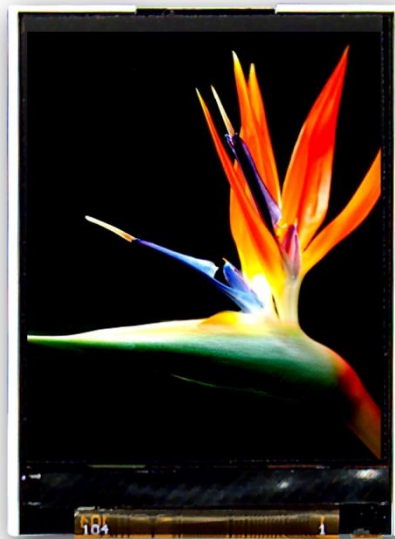




# EZL-176



## Datasheet (release 1.1)

## Table of Contents

<b>1</b>	<b>Description</b> .....	<b>4</b>
1.1	Features.....	4
<b>2</b>	<b>EZL-176 Hardware</b> .....	<b>5</b>
2.1	Pin Configuration and Summary.....	6
2.2	Serial Interface (UART) .....	7
2.3	Button-Switch Interface.....	7
2.4	Sound.....	7
2.5	The SD Memory Card.....	8
2.6	System Pins.....	8
2.7	The 1.76" TFT display.....	8
2.8	The GOLDELOX-SGC processor.....	8
<b>3</b>	<b>EZL-176 Software Interface</b> .....	<b>9</b>
3.1	Serial Set-up: Auto-Baud .....	9
3.2	Power-Up and Reset .....	9
3.3	Splash Screen on Power-Up .....	9
3.4	Memory Card Script Program .....	9
<b>4</b>	<b>Command Set</b> .....	<b>10</b>
4.1	General Commands.....	11
4.1.1	<i>AutoBaud (55)</i> .....	12
4.1.2	<i>Version-Device Info Request (56)</i> .....	12
4.1.3	<i>Set Background Color (42)</i> .....	12
4.1.4	<i>Clear Screen (45)</i> .....	12
4.1.5	<i>Switch-Buttons-Joystick Status (4A)</i> .....	13
4.1.6	<i>Wait for Switch-Buttons-Joystick Status (6A)</i> .....	13
4.1.7	<i>Sound (4E)</i> .....	13
4.2	Graphics Commands.....	14
4.2.1	<i>Add User Bitmap Character - 41hex</i> .....	15
4.2.2	<i>Draw Circle (43)</i> .....	15
4.2.3	<i>Draw User Bitmap Character (44)</i> .....	16
4.2.4	<i>Draw Triangle (47)</i> .....	16
4.2.5	<i>Draw Image-Icon (49)</i> .....	17
4.2.6	<i>Draw Line (4C)</i> .....	17
4.2.7	<i>Draw Pixel (50)</i> .....	18
4.2.8	<i>Read Pixel (52)</i> .....	18
4.2.9	<i>Screen Copy-Paste (63)</i> .....	19
4.2.10	<i>Draw Polygon (67)</i> .....	19
4.2.11	<i>Set Pen Size (70)</i> .....	20
4.2.12	<i>Draw Rectangle (72)</i> .....	20
4.3	Text Commands.....	21
4.3.1	<i>Set Font (46)</i> .....	22
4.3.2	<i>Set Transparent-Opaque Text (4F)</i> .....	22
4.3.3	<i>Draw "String" of ASCII Text in graphics format (53)</i> .....	23
4.3.4	<i>Draw ASCII Character in text format (54)</i> .....	24
4.3.5	<i>Draw Text Button (62)</i> .....	25
4.3.6	<i>Draw "String" of ASCII Text in text format (73)</i> .....	26
4.3.7	<i>Draw ASCII Character in graphics format (74)</i> .....	27

4.4	SD Memory Card Commands.....	28
4.4.1	Set Address Pointer of Memory Card (40 41).....	29
4.4.2	Screen Copy – Save to Memory Card (40 43).....	29
4.4.3	Display Image-Icon from Memory Card (40 49).....	30
4.4.4	Display Object from Memory Card (40 4F).....	30
4.4.5	Run Script (4DSL) Program from Memory Card (40 50).....	31
4.4.6	Read Sector Block Data from Memory Card (40 52).....	31
4.4.7	Display Video-Animation Clip from Memory Card (40 56).....	32
4.4.8	Write Sector Block Data to Memory Card (40 57).....	32
4.4.9	Initialize Memory Card (40 69).....	33
4.4.10	Read Byte Data from Memory Card (40 72).....	33
4.4.11	Write Byte Data to Memory Card (40 77).....	33
4.5	Script Commands .....	34
4.5.1	Delay (07) .....	35
4.5.2	Set Counter (08) .....	35
4.5.3	Decrement Counter (09).....	36
4.5.4	Jump to Address If Counter Not Zero (0A) .....	36
4.5.5	Jump to Address (0B) .....	36
4.5.6	Exit-Terminate Script Program (0C).....	36
4.6	Summary List of Commands available for Scripting.....	37
<b>5</b>	<b>Online Resources .....</b>	<b>38</b>
<b>6</b>	<b>Dimensions .....</b>	<b>39</b>
<b>7</b>	<b>Specifications and Ratings .....</b>	<b>40</b>
<b>8</b>	<b>Proprietary Information .....</b>	<b>41</b>
<b>9</b>	<b>Disclaimer of Warranties &amp; Limitation of Liability .....</b>	<b>41</b>
<b>10</b>	<b>Contact Information .....</b>	<b>41</b>

# EZL-176

## Datasheet



## 1 Description

The **EZL-176** is a compact and cost effective all-in-one “SMART” serial display module using the latest state of the art TFT LCD technology with an embedded GOLDELOX-SGC graphics controller that delivers “stand-alone” functionality to any project.

Powerful graphics, text, image, animation and countless more features are built inside the module. It offers a simple yet effective serial interface to any host micro-controller that can communicate via a serial port. All screen related functions are sent using a simple protocol via the serial interface.

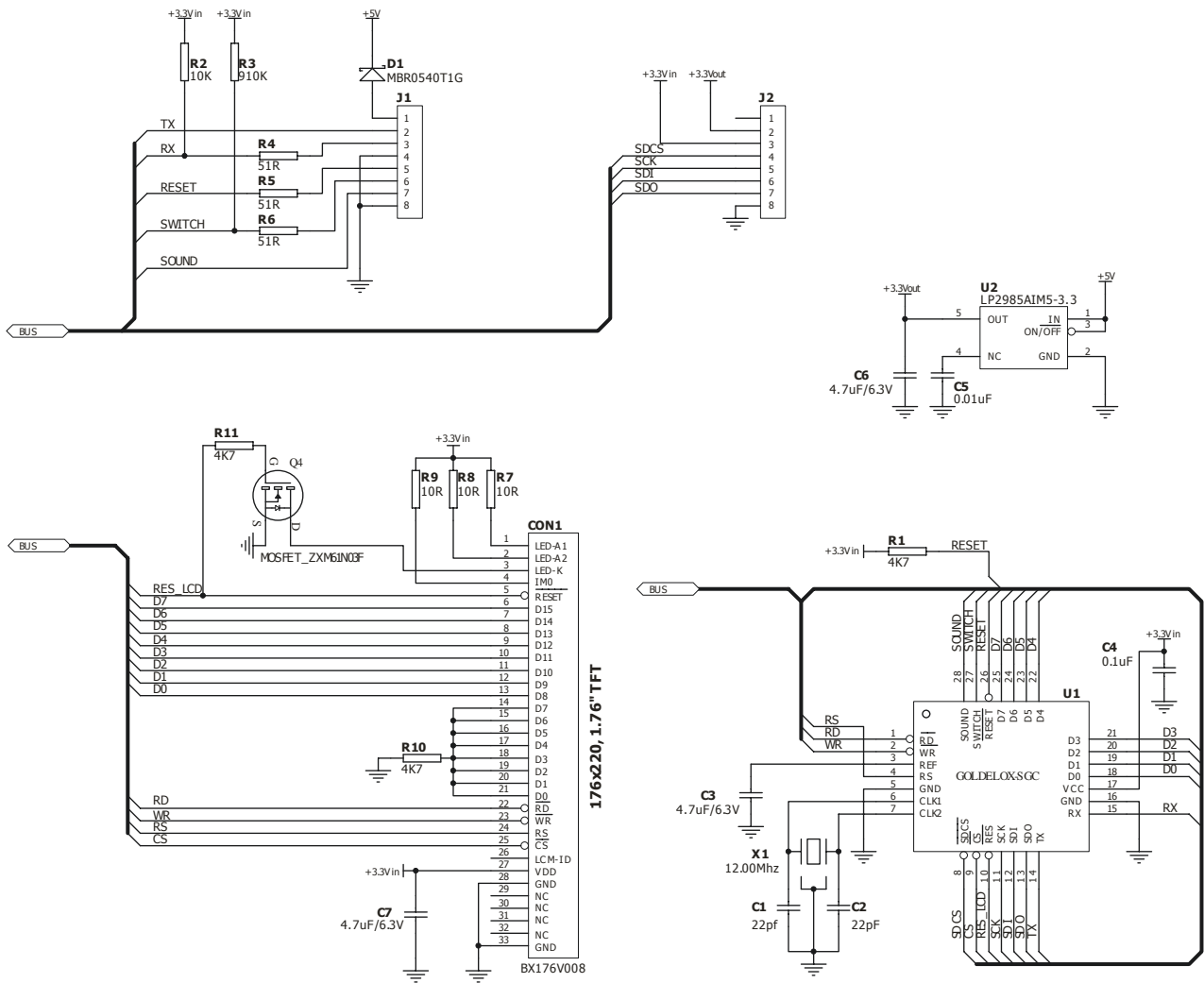
The serial platform allows users to develop their application using their favorite micro-controller and software development tools. In short, the EZL-176 offers one of the most flexible embedded graphics solutions available.

### 1.1 Features

- 1.76” diagonal size TFT display
- 176x220 pixels resolution
- 262K colors
- 28x35 mm active area.
- LED backlight.
- Easy 5 pin interface to any host device (VCC, TX, RX, GND, RESET).
- Serial TTL interface with auto-baud feature (300 to 256K baud).
- External SD/microSD card support (up to 2GB) for storing images, videos, scripts..
- 3.3V to 5.5V range operation (single supply).
- Powered by the 4D-Labs GOLDELOX-SGC processor.
- Comprehensive set of built-in high level graphics functions and algorithms that can draw lines, circles, text, show images and videos, and much more.
- Display full color images, animations, icons and video clips.
- Supports all available Windows fonts and characters (imported as external fonts).
- Multiple switch/button feature on a single pin.
- Dedicated sound pin with complex sound generation.
- RoHS Compliant.
- Size: 34x45x6 mm.

