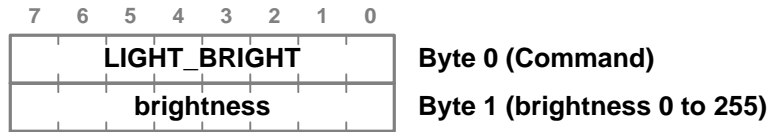
SET_COLORH 84 hex
GREEN_LSB  11100000 bin
GREEN_MSB  00000111 bin
SET_XHY    85 hex
0          0 dec (x MSB)
20         20 dec (x LSB)
60         60 dec (y)
H_LINEH  A0 hex
0          0 dec (x MSB)
170       170 dec (x LSB)

```

1.7.15 LIGHT_BRIGHT

Description: Sets the brightness of the screen backlight.

Code: 80_{hex}, 128_{dec}



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
```

1.7.16 LIGHT_OFF

Description: Turns off the screen backlight.

Code: 23_{hex}, 35_{dec}



See Also: [LIGHT_ON](#), [LIGHT_BRIGHT](#)

Example:

The following sequence will turn off the screen backlight.

`LIGHT_OFF` 23 hex

1.7.17 LIGHT_ON

Description: Turns on the screen backlight.

Code: 22_{hex}, 34_{dec}



See Also: [LIGHT_OFF](#), [LIGHT_BRIGHT](#)

Example:

The following sequence will turn on the screen backlight.

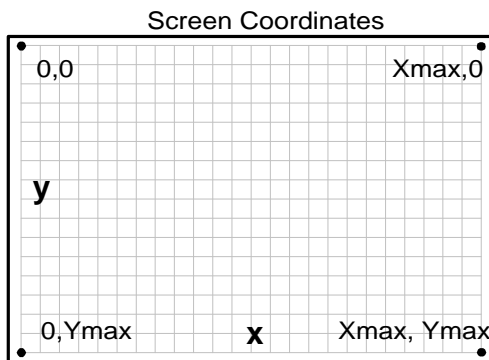
LIGHT_ON 22 hex

1.7.18 LINE_TO_XHY

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

Code: 88hex, 136dec

7	6	5	4	3	2	1	0	
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.

```

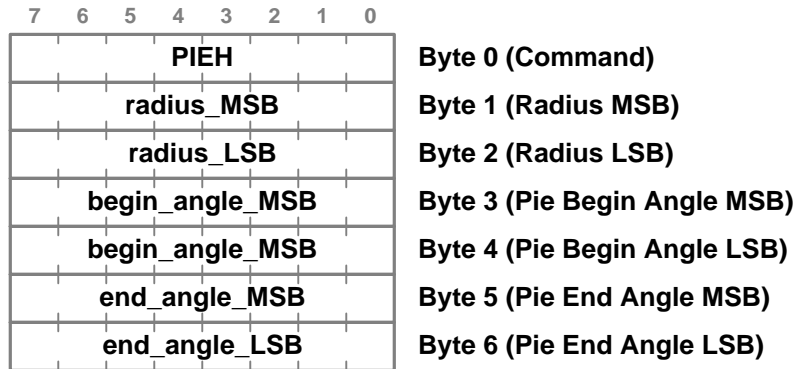
SET_COLORH 84 hex
RED_LSB    00000000 bin
RED_MSB    11111000 bin
SET_XHY    85 hex
0          0 dec (x0 MSB)
0          0 dec (x0 LSB)
0          0 dec (y0)
LINE_TO_XHY 88 hex
1          1 dec (x1 MSB)
63         63 dec (x1 LSB)
233        233 dec (y1)

```

1.7.19 PIEH

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

Code: 90_{hex}, 144_{dec}



See Also: [ARCH](#), [SET_XHY](#), [SET_COLORH](#), [CIRCLE_RH](#)

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

$$\text{Angle_lcd} = \text{Angle_deg} \times 2048 / 45$$

For example:

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

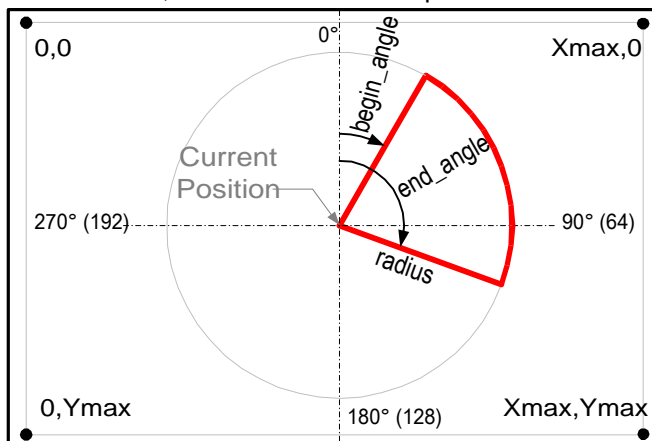
$$4096_{\text{dec}} = 1000_{\text{hex}} = 90^\circ$$

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

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

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

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



Example:

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

```
SET_COLORH 84 hex
GREEN_LSB  11100000 bin
GREEN_MSB  00000111 bin
SET_XHY    85 hex
  0        0 dec (x MSB)
160       160 dec (x LSB)
117       117 dec (y)
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)
```

1.7.20 PING

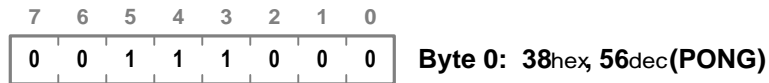
Description: Checks if the ezLCD is connected and ready to receive commands.

Code: 83_{hex}, 131_{dec}



ezLCD Response

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



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

Example:

The following sequence will check if the ezLCD is OK:

PING 83 hex

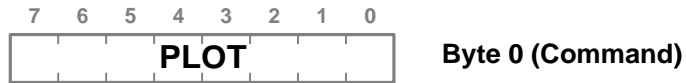
If the ezLCD is connected and ready to receive commands, it responds with:

38 hex

1.7.21 PLOT

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

Code: 26_{hex}, 38_{dec}



See Also: [SET_XHY](#), [SET_COLORH](#)

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

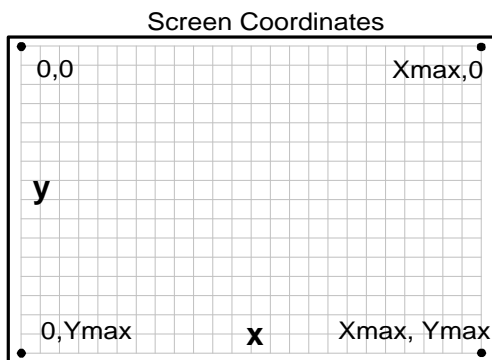
```

1.7.22 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).

```

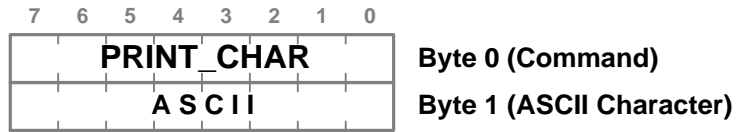
SET_COLORH 84 hex
RED_LSB    0000000 bin
RED_MSB    11111000 bin
PLOT_XHY   87 hex
1          1 dec (x MSB)
60        160 dec (x LSB 1*256+54=310)
117      117 dec (y)

```

1.7.23 PRINT_CHAR

Description: Prints a character at Current Position.

Code: 2Chex, 44dec



See Also: [SELECT_FONT](#), [PRINT_STRING](#)

Example:

The following sequence will print a black character 'M' using Font 2.

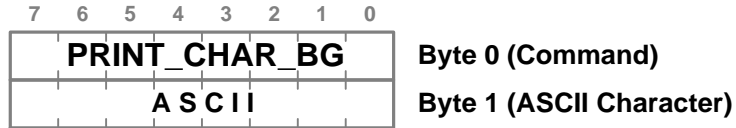
```

SELECT_FONT  2B hex
2            2 dec
SET_COLORH  84 hex
BLACK_LSB   0000000 bin
BLACK_MSB   0000000 bin
PRINT_CHAR  2C hex
'M'        4D hex
  
```

1.7.24 PRINT_CHAR_BG

Description: Prints a character at Current Position on the background specified by the [SET_BG_COLORH](#) command.

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



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.

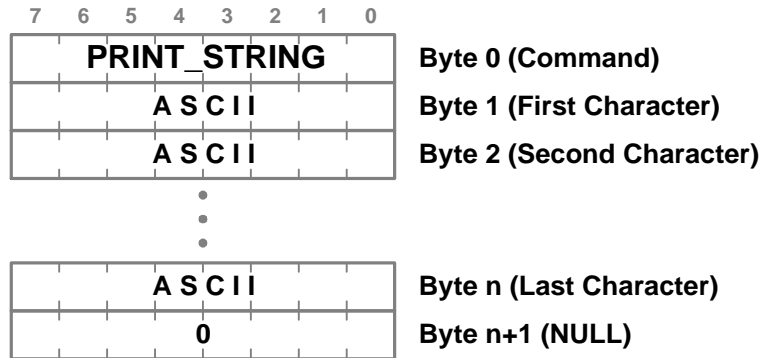
```

SELECT_FONT      2B hex
2                2 dec
SET_BG_COLORH   94 hex
BLACK_LSB       0000000 bin
BLACK_MSB       0000000 bin
SET_COLORH      84 hex
WHITE_LSB       1111111 bin
WHITE_MSB       1111111 bin
PRINT_CHAR_BG   3C hex
'M'             4D hex
  
```

1.7.25 PRINT_STRING

Description: Prints a null-terminated String starting at Current Position.

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



See Also: [SELECT_FONT](#), [PRINT_CHAR](#)

Example:

The following sequence will print "LCD" in purple using Font 1 at (160, 117).

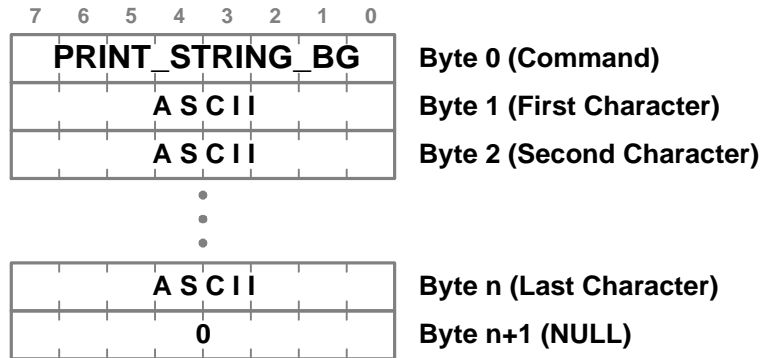
```

SELECT_FONT  2B hex
  1          1 dec
SET_COLORH   84 hex
PURPLE_LSB   00010000 bin
PURPLE_MSB   10000000 bin
SET_XHY     85 hex
  0         0 dec (x MSB)
160         160 dec (x LSB)
117        117 dec (y)
PRINT_STRING 2D hex
  'L'       4C hex
  'C'       43 hex
  'D'       44 hex
NULL        0 hex
  
```

1.7.26 PRINT_STRING_BG

Description: Prints null-terminated String starting at Current Position on the background specified by [SET_BG_COLORH](#) command

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



See Also: [SELECT_FONT](#), [SET_BG_COLORH](#), [PRINT_CHAR_BG](#)

Example:

The following sequence print "LCD" in yellow on a navy background in the middle of the screen using Font 0.

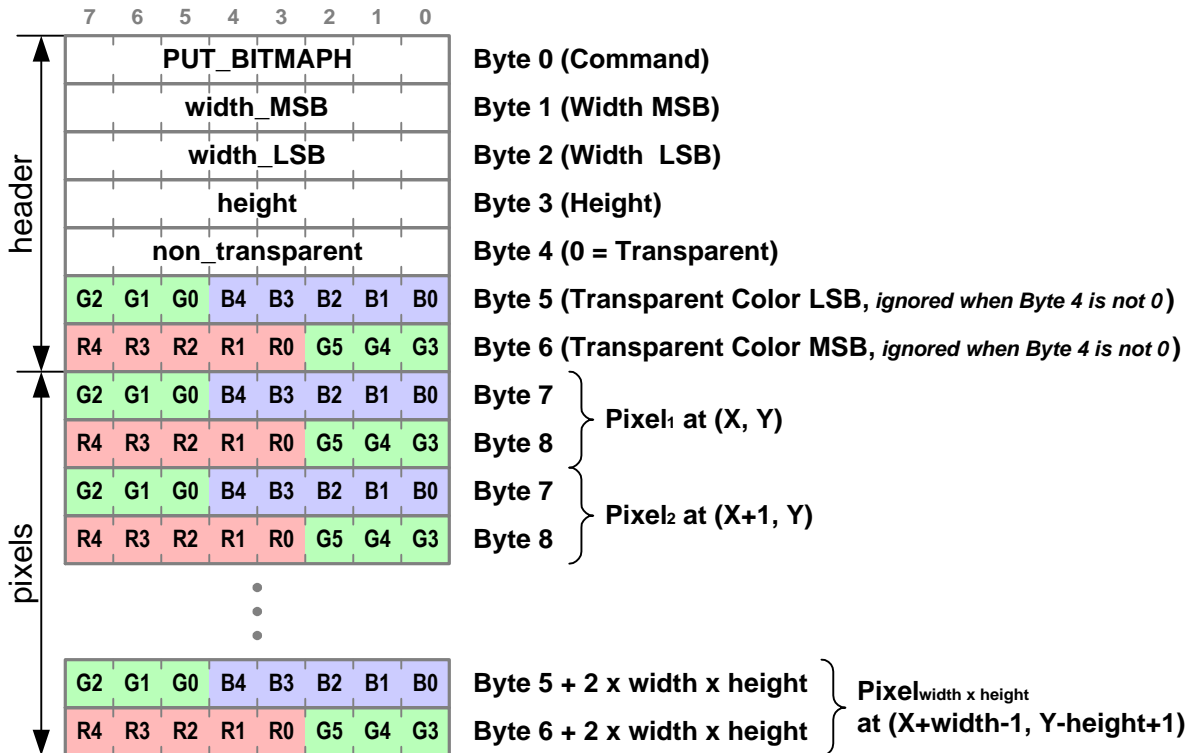
```

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

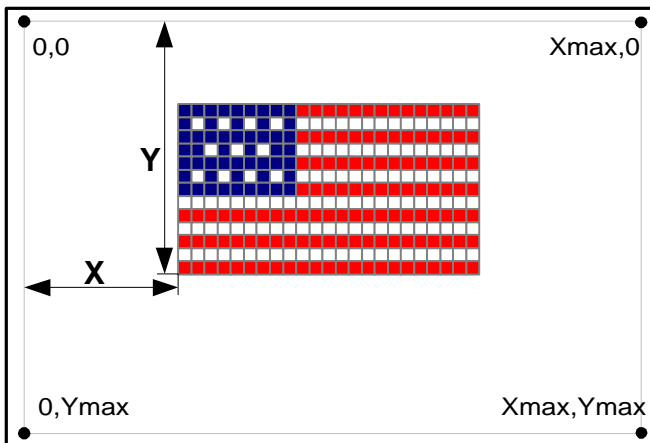
```

1.7.27 PUT_BITMAP

Description: Displays a Bitmap on the screen starting at Current Position, then UP and RIGHT?
Code: 9E_{hex}, 158_{dec}



- Notes:**
1. The total number of bytes is: $2 \times \text{width} \times \text{height} + 7$
 2. When Byte 4 = 0, Bytes 5 and 6 specify the Transparent Color. Pixels equal to the Transparent Color are ignored during bitmap drawing. All pixels are drawn when Byte 4 is not 0.



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



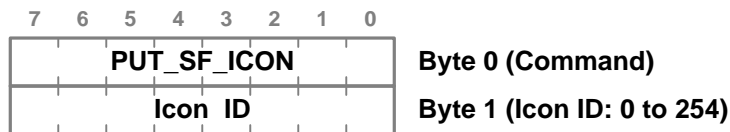
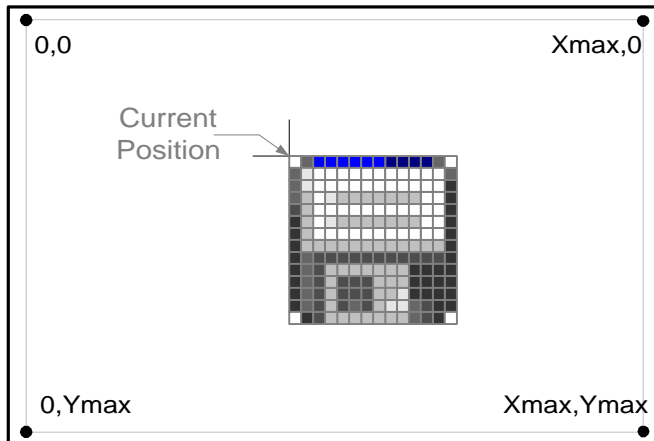
Pixel 1

See Also: [SET_XHY](#), [SET_COLORH](#)

1.7.28 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}



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

See Also: [SET_XHY](#)

Example:

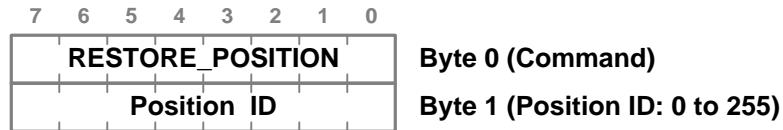
The following sequence will display Icon No. 3 with its upper-left corner positioned at (60, 43).

