µLCD 320 PMD2

µSD

USERS MANUAL

(Serial Command Platform)

Revision 1.1

4D Systems

PROPRIETORY INFORMATION

DISCLAIMER OF WARRANTIES & LIMITATION OF LIABILITY

Table of contents

1 µLCD Description

1.1 Introduction

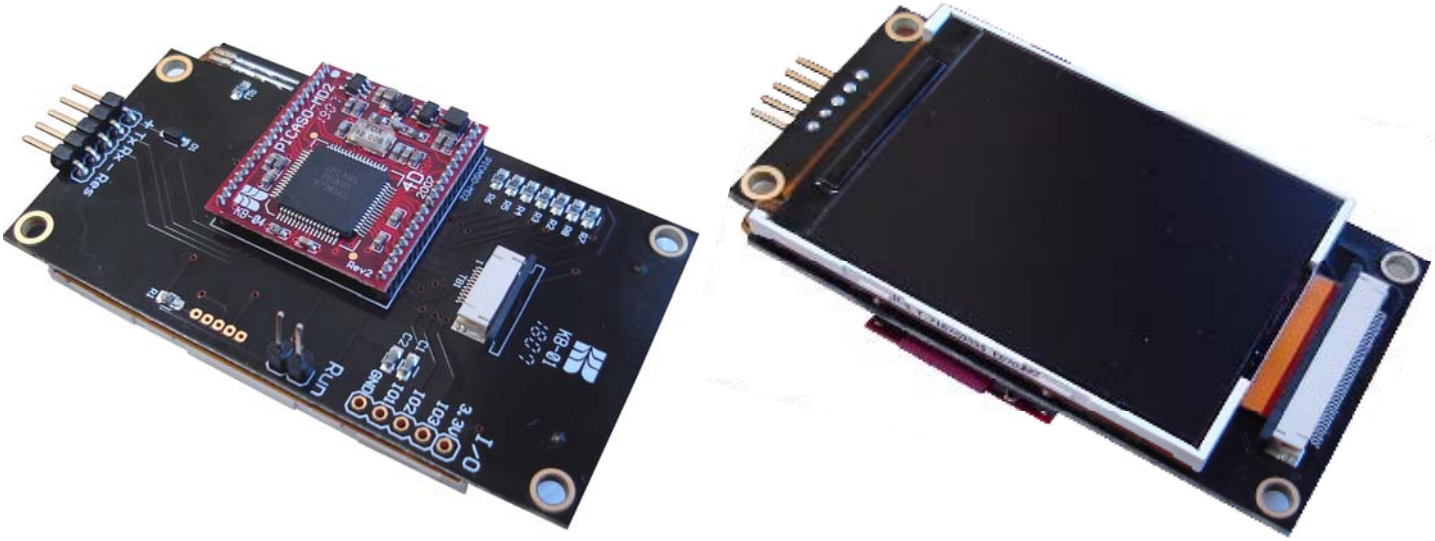
The µLCD is a compact & cost effective all in one ‘SMART” LCD module with an embedded graphics controller that will deliver ‘stand-alone’ functionality to your project. The ‘simple to use’ embedded commands not only control background colour but can produce text in a variety of sizes as well as draw shapes (which can include user definable bitmapped characters such as logos) in 256 or 65,536 colours whilst freeing up the host processor from the ‘processor hungry ‘ screen control functions. This means a simple micro-controller with a standard serial or USB interface can drive the µLCD module with total ease.

Figures below show some of the graphics capability of the µLCD.

1.2 µLCD-320-PMD2 Features

The **µLCD-320-PMD2** is aimed at being integrated into a variety of different applications via a wealth of features designed to facilitate any given functionality quickly and cost effectively and thus reduce 'time to market'. These features are as follows:

- 240 x 320 pixel resolution, 256 or 65K true to life colours, Enhanced LCD screen.
- 2.2" diagonal. Module Size: 74.5 x 40.5 x 10.3mm. Active Area: 33.75 x 45.25mm.
- LED backlighting with excellent viewing angle.
- Easy 5 pin interface to any host device: VCC, TX, RX, GND, RESET
- Voltage supply from 3.6V to 6.0V, current @ 60mA nominal when using a 5.0V supply source. **Note**: Due to the characteristics of some memory cards the module may require input voltages greater than 4.0 Volts.
- Serial RS-232 (0V to 3.3V) with auto-baud feature (2400 to 500K baud). If interfacing to a system greater than 3.6V supply, a series resistor (100 to 220 Ohms) maybe required between the module RX and the host TX line.
- Powered by the fully integrated **PICASO-MD2** module (also available as separate OEM modules for volume users).
- Optional USB to Serial interface via the 4D micro-USB (**uUSB-MB5**) module.
- Onboard micro-SD (**µSD**) memory card adaptor for storing of icons, images, animations, etc. 64Mb to 2Gig µSD memory cards can be purchased separately.
- Four selectable font sizes (5x7, 8x8 8x12 and 12x16) for ASCII characters as well as user-defined bitmapped characters.
- Built in graphics commands such as: LINE, CIRCLE, RECTANGLE, TEXT, USER BITMAP, BACKGROUND COLOUR, PUT PIXEL, IMAGE, etc. just to name a few.



2 µLCD-320-PMD2 Serial Command Set

The heart of the µLCD-320-PMD2 is the easy to understand command set. This comprises of a handful of easy to learn instructions that can draw lines, circles, squares, etc, to provide a full text and graphical user interface. The commands are sent to the µLCD-320-PMD2 via its serial connection (5 pin header). The command set is grouped into 3 sections:

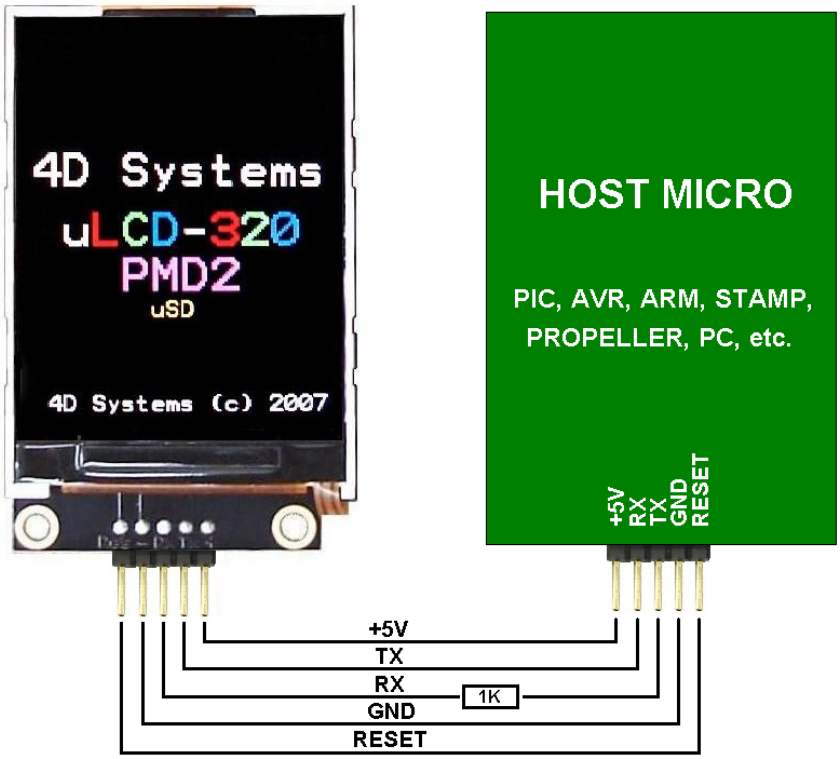
- General Command Set
- Display Specific Command Set
- Extended Command Set

Each Command set is described in detail in the following sections.

NOTE!

The RX and the TX signals are at 3.3V levels. If interfacing to a host system running at voltages greater than 3.6V levels, say 5.0Volts, then a 100 to 220 Ohms series resistor must be inserted between the Host Tx and the µLCD-320-PMD2 Rx signals.

Serial Data Format: 8 Bits, No Parity, 1 Stop Bit.



2.1 Command Protocol

The following are each of the commands with the correct syntax. Please note that all command examples listed below are in hex (**00**hex). Due to the high colour depth of the µLCD, a pixel colour value will not fit into a single byte, a byte can only hold a maximum value of 255. Therefore the colour is represented as a 2 byte value, **colour**(msb:lsb). The most significant byte (msb) is transmitted first followed by the least significant byte (lsb). This format is called the big endian. So for a 2 byte colour value of **013F**hex the byte order can be shown as (**01**hex),(**3F**hex). The 2 byte value also applies to the **y** coordinates as the vertical resolution is 320 pixels high.

NOTE: When transmitting the command and data bytes to the µLCD, do not include any separators such as commas ‘,’ or spaces ‘ ‘ or brackets ‘(‘ ‘)’ between the bytes. The examples show these separators purely for legibility; these must not be included when transmitting data to the µLCD.

When a command is sent, the µLCD will reply back with a single acknowledge byte called the **ACK** (**06**hex). This tells the host that the command was understood and the operation is completed. It will take the µLCD anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the µLCD has to perform. If the µLCD receives a command that it does not understand it will reply back with a negative acknowledge called the **NAK** (**15**hex).

If a command that has 5 bytes but only 4 bytes are sent, the command will not be executed and the µLCD will wait until another byte is sent before trying to execute the command. There is no timeout on the µLCD when incomplete commands are sent. The µLCD will reply back with a **NAK** for each invalid command it receives. For correct operation make sure the command bytes are sent in the correct sequence.

2.2 General Command Set

General Command Set	Live	Object	Memory
(A) **A**dd User Bitmapped Character	√		
(B) Set **B**ackground Colour	√	√	√
(b) Place Text **b**utton	√	√	√
(C) Draw **C**ircle	√	√	√
(c) Block **c**opy and Paste (bitmap copy)	√		
(D) **D**isplay User Bitmapped Character	√		
(e) Draw **e**llipse	√	√	√
(E) **E**rase Screen	√	√	√
(F) **F**ont Size	√	√	√
(G) Draw Trian**G**le	√	√	√
(g) Draw Poly**g**on	√	√	√
(I) Display **I**mage	√		
(L) Draw **L**ine	√	√	√
(O) **O**paque or Transparent Text	√	√	√
(P) Put **P**ixel	√		
(p) Set **p**en Size	√	√	√
(R) **R**ead Pixel	√		
(r) Draw **r**ectangle	√	√	√
(S) Place **S**tring of ASCII Text (unformatted)	√	√	√
(s) Place **s**tring of ASCII Text (formatted)	√	√	√
(T) Place **T**ext Character (formatted)	√	√	√
(t) Place **t**ext Character (unformatted)	√	√	√
(V) **V**ersion/Device Info Request	√		
(Y) LCD Displa**Y** Control functions	√	√	√

NOTES:

Live : **Those commands that can be sent via the serial link and executed by the uLCD module.**

Object : **Those commands that can be recalled from the memory card at any time by the host and displayed on the screen using the "Display Object from Memory Card" command.**

Memory: **Those commands that can reside and be executed from inside the memory card.**

2.2.1 Add User Bitmapped Character (A)

Syntax : cmd, group, char#, data1, data2, ……., dataN

cmd : 41hex**, A**ascii

group : selects the appropriate bitmap format
00hex selects the 8x8 bitmap format
01hex selects the 16x16 bitmap format
02hex selects the 32x32 bitmap format

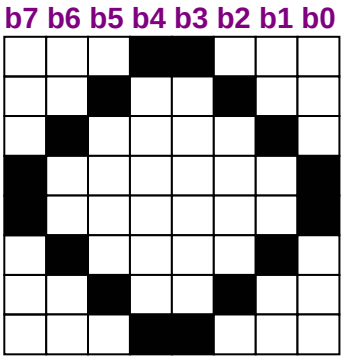
char# : bitmap character number to add to memory:
0 to 63 (**00**h to **3F**h), 64 characters of 8x8 format when group = **00**h
0 to 15 (**00**h to **0F**h), 16 characters of 16x16 format when group = **01**h
0 to 7 (**00**h to **07**h), 8 characters of 32x32 format when group = **02**h

data1 to dataN : number of data bytes that make up the composition and format of the bitmapped character. The 8x8 bitmap composition is 1 byte wide (8bits) by 8 bytes deep which makes N = 1x8 = 8. The 16x16 bitmap composition is 2 bytes wide (16bits) by 16 bytes deep which makes N = 2x16 = 32. The 32x32 bitmap composition is 4 bytes wide (32bits) by 32 bytes deep hence N = 4x32 = 128.