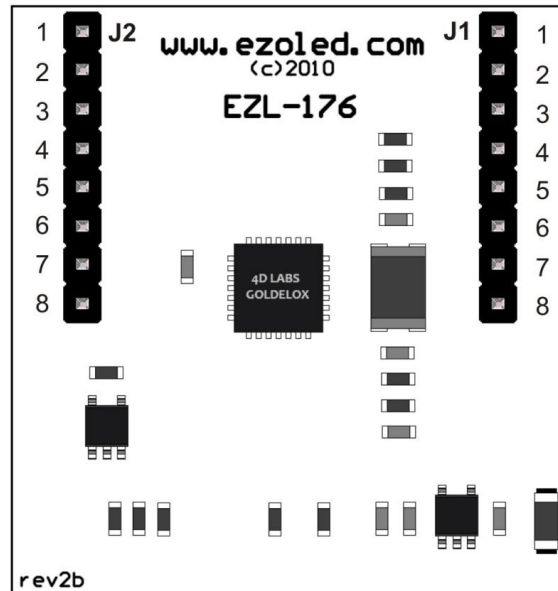
## 2 EZL-176 Hardware

The EZL-176 provides both hardware and software interface. This section describes in detail the hardware interface pins.



EZL-176 Schematics

## 2.1 Pin Configuration and Summary



Pin	Symbol	I/O	Description	
J1.1	VCC	I	5V Main Voltage Supply (reverse polarity protected), range is 4.0V to 5.5V, nominal 5.0V.	
J1.2	TX	O	Asynchronous Serial Transmit pin. Connect this pin to host micro-controller Serial Receive (Rx) signal. The host receives data from EZL-176 via this pin. This pin is tolerant up to 5.0V levels.	
J1.3	RX	I	Asynchronous Serial Receive pin. Connect this pin to host micro-controller Serial Transmit (TX) signal. The host transmits commands and data to the EZL-176 via this pin. This pin is tolerant up to 5.0V levels.	
J1.4	GND	P	Supply Ground.	
J1.5	RESET	I	Master Reset signal. Internally pulled up to 3.3V via a 4.7K resistor. An active Low pulse greater than 2 micro-seconds will reset the module. If the module needs to be reset externally, only use open collector type circuits. This pin is not driven low by any internal conditions. The host should control this pin via one of its port pins using an open collector/drain arrangement.	
J1.6	SWITCH	I	Multi Button or Joystick switch input pin. Option is available to connect from 1 up to 5 push buttons. <b>Note:</b> If connected to GND on power-up it will auto-run a script program from the memory card.	
J1.7	SOUND	P	Sound generation output pin. Connect this pin to a simple speaker circuit described in section 2.4. Leave open if unused.	
J1.8	GND	P	Ground.	
J2.1	-	-	Not connected.	
J2.2	3.3Vout	P	3.3V regulated output. From internal 5V/3.3V voltage regulator.	<b>Note:</b> if 5V Main Voltage is used then the two pins J2.2 and J2.3 must be closed and not connected to any external device but the external microSD card.
J2.3	3.3Vin	P	3.3V regulated input.	
J2.4	SDCS	O	External SD/microSD card interface (see section 2.5). The EZL-176 supports cards up to 2GB.	
J2.5	SCK	O		
J2.6	SDI	I		
J2.7	SDO	O		
J2.8	GND	P	Ground.	

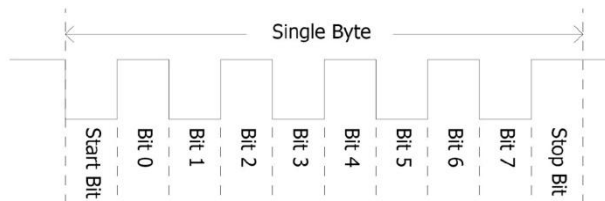
## 2.2 Serial Interface (UART)

The EZL-176 has a dedicated hardware UART that can communicate with a host micro-controller via its serial port. This is the main interface used by the host micro-controller to communicate with the EZL-176 module to send commands and receive back data.

The primary features are:

- Full-Duplex 8 bit data transmission and reception through the TX and RX pins.
- Data format: 8 bits, No Parity, 1 Stop bit.
- Auto Baud feature.
- Baud rates from 300 baud up to 256K baud.

A single byte serial transmission consists of the start bit, 8-bits of data followed by the stop bit. The start bit is always 0, while a stop bit is always 1. The LSB (Least Significant Bit, Bit 0) is sent out first following the start bit. Figure below shows a single byte transmission timing diagram.



The Serial port is also the primary interface for updating and programming the on board GOLDELOX-SGC processor with future serial command upgrades and enhancements. Please refer to **Section 6. Programming-System Updates** for more details.

**TX pin J1.2 (Serial Transmit):** Asynchronous Serial port Transmit pin, TX. Connect this pin to host micro-controller Serial Receive (Rx) signal. The host receives data from the EZL-176 module via this pin.

**RX pin J1.3 (Serial Receive):** asynchronous Serial port Receive pin, RX. Connect this pin to host micro-controller Serial Transmit (Tx) signal. The host transmits data to the EZL-176 module via this pin.

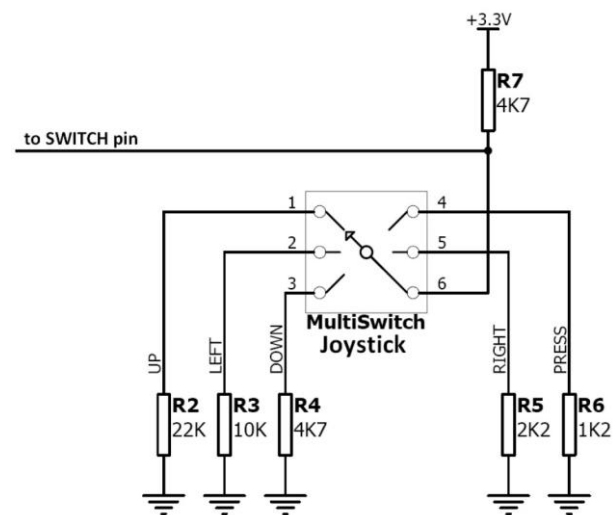
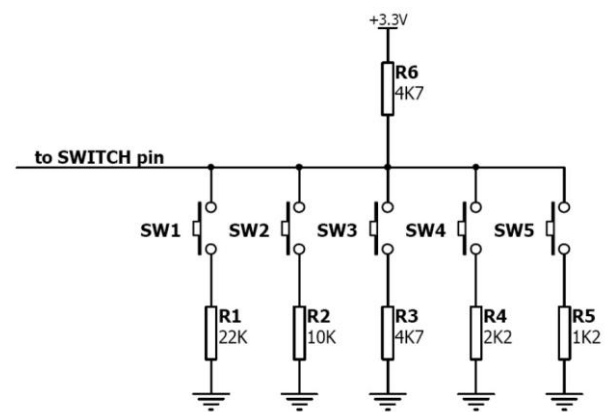
## 2.3 Button-Switch Interface

Multiple Buttons can be connected to a single pin on the EZL-176 module. Up to 5 buttons or a 5 position multi switch connects to a junction of a resistor ladder network that forms a voltage divider. The SWITCH pin internally reads the analogue value and decodes it accordingly.

**SWITCH pin J1.6 (Multiple Button Input):** connect up to 5 push buttons or a 5 position multi-switch as shown in the diagram below. Each consecutive button must be connected to ground via its matching resistor.

Unused buttons do not need resistors to be connected to the circuit. Table below lists the buttons and corresponding resistor values. If no buttons or switches are used then connect this pin to GND.

Number of Buttons	Button Number	Resistor Value
1	SW1	22K
2	SW2	10K
3	SW3	4.7K
4	SW4	2.2K
5	SW5	1.2K

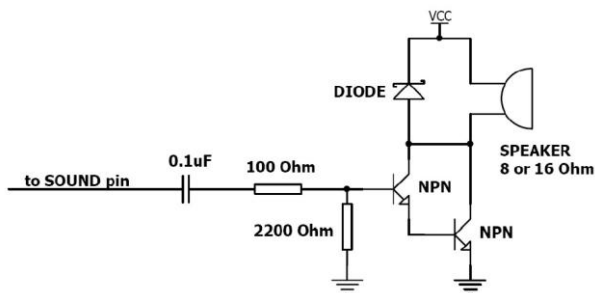


**Note:** If this pin is connected to GND on power-up, it will auto-run a script program from the memory card.

## 2.4 Sound

The EZL-176 module is capable of generating complex sounds and audio from its SOUND pin **J1.7**. A simple speaker circuit as shown below can be utilized. For a complete list of sound commands please refer to the section 4.

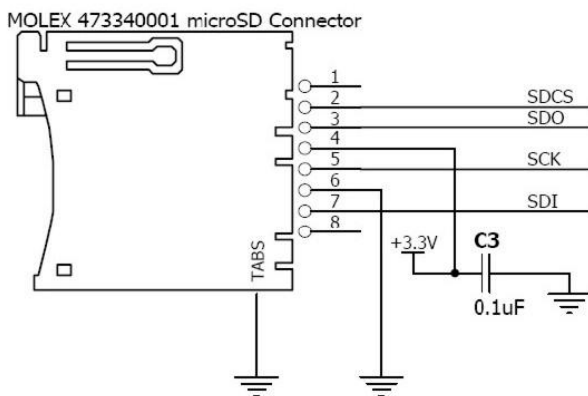
**SOUND pin J1.7** (Sound-Audio Output): sound and audio waveforms are generated from this pin. Connect this pin to a simple speaker circuit detailed above. If unused then this pin must be left open or floating.



## 2.5 The SD Memory Card

The module supports micro-SD memory cards via the off-board SD (or microSD) socket which connects on the pins: **SDCS**, **SDO**, **SCK**, **SDI**.

The memory card is used for all multimedia file retrieval such as images, animations and movie clips. The memory card can also be used as general purpose storage for data logging applications. Support is available for off the shelf SD/microSD with up to 2 GB capacity.



## 2.6 System Pins

**VCC pin 1** (Module Supply Voltage Input): module supply voltage input pin. This pin must be connected to a regulated supply voltage in the range of 4.0 Volts to 5.5 Volts DC. Nominal operating voltage is 5.0 Volts.