```
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
```

1.7.29 RESTORE_POSITION

Description: Restores the Current Position saved by the [SAVE_POSITION](#) command.

Code: 36hex, 54dec



See Also: [SAVE_POSITION](#), [SET_XHY](#)

Example:

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

```

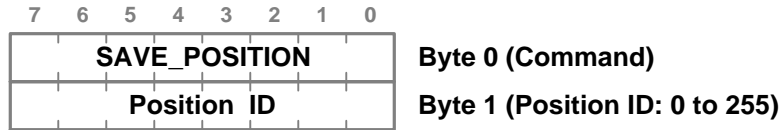
SET_XHY          85 hex
  0              0 dec (x MSB)
 160             160 dec (x LSB)
 117             117 dec (y)
SAVE_POSITION    35 hex
 12             12 dec (Position ID)
LINE_TO_XHY     88 hex
  0              0 dec (x1 MSB)
 247            247 dec (x1 LSB)
 67             67 dec (y1)
RESTORE_POSITION 36 hex
 12             12 dec (Position ID)
LINE_TO_XHY     88 hex
  0              0 dec (x1 MSB)
 73             73 dec (x1 LSB)
 67             67 dec (y1)
RESTORE_POSITION 36 hex
 12             12 dec (Position ID)
V_LINE          41 hex
 217           217 dec

```

1.7.30 SAVE_POSITION

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

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



See Also: [RESTORE_POSITION](#), [SET_XHY](#)

Example:

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

```

SET_XHY          85 hex
  0              0 dec (x MSB)
 160             160 dec (x LSB)
 117             117 dec (y)
SAVE_POSITION  35 hex
  12             12 dec (Position ID)
LINE_TO_XHY     88 hex
  0              0 dec (x1 MSB)
 247             247 dec (x1 LSB)
  67             67 dec (y1)
RESTORE_POSITION 36 hex
  12             12 dec (Position ID)
LINE_TO_XHY     88 hex
  0              0 dec (x1 MSB)
  73             73 dec (x1 LSB)
  67             67 dec (y1)
RESTORE_POSITION 36 hex
  12             12 dec (Position ID)
V_LINE          41 hex
 217            217 dec

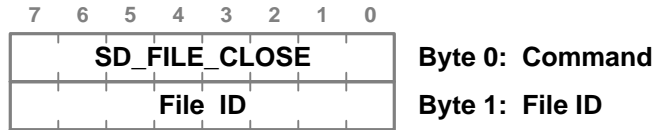
```

1.7.31 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}



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.

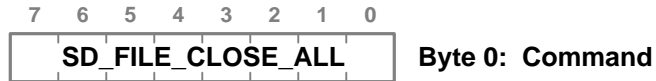
```
SD_FILE_CLOSE  72 hex
1              1 dec (File ID)
```

1.7.32 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}



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.

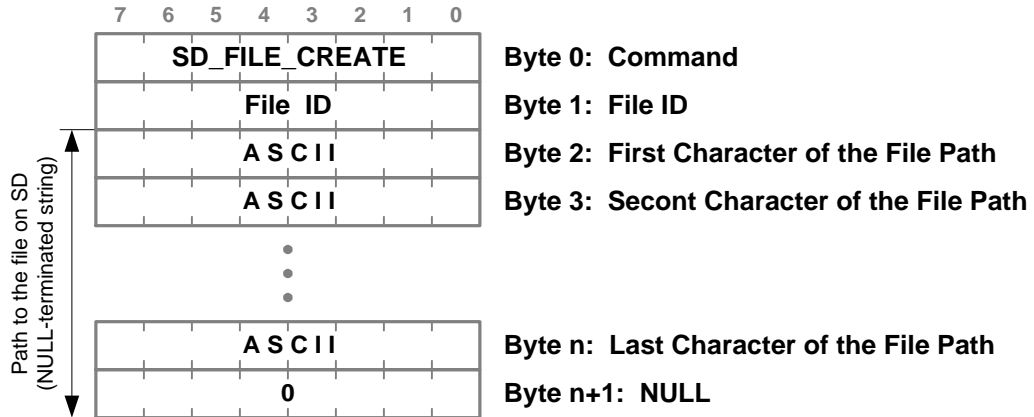
SD_FILE_CLOSE_ALL 73 hex

1.7.33 SD_FILE_CREATE

Description: Creates a new SD Flash file and opens it for writing. File Position Index is set to 0.

Supported file systems: FAT12, FAT16, FAT32

Code: 76hex, 118dec



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

See Also: [SD_FILE_OPEN](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

About the File ID:

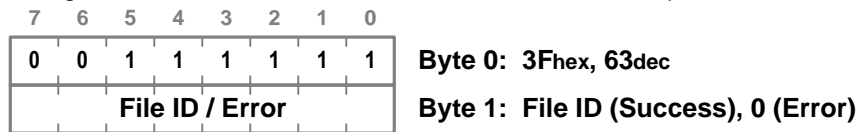
File ID identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

About the File Path:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A: /Cat/Jumped/Over.txt and cat / jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 64 bytes.

ezLCD Response

After receiving the SD_FILE_CREATE command, the ezLCD responds with the following sequence:



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

Touch Screen Processing

SD_FILE_CREATE command temporarily disables the touch screen. The touch screen will be automatically re-enabled when all files are closed. This can be done by issuing the [SD_FILE_CLOSE](#) or [SD_FILE_CLOSE_ALL](#) command.

Note: The touch screen is temporary disabled, even if due to error no file is created. If this is the case, issuing "dummy" [SD_FILE_CLOSE](#) or [SD_FILE_CLOSE_ALL](#) command will re-enable the touch screen.

Example:

The following sequence will create and open file MyFile.dat

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

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

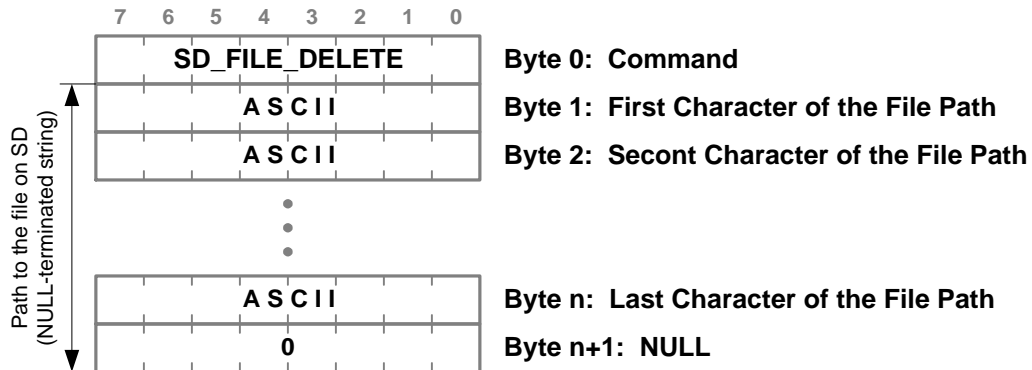
```
3F hex
 1 dec
```

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

```
3F hex
 0 dec
```

1.7.34 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.

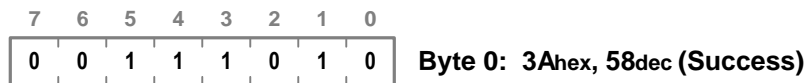
About the File Path:

- 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 64 bytes.
- Wildcards are not allowed.

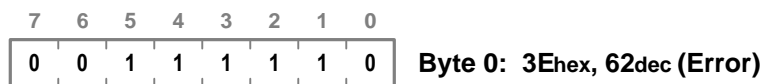
ezLCD Response

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

In case of the **success**:



In case of an **error**:



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

SD_FILE_DELETE example is shown on the next page

Example:

The following sequence will delete file MyFile.dat

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

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

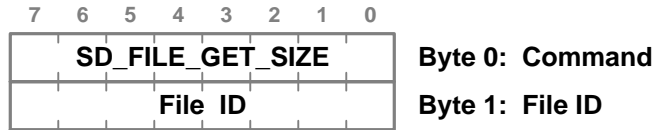
In case of the failure, the following sequence will be sent by the ezLCD:
3E hex

1.7.35 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}



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

See Also: [SD_FILE_OPEN](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

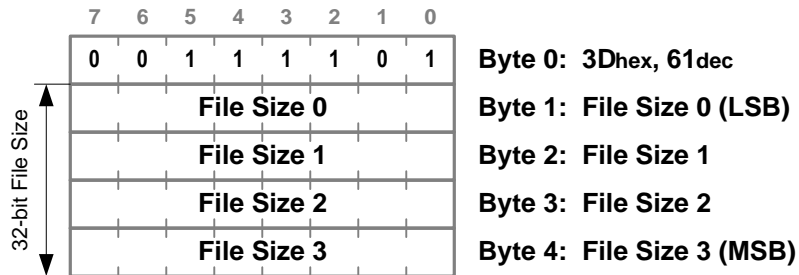
About the File ID:

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

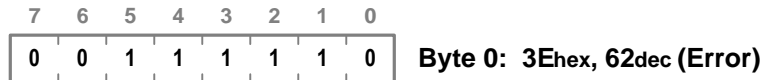
ezLCD Response

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

In case of the **success**:



In case of an **error**:

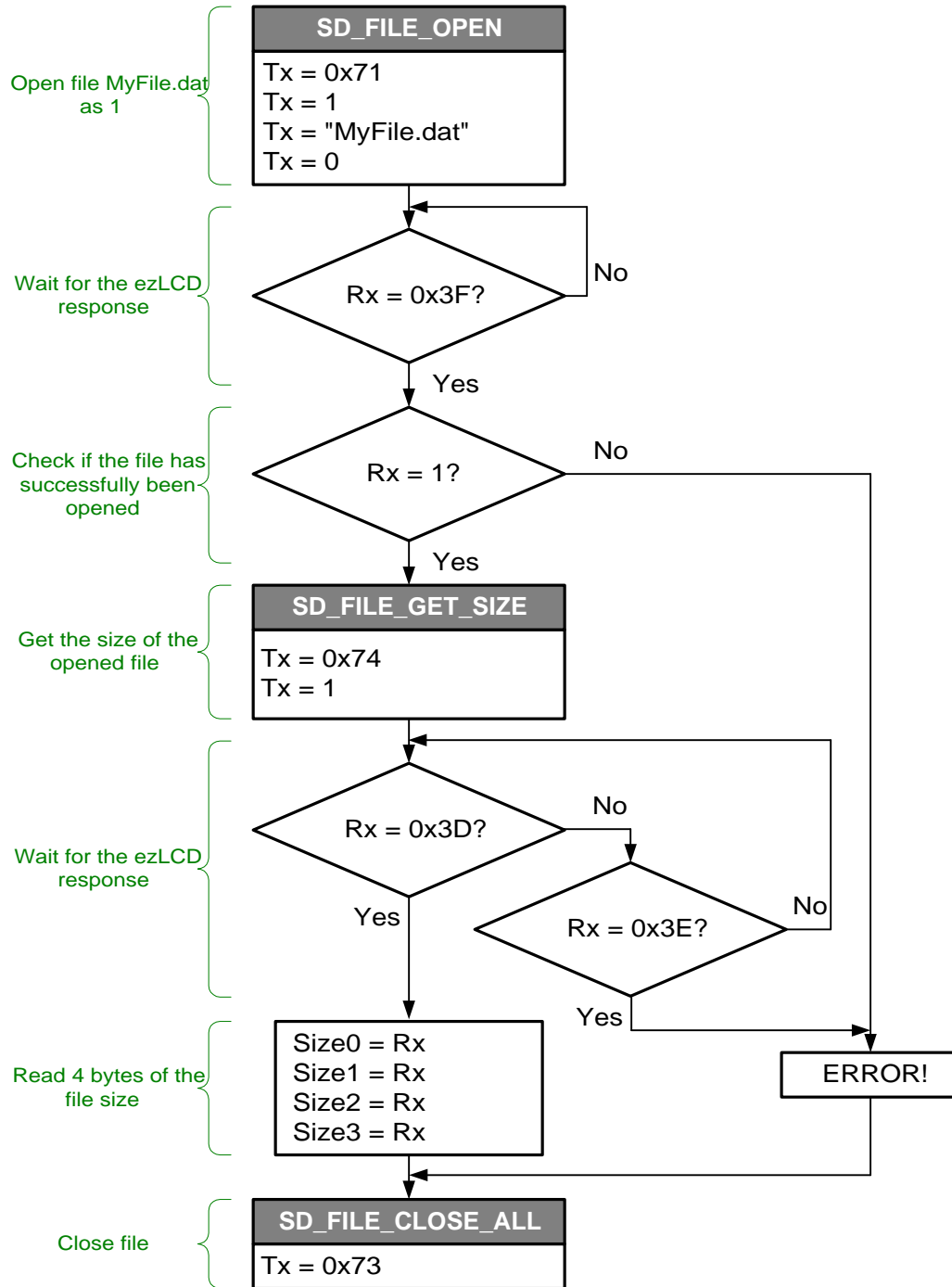


The ezLCD response is sent through the same interface, which received the SD_FILE_GET_SIZE command.

SD_FILE_GET_SIZE example is shown on the next page.

Example:

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

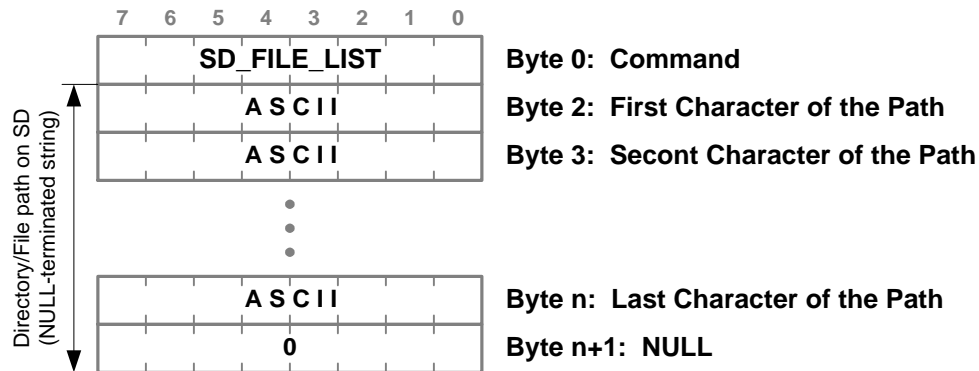


1.7.36 SD_FILE_LIST

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

Supported file systems: FAT12, FAT16, FAT32

Code: 79hex, 121dec



Notes: 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 64 bytes..

ezLCD Response

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

In case of the **success**:

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

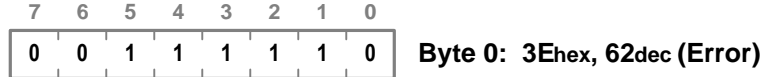
- Entries (files or sub-directories) are separated by the Line Feed character (**0Ahex or 10dec**)
- Entries are sent in no particular order
- Sub-directories have '/' as their last character

For example:

```

3Ahex      Start
whatever.txt file
Pictures/   directory
Cat.doc     file
ezLCD.bin  file
SOURCES/    directory
0         End (NULL)
  
```

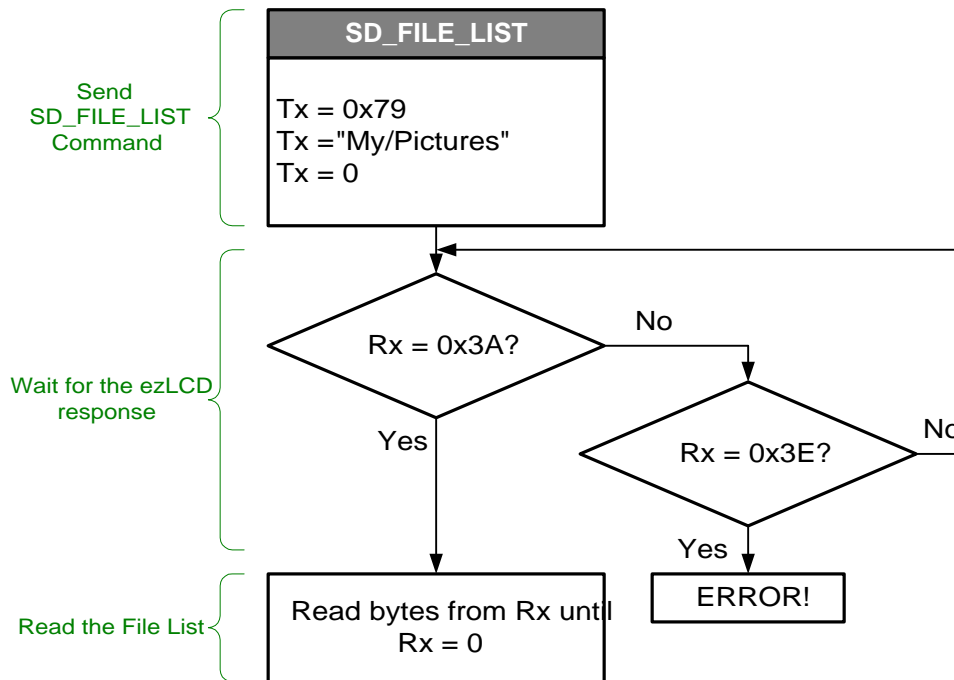
In case of an **error**:



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

Example:

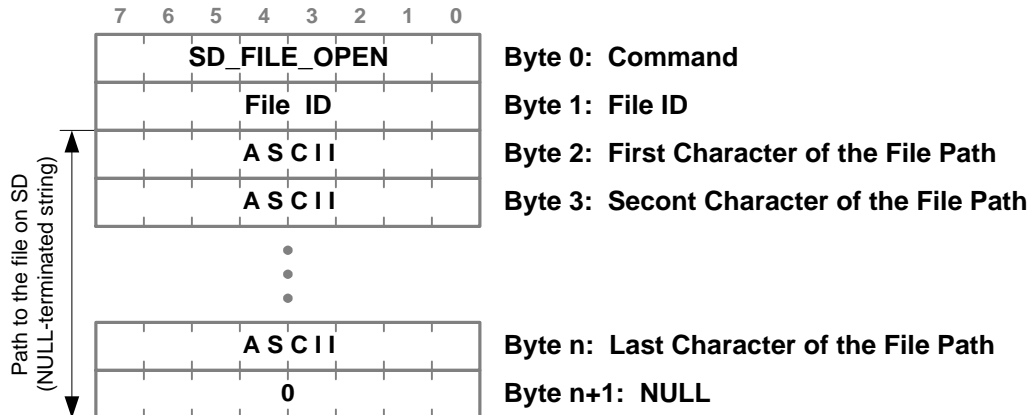
The following flow chart shows an example of reading the file list from the directory My/Pictures



1.7.37 SD_FILE_OPEN

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

Code: 71hex, 113dec



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

See Also: [SD_FILE_CREATE](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

About the File ID:

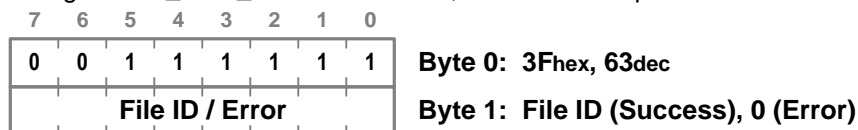
File ID identifies the file after it has been opened. Since maximum 2 files may be concurrently opened, the File ID should be: 1 or 2. Values higher than 2 are interpreted as 2 and 0 is interpreted as 1.

About the File Path:

- File Path specifies the full path to the file on SD including directory, filename and extension
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- File Path is not case-sensitive. The drive and root directory do not have to be indicated, for example, both: A:/Cat/Jumped/Over.txt and cat/jumped/over.TXT specify the same file.
- Long file names are supported, however the File Path (directory + filename + extension + NULL) may not exceed 64 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:

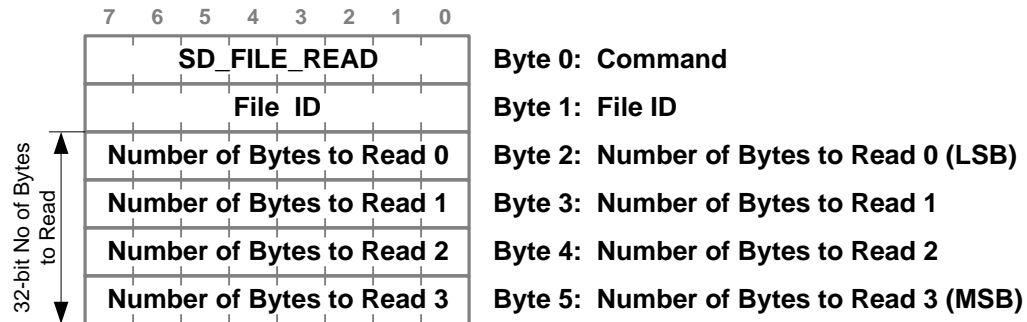
```
3F hex
 0 dec
```

1.7.38 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](#)

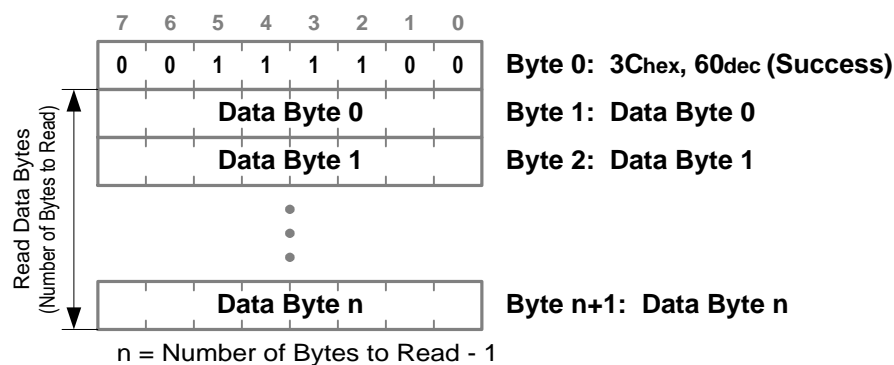
About the File ID:

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

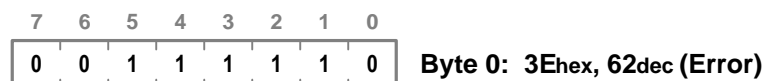
After receiving the SD_FILE_READ command, the ezLCD responds with either of the following sequences:

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.

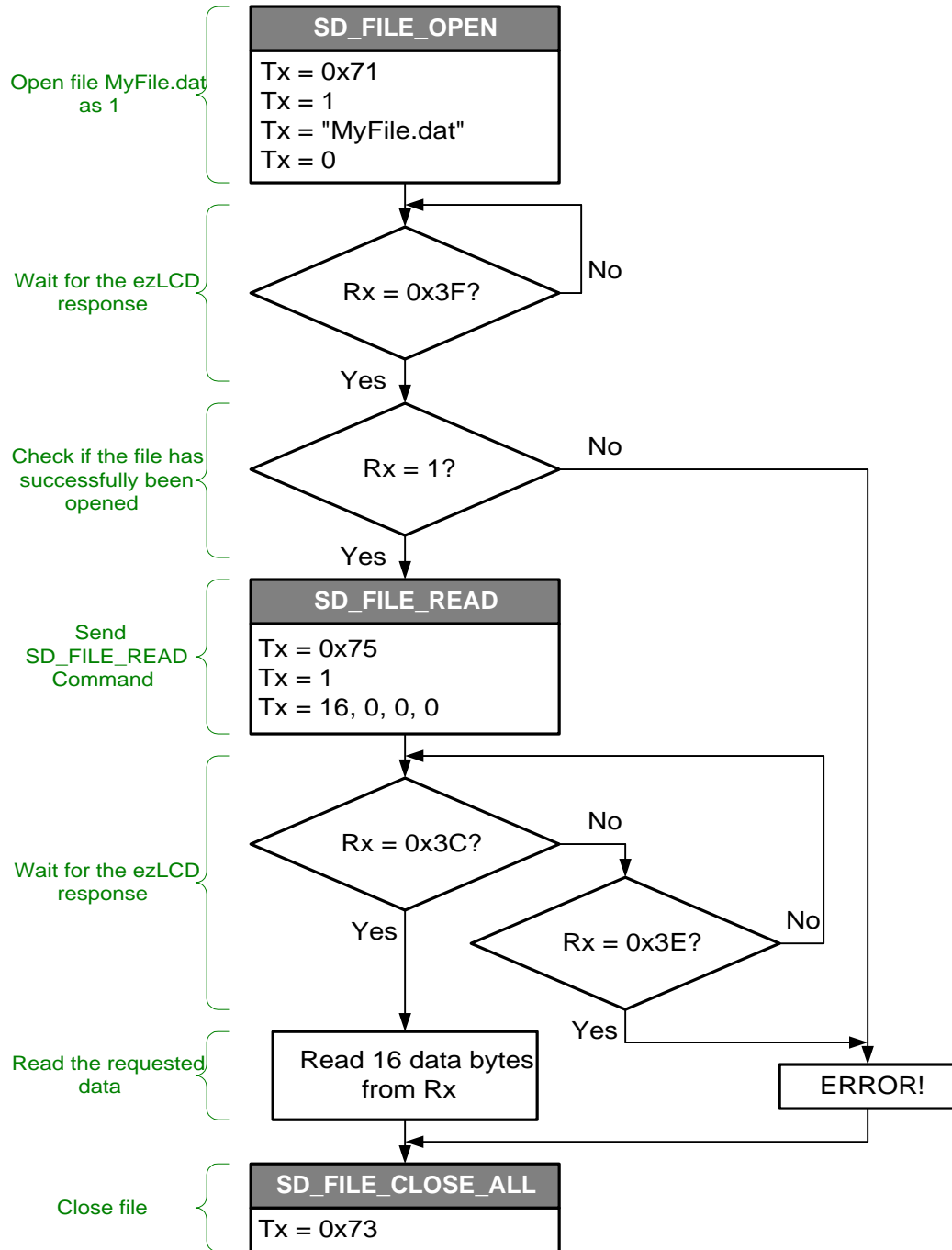
In case of an **error**:



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

Example:

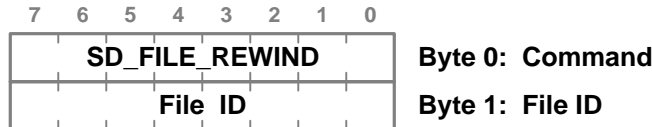
The following flow chart shows an example of reading first 16 bytes from the file MyFile.dat



1.7.39 SD_FILE_REWIND

Description: Moves the File Position Index to the beginning of the opened SD Flash file.
Supported file systems: FAT12, FAT16, FAT32

Code: 7Ah_{hex}, 122_{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_OPEN](#), [SD_FILE_SEEK](#), [SD_FILE_READ](#), [SD_FILE_WRITE](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

About the File ID:

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

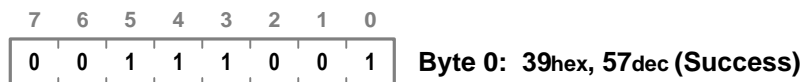
About the File Position Index

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

ezLCD Response

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

In case of the **success**:



In case of an **error**:



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

Example:

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

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

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

39 hex

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

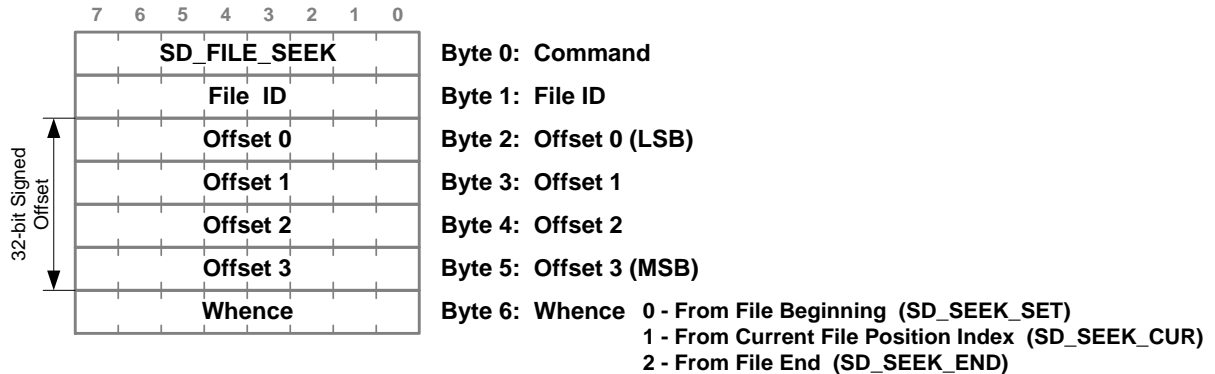
3E hex

1.7.40 SD_FILE_SEEK

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

Supported file systems: FAT12, FAT16, FAT32

Code: 7Chex, 124dec



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

This command works only if the file is already opened by the [SD_FILE_OPEN](#) command, or created and opened by the [SD_FILE_CREATE](#) command.

See Also: [SD_FILE_OPEN](#), [SD_FILE_REWIND](#), [SD_FILE_READ](#), [SD_FILE_WRITE](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

About the Offset:

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

About the File Position Index

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

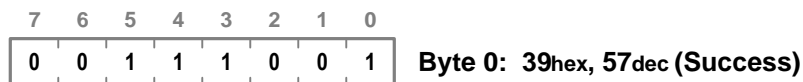
About the File ID:

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

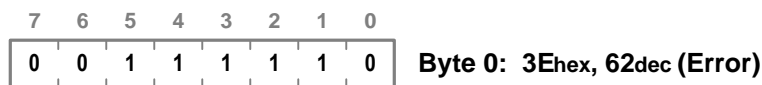
ezLCD Response

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

In case of the **success**:



In case of an **error**:



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

Example:

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

```
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:

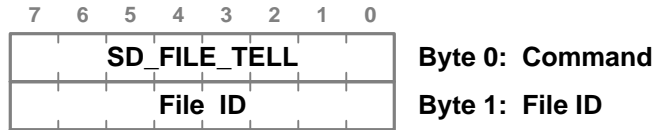
```
3E hex
```

1.7.41 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}



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

This command works only if the file is already opened by the [SD_FILE_OPEN](#) command, or created and opened by the [SD_FILE_CREATE](#) command.