Description : This command will add a user defined bitmapped character into the internal memory. There are 3 different size bitmaps available, 8x8 format, 16x16 format and 32x32 format. The desired format is selected by specifying the appropriate value in '**group**'.

Example1: 41hex**, 00**hex**, 01**hex**, 18**hex**, 24**hex**, 42**hex**, 81**hex**, 81**hex**, 42**hex**, 24**hex**, 18**hex

This adds and saves user defined 8x8 bitmap (group 0) as character number 1 into memory as seen below.



data1 (hex = 18h)
data2 (hex = 24h)
data3 (hex = 42h)
data4 (hex = 81h)
data5 (hex = 81h)
data6 (hex = 42h)
data7 (hex = 24h)
data8 (hex = 18h)

Example of a 8x8 user defined bitmap

Example2: **41**hex , **01**hex, **03**hex, **3F**hex, **FC**hex, **40**hex, **02**hex, **80**hex, **01**hex, **BC**hex, **3D**hex, **98**hex, **19**hex, **98**hex, **19**hex, **81**hex, **81**hex, **81**hex, **81**hex, **80**hex, **01**hex, **40**hex, **02**hex, **24**hex, **24**hex, **23**hex, **C4**hex, **21**hex, **84**hex, **10**hex, **08**hex, **08**hex, **10**hex, **07**hex, **E0**hex

This adds and saves user defined 16x16 bitmap (group 1) as character number 3 into memory as seen below.

	b7 b6 b5 b4 b3 b2 b1 b0 b7 b6 b5 b4 b3 b2 b1 b0	
data1(hex=3Fh)	▯ ▯ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ▯ ▯	data2(hex=FCh)
data3(hex=40h)	▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ■ ▯ ▯	data4(hex=02h)
data5(hex=80h)	■ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ■	data6(hex=01h)
data7(hex=BCh)	■ ▯ ■ ■ ■ ■ ▯ ▯ ▯ ▯ ■ ■ ■ ■ ▯ ■	data8(hex=3Dh)
data9(hex=98h)	■ ▯ ▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ■ ■ ▯ ▯ ■	data10(hex=19h)
data11(hex=98h)	■ ▯ ▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ■ ■ ▯ ▯ ■	data12(hex=19h)
data13(hex=81h)	■ ▯ ▯ ▯ ▯ ▯ ▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ■	data14(hex=81h)
data15(hex=81h)	■ ▯ ▯ ▯ ▯ ▯ ▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ■	data16(hex=81h)
data17(hex=80h)	■ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ■	data18(hex=01h)
data19(hex=40h)	▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ■ ▯ ▯	data20(hex=02h)
data21(hex=24h)	▯ ▯ ■ ■ ▯ ▯ ■ ▯ ▯ ▯ ▯ ■ ▯ ▯ ■ ■ ▯ ▯	data22(hex=24h)
data23(hex=23h)	▯ ▯ ■ ■ ▯ ▯ ▯ ■ ■ ■ ■ ▯ ▯ ▯ ■ ■ ▯ ▯	data24(hex=C4h)
data25(hex=21h)	▯ ▯ ■ ■ ▯ ▯ ▯ ■ ■ ▯ ▯ ▯ ▯ ▯ ■ ■ ▯ ▯	data26(hex=84h)
data27(hex=10h)	▯ ▯ ▯ ■ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ▯ ■ ▯ ▯ ▯ ▯	data28(hex=08h)
data29(hex=08h)	▯ ▯ ▯ ▯ ■ ▯ ▯ ▯ ▯ ▯ ▯ ■ ▯ ▯ ▯ ▯ ▯	data30(hex=10h)
data31(hex=07h)	▯ ▯ ▯ ▯ ▯ ■ ■ ■ ■ ■ ■ ▯ ▯ ▯ ▯ ▯	data32(hex=E0h)

Example of a 16x16 user defined bitmap

2.2.2 Set Background Colour (B)

Syntax : cmd, colour(msb:lsb)

cmd : 42hex**, B**ascii

colour(msb:lsb) **:** pixel colour value: 2 bytes (16 bits) msb:lsb
65,536 colours to choose from
Black = **0000**hex, **0**dec
White = **FFFF**hex, **65,535**dec, **1111111111111111**bin

Description : This command sets the current background colour. Once this command is sent, only the background colour will change. Any other object on the screen with a different colour value will not be affected.

Example : 42hex**, FFFF**hex
Set the background colour to value 65,535 (**white**).

2.2.3 Text button (b)

Syntax : cmd, state, x, **y**(msb:lsb), **buttonColour**(msb:lsb), **font, textColour**(msb:lsb)**, textWidth, textHeight, char1, .., charN, terminator**

cmd : 62hex**, b**ascii

state : Specifies whether the displayed button is drawn as **UP** (not pressed) or **DOWN** (pressed). 0 = Button Down (pressed)
1 = Button Up (not pressed)

x : top left horizontal start position of the button

y(msb:lsb) **:** top left vertical start position of the button , 2 bytes

buttonColour(msb:lsb) **:** 2 byte button colour value

font : 0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font. This has precedence and does not affect the Font command.

textColour(msb:lsb) **:** 2 byte text colour value

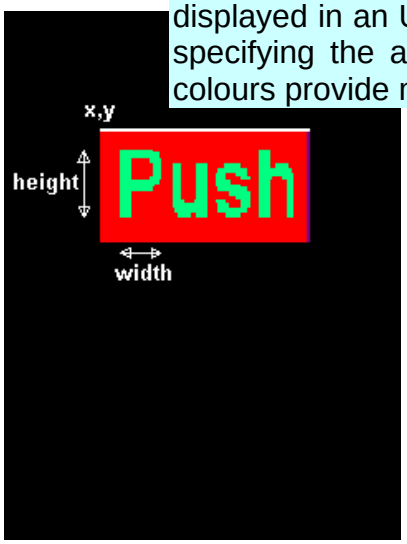
textWidth : horizontal size of the character, effects the width of the button.

textHeight : vertical size of the character, effects the height of the button.

char1..charN : string of ASCII characters (limit the string to line width)

terminator : the string must be terminated with **00**hex

Description : This command will place a Text button similar to the ones used in a PC Windows environment. **(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 text relatively justified inside the button box. 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. Separate button and text colours provide many variations in appearance and format.