**3.3Vout pin 10** (3.3V Regulated Output): external circuitry that requires a regulated 3.3V supply can be powered up via this pin. Maximum available current is 50ma.

**GND pins 7, 8** (Module Ground): device ground pins. These pins must be connected to ground.

**RESET pin 9** (Module Master Reset): module Master Reset pin. An active low pulse of greater than 2 micro-seconds will reset the module (internally pulled up to 3.3V via 4.7K resistor). Only use open collector type circuits to reset the device

if an external reset is required.

## 2.7 The 1.76" TFT display

The EZL-176 is equipped with a full color TFT LCD screen. Some of the features of the screen are:

- Screen Size: 1.76" diagonal
- Screen Dimensions: 33.5 x 33.5mm.
- Viewing Area: 27 x 27mm
- 65K true to life colours
- Brightness: 100 cd/m2
- Contrast Ratio: 5000:1
- Viewing Angle: greater than 160 degrees
- No Back lighting

## 2.8 The GOLDELOX-SGC processor

The module is designed around the GOLDELOX-SGC Serial Graphics Controller from 4D Labs.



The **GOLDELOX-SGC** is an intelligent Serial Graphics Controller and provides plug-n-play interfacing with virtually any LCD or OLED display that supports an 80-series 8 bit wide CPU interface (max 256 x 256 pixels). All of the data and control signals are provided by the chip to interface directly to the display.

Powerful graphics, text, image, animation and countless more features are built right inside the chip. It offers a simple yet effective serial interface to any host micro-controller that can communicate via a serial port.

The embedded GOLDELOX-SGC processor on the EZL-176 module can be re-programmed with the latest firmware update, available as a "PmmC" (Personality-module-micro-Code) file.

The programming must be performed over the serial interface. All of the high level software interface commands are part of the PmmC configuration file so please check regularly for the latest updates and enhancements.

The PmmC file is programmed into the device with the aid of "**PmmC Loader**", a PC based software tool provided by 4D Systems.



### Download

GOLDELOX-SGC Datasheet:

<http://www.4dsystems.com.au/downloads/Semiconductors/GOLDELOX-SGC/Docs/>

### 3 EZL-176 Software Interface

The EZL-176 module is a slave device peripheral device and it provides a bidirectional serial interface to a host controller via its UART. All communications between the host and the device occur over this serial interface.

**Note:** Serial Data Format: **8 Bits, No Parity, 1 Stop Bit**. Serial data is true and not inverted.

The software interface provided by the EZL-176 module is a set of easy-to-use serial commands. Each command (explained in detail in section 4) is made up of a sequence of data bytes.

When a command is sent to the device and the operation is completed, it will always return a response. For a command that has no specific response the device will send back a single acknowledge byte called the **ACK (06hex)**, in the case of success, or **NAK (15hex)**, in the case of failure.

Commands having specific responses may send back varying numbers of bytes, depending upon the command and response. It will take the device a certain amount of time to respond, depending on the command type and the operation that has to be performed.

If the module receives a command that it does not understand it will reply back with a NAK. Since a command is only identified by its position in the sequence of data bytes sending incorrect data can result in wildly incorrect operation.

#### 3.1 Serial Set-up: Auto-Baud

The EZL-176 module has an auto-baud feature which can automatically detect the host speed and can set its internal baud rate to operate from 300 to 256K baud.

Prior to any commands being sent to the module, it must first be initialized by sending the auto-baud character **U (55hex)** after any power-up or reset.

This will allow the module to determine and lock on to the baud rate of the host automatically without needing any further set up. Once the device has locked onto the host baud rate it will respond with an ACK byte.

If the host needs to change the baud rate, the module must be power/reset cycled. The auto-baud command cannot be used to change the baud rate in the middle of normal usage.

**Note:** the auto-baud procedure must be performed each time the device is powered-up or reset.

#### 3.2 Power-Up and Reset

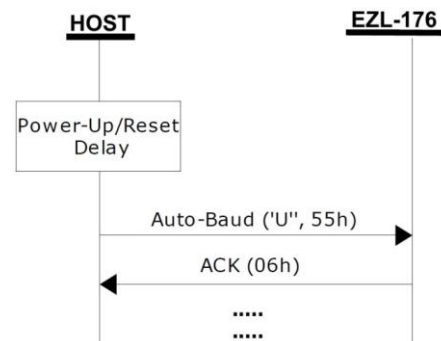
When the EZL-176 module comes out of a power-up or external reset, a sequence of events must be observed before attempting to communicate with the module:

1. Allow up to 500ms delay after power-up or reset for the module to settle. Do not attempt to communicate with the module during this period. The module may send garbage on its TX Data line during this period; the host should disable its RX data reception.

2. Within 100ms of powering up, the host should make sure it has its TX line pulled HIGH. If the host TX (EZL-176 RX) is LOW or floating after the 100ms period, the module may misinterpret this as the START bit of the auto-baud character and lock onto some undesired baud rate.

3. The host transmits the auto-baud character (U, 55hex) as the first command so the module can lock onto the host's baud rate.

Once the host receives the ACK, the EZL-176 module is ready to accept commands.



#### 3.3 Splash Screen on Power-Up

The EZL-176 will wait up to 5 seconds with its screen blank for the host to transmit the Auto-Baud command.

If the host has not transmitted the Auto-Baud command by the end of this period the module will display a built-in splash screen.

If the host has transmitted the Auto-Baud command, the screen will remain blank. This wait period is for those customer specific applications where the splash screen is undesired.

#### 3.4 Memory Card Script Program

The command execution is not only limited to the host sending commands via the serial interface. The majority of them can be composed as a script and written into the optional memory card (please see section 4.5). A script program is a sequence of those commands, and can be a combination of graphics, text, image, video and audio commands.

## 4 Command Set

The command interface between the EZL-176 module and the host is via the serial interface. A handful of easy-to-learn commands provide complete access to all the available functions.

The simplified command set also means that very low overheads are imposed on the host controller. Commands and responses can be either single bytes or many bytes.

All commands return a response, either an acknowledgement or data.

The command set is grouped into following sections:

- 4.1 General Commands
- 4.2 Graphics Commands
- 4.3 Text Commands
- 4.4 SD Memory Card Commands
- 4.5 Scripting Language Commands

**Note:** Separation characters such as commas “,” or spaces “ ” or brackets “()” between bytes that are shown in the command/response syntax descriptors are purely for legibility purposes and must not be considered as part of any transmitted/received data unless specifically stated.

**Unless specifically stated all the numeric values are in Hexadecimal format (hex).**

## 4.1 General Commands

Command	Code (HEX)
AutoBaud	55
Version-Device Info Request	56
Set Background Color	42
Clear Screen	45
Switch-Buttons-Joystick Status	4A
Switch-Buttons-Joystick Wait for Status	6A
Sound	4E

**4.1.1 AutoBaud (55)**

<b>Command</b>	<b>cmd</b>	
	cmd	<b>55</b> or <b>U</b> (ascii)
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful
<b>Description</b>	This must be the very first command sent to the module after power-up or reset. This will enable the device to lock on to the host baud rate.	

**4.1.2 Version-Device Info Request (56)**

<b>Command</b>	<b>Cmd</b>	
	Cmd	<b>56</b> or <b>V</b> (ascii)
<b>Response</b>	<b>device_type, silicon_rev, pmmc_rev, reserved1, reserved2</b>	
	device_type	This byte ( value <b>03</b> ) indicates the device type is GOLDELOX-SGC
	silicon_rev	This byte indicates the GOLDELOX silicon revision
	pmmc_rev	This byte indicates the firmware revision
	reserved1	This byte is reserved for future support. If the value is 0 then ignore it
	reserved2	This byte is reserved for future support. If the value is 0 then ignore it
<b>Description</b>	This command requests all the necessary information from the module about its characteristics and capability.	

**4.1.3 Set Background Color (42)**

<b>Command</b>	<b>cmd, color(msb:lsb)</b>	
	cmd	<b>42</b> or <b>B</b> (ascii)
	color	2 bytes (16 bits) define the background color in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0  Where: msb: R4R3R2R1R0G5G4G3 lsb: G2G1G0B4B3B2B1B0
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command sets the current background color. Once this command is sent, only the background color will change. Any other object on the screen with a different color value will not be affected.	
<b>Example</b>	<b>42, FF, FF</b> This example sets the background color value to FFFF (White).	

**4.1.4 Clear Screen (45)**

<b>Command</b>	<b>cmd</b>	
	cmd	<b>45</b> or <b>E</b> (ascii)
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command clears the entire screen using the current background color	

#### 4.1.5 Switch-Buttons-Joystick Status (4A)

<b>Command</b>	<b>cmd, option</b>	
	cmd	<b>4A</b> or <b>J</b> (ascii)
	option	<b>08</b> : Return Buttons-Joystick Status. <b>0F</b> : Wait for Buttons-Joystick to be pressed and released. <b>00</b> : Wait until any Buttons-Joystick pressed. <b>01</b> : Wait until SW1 (UP) released. <b>02</b> : Wait until SW2 (LEFT) released. <b>03</b> : Wait until SW3 (DOWN) released. <b>04</b> : Wait until SW4 (RIGHT) released. <b>05</b> : Wait until SW5 (FIRE) released.
<b>Response</b>	<b>status</b>	
	status	<b>00</b> : No Buttons pressed (or pressed button has been released). <b>01</b> : SW1 (UP) pressed. <b>02</b> : SW2 (LEFT) pressed. <b>03</b> : SW3 (DOWN) pressed. <b>04</b> : SW4 (RIGHT) pressed. <b>05</b> : SW5 (FIRE) pressed.
<b>Description</b>	This command returns the status of the Buttons-Joystick in several options.	

#### 4.1.6 Wait for Switch-Buttons-Joystick Status (6A)

<b>Command</b>	<b>cmd, option, waitTime(msb:lsb)</b>	
	cmd	<b>6A</b> or <b>j</b> (ascii)
	option	<b>00</b> : Wait until any Buttons-Joystick pressed. <b>01</b> : Wait until SW1 (UP) released. <b>02</b> : Wait until SW2 (LEFT) released. <b>03</b> : Wait until SW3 (DOWN) released. <b>04</b> : Wait until SW4 (RIGHT) released. <b>05</b> : Wait until SW5 (FIRE) released.
	waitTime	2 bytes (big endian) define the wait time (in milliseconds).
<b>Response</b>	<b>status</b>	
	status	<b>00</b> : Time-Out (or Button released). <b>01</b> : SW1 (UP) pressed. <b>02</b> : SW2 (LEFT) pressed. <b>03</b> : SW3 (DOWN) pressed. <b>04</b> : SW4 (RIGHT) pressed. <b>05</b> : SW5 (FIRE) pressed.
<b>Description</b>	This command asks for the status of the Buttons-Joystick in several options with a wait time.	