See Also: [SD_FILE_OPEN](#), [SD_FILE_SEEK](#), [SD_FILE_CLOSE](#), [SD_FILE_CLOSE_ALL](#)

About the File ID:

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

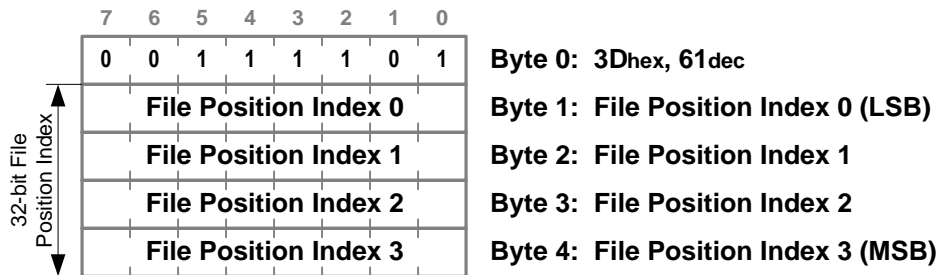
About the File Position Index

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

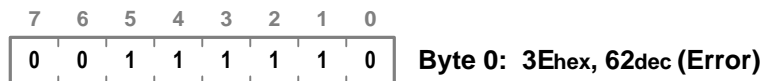
ezLCD Response

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

In case of the **success**:



In case of an **error**:

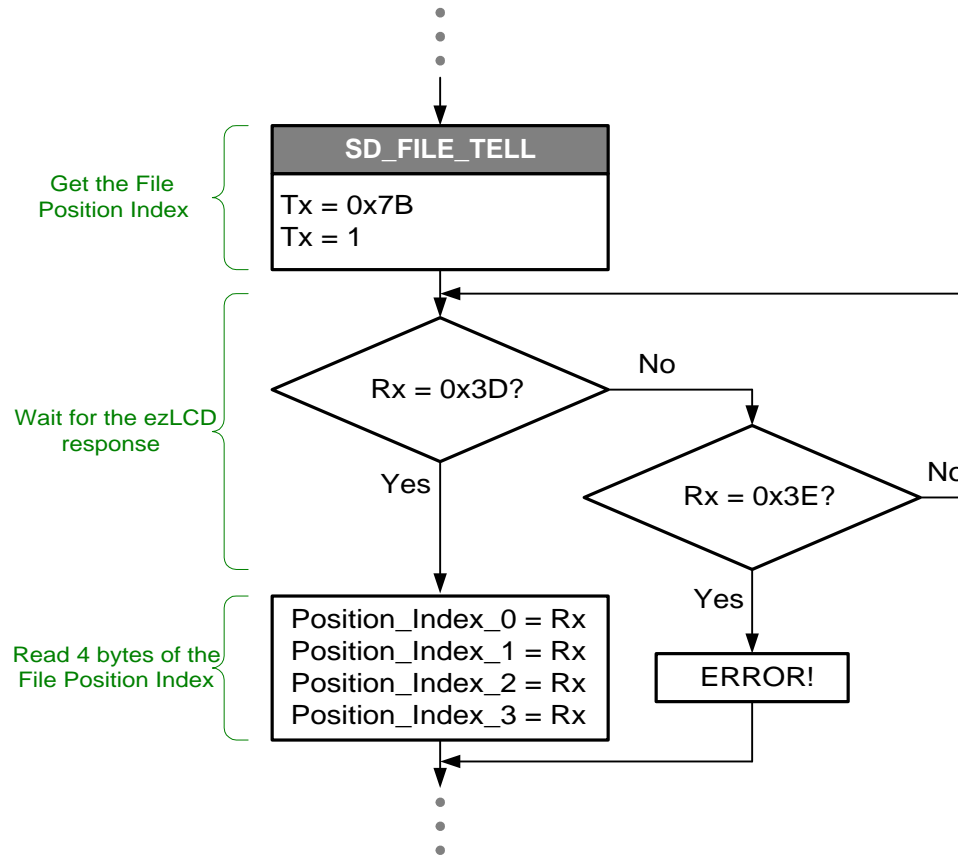


The ezLCD response is sent through the same interface, which received the SD_FILE_TELL command.

SD_FILE_TELL example is shown on the next page.

Example:

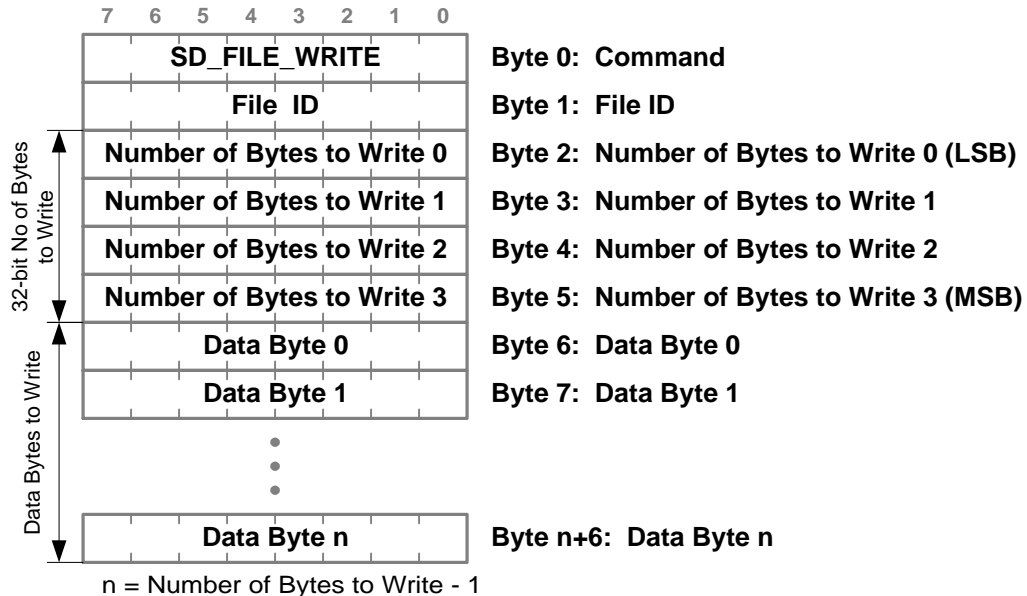
The following flow chart shows an example of getting the File Position Index of the opened file with File Id = 1.



1.7.42 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](#)

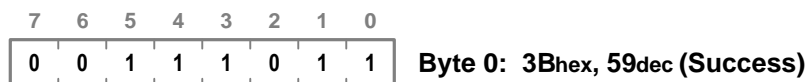
About the File ID:

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.

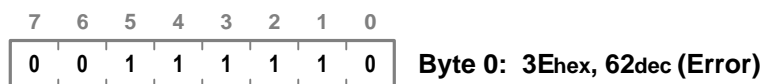
ezLCD Response

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

In case of the **success**:



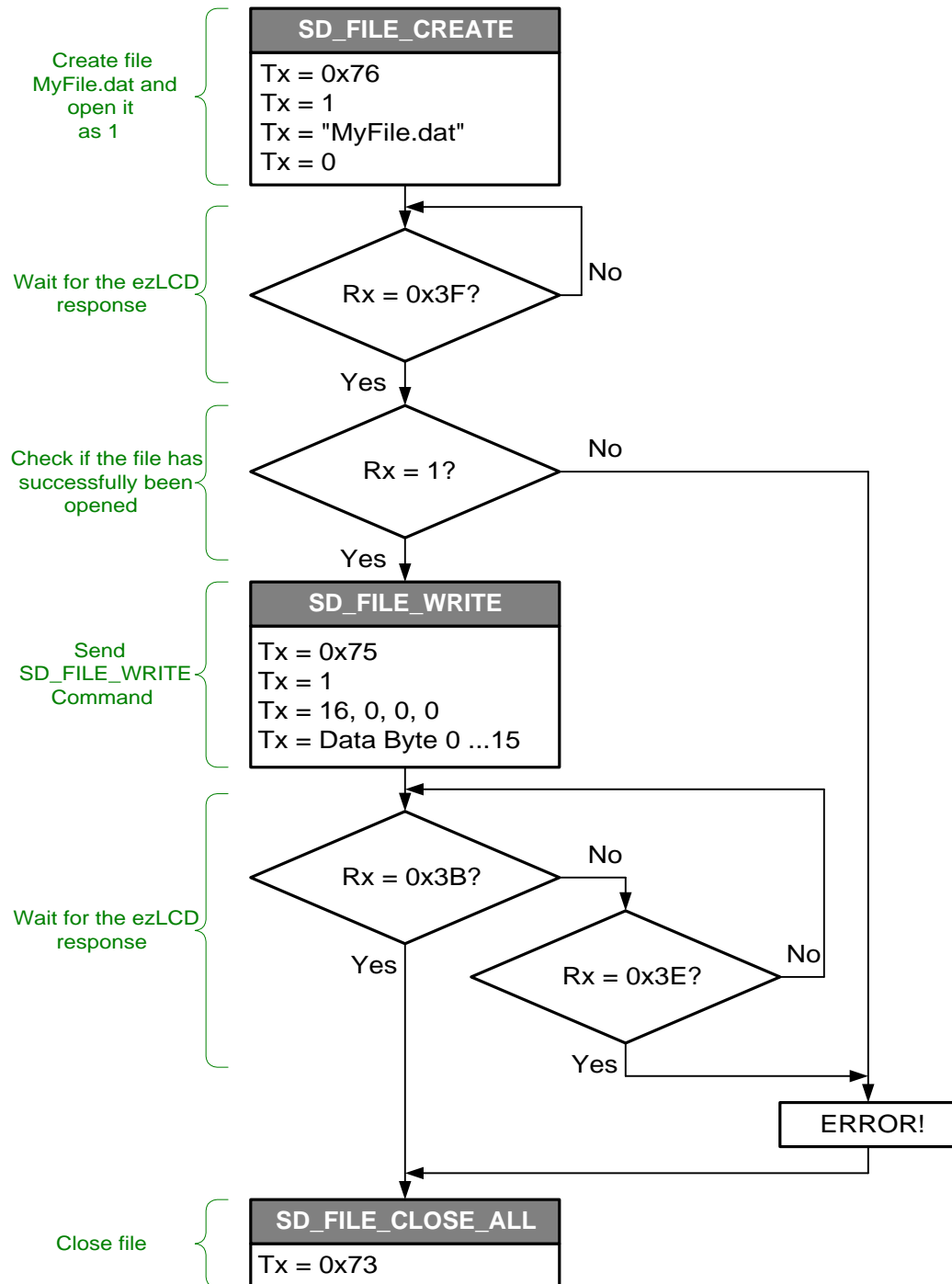
In case of an **error**:



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

Example:

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

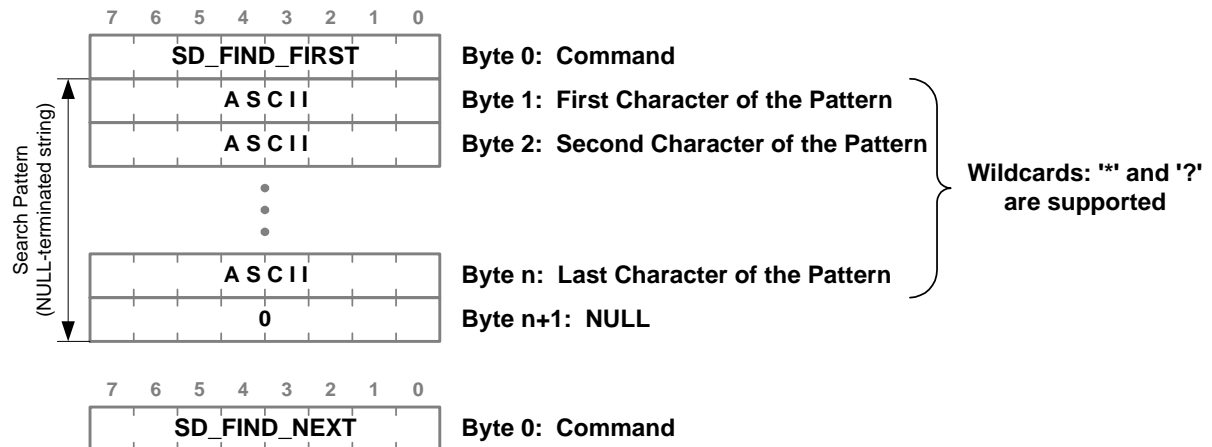


1.7.43 SD_FIND_FIRST and SD_FIND_NEXT

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

Supported file systems: FAT12, FAT16, FAT32

Codes: **SD_FIND_FIRST:** 4A_{hex}, 74_{dec}
 SD_FIND_NEXT : 4B_{hex}, 75_{dec}



SD_FIND_FIRST gets only the first found file or directory, which matches the search pattern. Each time, the SD_FIND_NEXT is issued, it finds the next file or directory, which matches the search pattern specified in the last SD_FIND_FIRST command.

The difference between the above described mechanism and [SD_FILE_LIST](#) command is that it obtains the files and directories one by one, while SD_FILE_LIST obtains them all at once.

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

About the Search Pattern:

- Specifies the path to the SD directory, SD file or group of files and sub-directories.
- Wildcards: '*' and '?' are supported
- Directories should be separated by: / (**not by:** \ like in Windows and DOS).
- Search Pattern is not case-sensitive. The drive and root directory do not have to be indicated, for example: A: /Cat/Jumped/Over, CAT/juMpEd/OvEr/ and cat/jumped/over specify the same.
- Long directory and file names are supported, however the Search Pattern + NULL may not exceed 64 bytes..

See Also: [SD_FILE_LIST](#)

ezLCD Response

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

In case of the **success:**

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

Examples:

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

or

3A_{hex} *Start*
 Pictures/ *directory*
0 *End (NULL)*

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

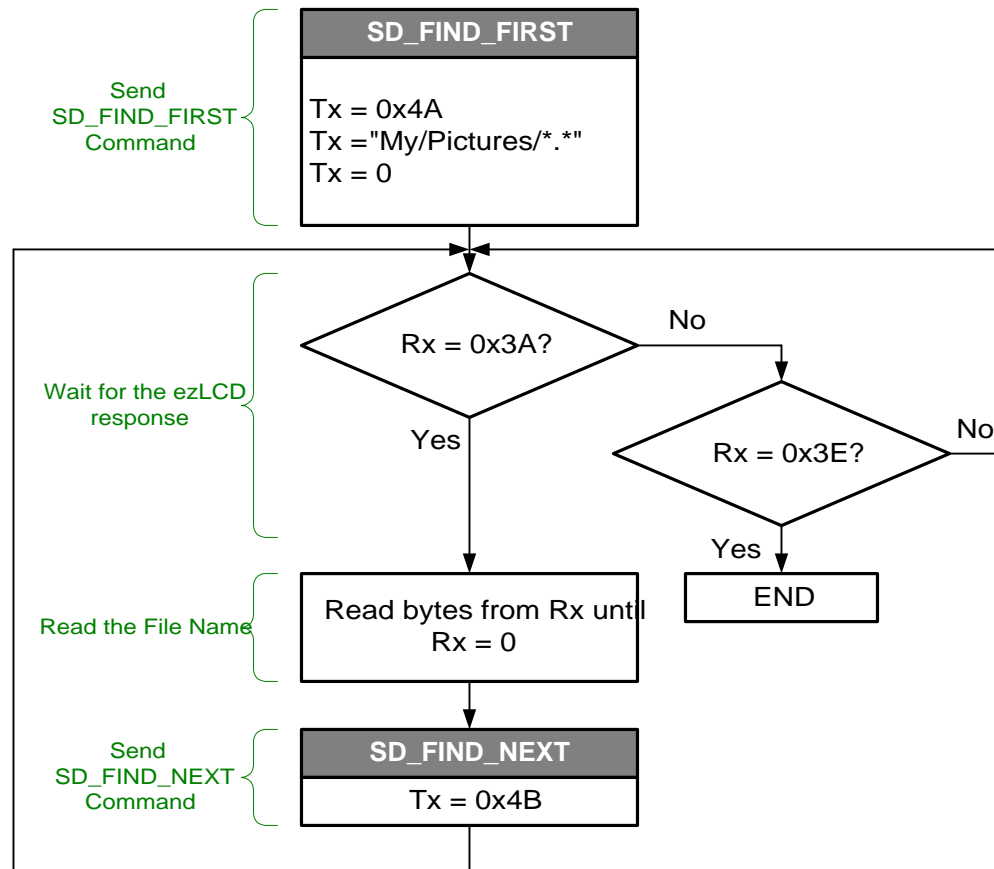
7	6	5	4	3	2	1	0
0	0	1	1	1	1	1	0

Byte 0: 3E_{hex}, **62**_{dec} (Error)