2.2.4 Draw Circle (C)

Syntax : **cmd, x**, **y**(msb:lsb), **rad**, **colour**(msb:lsb)

cmd : 43hex**, C**ascii

x : circle centre horizontal position. **0**dec to **239**dec (**00**hex to **EF**hex).

y(msb:lsb) **:** circle centre vertical position. **0**dec to **319**dec (**0000**hex to **013F**hex).

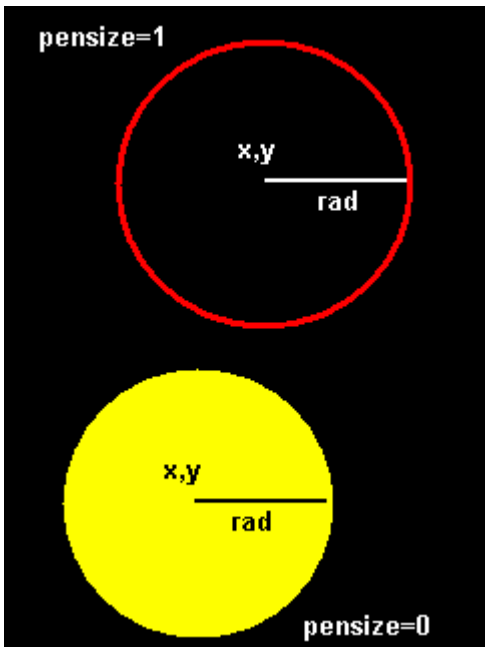
rad : radius size of the circle. **0**dec to **239**dec (**00**hex to **EF**hex).

colour(msb:lsb) **:** 2 byte circle colour value

Description : This command will draw a coloured circle centred at **(x, y)** with a radius determined by the value of **rad**. The circle can be either solid or wire frame (empty) depending on the value of the Pen Size (see **Set Pen Size** command). When Pen Size = 0 circle is solid, Pen Size = 1 circle is wire frame.

Example : **43**hex**, 3F**hex**, 00**hex**, 3F**hex**, 22**hex, **00**hex**, 1F**hex

Draws a RED circle (**001F**hex) centred at x = **63**dec (**3F**hex) and y = **63**dec (**003F**hex) with a radius of **34**dec (**22**hex).



2.2.5 Block copy & Paste (Screen Bitmap Copy) (c)

Syntax : **cmd, xs**, **ys**(msb:lsb), **xd**, **yd**(msb:lsb), **width**, **height**(msb:lsb)

cmd : 63hex**, c**ascii

xs: top left horizontal start position of block to be copied (source)

ys(msb:lsb)**:** top left vertical start position of block to be copied (source)

xd: top left horizontal start position of where copied block is to be pasted (destination)

yd(msb:lsb)**:** top left vertical start position of where the copied block is to be pasted (destination)

width: width of block to be copied (source)

height(msb:lsb)**:** height of block to be copied (source)

Description : This command copies an area of a bitmap block of specified size. 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).
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.

2.2.6 Display User Bitmapped Character (D)

Syntax : **cmd, group, char#, x**, **y**(msb:lsb), **colour**(msb:lsb)

cmd : 44hex**, D**ascii

group : selects the appropriate bitmap format:
00hex selects the 8x8 bitmap format
01hex selects the 16x16 bitmap format
02hex selects the 32x32 bitmap format

char# : which user defined character number to display from the selected group.
0 to 63 (**00**hex to **3F**hex), of 8x8 format when group = **00**hex
0 to 15 (**00**hex to **0F**hex), of 16x16 format when group = **01**hex
0 to 7 (**00**hex to **07**hex), of 32x32 format when group = **02**hex

x : horizontal display position of the character.

y(msb:lsb) **:** vertical display position of the character. 2 bytes.

colour(msb:lsb) **:** 2 byte circle colour value

Description : This command displays the previously defined user bitmapped character at location **(x, y)** on the screen. User defined bitmaps allow drawing & displaying unlimited graphic patterns quickly & effectively.

Example 1: **44**hex**, 00**hex**, 01**hex **, 00**hex**, 00**hex**, 00**hex**, F8**hex**, 00**hex
Display 8x8 bitmap character number 1 at x = 0, y = 0, colour = red

Example 2: **44**hex**, 00**hex**, 01**hex**, 08**hex**, 00**hex**, 00**hex**, 07**hex**, F0**hex
Display 8x8 bitmap character number 1 at x = 8, y = 0, colour = green

Example 3: **44**hex **, 00**hex**, 01**hex**, 10**hex**, 00**hex**, 00**hex**, 00**hex**, 1F**hex
Display 8x8 bitmap character number 1 at x = 16, y = 0, colour = blue

Example 4: **44**hex **, 01**hex**, 03**hex**, 10**hex**, 00**hex**, 20**hex**, FF**hex**, FF**hex
Display 16x16 bitmap character number 3 at x = 16, y = 32, colour = white

2.2.7 Draw ellipse (e)

Syntax : **cmd, x**, **y**(msb:lsb), **rx**, **ry**(msb:lsb), **colour**(msb:lsb)

cmd : 65hex**, e**ascii

x : ellipse centre horizontal position. **0**dec to **239**dec (**00**hex to **EF**hex).