#### 4.1.7 Sound (4E)

<b>Command</b>	<b>cmd, note(msb:lsb), duration(msb:lsb)</b>	
	cmd	<b>4E</b> or <b>N</b> (ascii)
	note	2 bytes (big endian) define the note or frequency of the sound. <b>0</b> : No sound, silence. <b>1-84</b> : 5 octaves piano range + 2 more. <b>100-20000</b> : Frequency in Hz.
	duration	2 bytes (big endian) define the duration of the note (in milliseconds).
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will generate a specified note or frequency for certain duration.	

## 4.2 Graphics Commands

<b>Command</b>	<b>Code (HEX)</b>
Add User Bitmap Character	<b>41</b>
Draw Circle	<b>43</b>
Draw User Bitmap Character	<b>44</b>
Draw Triangle	<b>47</b>
Draw Image-Icon	<b>49</b>
Draw Line	<b>4C</b>
Draw Pixel	<b>50</b>
Read Pixel	<b>52</b>
Screen Copy-Paste	<b>63</b>
Draw Polygon	<b>67</b>
Set Pen Size	<b>70</b>
Draw Rectangle	<b>72</b>

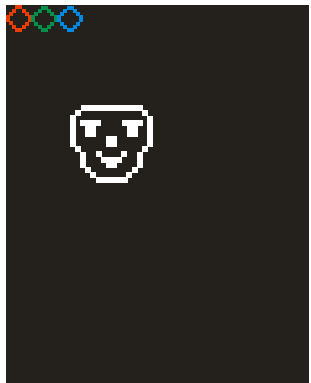
**4.2.1 Add User Bitmap Character - 41hex**

<b>Command</b>	<b>cmd, char_idx, data1, data2, data2, data3, data4, data5, data6, data7, data8</b>																																																																																										
	cmd	41(hex) or A(ascii) : Command header byte																																																																																									
	char_idx	Bitmap character index to add to memory. Range is 0 to 31 (00 to 1F), 32 characters of 8x8 format.																																																																																									
	data1..data8	8 data bytes that make up the composition of the bitmap character. The 8x8 bitmap composition is 1 byte wide (8 bits) by 8 bytes deep.																																																																																									
<b>Response</b>	<b>acknowledge</b>																																																																																										
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful																																																																																									
<b>Description</b>	This command will add a user defined bitmap character into the internal memory.																																																																																										
	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="8">Data Bits</th> <th></th> </tr> <tr> <th>b7</th> <th>b6</th> <th>b5</th> <th>b4</th> <th>b3</th> <th>b2</th> <th>b1</th> <th>b0</th> <th></th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td style="background-color: black;"></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td>data1 (18)</td> </tr> <tr> <td></td> <td></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td style="background-color: black;"></td> <td></td> <td>data2 (24)</td> </tr> <tr> <td></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td style="background-color: black;"></td> <td>data3 (42)</td> </tr> <tr> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>data4 (81)</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>data5 (81)</td> </tr> <tr> <td></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td style="background-color: black;"></td> <td>data6 (42)</td> </tr> <tr> <td></td> <td></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>data7 (24)</td> </tr> <tr> <td></td> <td></td> <td></td> <td style="background-color: black;"></td> <td style="background-color: black;"></td> <td></td> <td></td> <td></td> <td>data8 (18)</td> </tr> </tbody> </table> <p style="text-align: center;">Example of 8x8 User defined bitmap</p>		Data Bits									b7	b6	b5	b4	b3	b2	b1	b0										data1 (18)									data2 (24)									data3 (42)									data4 (81)									data5 (81)									data6 (42)									data7 (24)								
Data Bits																																																																																											
b7	b6	b5	b4	b3	b2	b1	b0																																																																																				
								data1 (18)																																																																																			
								data2 (24)																																																																																			
								data3 (42)																																																																																			
								data4 (81)																																																																																			
								data5 (81)																																																																																			
								data6 (42)																																																																																			
								data7 (24)																																																																																			
								data8 (18)																																																																																			
<b>Example</b>	<b>41, 01, 18, 24, 42, 81, 81, 42, 24, 18</b> This example adds and saves a user defined 8x8 bitmap as character index 1 into memory.																																																																																										

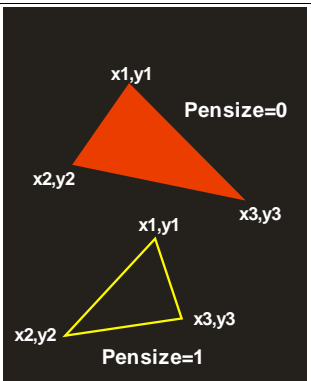
**4.2.2 Draw Circle (43)**

<b>Command</b>	<b>cmd, x, y, radius, color(msb:lsb)</b>	
	cmd	43(hex) or C(ascii) : Command header byte
	x	Horizontal position of the circle centre.
	y	Vertical position of the circle centre.
	radius	Radius of the circle.
	color	2 bytes define the circle color.
<b>Response</b>	<b>acknowledge</b>	
<b>Description</b>	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
	This command will draw a colored circle centered at (x, y) with a radius determined by the value set in the 'radius' byte. The circle can be either solid or wire frame (empty) depending on the value of the Pen Size (see Set Pen Size command): Pen Size=0: circle is solid. Pen Size=1: circle is wire frame.	
<b>Example</b>	<b>43, 3F, 3F, 22, 00, 1F</b> Draws a RED circle (001F) centered at x=3F (63dec) and y=3F (63dec) with a radius of 22 (34dec).	

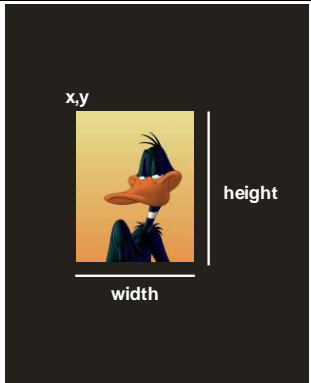
### 4.2.3 Draw User Bitmap Character (44)

<b>Command</b>	<b>cmd, char_idx, x, y, color(msb:lsb)</b>	
	cmd	<b>44</b> or <b>D</b> (ascii)
	char_idx	Bitmap character index to draw from the previously added bitmap characters into memory. Range is 0 to 31 (00 to 1F), 32 characters of 8x8 format.
	x	Horizontal display position of the bitmap character.
	y	Vertical display position of the bitmap character.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command draws the previously defined user bitmap character at location (x, y) on the screen. User defined bitmaps allow drawing and displaying unlimited graphic patterns quickly & effectively.	
<b>Examples</b>	<p><b>44, 01, 00, 00, F8, 00</b> Display 8x8 bitmap character index 1 at x=0, y=0, color=RED.</p> <p><b>44, 02, 08, 00, 07, E0</b> Display 8x8 bitmap character index 2 at x=8, y=0, color=GREEN.</p> <p><b>44, 03, 10, 08, 00, 1F</b> Display 8x8 bitmap character index 3 at x=16, y=8, color=BLUE.</p>	

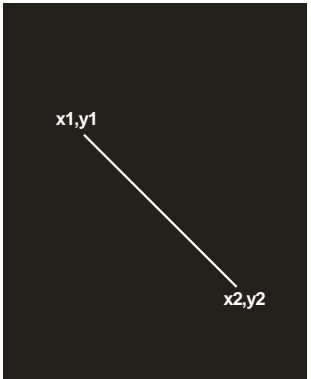
### 4.2.4 Draw Triangle (47)

<b>Command</b>	<b>cmd, x1, y1, x2, y2, x3, y3, color(msb:lsb)</b>	
	cmd	<b>47</b> or <b>G</b> (ascii)
	x1, y1, x2, y2, x3, y3	3 vertices of the triangle. These must be specified in an anti-clockwise fashion.
	color	2 bytes (big endian) triangle color value.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	<p>This command draws a Solid/Wire-Frame triangle. The vertices must be specified in an anti-clock wise manner, i.e. <math>x_2 &lt; x_1 : x_3 &gt; x_2 : y_2 &gt; y_1 : y_3 &gt; y_1</math></p> <p>A solid or a wire frame triangle is determined by the value of the Pen Size setting. Pen Size = 0: triangle is solid Pen Size = 1: triangle is wire frame</p>	

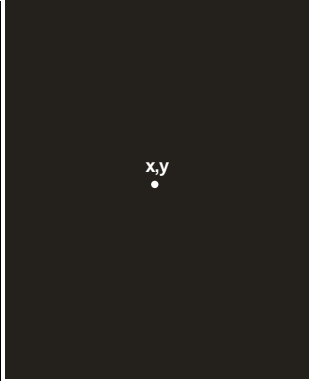
**4.2.5 Draw Image-Icon (49)**

<b>Command</b>	<b>cmd, x, y, width, height, colourMode, pixel1, .. pixelN</b>	
	cmd	<b>49</b> or <b>I</b> (ascii)
	x	Image horizontal start position (top left corner).
	y	Image vertical start position (top left corner).
	width	Horizontal size of the image.
	height	Vertical size of the image.
	colourMode	<b>08</b> : 256 color mode, 8bits/1byte per pixel. <b>10</b> : 65K color mode, 16bits/2bytes per pixel.
pixel1..pixelN	Image pixel data where N is the total number of pixels. N = width x height (when colourMode = 08). N = 2 x width x height (when colourMode = 10).	
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command displays a bitmap image on to the screen with the top left corner specified by (x, y) and the size of the image specified by width and height parameters.	
	This command is more effective than using the "Put Pixel" command, where there are no overheads in specifying the x, y location of each pixel.	
		

**4.2.6 Draw Line (4C)**

<b>Command</b>	<b>cmd, x1, y1, x2, y2, color(msb:lsb)</b>	
	cmd	<b>4C</b> or <b>L</b> (ascii)
	x1	Top left horizontal start position of line.
	y1	Top left vertical start position of line.
	x2	Bottom right horizontal end position of line.
	y2	Bottom right vertical end position of line.
	color	2 bytes define the Line color.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will draw a colored line from point (x1, y1) to point (x2, y2) on the screen.	
<b>Example</b>	<b>4C, 00, 00, 7F, 7F, FF, FF</b>	
	Draws a WHITE line (FFFF) from (x1=00, y1=00) to (x2=7F, y2=7F).	
		