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

Example:

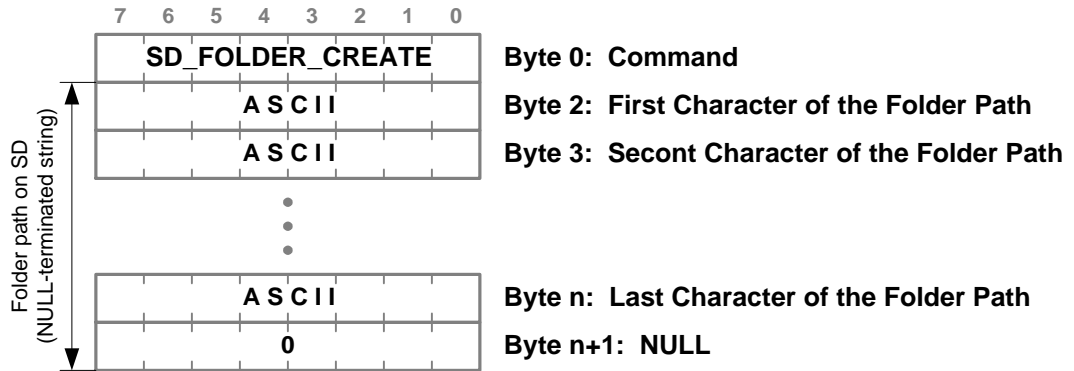
The following flow chart shows an example of reading the file list from the directory My/Pictures



1.7.44 SD_FOLDER_CREATE

Description: Creates a new folder (directory) on the SD.
This command is similar to the DOS "mkdir" command.
Supported file systems: FAT12, FAT16, FAT32

Code: 46hex, 70dec



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

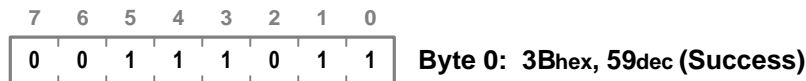
About the Folder Path:

- Folder Path specifies the full path to the directory on the SD.
- Directories (folders) should be separated by: / (**not by:** \ like in Windows and DOS).
- Long names are supported, however the Folder Path (+ NULL) may not exceed 64 bytes.

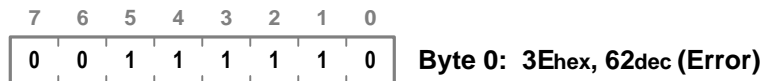
ezLCD Response

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
```

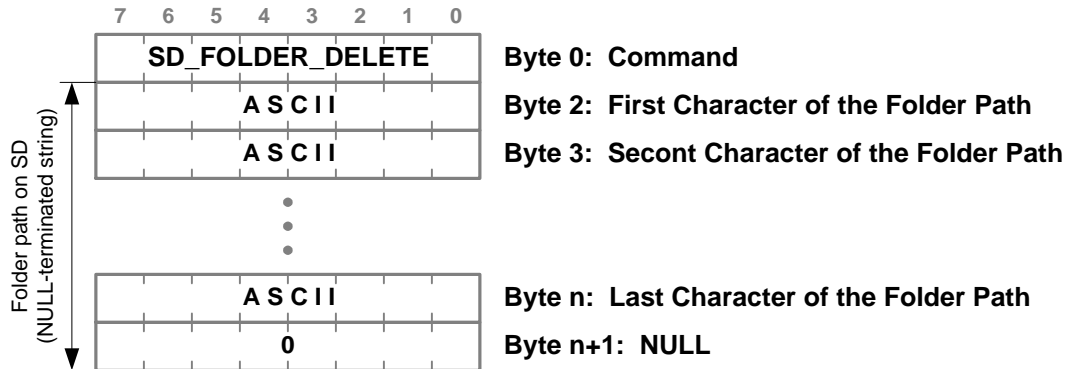
In case of the failure, the following sequence will be sent by the ezLCD:

```
3E hex
```

1.7.45 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

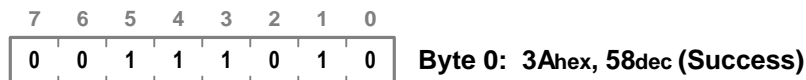
About the Folder Path:

- 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 64 bytes.
- Wildcards are not allowed.

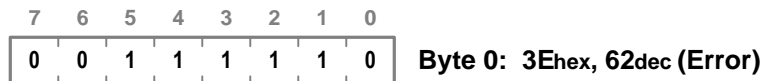
ezLCD Response

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

In case of the **success**:



In case of an **error**:



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

SD_FOLDER_DELETE example is shown on the next page

Example:

The following sequence will delete folder MyDir from the root directory

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

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

3A hex

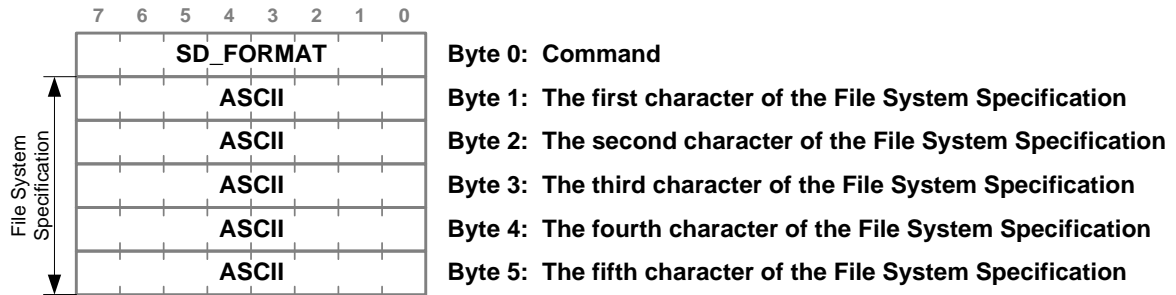
In case of the failure, the following sequence will be sent by the ezLCD:

3E hex

1.7.46 SD_FORMAT

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

Code: 4F_{hex}, 79_{dec}



Warning: This command will erase all files on the SD

About the File System Specification:

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

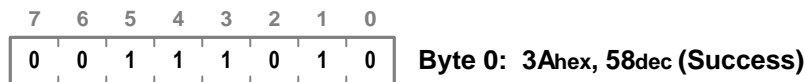
About the supported file systems

	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

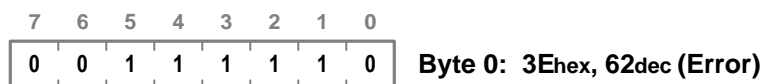
ezLCD Response

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

In case of the **success**:



In case of an **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

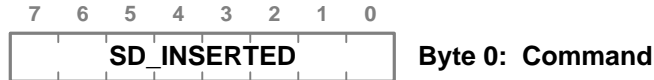
If the folder has successfully been deleted, the ezLCD responds with the following sequence:
3A hex

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

1.7.47 SD_INSERTED

Description: Checks if the SD card is inserted

Code: 49_{hex}, 73_{dec}



ezLCD Response

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

If an SD card is inserted in the SD slot:



If there is no card inserted in the SD slot:



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

Example:

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

SD_INSERTED 49 hex

If an SD card is present in the SD slot:

3D hex

If there is no card inserted in the SD slot:

3E hex

Example:

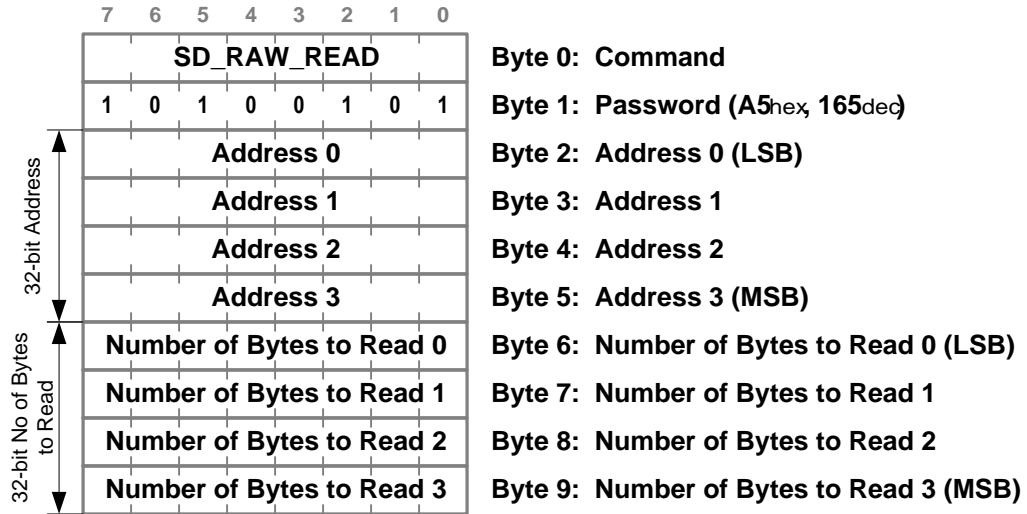
The following sequence will display the bitmap from file ezLCD.ezp with its upper-left corner positioned at (60, 43).