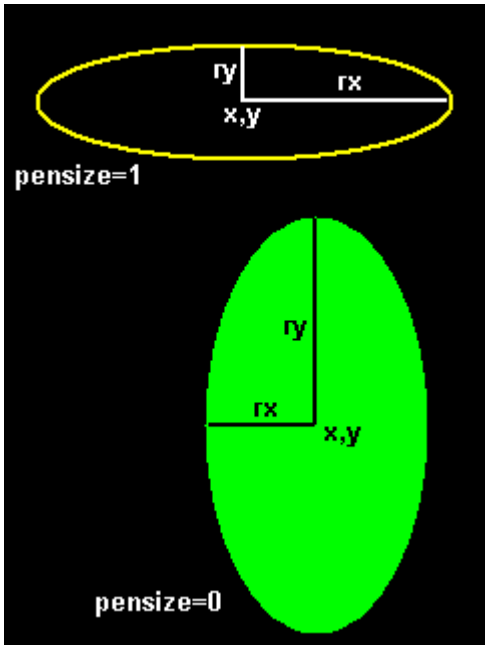
y(msb:lsb) **:** ellipse centre vertical position. **0**dec to **319**dec (**0000**hex to **013F**hex).

rx : radius in the x axis. **0**dec to **239**dec (**00**hex to **EF**hex).

ry(msb:lsb) **:** radius in the y axis. **0**dec to **319**dec (**0000**hex to **013F**hex).

colour(msb:lsb) **:** 2 byte ellipse colour value

Description : This command will draw a coloured ellipse centred at **(x, y)** with its shape determined by the values of **rx** (the x radius) and **ry** (the y radius). The ellipse can be either solid or wire frame (empty) depending on the value of the Pen Size (see **Set Pen Size** command). When Pen Size = 0 ellipse is solid, Pen Size = 1 ellipse is wire frame.



2.2.8 Erase Screen (E)

Syntax : cmd

cmd : 45hex**, E**ascii

Description : This command clears the entire screen using the current background colour.

Example : 45hex
Clear the screen.

2.2.9 Set Font Size (F)

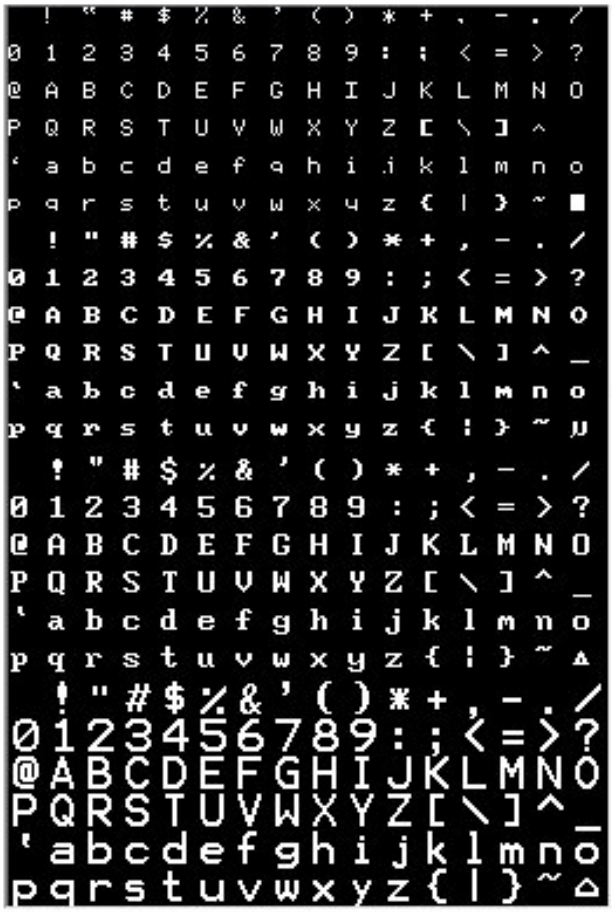
Syntax : cmd, size

cmd : 46hex**, F**ascii

size : = **00**hex : 5x7 small size font
= **01**hex : 8x8 medium size font
= **02**hex : 8x12 large size font
= **03**hex : 12x16 largest size font

Description : This command will change the size of the font according to the value set by **size**. Changes take place after the command is sent. Any character on the screen with the old font size will remain as it was.

Example1:	**46**hex**, 00**hex	Select small 5x7 fonts
Example1:	**46**hex**, 01**hex	Select medium 8x8 fonts
Example1:	**46**hex**, 02**hex	Select large 8x12 fonts
Example1:	**46**hex**, 03**hex	Select largest 12x16 fonts



2.2.10 Draw TrianGle (G)

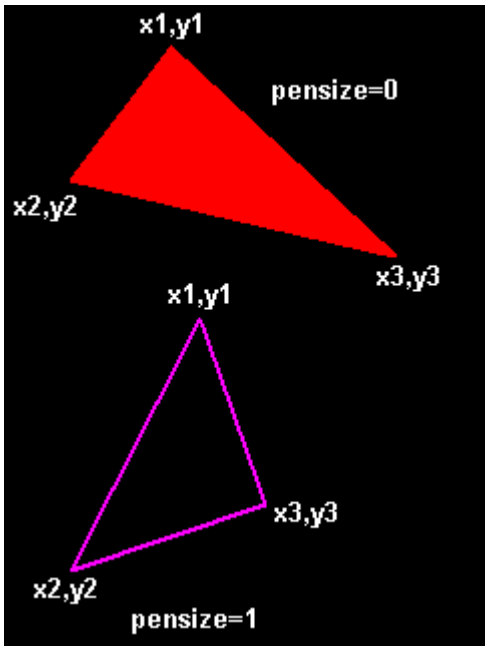
Syntax : cmd, x1, **y1**(msb:lsb), **x2**, **y2**(msb:lsb), **x3**, **y3**(msb:lsb), **colour**(msb:lsb)

cmd : 47hex**, G**ascii

x1, y1(msb:lsb)**, x2, y2**(msb:lsb)**, x3, y3**(msb:lsb) **:** 3 vertices of the triangle.
These must be specified in an anti-clockwise fashion.

colour(msb:lsb) **:** 2 byte triangle colour value

Description : This command draws a Solid/Empty triangle. The vertices must be specified in an anti-clock wise manner, i.e.
x2 < x1, x3 > x2, y2 > y1, y3 > y1.
A solid or a wire frame triangle is determined by the value of the Pen Size setting, i.e. 0 = solid, 1 = wire frame.