**4.2.7 Draw Pixel (50)**

<b>Command</b>	<b>cmd, x, y, color(msb:lsb)</b>	
	cmd	<b>50</b> or <b>P</b> (ascii)
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
	color	2 bytes (16 bits) define the pixel color in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0  Where: msb: R4R3R2R1R0G5G4G3 lsb: G2G1G0B4B3B2B1B0
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will draw a colored pixel at location (x, y) on the screen.	
<b>Example</b>	<b>50, 01, 0A, FF, FF</b>	
	Draw a WHITE pixel (FFFF) at location (x=01, y=0A).	
		

**4.2.8 Read Pixel (52)**

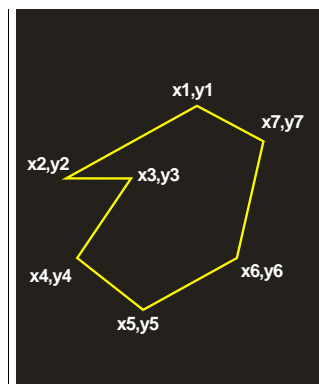
<b>Command</b>	<b>cmd, x, y</b>	
	cmd	<b>52</b> or <b>R</b> (ascii)
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
<b>Response</b>	<b>color(msb:lsb)</b>	
	color	Returns back 2 bytes (16 bits) pixel color in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0  Where: msb: R4R3R2R1R0G5G4G3 lsb: G2G1G0B4B3B2B1B0
<b>Description</b>	This command will read the color value of a pixel at location (x, y) on the screen and return it to the host. This is a useful command when for example a white pointer is moved across the screen and the host can read the color on the screen and switch the color of the pointer when it's on top of a light colored area.	
<b>Example</b>	<b>52, 01, 0A</b>	
	Module response: <b>00, 1F</b> Reads a BLUE pixel (001F) at location (x=01, y=0A).	

#### 4.2.9 Screen Copy-Paste (63)

<b>Command</b>	<b>cmd, xs, ys, xd, yd, width, height</b>	
	cmd	<b>63</b> or <b>c</b> (ascii)
	xs	Top left horizontal start position of screen area to be copied (source).
	ys	Top left vertical start position of screen area to be copied (source).
	xd	Top left horizontal start position of where copied area is to be pasted (destination).
	yd	Top left vertical start position of where copied area is to be pasted (destination).
	width	Width of screen area to be copied (source).
	height	Height of screen area to be copied (source).
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	<p>This command copies a specified area of the screen as a bitmap block. The start location of the block to be copied is represented by xs, ys (top left corner) and the size of the area to be copied is represented by width and height parameters. The start location of where the block is to be pasted (destination) is represented by xd, yd (top left corner).</p> <p>This is a very powerful feature for animating objects, smooth scrolling, implementing a windowing system or copying patterns across the screen to make borders or tiles.</p>	

#### 4.2.10 Draw Polygon (67)

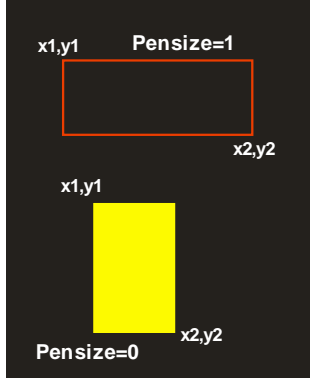
<b>Command</b>	<b>cmd, vertices, x1, y1, .. , xn, yn, color(msb:lsb)</b>	
	cmd	<b>67</b> or <b>g</b> (ascii)
	vertices	Number of vertices from 3 to 7. This byte specifies the number of vertices of the polygon.
	x1,y1,..xn, yn	Vertices of the triangle. These can be specified in any fashion.
	color	2 bytes triangle color value.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	<p>This command draws an Empty/Wire-Frame polygon. Up to 7 vertices can be specified in any manner.</p> <p><b>Note:</b> Only a wire frame polygon is supported.</p>	



**4.2.11 Set Pen Size (70)**

<b>Command</b>	<b>cmd, size</b>	
	cmd	<b>70</b> or <b>p</b> (ascii)
	size	Selects one of the 2 options: <b>00</b> : All graphics objects are drawn solid. <b>01</b> : All graphics objects are drawn wire-frame. <b>Note</b> : Does not apply to polygon command.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command determines if certain graphics objects are drawn in solid or wire frame fashion.	
<b>Examples</b>	<b>70, 00</b> All objects will be drawn solid.	
	<b>70, 01</b> All objects will be drawn wire-frame.	

**4.2.12 Draw Rectangle (72)**

<b>Command</b>	<b>cmd, x1, y1, x2, y2, color(msb:lsb)</b>	
	cmd	<b>72</b> or <b>r</b> (ascii)
	x1	Top left horizontal start position of rectangle.
	y1	Top left vertical start position of rectangle.
	x2	Bottom right horizontal end position of rectangle.
	y2	Bottom right vertical end position of rectangle.
	color	2 bytes define the rectangle color.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will draw a colored rectangle from point (x1, y1) to point (x2, y2) on the screen. If color is chosen to be that of the background then the effect will be erasure. If Pen Size value was previously set to 0, the rectangle will be solid; otherwise it will be wire-frame if value was 1.	
		

### 4.3 Text Commands

The EZL-176 module is shipped with 3 internal fonts. These fonts can be altered, deleted and replaced with new fonts, using the **FONT-Tool**, a free software tool that can assist in the conversion of any Windows fonts into the bitmap format that can be used by the module.

The converted font set can then be exported into the **DISP-Tool** utility which can then be downloaded into the module's on-chip (GOLDELOX-SGC) flash memory.

Command	Code (HEX)
Set Font	46
Set Transparent-Opaque Text	4F
Draw "String" of ASCII Text (graphics format)	53
Draw ASCII Character (text format)	54
Draw Text Button	62
Draw "String" of ASCII Text (text format)	73
Draw ASCII Character (graphics format)	74



#### Download

FONT-tool: <http://www.4dsystems.com.au/downloads/Software/Font-Tool/>

DISP-Tool: <http://www.4dsystems.com.au/downloads/Software/Disp-Tool/>

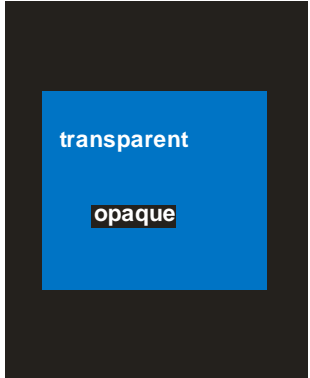
---

**4.3.1 Set Font (46)**

<b>Command</b>	<b>cmd, fontSet</b>	
	cmd	<b>46</b> or <b>F</b> (ascii)
<b>Response</b>	fontSet	Selects one of internal fonts. The supplied 3 fonts are: <b>00</b> : 5x7 small size font set <b>01</b> : 8x8 medium size font set <b>02</b> : 8x12 large size font set  These fonts can be altered and other fonts can be added.
	<b>acknowledge</b>	
<b>Description</b>	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
	<p>This command selects one of the available internal fonts. Changes take place after the command is sent. Any character on the screen with the previous font set will remain as it was.</p> <p><b>Note:</b> the module is shipped with three fonts displaying the characters 0x20 to 0x7F. The user can alter the number of fonts, delete existing fonts, and, or, add extra fonts, up to the amount of available user flash (a very limited resource). A font does not need to start at 0x20, or end at 0x7F. It could, for example start at 0x30 (“0”) and end at 0x39 (“9”).</p>	
<b>Examples</b>	<b>46, 00</b> Select small 5x7 font.	
	<b>46, 01</b> Select medium 8x8 font.  Command Data: <b>46, 02</b> Select large 8x12 font.	

**4.3.2 Set Transparent-Opaque Text (4F)**

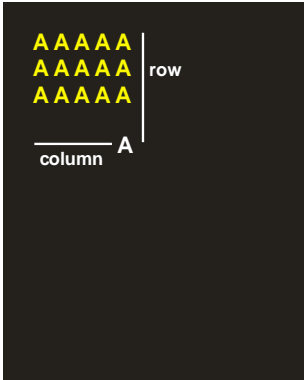
<b>Command</b>	<b>cmd, mode</b>	
	cmd	<b>4F</b> or <b>O</b> (ascii)
<b>Response</b>	mode	Select one of the following options for text appearance: <b>00</b> : Transparent, objects behind text are visible. <b>01</b> : Opaque, objects behind text blocked by background.
	<b>acknowledge</b>	
<b>Description</b>	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
	<p>This command will change the attribute of the text so that an object behind the text can either be blocked or transparent. Changes take place after the command is sent.</p>	
<b>Examples</b>	<b>4F, 00</b> Transparent text mode.	
	<b>4F, 01</b> Opaque text mode.	



### 4.3.3 Draw "String" of ASCII Text in graphics format (53)

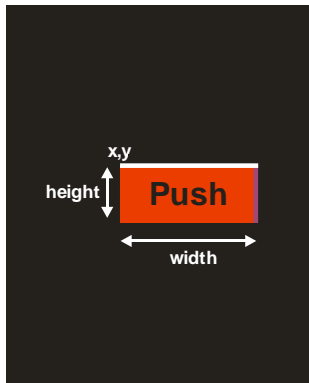
Command	cmd, x, y, font, stringColour(msb:lsb), width, height, "string", terminator	
	cmd	<b>53</b> or <b>S</b> (ascii)
	x	Top left horizontal start position of the string (pixel units).
	y	Top left vertical start position of the string (pixel units).
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: <b>00</b> : 5x7 internal font <b>01</b> : 8x8 internal font <b>02</b> : 8x12 internal font These fonts can be altered and other fonts can be added. ORing the fonts with 0x10 will cause the string to be displayed in a proportional manner (e.g. 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text color.
	width	This byte defines the width or horizontal size multiplier of the character in the string. Affects the total width of the string.
	height	This byte defines the height or vertical size multiplier of the character in the string. Affects the total height of the string.
	"string"	String of ASCII characters to be displayed (max. 256 characters).
	terminator	The string must be terminated with 00.
Response	acknowledge	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
Description	<p>This command will draw/display a string of ASCII text anywhere on the screen in pixel coordinates specified by x and y parameters.</p> <p>The horizontal start position of the string is specified by x and the vertical position is specified by y. The string must be terminated with 00.</p> <p>The size of the characters is determined by the width and height parameters. If the length of the string is longer than the maximum number of characters per line, a wrap around will occur on to the next line.</p> <p>Maximum string length is 256 bytes.</p>	