```
SET_XHY      85 hex
  0           0 dec (x MSB)
  60          60 dec (x LSB)
  43          43 dec (y)
SD_PUT_ICON  70 hex
'e'          65 hex
'z'          7A hex
'L'          4C hex
'C'          43 hex
'D'          44 hex
'.'          2E hex
'e'          65 hex
'z'          7A hex
'p'          70 hex
NULL         0 hex
```

1.7.49 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}

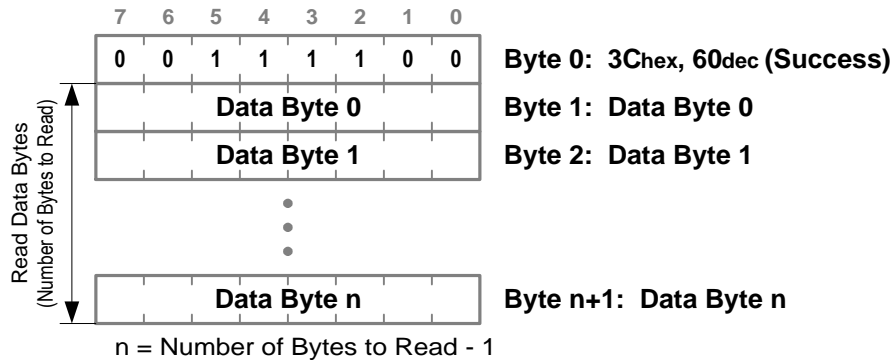


See Also: [SD_RAW_WRITE](#)

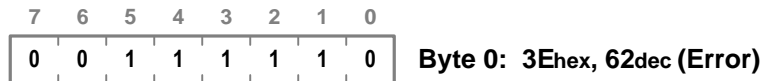
ezLCD Response

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

In case of the **success**:



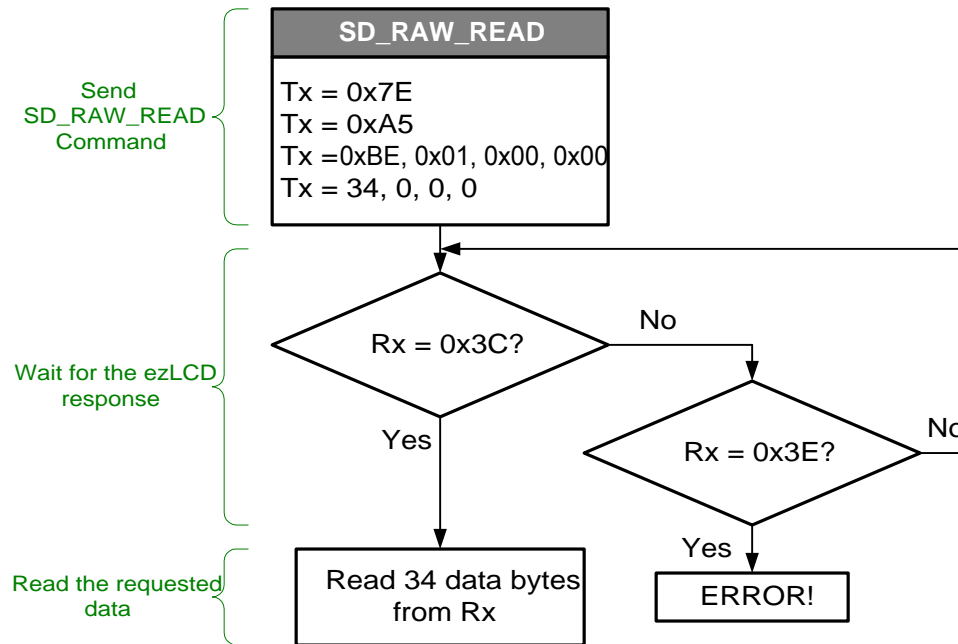
In case of an **error**:



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

Example:

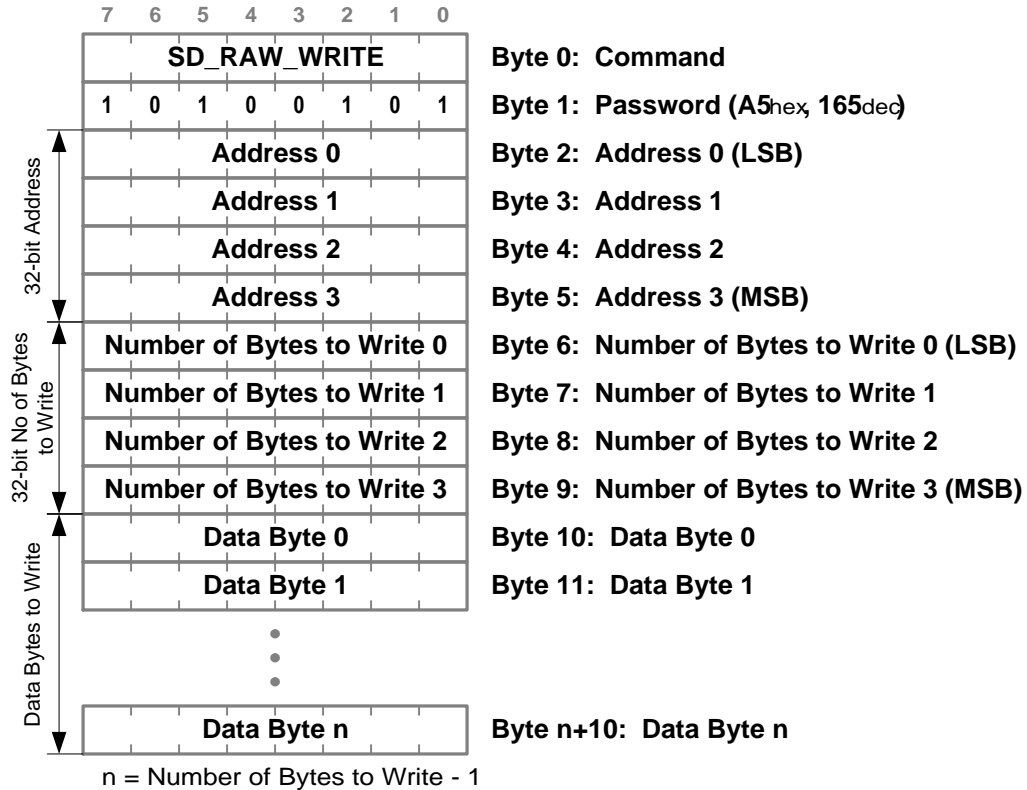
The following flow chart shows an example of reading 34 bytes starting from the SD address 000001BE_{hex}.



1.7.50 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}



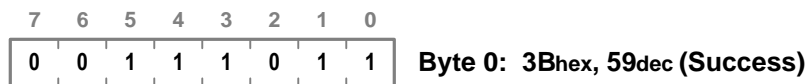
Warning! This command performs raw write on SD. Use it with caution. It may overwrite the existing SD files or corrupt the SD file formatting.

See Also: [SD_RAW_READ](#)

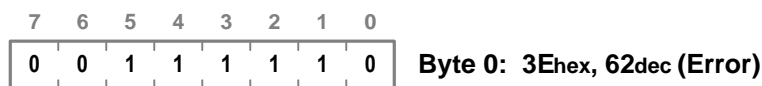
ezLCD Response

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

In case of the **success**:



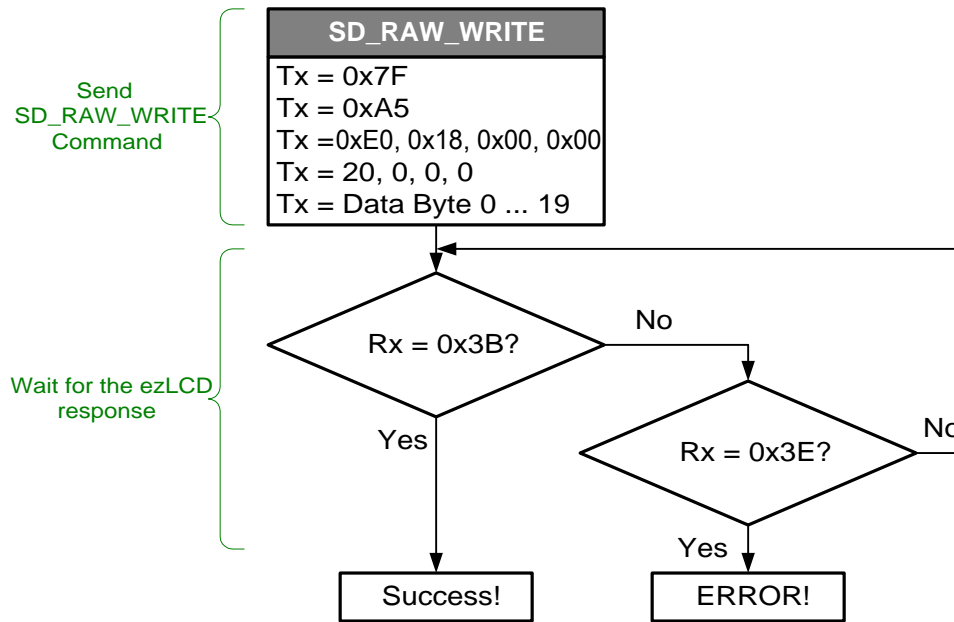
In case of an **error**:



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

Example:

The following flow chart shows an example of writing 20 bytes starting from the SD address 000018E0_{hex}.

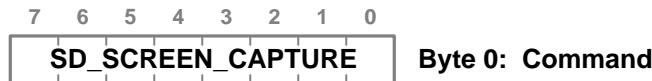


1.7.51 SD_SCREEN_CAPTURE

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}



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

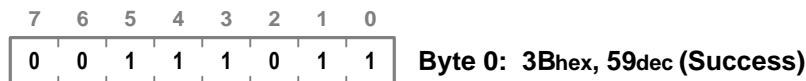
Screen capture files have names "Scr_xxxx.bmp", where xxxx is a consecutive number. For example: Scr_0001.bmp, Scr_0002.bmp, etc. The files are created in the "Scr_Cap" SD folder. If the SD does not have the "Scr_Cap" folder, it will be created automatically.

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

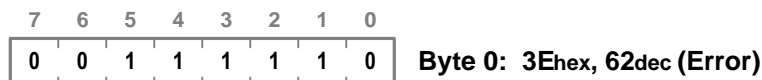
ezLCD Response

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

In case of the **success**:



In case of an **error**:



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

Example:

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

SD_SCREEN_CAPTURE 44 hex

If the screen image has been written to the .bmp file, the ezLCD responds with:

3B hex

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

3E hex

1.7.52 SD_SIZE

Description: Gets the physical size (in bytes) of the SD Card.

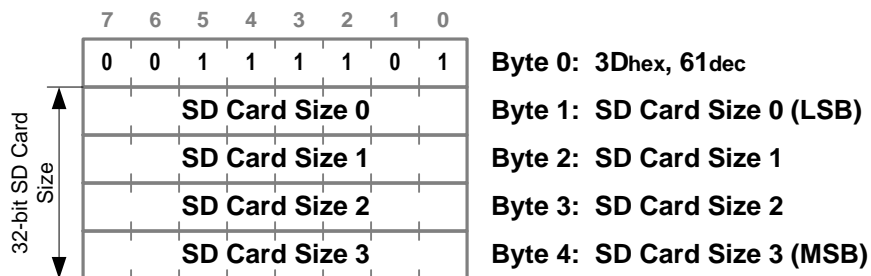
Code: 78_{hex}, 120_{dec}



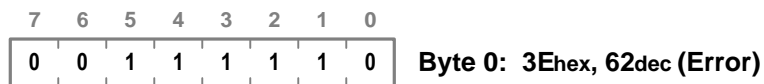
ezLCD Response

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

In case of the **success**:



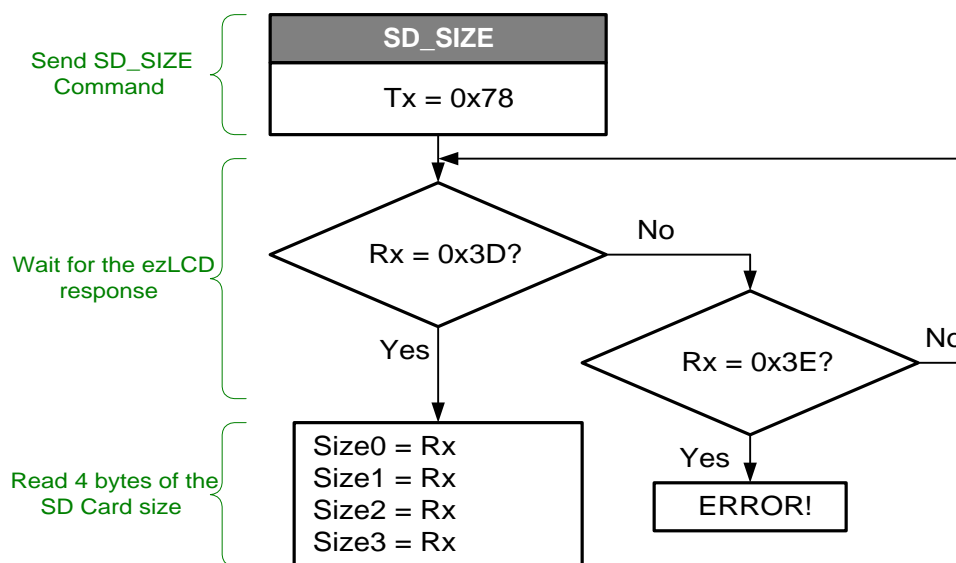
In case of an **error**:



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

Example:

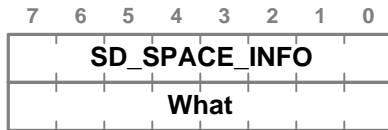
The following flow chart shows an example of getting the size of the SD Card.



1.7.53 SD_SPACE_INFO

Description: Gets the information about the space usage (in bytes) of the formatted SD Card.
Supported file systems: FAT12, FAT16, FAT32

Code: 48hex, 72dec



Byte 0: Command

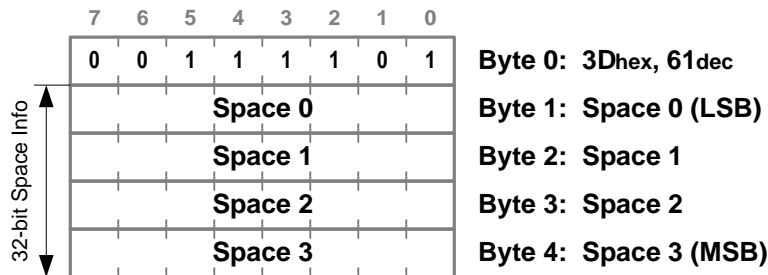
Byte 1: What 1 - Get Free Space
 2 - Get Used Space
 3 - Get Bad Space
 Any Other Number - Get Total Formatted Space

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

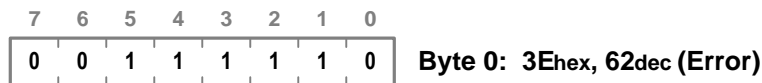
ezLCD Response

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

In case of the **success**:



In case of an **error**:

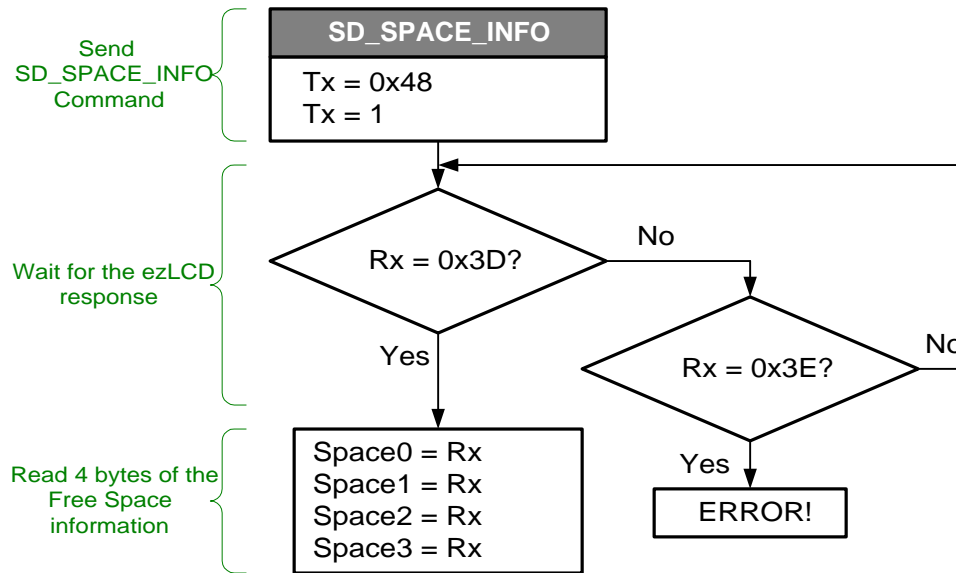


The ezLCD response is sent through the same interface, which received the SD_SPACE_INFO command.

SD_SPACE_INFO example is shown on the next page.

Example:

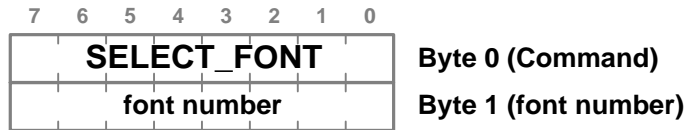
The following flow chart shows an example of getting the number of the available bytes on the formatted SD card.



1.7.54 SELECT_FONT

Description: Sets the Current Font.

Code: 2B_{hex}, 43_{dec}



Note: The following fonts are implemented

Font 0: ezLCD

Font 1: ezLCD

Font 2: ezLCD

Font 3: ezLCD

Font 4: ezLCD

Font 5: ezLCD

See Also: [PRINT_STRING](#), [PRINT_CHAR](#)

Example:

The following sequence will print a black character 'M' in the middle of the screen using Font 2.

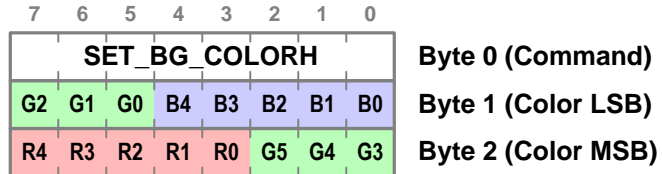
```
SELECT_FONT  2B hex
2            2 dec
SET_COLORH  84 hex
BLACK_LSB   0000000 bin
BLACK_MSB   0000000 bin
PRINT_CHAR  2C hex
'M'         4D hex
```

1.7.55 SET_BG_COLORH

Description: Sets the Background Color for the following instructions:

[PRINT_CHAR_BG](#)
[PRINT_STRING_BG](#)

Code: **94**_{hex}, **148**_{dec}



See Also: [PRINT_CHAR_BG](#), [PRINT_STRING_BG](#)

Example:

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

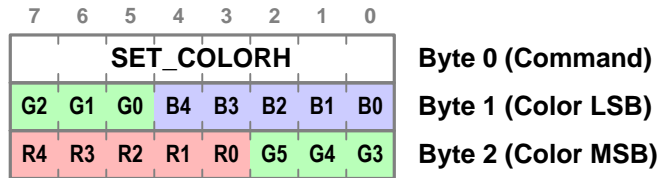
```

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
  
```

1.7.56 SET_COLORH

Description: Sets the Current Color.

Code: 84_{hex}, 132_{dec}



See Also: [CLS](#), [PLOT](#)

Example:

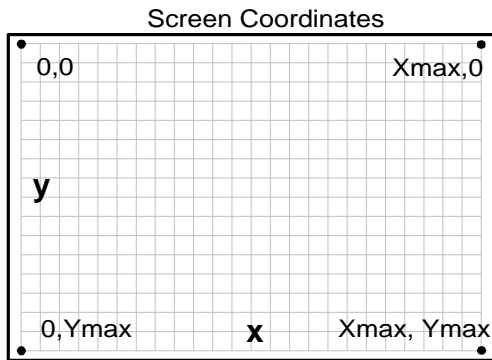
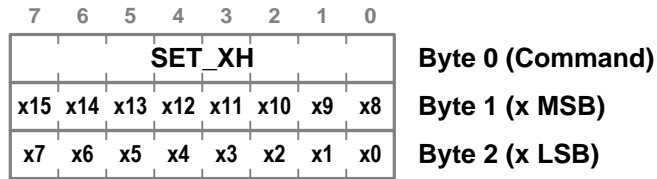
The following sequence will fill the whole screen with green.