2.2.11 Draw Polygon (g)

Syntax : cmd, vertices, x1, **y1**(msb:lsb), **.. .. .. .. xn**, **yn**(msb:lsb), **colour**(msb:lsb)

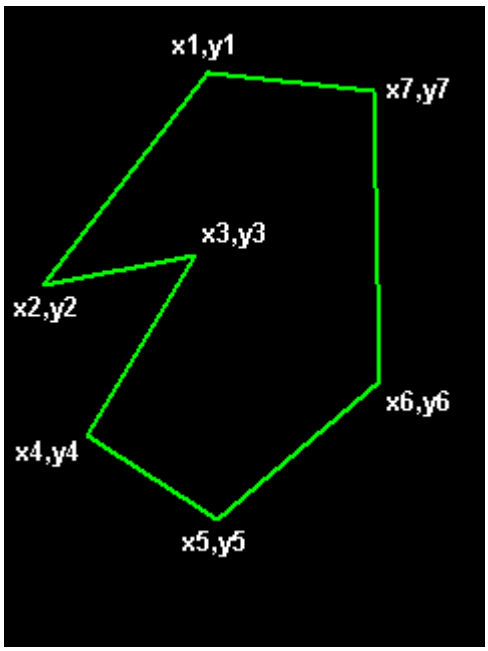
cmd : 67hex**, g** ascii

vertices : number of vertices from 3 to 7. Specifies the number of vertices of the polygon.

(x1, y1(msb:lsb)**) .. .. .. (xn, yn**(msb:lsb)**) :** vertices of the polygon. These can be specified in any fashion.

colour(msb:lsb) **:** 2 byte polygon colour value

Description : This command draws an Empty/Wire Frame polygon. Up to 7 vertices can be specified in any manner. Currently only a wire frame polygon is supported.



2.2.12 Display Image (I)

Syntax : **cmd, x**, **y**(msb:lsb), **width**, **height**(msb:lsb), **colourMode**, **pixel1**, .. **pixelN**

cmd : **49**hex**, I**ascii

x : Image horizontal start position (top left corner)

y(msb:lsb) **:** Image vertical start position (top left corner)

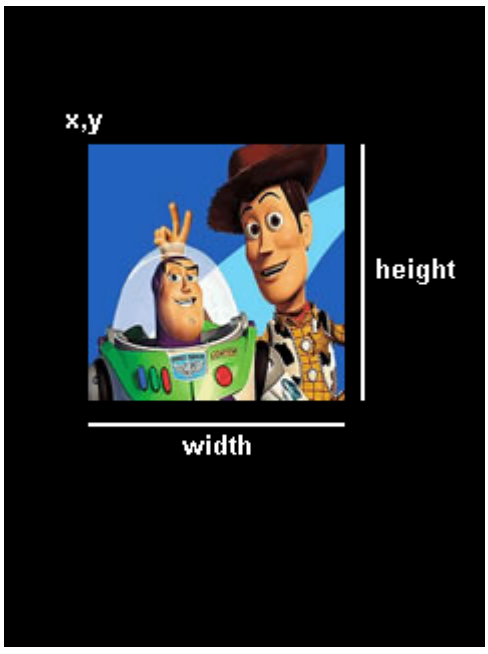
width : horizontal size of the image

height(msb:lsb) **:** vertical size of the image

colourMode : **8**dec = 256 colour mode, 8bits/1byte per pixel
16dec = 65K colour mode, 16bits/2bytes per pixel (msb:lsb)

pixel1..pixelN : image pixel data and N is the total number of pixels
N = height x width when colourMode = 8
N = height x width x 2 when colourMode = 16

Description : This command displays a bitmap image on to the screen with the top left corner specified by **(x, y)** and 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.



2.2.13 Draw Line (L)

Syntax : **cmd, x1**, **y1**(msb:lsb), **x2**, **y2**(msb:lsb), **colour**(msb:lsb)

cmd : **4C**hex**, L**ascii

x1 : horizontal position of line start. **0**dec to **239**dec (**00**hex to **EF**hex).

y1(msb:lsb) **:** vertical position of line start. **0**dec to **319**dec (**00**hex to **013F**hex).

x2 : horizontal position of line end. **0**dec to **239**dec (**00**hex to **EF**hex).

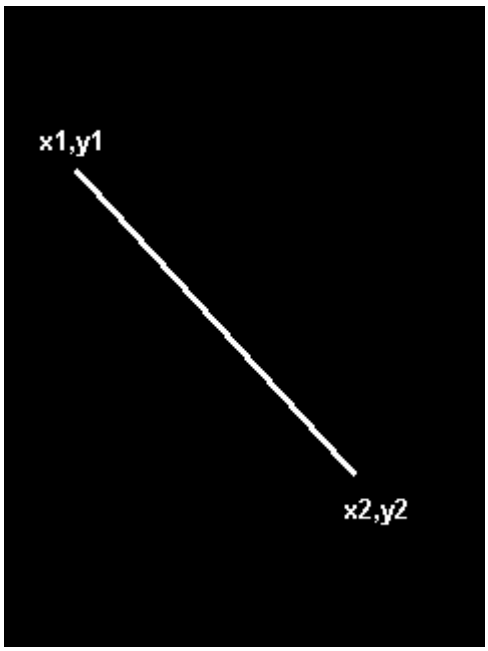
y2(msb:lsb) **:** vertical position of line end. **0**dec to **319**dec (**00**hex to **013F**hex).

colour(msb:lsb) **:** 2 byte line colour value

Description : This command will draw a coloured line from point **(x1, y1)** to point **(x2, y2)** on the screen.

Example : **4C**hex**, 00**hex**, 00**hex**, 00**hex**, 7F**hex**, 00**hex**, 7F**hex**, FF**hex**, FF**hex

Draws a white line from (x1=0, y1=0) to (x2=127, y2=127).