**4.3.4 Draw ASCII Character in text format (54)**

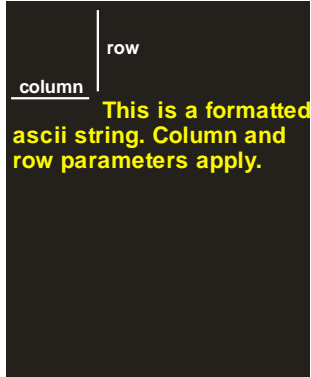
<b>Command</b>	<b>cmd, char, column, row, charColour(msb:lsb)</b>	
	cmd	<b>54</b> or <b>T</b> (ascii)
	char	Inbuilt standard ASCII character. Range: 32dec – 127dec (20hex - 7Fhex).
	column	Horizontal position of the character (character units). Range: 0 to <b>29</b> dec for 5x7 font. Range: 0 to <b>22</b> dec for 8x8 and 8x12 fonts.
	row	Vertical position of the character (character units). Range: 0 to <b>27</b> dec for 5x7 and 8x8 fonts. Range: 0 to <b>18</b> dec for 8x12 font.
	charColour	2 bytes define the character color.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will draw/display an ASCII character anywhere on the screen in character unit coordinates. The horizontal position of the character is specified by the column and the vertical position is specified by the row parameters.	
<b>Example</b>	<p><b>54, 41, 00, 00, FF, FF</b></p> <p>Draw/Display character "A" (41) at:                  Column=0                  Row=0                  Color=white (FFFF)</p>	

**4.3.5 Draw Text Button (62)**

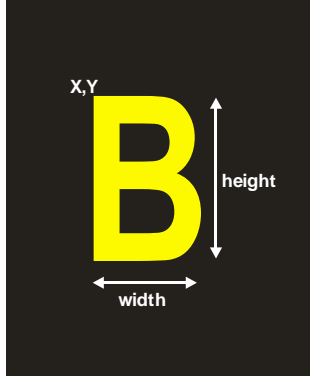
<b>Command</b>	<b>cmd, state, x, y, buttonColour(msb:lsb), font, stringColour(msb:lsb), width, height, "string", terminator</b>		
	cmd	<b>62</b> or <b>b</b> (ascii)	
	state	This byte specifies whether the displayed button is drawn UP (not pressed) or DOWN (pressed): <b>00</b> : Button Down (pressed) <b>01</b> : Button Up (not pressed)	
	x	Top left horizontal start position of the button.	
	y	Top left vertical start position of the button.	
	buttonColour	2 bytes define the button color.	
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: <b>00</b> : 5x7 internal font <b>01</b> : 8x8 internal font <b>02</b> : 8x12 internal font  These fonts can be altered and other fonts can be added.	
	stringColour	2 bytes define the string text color.	
	width	This byte defines the width or horizontal size (x magnification) of the character in the string. Affects the total width of the string and button.	
	height	This byte defines the height or vertical size (y magnification) of the character in the string. Affects the total height of the string and button.	
	"string"	String of ASCII characters displayed inside the button. Limit the string to a single line width.	
	terminator	The string must be terminated with 00.	
	<b>Response</b>	<b>acknowledge</b>	
		acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	<p>This command will place a Text button similar to the ones used in a PC Windows environment.</p> <p>The (x, y) refers to the top left corner of the button and the size of the button is automatically calculated and drawn on the screen with the string text relatively justified inside the button.</p> <p>The button can be displayed in an UP (button not pressed) or DOWN (button pressed) position by specifying the appropriate value in the "state" byte.</p> <p>Separate button and text colors provide many variations in appearance and format.</p>		



**4.3.6 Draw “String” of ASCII Text in text format (73)**

<b>Command</b>	<b>cmd, column, row, font, stringColour(msb:lsb), “string”, terminator</b>	
	cmd	<b>73</b> or <b>s</b> (ascii)
	column	Horizontal position of the character (character units). Range: 0 to <b>29</b> dec for 5x7 font. Range: 0 to <b>22</b> dec for 8x8 and 8x12 fonts.
	row	Vertical position of the character (character units). Range: 0 to <b>27</b> dec for 5x7 and 8x8 fonts. Range: 0 to <b>18</b> dec for 8x12 font.
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: <b>00</b> : 5x7 internal font <b>01</b> : 8x8 internal font <b>02</b> : 8x12 internal font  These fonts can be altered and other fonts can be added. ORing the fonts with 0x10 will cause the string to be displayed in a proportional manner (e.g. 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text color.
	“string”	String of ASCII characters to be displayed (max. 256 characters).
	terminator	The string must be terminated with 00.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	<p>This command will draw/display a string of ASCII text anywhere on the screen in character unit coordinates. The horizontal start position of the string is specified by the column and the vertical position is specified by the row parameters. The string must be terminated with 00. If the length of the string is longer than the maximum number of characters per line, a wrap around will occur on to the next line. Maximum string length is 256 bytes.</p>	
		

#### 4.3.7 Draw ASCII Character in graphics format (74)

<b>Command</b>	<b>cmd, char, x, y, charColour(msb:lsb), width, height</b>	
	cmd	<b>74</b> or <b>t</b> (ascii)
	char	Inbuilt standard ASCII character. Range: 32dec – 127dec (20hex - 7Fhex).
	x	Horizontal position of the character (pixel units).
	y	Vertical position of the character (pixel units).
	charColour	2 bytes define the character color.
	height	This byte defines the height or vertical size (multiplier) of the character.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful
<b>Description</b>	This command will draw/display an ASCII character anywhere on the screen in pixel coordinates specified by x and y parameters. Unlike the “Draw ASCII Character in text format” command, this option allows text of any size (determined by width and height) to be placed at any position. The font of the character is determined by the “Set Font” command.	
		

## 4.4 SD Memory Card Commands

The commands detailed in this section utilize the SD/microSD memory card which must be connected to the SPI port of the module (pin J2.3 to J2.7). The memory card is used as the storage medium for all multimedia objects such as images, icons, animations and video clips which can be accessed and displayed. The memory card can also be used by the host controller as a general purpose storage medium, for example in data logging applications.

The following commands are related to Low-Level memory card operations and they are described in this section.

Command	Code (HEX)
Set Address Pointer of Memory Card	40 41
Screen Copy-Save to Memory Card	40 43
Display Image-Icon from Memory Card	40 49
Display Object from Memory Card	40 4F
Run Script (4DSL) Program from Memory Card	40 50
Read Sector Block Data from Memory Card	40 52
Display Video-Animation Clip from Memory Card	40 56
Write Sector Block Data to Memory Card	40 57
Initialize Memory Card	40 69
Read Byte Data from Memory Card	40 72
Write Byte Data to Memory Card	40 77

#### 4.4.1 Set Address Pointer of Memory Card (40 41)

<b>Command</b>	<b>ext_cmd, cmd, Address(Umsb:Ulsb:Lmsb:Llsb)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>41</b> or <b>A</b> (ascii)
	Address	A 4 byte card memory address (big endian) for byte wise access.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	This command sets the internal memory address pointer for byte wise reads and writes. After a byte read or write, the memory Address pointer is automatically incremented internally to the next byte address location.	

#### 4.4.2 Screen Copy – Save to Memory Card (40 43)

<b>Command</b>	<b>ext_cmd, cmd, x, y, width, height, SectorAdd(hi:mid:lo)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>43</b> or <b>C</b> (ascii)
	x	Top left horizontal start position of screen area to be copied.
	y	Top left vertical start position of screen area to be copied.
	width	Width of screen area to be copied (source).
	height	Height of screen area to be copied (source).
		SectorAdd
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	<p>This command copies an area of the screen of specified size. The start location of the block to be copied is represented by x, y (top left corner) and the size of the area to be copied is represented by width and height parameters.</p> <p>This is similar the “Screen Copy-Paste” command but instead of the copied screen area being pasted to another location on the screen it is stored into the memory card. The stored screen image can then be later recalled from the memory card and redisplayed onto the screen at the same or different location by using the “Display Image-Icon from Memory Card” command.</p> <p>This is a very powerful feature for animating objects, smooth scrolling, or implementing a windowing system.</p> <p><b>Notes:</b> the “Screen Copy-Save to Memory Card” command always stores that part of the screen as a 16 bit image, i.e. 2 bytes per pixel. The images or icons when stored into the memory card must be sector boundary aligned, i.e. the object start location must be at the start of a sector boundary.</p>	

#### 4.4.3 Display Image-Icon from Memory Card (40 49)

<b>Command</b>	<b>ext_cmd, cmd, x, y, width, height, colourMode, SectorAdd(hi:mid:lo)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>49</b> or I (ascii)
	x	Image horizontal start position (top left corner).
	y	Image vertical start position (top left corner).
	width	Horizontal size of the image.
	height	Vertical size of the image.
	colourMode	<b>08</b> : 256 color mode, 8bits/1byte per pixel. <b>10</b> : 65K color mode, 16bits/2bytes per pixel.
SectorAdd	3 bytes (big endian) sector address of a previously stored Image-Icon that is about to be displayed.	
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	<p>This command displays a bitmap image or an icon on the screen that has been previously stored at a particular sector address in the memory card.</p> <p>The screen position of the image to be displayed is specified by (x, y) and the size of the image by width and height parameters.</p> <p>If the previously stored image was in 8 bit color format (1 byte per pixel) or 16 bits (2 bytes per pixel) then this must be specified in the colourMode byte parameter.</p> <p>Do not store an image/icon in one color format then display it in another color format, this will result in a corrupted image.</p> <p><b>Notes:</b> the "Screen Copy-Save to Memory Card" command always stores that part of the screen as a 16 bit image, i.e. 2 bytes per pixel.</p> <p>The images or icons when stored into the memory card must be sector boundary aligned, i.e. the object start location must be at the start of a sector boundary.</p>	

#### 4.4.4 Display Object from Memory Card (40 4F)

<b>Command</b>	<b>ext_cmd, cmd, Address(Umsb:Ulsb:Lmsb:Llsb)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>4F</b> or O (ascii)
	Address	A 4 byte card memory address (big endian) of a previously stored Object that is about to be displayed.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	<p>Some of the commands can be stored as objects in the memory card which can be later recalled by the host on demand and displayed or executed.</p> <p>The user must make sure the 32 bit address of each stored command/object is known before using this feature.</p> <p>For example, a series of images can be stored as icons and later displayed as the application requires them.</p> <p>The table at the end of this section lists all of the commands that can be stored as objects within the memory card.</p>	