```
SET_COLORH 84 hex
GREEN_LSB 11100000 bin
GREEN_MSB 0000111 bin
CLS 21 hex
```

1.7.57 SET_XH

Description: Sets only the X-coordinate of the Current Position. Y coordinate remains unchanged

Code: 6E_{hex}, 110_{dec}



See Also: [SET_Y](#), [SET_XHY](#)

Example:

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

```

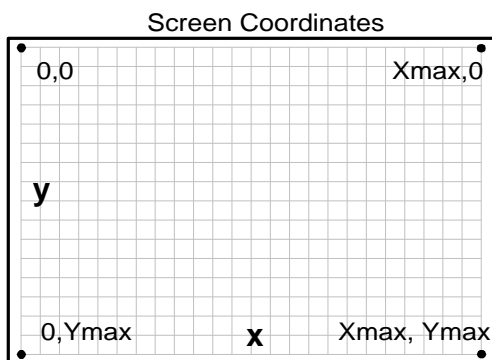
SET_COLORH 84 hex
BLUE_LSB   00011111 bin
BLUE_MSB   00000000 bin
SET_XH     6E hex
  0         0 dec (x MSB)
160        160 dec (x LSB)
PLOT       26 hex
SET_XH     6E hex
  0         0 dec (x MSB)
170        170 dec (x LSB)
PLOT       26 hex
  
```

1.7.58 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)



See Also: [PLOT](#), [LINE_TO_XHY](#), [CIRCLE_RH](#)

Example:

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

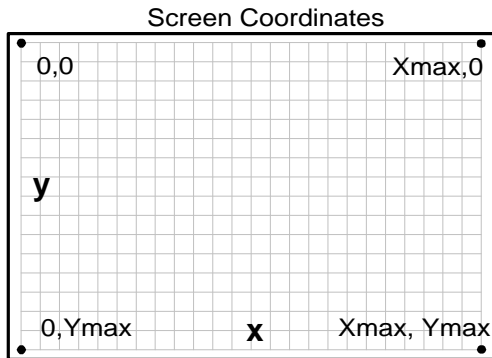
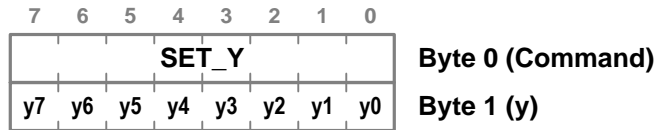
```

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
  
```

1.7.59 SET_Y

Description: Sets only the Y-coordinate of the Current Position. X coordinate remains unchanged

Code: **5F**_{hex}, **95**_{dec}



See Also: [SET_XH](#), [SET_XHY](#)

Example:

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

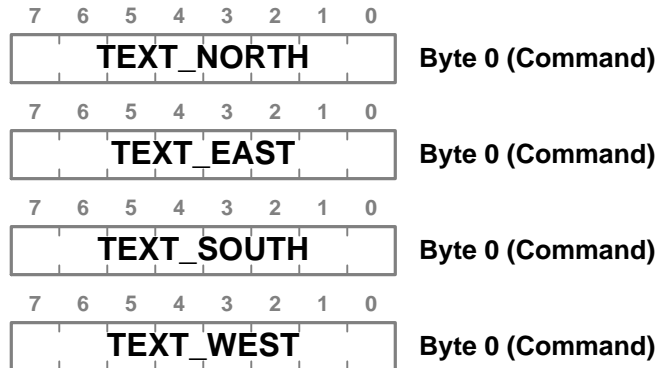
```

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
  
```

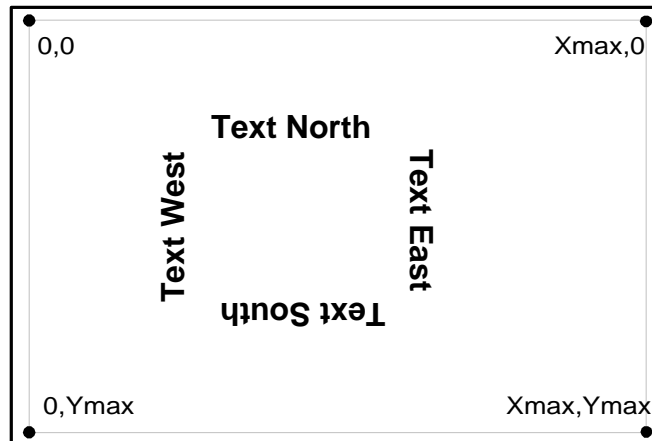
1.7.60 TEXT_EAST

Description: Sets the orientation of the text, as shown on the picture below.

Code:
TEXT_NORTH: 60hex, 96dec
TEXT_EAST : 61hex, 97dec
TEXT_SOUTH: 62hex, 98dec
TEXT_WEST : 2Fhex, 99dec



Note: TEXT_NORTH is the default text orientation



See Also: [PRINT_CHAR](#), [PRINT_STRING](#), [SELECT_FONT](#)

Example:

The following sequence will print a text pattern similar to the one pictured above.

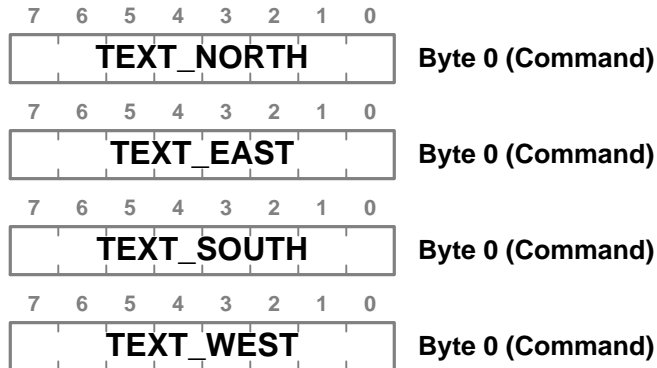
```
SET_XHY      85 hex
  0          0 dec (x MSB)
  60        60 dec (x LSB)
  10        10 dec (y)
SELECT_FONT  2B hex
  0          0 dec
TEXT_NORTH   60 hex
PRINT_STRING 2D hex
"Text North "
```

```
NULL          0 hex
TEXT_EAST    61 hex
PRINT_STRING  2D hex
" Text East  "
NULL          0 hex
TEXT_SOUTH    62 hex
PRINT_STRING  2D hex
" Text South "
NULL          0 hex
TEXT_WEST     63 hex
PRINT_STRING  2D hex
" Text West  "
NULL          0 hex
```

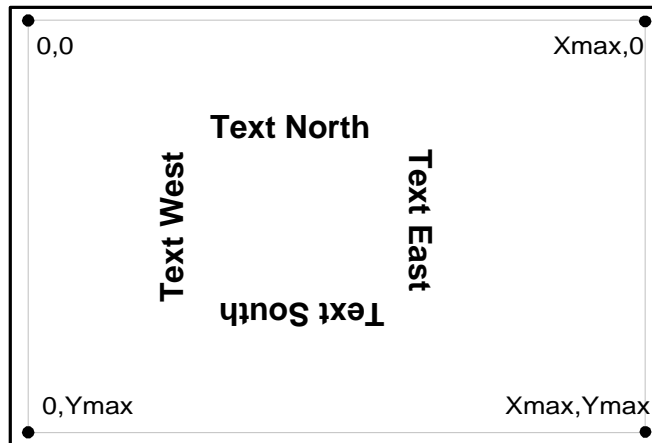
1.7.61 TEXT_NORTH

Description: Sets the orientation of the text, as shown on the picture below.

Code:
TEXT_NORTH: 60hex, 96dec
TEXT_EAST : 61hex, 97dec
TEXT_SOUTH: 62hex, 98dec
TEXT_WEST : 2Fhex, 99dec



Note: TEXT_NORTH is the default text orientation



See Also: [PRINT_CHAR](#), [PRINT_STRING](#), [SELECT_FONT](#)

Example:

The following sequence will print a text pattern similar to the one pictured above.

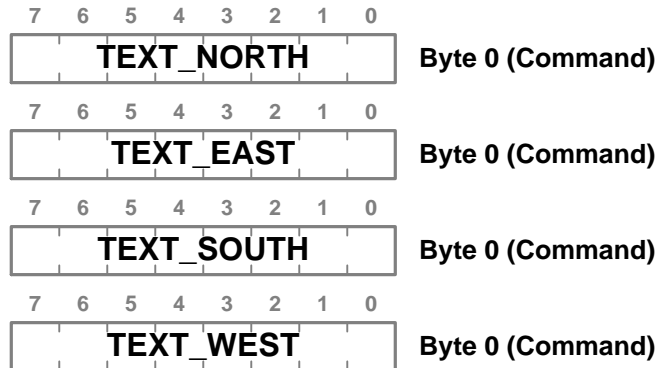
```
SET_XHY      85 hex
  0          0 dec (x MSB)
  60        60 dec (x LSB)
  10        10 dec (y)
SELECT_FONT  2B hex
  0          0 dec
TEXT_NORTH 60 hex
PRINT_STRING 2D hex
"Text North "
```

```
NULL          0 hex
TEXT_EAST     61 hex
PRINT_STRING  2D hex
" Text East  "
NULL          0 hex
TEXT_SOUTH    62 hex
PRINT_STRING  2D hex
" Text South "
NULL          0 hex
TEXT_WEST     63 hex
PRINT_STRING  2D hex
" Text West  "
NULL          0 hex
```

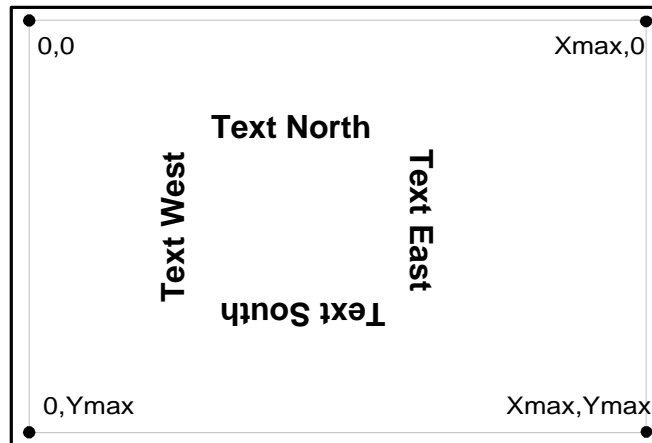
1.7.62 TEXT_SOUTH

Description: Sets the orientation of the text, as shown on the picture below.

Code:
TEXT_NORTH: 60hex, 96dec
TEXT_EAST: 61hex, 97dec
TEXT_SOUTH: 62hex, 98dec
TEXT_WEST: 2Fhex, 99dec



Note: TEXT_NORTH is the default text orientation



See Also: [PRINT_CHAR](#), [PRINT_STRING](#), [SELECT_FONT](#)

Example:

The following sequence will print a text pattern similar to the one pictured above.

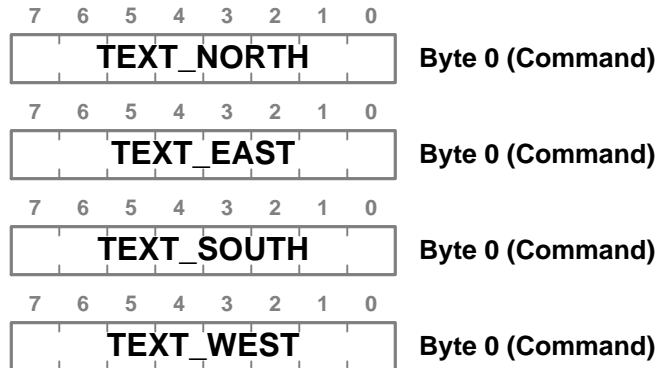
```
SET_XHY      85 hex
  0          0 dec (x MSB)
  60        60 dec (x LSB)
  10        10 dec (y)
SELECT_FONT  2B hex
  0          0 dec
TEXT_NORTH   60 hex
PRINT_STRING 2D hex
"Text North "
```

```
NULL          0 hex
TEXT_EAST     61 hex
PRINT_STRING  2D hex
" Text East  "
NULL          0 hex
TEXT_SOUTH   62 hex
PRINT_STRING  2D hex
" Text South "
NULL          0 hex
TEXT_WEST     63 hex
PRINT_STRING  2D hex
" Text West  "
NULL          0 hex
```

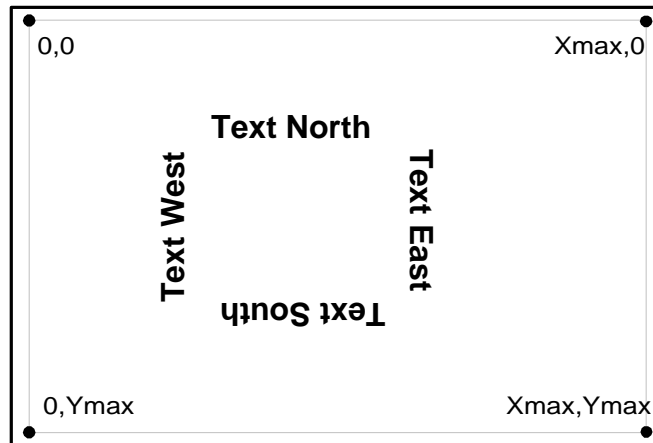
1.7.63 TEXT_WEST

Description: Sets the orientation of the text, as shown on the picture below.

Code:
TEXT_NORTH: 60hex, 96dec
TEXT_EAST : 61hex, 97dec
TEXT_SOUTH: 62hex, 98dec
TEXT_WEST : 2Fhex, 99dec



Note: TEXT_NORTH is the default text orientation



See Also: [PRINT_CHAR](#), [PRINT_STRING](#), [SELECT_FONT](#)

Example:

The following sequence will print a text pattern similar to the one pictured above.

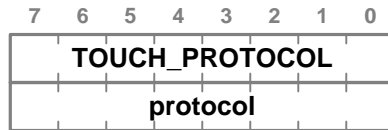
```
SET_XHY      85 hex
  0          0 dec (x MSB)
  60        60 dec (x LSB)
  10        10 dec (y)
SELECT_FONT  2B hex
  0          0 dec
TEXT_NORTH  60 hex
PRINT_STRING 2D hex
"Text North "
```

```
NULL          0 hex
TEXT_EAST     61 hex
PRINT_STRING  2D hex
" Text East  "
NULL          0 hex
TEXT_SOUTH    62 hex
PRINT_STRING  2D hex
" Text South "
NULL          0 hex
TEXT_WEST    63 hex
PRINT_STRING  2D hex
" Text West  "
NULL          0 hex
```

1.7.64 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](#) command.
 - ezLCD sends Button Down and Button Up events for the buttons defined by the [BUTTON_DEF](#) 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 by the TOUCH_PROTOCOL command.

See Also: [BUTTON_DEF](#), [BUTTON_STATE](#), [BUTTONS_ALL_UP](#), [BUTTONS_DELETE_ALL](#)

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

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

Example:

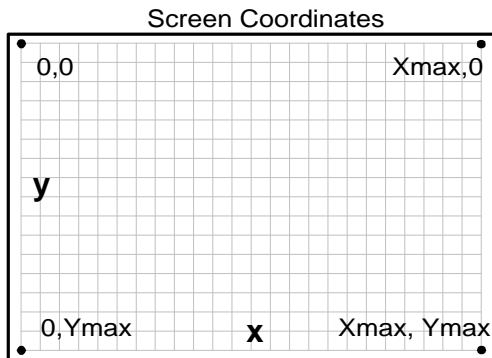
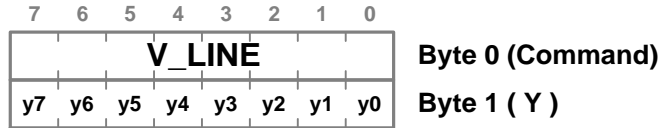
The following sequence will change the Touch Protocol to ezButton.

```
TOUCH_PROTOCOL B2 hex (Command)
                1          1 dec (ezButton Protocol)
```

1.7.65 V_LINE

Description: Quickly draws a vertical line from Current Position, to the row specified by the parameter.

Code: 41hex, 65dec



See Also: [H_LINEH](#), [SET_XHY](#)

Example:

The following sequence will draw a blue vertical line from (95, 10) to (95, 110).