2.2.14 Opaque / Transparent Text (O)

Syntax : cmd, mode

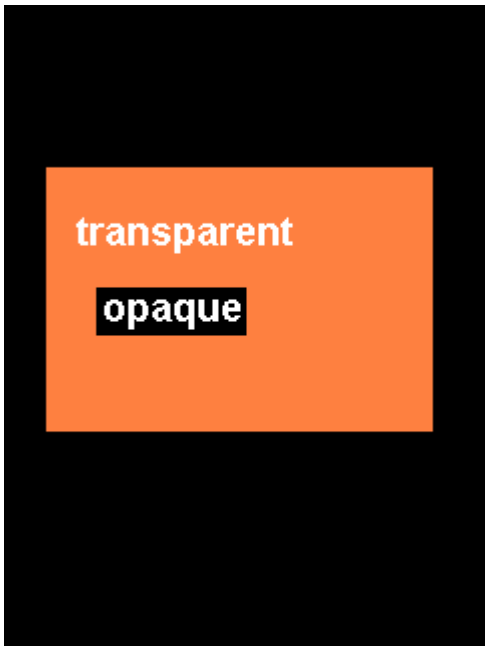
cmd : 4Fhex**, O**ascii

mode : = **00**hex : Transparent Text, objects behind the text can be seen.
= **01**hex: Opaque Text, objects behind text is blocked by background

Description : 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.

This command will change the attribute so that when a character is written, it will either write just the character alone (Transparent Mode) so any original character will be seen as well as the new, or overwrite any existing data with the new character.

Example1: **4F**hex**, 00**hex Transparent Text Mode
Example2: **4F**hex**, 01**hex Opaque Text Mode



2.2.15 Put Pixel (P)

Syntax : **cmd, x**, **y**(msb:lsb), **colour**(msb:lsb)

cmd : 50hex**, P**ascii

x : horizontal pixel position. **0**dec to **239**dec (**00**hex to **EF**hex).

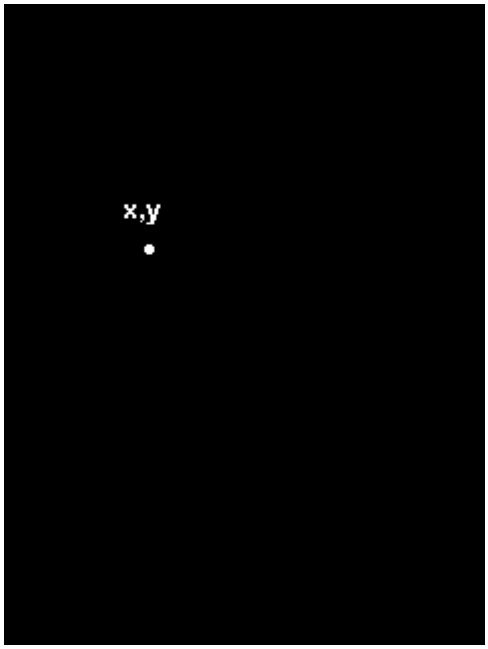
y(msb:lsb) : vertical pixel position. **0**dec to **319**dec (**0000**hex to **013F**hex).

colour : pixel colour value: 2 bytes (16 bits) msb, lsb
65,536 colours to choose from
Black = **0000**hex, **0**dec
White = **FFFF**hex, **65,535**dec, **1111111111111111**bin

Description : This command will put a coloured pixel at location **(x, y)** on the screen.

Example : **50**hex**, 01**hex**, 00**hex**, 0A**hex**, FF**hex**, FF**hex

Puts a white (**FFFF**hex) pixel at location x = **01**dec (**01**hex) and y = **10**dec (**0A**hex).



2.2.16 Set pen Size (p)

Syntax : cmd, size

cmd : 70hex**, p** ascii

size : = **00**hex : All objects such as circles, rectangles, triangles, etc are solid
= **01**hex : All objects are wire frame (empty)

Description : This command determines if certain graphics objects are drawn in solid or wire frame fashion.

Example1: 70hex**, 00**hex All objects will be drawn solid
Example1: 70hex**, 01**hex All objects will be drawn wire frame.

2.2.17 Read Pixel (R)

Syntax : **cmd, x**, **y**(msb:lsb)

cmd : 52hex**, R**ascii

x : horizontal pixel position. **0**dec to **239**dec (**00**hex to **EF**hex).

y(msb:lsb) **:** vertical pixel position. **0**dec to **319**dec (**0000**hex to **013F**hex).

Description : This command will read the colour value of 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 colour on the screen and switch the colour of the pointer when it's on top of a light coloured area.

Example : **52**hex**, 01**hex**, 00**hex, **01**hex
µLCD reply : 00hex, **1F**hex

Reads a blue (**001F**hex) pixel at location x = **1**dec (**01**hex) and y = **1**dec (**0001**hex).

2.2.18 Draw rectangle (r)

Syntax : **cmd, x1**, **y1**(msb:lsb), **x2**, **y2**(msb:lsb), **colour**(msb:lsb)

cmd : 72hex**, r** ascii

x1 : top left horizontal start position of rectangle.

y1(msb:lsb) **:** top left vertical start position of rectangle. 2 bytes.

x2 : bottom right horizontal end position.

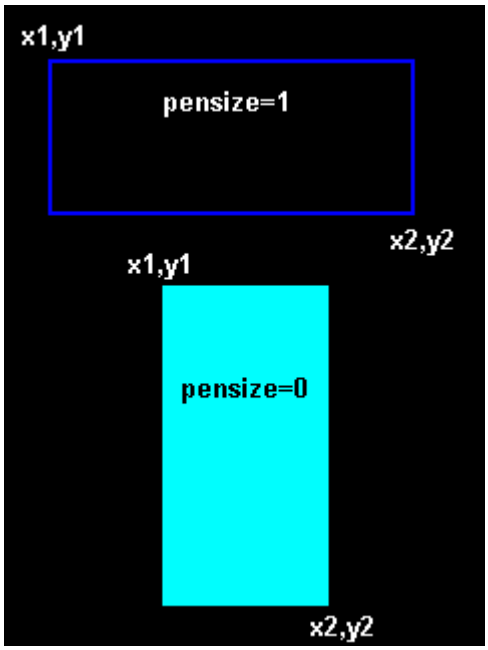
y2(msb:lsb) **:** bottom right vertical end position. 2 bytes.

colour(msb:lsb) **:** 2 byte rectangle colour value

Description : This command will draw a rectangle of specified area on the screen. **x1, y1** refers to the top left corner of the area and **x2, y2** refers to the bottom right hand corner of the rectangle on the screen. If colour is chosen to be that of the background then the effect will be erasure. If Pen Size value was previously set to 0 rectangle will be solid, otherwise wire frame if value was 1.