#### 4.4.5 Run Script (4DSL) Program from Memory Card (40 50)

<b>Command</b>	<b>ext_cmd, cmd, Address(Umsb:Ulsb:Lmsb:Llsb)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>50</b> or <b>P</b> (ascii)
	Address	A 4 byte card memory start address (big endian) of a 4DSL (4D Scripting Language) program.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	The majority of the commands can be composed as a script and written into memory card. A script program is a sequence of those commands that reside and can be executed from inside the memory card and these can be a combination of graphics, text, image, video and audio commands. Complete list of commands available for the scripting program is listed in section 4.5.	
	This command forces the 32bit internal memory pointer to jump to the specified address and automatically start executing a script program, from the memory card without any further interaction by the host processor. It will sequentially execute any valid instruction and commands until it gets to the end of the program.	
<b>Example</b>	A sample script program inside the memory card:	
	<u>Address</u>	<u>Command</u>
	00000000	45
	00000001	43 64 32 14 00 1F
	0000000A	07 03 E8
	0000000D	72 00 00 3C 3C 07 E0
	00000018	40 56 00 00 46 32 10 0A 02 5F 00 10 00
00000029	0B 00 00 00 00	
	<u>Comment</u>	Erase Screen
		Draw Circle
		Delay(1second)
		Draw Rectangle
		Play video from card
		Goto Address 00000000

#### 4.4.6 Read Sector Block Data from Memory Card (40 52)

<b>Command</b>	<b>ext_cmd, cmd, SectorAdd(hi:mid:lo)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>52</b> or <b>R</b> (ascii)
	SectorAdd	3 bytes (big endian) sector address. Sector addresses range from 0 to 16,777,215 depending on the capacity of the card. Each sector is 512 bytes in size. There are 2048 sectors per every 1Mb of card memory.
<b>Response</b>	<b>data(1..512)</b>	
	data	512 bytes of sector data
<b>Description</b>	This command will return 512 bytes of data relating to a sector.	

#### 4.4.7 Display Video-Animation Clip from Memory Card (40 56)

<b>Command</b>	<b>ext_cmd, cmd, x,y,width, height, colourMode, delay, frames(msb:lsb), SectorAdd(hi:mid:lo)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>56</b> or <b>V</b> (ascii)
	x	Video horizontal start position (top left corner).
	y	Video vertical start position (top left corner).
	width	Horizontal size of the video-animation.
	height	Vertical size of the video-animation.
	colourMode	08(hex) : 256 color mode, 8bits/1byte per pixel. 10(hex) : 65K color mode, 16bits/2bytes per pixel .
	delay	1 byte inter-frame delay in milliseconds.
	frames	2 bytes (big endian) total frame count in the video-animation clip.
	SectorAdd	3 bytes (big endian) sector address of a previously stored video-animation clip that is about to be displayed.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	This command plays a video or an animation clip on the screen that has been previously stored at a particular sector address in the memory card. The screen position of the clip to be played is specified by (x, y) and the size of the clip by width and height parameters	

#### 4.4.8 Write Sector Block Data to Memory Card (40 57)

<b>Command</b>	<b>ext_cmd, cmd, SectorAdd(hi:mid:lo), data(1..512)</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>57</b> or <b>W</b> (ascii)
	SectorAdd	3 bytes (big endian) sector address.
	data	512 bytes of sector data. Data length must be 512 bytes.
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	<p>This command allows downloading and writing blocks of sector data to the card. The data block must always be 512 bytes in length. For large volumes of data such as images, the data must be broken up into multiple sectors (chunks of 512 bytes) and this command then maybe used many times until all of the data is written. If the data block to be written is less than 512 bytes in length, then make sure the rest of the remaining data are padded with 00 or FF (it can be anything).</p> <p>If only few bytes of data are to be written then the "Write Byte Data to Memory Card" command can be used.</p> <p>Once this command is sent, the device will take a few milliseconds to write the data into its memory card and at the end of which it will respond.</p> <p>Only data(1..512) are written to the sector. Other bytes in the command message do not get written.</p>	

**4.4.9 Initialize Memory Card (40 69)**

<b>Command</b>	<b>ext_cmd, cmd</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>69</b> or i (ascii)
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	This command initializes the memory card. The memory card is always initialized upon Power-Up or Reset cycle, if the card is present. If the card is inserted after the power up or a reset then this command must be used to initialize the card.	
	<b>Note:</b> There is no card insert/remove auto detect facility.	

**4.4.10 Read Byte Data from Memory Card (40 72)**

<b>Command</b>	<b>ext_cmd, cmd</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>72</b> or r (ascii)
<b>Response</b>	<b>data_byte</b>	
	data_byte	1 byte of card data
<b>Description</b>	This command provides a means of reading a single byte of data back from the card. Before this command can be used, memory address location must be set using the "Set Address Pointer of Memory Card" command. Once this command is sent, the device will return 1 byte of data relating to that memory location set by the memory address pointer. The memory address location pointer is automatically incremented to the next byte address location.	

**4.4.11 Write Byte Data to Memory Card (40 77)**

<b>Command</b>	<b>ext_cmd, cmd, data</b>	
	ext_cmd	<b>40</b> or @ (ascii)
	cmd	<b>77</b> or w (ascii)
	data	1 byte of card data
<b>Response</b>	<b>acknowledge</b>	
	acknowledge	<b>ACK</b> (06) byte if successful <b>NAK</b> (15) byte if unsuccessful or card not present
<b>Description</b>	This command permits writing single bytes of data to the card. This is useful for writing small chunks of data at irregular intervals quickly. For large data blocks it is more efficient to use the "Write Sector Block Data to Memory Card" command described previously. Before this command can be used, the card memory address location must be set using the "Set Address Pointer of Memory Card" command. Once the Write Byte command is sent, a single byte of data will be stored to that memory location set by the memory address pointer. The memory address pointer is automatically incremented to the next location. Only the data byte is written. Other bytes in the command message are not stored.	

## 4.5 Script Commands

The command execution is not only limited to the host sending commands via the serial interface. The majority of them can be composed as a script and written into the optional memory card. A script program is a sequence of those commands, and can be a combination of graphics, text, image, video and audio commands.

The commands detailed in this section must reside in the SD/microSD memory card. They form the heart of a simple scripting language that can be sequentially executed and run from the card.

Majority of the commands described in the previous sections can also be included and executed within the script.

To create scripts and save them into the SD card you need to use the FAT Controller software from 4D Systems. For additional details, please refer to the separate user manual of the FAT Controller.

The following commands are related to Low-Level memory card operations and they are described in this section.

Command	Code (HEX)
Delay	07
Set Counter	08
Decrement Counter	09
Jump to Address If Counter Not Zero	0A
Jump to Address	0B
Exit-Terminate Script Program	0C




### Download

FAT Controller: <http://www.4dsystems.com.au/downloads/Software/FAT-Controller/>

**4.5.1 Delay (07)**

<b>Command</b>	<b>ScriptCmd, value(msb:lsb)</b>	
	scriptCmd	<b>07</b>
<b>Description</b>	value	2 byte (big endian) delay value in milliseconds.
	When commands are executed within the script program a delay can be inserted between subsequent commands. A delay basically has the same effect as a NOP (No Operation) which can be used as a pause between drawing objects or displaying images-videos etc.	

**4.5.2 Set Counter (08)**

<b>Command</b>	<b>ScriptCmd, value</b>																				
	scriptCmd	<b>08</b>																			
<b>Description</b>	value	1 byte counter value that can be used with “Decrement Counter” and “Jump to Address If Counter Not Zero” commands to form loops. Practical values should be between 2 and 255.																			
	<p>Series of images that might be part of an animation may need to be redisplayed over and over to achieve a lengthy viewing. This command when used in conjunction with “Decrement Counter” and “Jump to Address If Counter Not Zero” commands allow the user to determine exactly how many times the series of images are looped.</p> <p>For example, we may want to animate the Globe rotating. Let’s say we have 10 image slides of the Globe at different rotated positions residing in the memory card. When the images are displayed sequentially, the effective duration will only be the length of time it takes to display the 10 image frames. We can increase that length by looping through the animation a number of times depending on the value set in the counter. When the display reaches the end of the last frame and encounters the Decrement Counter followed by Jump to Address If Counter Not Zero commands, the counter will be decremented and then the internal pointer will jump to the memory Address specified in the “Jump to Address If Counter Not Zero” command. This sequence will repeat until the value in the counter reaches zero.</p> <p>The following demonstrates how this may be used:</p> <table border="0"> <thead> <tr> <th><u>Address</u></th> <th><u>Comment</u></th> </tr> </thead> <tbody> <tr> <td>00000000</td> <td>Set Counter (value = 25)</td> </tr> <tr> <td>00000002</td> <td>Display Image from Memory Card (image1)</td> </tr> <tr> <td>00000012</td> <td>Delay(10ms)</td> </tr> <tr> <td>00000015</td> <td>Display Image from Memory Card (image2)</td> </tr> <tr> <td>00000025</td> <td>Delay(10ms)</td> </tr> <tr> <td>00000119</td> <td>Display Image from Memory Card (image10)</td> </tr> <tr> <td>00000129</td> <td>Delay(10ms)</td> </tr> <tr> <td>00000132</td> <td>Decrement Counter</td> </tr> <tr> <td>00000134</td> <td>Jump to Address if Counter Not Zero (Address = 00000002)</td> </tr> </tbody> </table> <div style="text-align: center;">  </div> <p><b>Note:</b> The above example is typical of how a series of commands might be loaded into the memory card and then executed by using the Run Program from Memory Card command. The commands would of course be the series of hex codes.</p>		<u>Address</u>	<u>Comment</u>	00000000	Set Counter (value = 25)	00000002	Display Image from Memory Card (image1)	00000012	Delay(10ms)	00000015	Display Image from Memory Card (image2)	00000025	Delay(10ms)	00000119	Display Image from Memory Card (image10)	00000129	Delay(10ms)	00000132	Decrement Counter	00000134
<u>Address</u>	<u>Comment</u>																				
00000000	Set Counter (value = 25)																				
00000002	Display Image from Memory Card (image1)																				
00000012	Delay(10ms)																				
00000015	Display Image from Memory Card (image2)																				
00000025	Delay(10ms)																				
00000119	Display Image from Memory Card (image10)																				
00000129	Delay(10ms)																				
00000132	Decrement Counter																				
00000134	Jump to Address if Counter Not Zero (Address = 00000002)																				

**4.5.3 Decrement Counter (09)**

<b>Command</b>	<b>ScriptCmd</b>	
	scriptCmd	<b>09</b>
<b>Description</b>	Decrements the Counter. See detailed description on how this command can be used effectively in the "Set Counter" command section.	