```

SET_COLORH  84 hex
BLUE_LSB    00011111 bin
BLUE_MSB    00000000 bin
SET_XHY     85 hex
  0          0 dec (x MSB)
  95        95 dec (x LSB)
  10        10 dec (y)
V_LINE      41 hex
110        110 dec

```

1.7.66 Legacy Commands

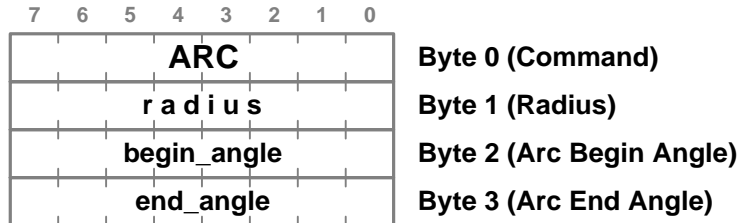
The ezLCD-004 can interpret the commands used by the smaller ezLCD displays. Those displays (ezLCD-001 and ezLCD-002), have maximum resolution of 240x160 and are capable of displaying maximum 256 colors.

ARC
BOX
BOX_FILL
CIRCLE_R
CIRCLE_R_FILL
LINE_TO_XY
PLOT_XY
PUT_BITMAP
SET_BG_COLOR
SET_COLOR
SET_X
SET_XY

1.7.66.1 ARC

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

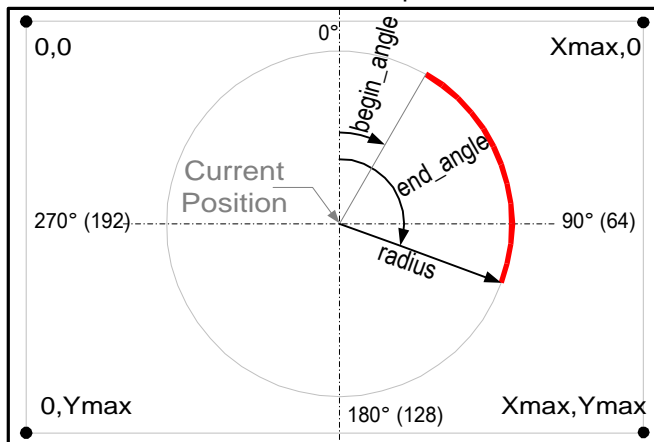
Code: **2F**_{hex}, **47**_{dec}



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:
 $Angle_lcd = Angle_deg \times 32 / 45$
 For example:
 32 = 45°
 64 = 90°
 128 = 180°
 192 = 270°
 0 = 0° = 360°

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

**Example:**

The following sequence will draw a green arc from 45 to 225 degrees with the center positioned in the middle of a screen.

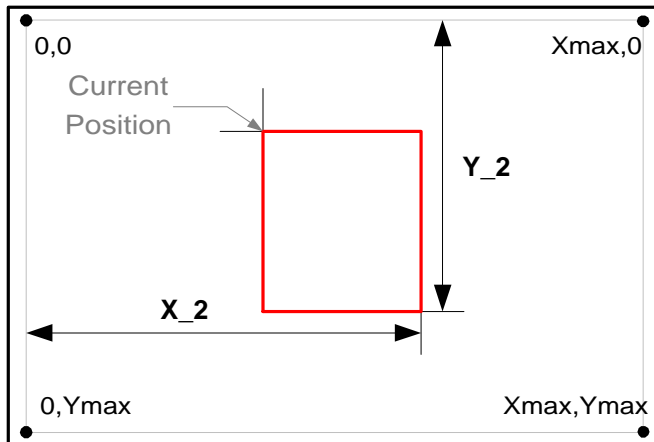
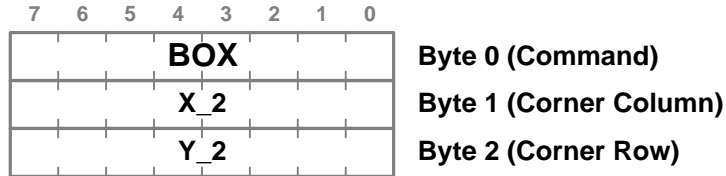
```
SET_COLOR      24 hex
GREEN         00111000 bin
SET_XY       25 hex
120          120 dec
80           80 dec
ARC          2F hex
60           60 dec (radius)
```

```
32          32 dec (begin_angle = 45 degrees)
160        160 dec (end_angle = 225 degrees)
\
```

1.7.66.2 BOX

Description: Draws a rectangle.

Code: 42_{hex}, 66_{dec}



See Also: [SET_XY](#), [BOX_FILL](#)

Example:

The following sequence will draw the red rectangle

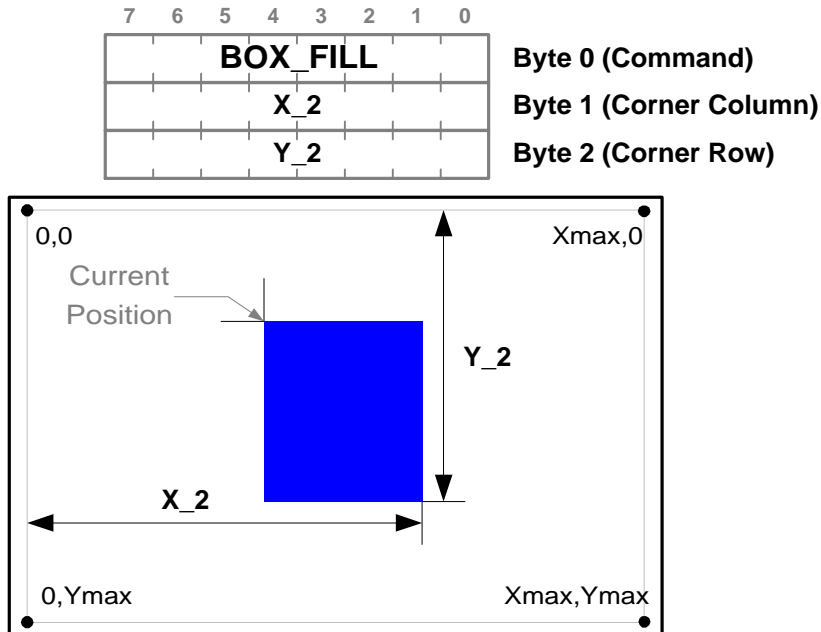
```

SET_COLOR  24 hex
RED        0000111 bin
SET_XY     25 hex
           95      95 dec
           40      10 dec
BOX        42 hex
180        180 dec (X_2)
120        120 dec (Y_2)
  
```

1.7.66.3 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

```

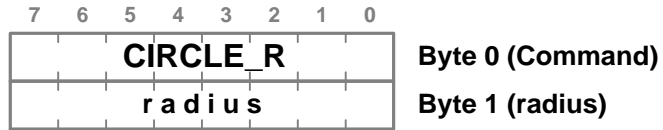
SET_COLOR  24 hex
RED        11000000 bin
SET_XY     25 hex
           95      95 dec
           40      10 dec
BOX_FILL   43 hex
180        180 dec (X_2)
120        120 dec (Y_2)

```

1.7.66.4 CIRCLE_R

Description: Draws a circle in Current Color at Current Position

Code: 29_{hex}, 41_{dec}



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
```

1.7.66.5 CIRCLE_R_FILL

Description: Draws a circle in Current Color at Current Position, filled with Current Color
Code: 39_{hex}, 57_{dec}



See Also: [SET_XY](#), [SET_COLOR](#)

Example:

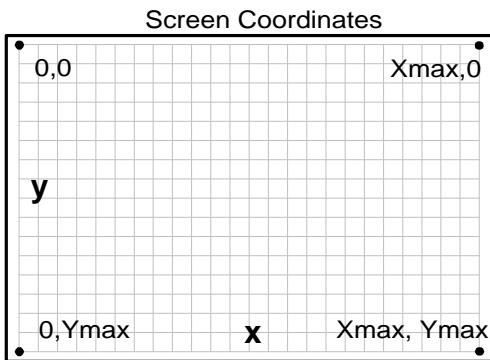
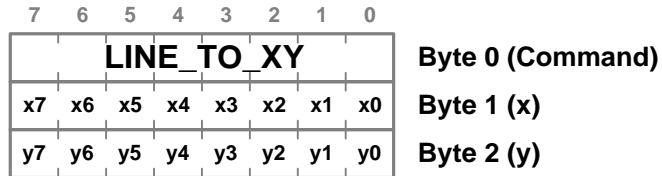
The following sequence will draw a red filled circle in the middle of the screen.

```
\SET_COLOR      24 hex
RED             0000111 bin
SET_XY         25 hex
120            120 dec
80             80 dec
CIRCLE_R_FILL  39 hex
60            60 dec
```

1.7.66.6 LINE_TO_XY

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

Code: 28_{hex}, 40_{dec}



See Also: [SET_XY](#), [SET_COLOR](#), [PLOT](#)

Example:

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

```

SET_COLOR      24 hex
RED            0000111 bin
SET_XY        25 hex
0              0 dec
0              0 dec
LINE_TO_XY    28 hex
239           239 dec
159           159 dec

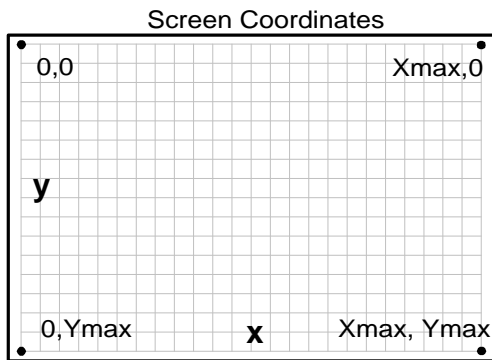
```

1.7.66.7 PLOT_XY

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

Code: 27_{hex}, 39_{dec}

7	6	5	4	3	2	1	0	
PLOT_XY								Byte 0 (Command)
x7	x6	x5	x4	x3	x2	x1	x0	Byte 1 (x)
y7	y6	y5	y4	y3	y2	y1	y0	Byte 2 (y)



See Also: [SET_XY](#), [SET_COLOR](#), [PLOT](#)

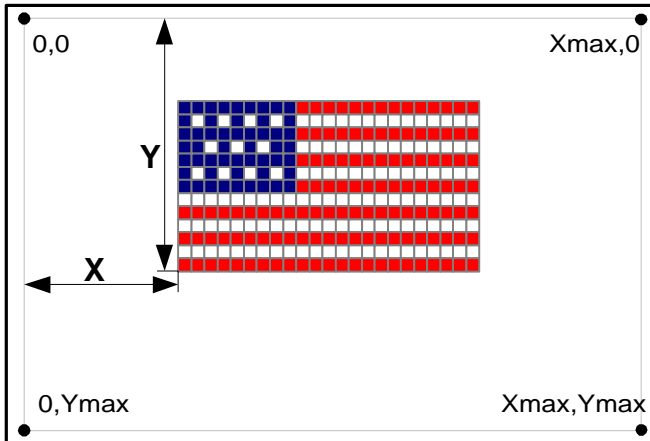
Example:

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

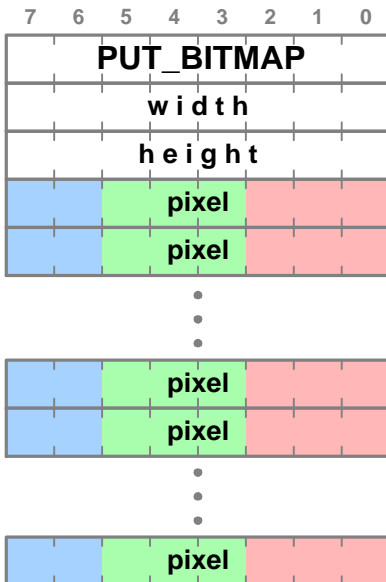
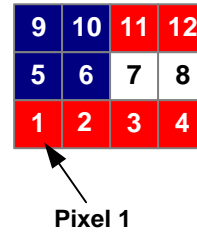
```
SET_COLOR 24 hex
RED       00000111 bin
PLOT_XY  27 hex
120      120 dec
80       80 dec
```

1.7.66.8 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
pixel (x = 60, y = 80)
```

```
pixel (x = 61, y = 80)
pixel (x = 62, y = 80)
pixel (x = 63, y = 80)
pixel (x = 60, y = 79)
pixel (x = 61, y = 79)
pixel (x = 62, y = 79)
pixel (x = 63, y = 79)
pixel (x = 60, y = 78)
pixel (x = 61, y = 78)
pixel (x = 62, y = 78)
pixel (x = 63, y = 78)
```

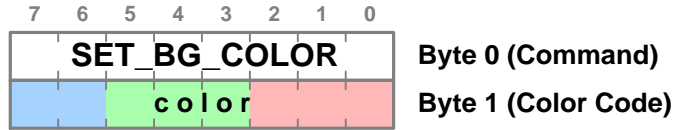
TOTAL:
4 x 3 + 3 = 15 bytes

1.7.66.9 SET_BG_COLOR

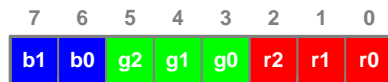
Description: Sets the Background Color for the following instructions:

[PRINT_CHAR_BG](#)
[PRINT_STRING_BG](#)

Code: 34_{hex}, 52_{dec}



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.

```

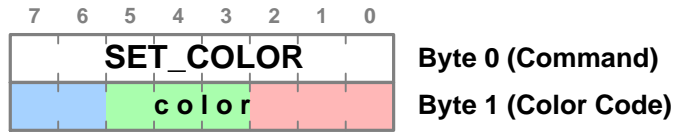
SET_BG_COLOR      34 hex
NAVY              10000000 bin
SET_COLOR        24 hex
YELLOW          00111111 bin
SET_XY          25 hex
120             120 dec
80              80 dec
SELECT_FONT     2B hex
0               0 dec
PRINT_STRING_BG 3D hex
'L'            4C hex
'C'            43 hex
'D'            44 hex
NULL           0 hex

```

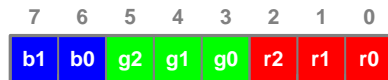
1.7.66.10 SET_COLOR

Description: Sets the Current Color

Code: 24_{hex}, 36_{dec}



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



See Also: [CLS](#), [PLOT](#)

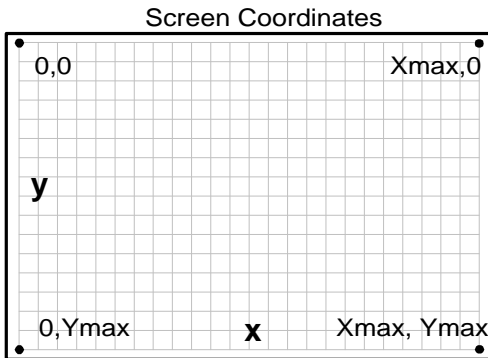
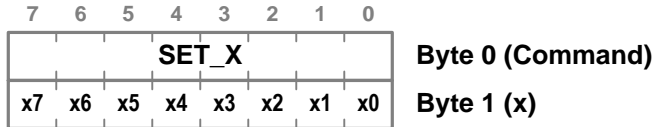
Example:

The following sequence will fill the whole display with green

```
SET_COLOR 24 hex
GREEN 00111000 bin
CLS 21 hex
```

1.7.66.11 SET_X

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



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
```


Index

- + -

+3.3V 5
+5V 5
+5V ext 5

- 8 -

8 bit parallel 1

- A -

ARC 32, 122
ARCH 32
ATmega128 1

- B -

BOX 34, 124
BOX_FILL 35, 125
BOXH 34
BOXH_FILL 35
BUTTON_DEF 36
BUTTON_STATE 38
BUTTONS_ALL_UP 39
BUTTONS_DELETE_ALL 40

- C -

CalibratedXY 21
calibration 15
CAN 5
CAN RX 5
CAN TX 5
CIRCLE_R 41, 126
CIRCLE_R_FILL 42, 127
CIRCLE_RH 41
CIRCLE_RH_FILL 42
CLS 43
CN1 46
Command Buffer 2
Command Interpreter 2

commands 1
Connectors 5
cuButton 19

- D -

Data Protocols 16
DM 5
DP 5

- E -

ezButton 17
ezLCD-001 1
ezNow 44, 46
EZNOW_BUZZER_BEEP 44
EZNOW_BUZZER_OFF 45
EZNOW_BUZZER_ON 46

- F -

Firmware Upgrade 11

- G -

GND 5
GND Pwr 5
GND sig 5

- H -

H_LINE 47
H_LINEH 47

- I -

I2C 1, 5
Interface 5

- L -

LCD Controller 1, 2
LIGHT_BRIGHT 48
LIGHT_OFF 49
LIGHT_ON 50

LINE_TO_XHY 51
LINE_TO_XY 51, 128

- M -

MCCDA 5
MCKK 5
MCDA0 5
MCDA1 5
MCDA2 5
MCDA3 5
MISO 5
MMC/SD slot 10
MOSI 5

- O -

ON/OFF 5

- P -

Packets 22
PIEH 52
Pin Configuration 5
PLOT 55
PLOT_XHY 56
PLOT_XY 56, 129
PRINT_CHAR 57
PRINT_CHAR_BG 58
PRINT_STRING 59
PRINT_STRING_BG 60
PROG 5
PROG# 5
PUT_BITMAP 61, 130
PUT_BITMAPH 61
PUT_SF_ICON 62

- R -

RS232 1, 5
RS232 TTL RX 5
RS232 TTL TX 5
RX 5

- S -

SCL 5
SD/MMC 5, 10
SDA 5
SED1375 1
SELECT_FONT 105
SET_BG_COLOR 106, 132
SET_BG_COLORH 106
SET_COLOR 107, 133
SET_COLORH 107
SET_XHY 109
SET_XY 109, 135
SPCK 5
SPI 1, 5
SS 5
SS# 5

- T -

TEXT 111
TEXT_EAST 111
TEXT_NORTH 111
TEXT_SOUTH 111
TEXT_WEST 111
touch screen 14
TOUCH_PROTOCOL 119
TX 5

- U -

USB 1, 5

- V -

V_LINE 120
VBUS 5
voltage 5