**4.5.4 Jump to Address If Counter Not Zero (0A)**

<b>Command</b>	<b>ScriptCmd, Address(Umsb:Ulsb:Lmsb:Llsb)</b>	
	scriptCmd	<b>0A</b>
<b>Description</b>	Address	A 4 byte (big endian) card memory jump address if counter is not zero.
	If the internal counter is not zero the program pointer will jump to the specified address. If the counter is zero then it will continue executing the next script command. Please see detailed description on how this command can be used effectively in the "Set Counter" command section.	

**4.5.5 Jump to Address (0B)**

<b>Command</b>	<b>ScriptCmd, Address(Umsb:Ulsb:Lmsb:Llsb)</b>	
	scriptCmd	<b>0B</b>
	Address	A 4 byte (big endian) card memory jump address.
<b>Description</b>	This command will force the internal 32 bit program memory pointer to jump unconditionally to the specified address and start executing commands from there.	

**4.5.6 Exit-Terminate Script Program (0C)**

<b>Command</b>	<b>ScriptCmd</b>	
	scriptCmd	<b>0C</b>
<b>Description</b>	This command forces the program to stop executing from the memory card and ready to accept and execute commands from the host via the serial interface. When the internal program memory pointer encounters this command it will force the command execution from memory card to terminate. It can also be sent, by the host, via the serial link to terminate a program currently executing from the memory card.	

## 4.6 Summary List of Commands available for Scripting

The commands listed below are all of the available commands for composing a script program that can be executed within the memory card.

<b>Command</b>	<b>Code (HEX)</b>
Set Background Color	<b>42</b>
Clear Screen	<b>45</b>
Display Control Functions	<b>59</b>
Switch-Buttons-Joystick Status	<b>4A</b>
Switch-Buttons-Joystick Wait for Status	<b>6A</b>
Sound	<b>4E</b>
Draw Circle	<b>43</b>
Draw Triangle	<b>47</b>
Draw Line	<b>4C</b>
Draw Pixel	<b>50</b>
Draw Polygon	<b>67</b>
Set Pen Size	<b>70</b>
Draw Rectangle	<b>72</b>
Set Font	<b>46</b>
Set Transparent-Opaque Text	<b>4F</b>
Draw "String" of ASCII Text (graphics format)	<b>53</b>
Draw ASCII Character (text format)	<b>54</b>
Draw Text Button	<b>62</b>
Draw "String" of ASCII Text (text format)	<b>73</b>
Draw ASCII Character (graphics format)	<b>74</b>
Display Image-Icon from Memory Card	<b>40 49</b>
Display Video-Animation Clip from Memory Card	<b>40 56</b>
Delay	<b>07</b>
Set Counter	<b>08</b>
Decrement Counter	<b>09</b>
Jump to Address If Counter Not Zero	<b>0A</b>
Jump to Address	<b>0B</b>
Exit-Terminate Script Program	<b>0C</b>

## 5 Online Resources

All documentation, firmware, and development tools are available as a free download online at the links below. Please note that the documentation and software tools are updated periodically, so please be sure to check for the latest versions to take advantage of future updates and new features.

Any update notices will be posted on the EZOLED website at [www.ezoled.com](http://www.ezoled.com)



### Download

FONT-tool: <http://www.4dsystems.com.au/downloads/Software/Font-Tool/>

DISP-Tool: <http://www.4dsystems.com.au/downloads/Software/Disp-Tool/>

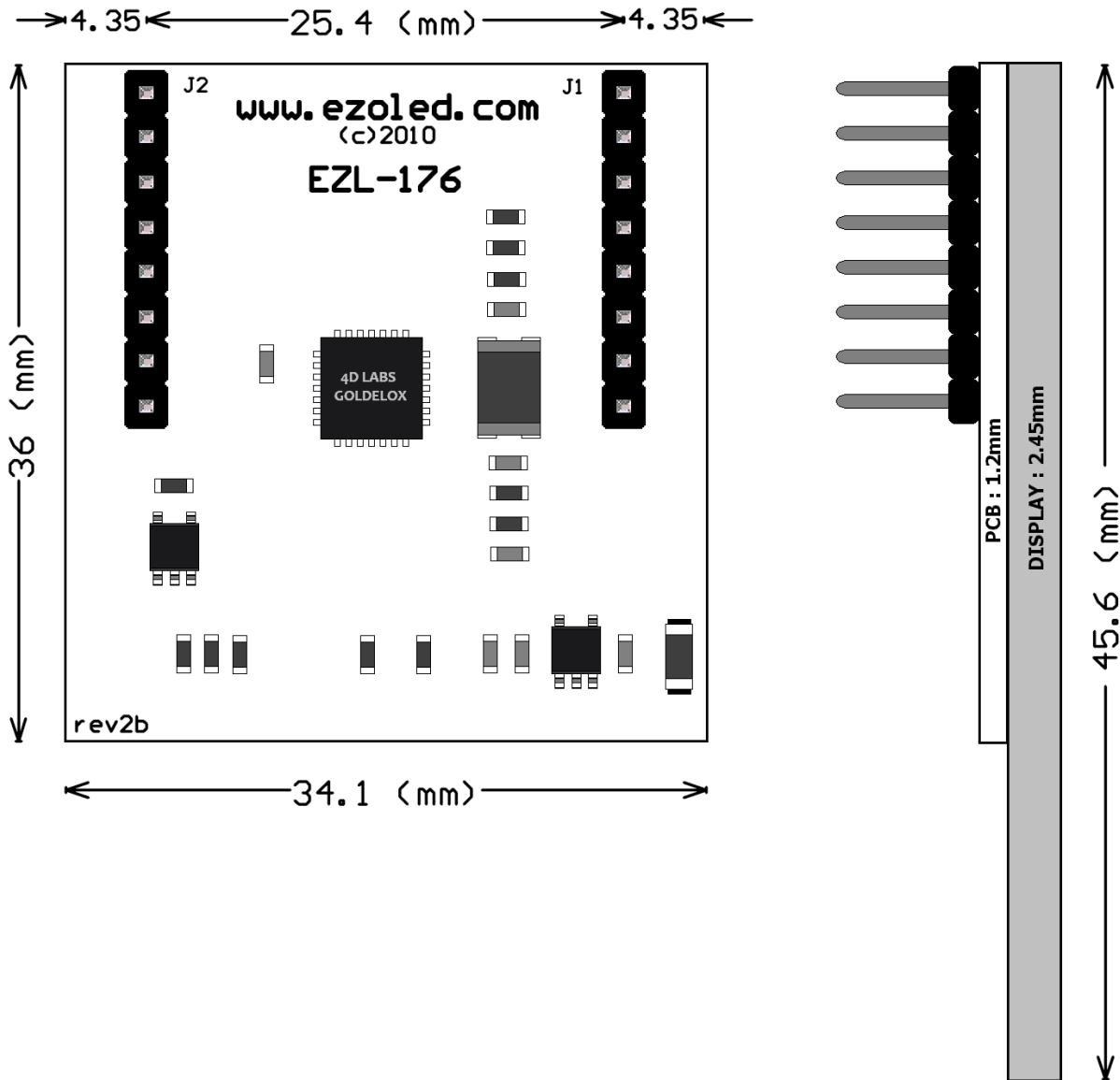
EZL-176 Firmware (PmmC): <http://www.ezoled.com>

EZL-176 Datasheet: <http://www.ezoled.com>

EZL-176 Software (FAT Controller): <http://www.ezoled.com>

---

## 6 Dimensions



## 7 Specifications and Ratings

Recommended Operating Conditions					
Parameter	Conditions	Min	Typ	Max	Units
Supply Voltage ( $V_{in}$ )	Pin J1.1	4.0	5.0	5.5	V
Supply Voltage ( $V_{3.3Vin}$ )	Pin J2.3	3.0		3.3	V
Operating Temperature		-20	--	+70	°C
Input Low Voltage	RX pin	GND	--	0.8	V
Input High Voltage	RX pin	2.0	3.3	5.0	V
Reset Pulse	External Open Collector	2.0	--	--	µs
Operational Delay	Power-Up or External Reset	1000	--	--	ms

Global Characteristics based on Operating Conditions					
Parameter	Conditions	Min	Typ	Max	Units
Supply Current ( $I_{in}$ )	$V_{in} = 5.0V$ , Backlight On	50	53	56	mA
Supply Current ( $I_{3.3Vin}$ )	$V_{3.3Vin} = 3.3V$ , Backlight On	46	47	50	mA
Output Low Voltage (VOL)	TX, SOUND pins, IOL = 3.4mA	--	--	0.4	V
Output High Voltage (VOH)	TX, SOUND pins, IOL = -2.0mA	2.4	--	3.3	V
A/D Converter Resolution	SWITCH pin	--	8	--	bits
Capacitive Loading	All pins	--	--	50	pF
Flash Memory Endurance	PmmC Programming	--	1000	--	E/W

Optical Characteristics					
Parameter	Conditions	Min	Typ	Max	Units
Luminance (L)	$V_{in} = 5.0V$	230	260	--	cd/m <sup>2</sup>
Viewing Angle (VA)	Temp = 25°			30	degree
				30	
				30	
				30	
Contrast Ratio (CR)	Temp = 25°C, $V_{in} = 5.0V$	300	450	--	--
Operational Lifetime (LT)	For Backlight			60000	hours

Ordering Information
Order Code: EZL-176
Package: 150mm x 95mm (ZIF Bag dimensions).
Packaging: Module sealed in antistatic padded ZIF bag.

## 8 Proprietary Information

The information contained in this document is the property of TIGAL KG and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

TIGAL KG endeavors to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of TIGAL KG products and services is continuous and published information may not be up to date. It is important to check the current position with TIGAL KG.

All trademarks belong to their respective owners and are recognized and acknowledged.

## 9 Disclaimer of Warranties & Limitation of Liability

TIGAL KG makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall TIGAL KG be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by TIGAL KG, or the use or inability to use the same, even if TIGAL KG has been advised of the possibility of such damages.

Use of TIGAL KG devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless TIGAL KG from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any TIGAL KG intellectual property rights.

## 10 Contact Information

For Technical Support: [support@ezoled.com](mailto:support@ezoled.com)

For Sales: [sales@ezoled.com](mailto:sales@ezoled.com)

Website: [www.ezoled.com](http://www.ezoled.com)

Copyright 2010 TIGAL KG  
EZOLED is a TIGAL KG Brand