Example : **70**hex**, 00**hex**, 00**hex**, 00**hex**, 10**hex**, 00**hex**, 10**hex**, 00**hex**, 1F**hex

Draws a RED (001Fhex) rectangle that has its top left corner at x1=0, y1=0 and its bottom right corner at x2=16, y2=16.



2.2.19 Place String of Ascii Text (unformatted) (S)

Syntax : **cmd, x**, **y**(msb:lsb), **font**, **colour**(msb:lsb), **width, height, char1, .. , charN, terminator**

cmd : 53hex**, S**ascii

x : the horizontal start position of string (in pixels).

y(msb:lsb) **:** the vertical start position of string (in pixels).

font : 0 = 5x7, 1 = 8x8, 2 = 8x12, 3 = 12x16 font. This has precedence over the Font command but does not effect the previous font selection.

colour(msb:lsb) **:** 2 byte colour value of the string.

width : horizontal size of the string characters, n x normal size

height : vertical size of the string characters, m x normal size

char1..charN : string of ASCII characters (max 512 characters)

terminator : the string must be terminated with **00**hex

Description : This command allows the display of a string of bitmapped (unformatted) ASCII characters. 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**hex. The sizes of the characters are determined by the **width** and **height** parameters. If the length of the string is longer than the maximum number of characters per line, then a wrap around will occur on to the next line. Maximum string length is **512 bytes**.

2.2.20 Place string of Ascii Text (formatted) (s)

Syntax : **cmd, column**, **row**, **font**, **colour**(msb:lsb), **char1,.., charN, terminator**

cmd : 73hex**, s**ascii

column : horizontal start position of string:
0 - 40 for 5x7, **0 - 30** for 8x8 and 8x12, **0 - 20** for 12x16 font.

row : vertical start position of string:
0 - 15 for 5x7 and 8x8 font, **0 – 9** for 8x12 font.

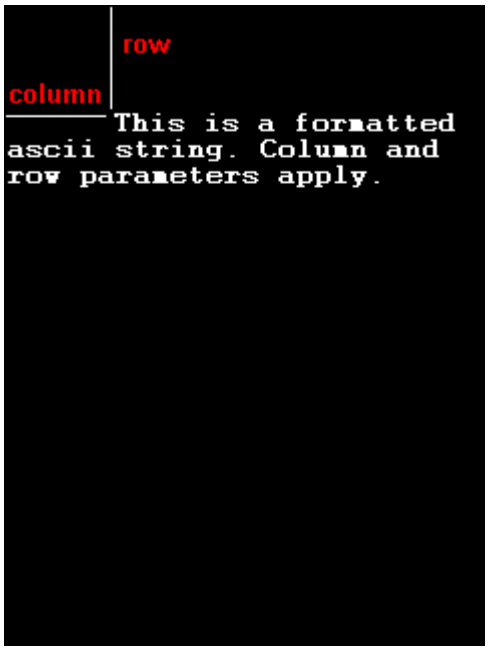
font : 0 = 5x7 font, 1 = 8x8 font, 2 = 8x12 font. This has precedence over the Font command.

colour(msb:lsb) **:** 2 byte colour value of the string.

char1..charN : string of ASCII characters (max 512 characters)

terminator : the string must be terminated with **00**hex

Description : This command allows the display of a string of ASCII characters. The horizontal start position of the string is specified by **column** and the vertical position is specified by **row**. The string must be **terminated** with **00**hex.
If the length of the string is longer than the maximum number of characters per line, then a wrap around will occur on to the next line. Maximum string length is **256 bytes**.



2.2.21 Place Text Character (formatted) (T)

Syntax : cmd, char, column, row, **colour**(msb:lsb)

cmd : 54hex**, T**ascii

char : inbuilt standard ASCII character, **32**dec to **127**dec (**20**hex to **7F**hex)

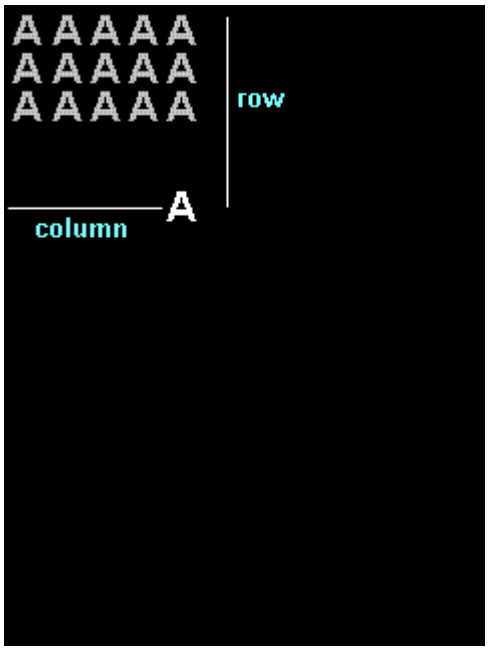
column : horizontal position of character, see range below:
0 - 25 for 5x7 font, **0 - 19** for 8x8 and 8x12 font.

row : vertical position of character:
0 - 15 for 5x7 and 8x8 font, **0 – 9** for 8x12 font.

colour(msb:lsb) **:** 2 byte colour value of the character.

Description : This command will place a coloured ASCII character (from the ASCII chart) on the screen at a location specified by **(column, row)**. The position of the character on the screen is determined by the predefined horizontal and vertical positions available, namely 0 to 25 columns by 0 to 15 rows.

Example : 54hex**, 41**hex**, 00**hex**, 00**hex**, FF**hex, **FF**hex
Place character 'A' (**41**hex) at column = 0, row = 0, colour = white (65,535).



2.2.22 Place text Character (unformatted) (t)

Syntax : cmd, char, **x**, **y**(msb:lsb), **colour**(msb:lsb), **width**, **height**

cmd : 74hex**, t**ascii

char : inbuilt standard ASCII character, **32**dec to **127**dec (**20**hex to **7F**hex)

x : the horizontal position of character (in pixel units).

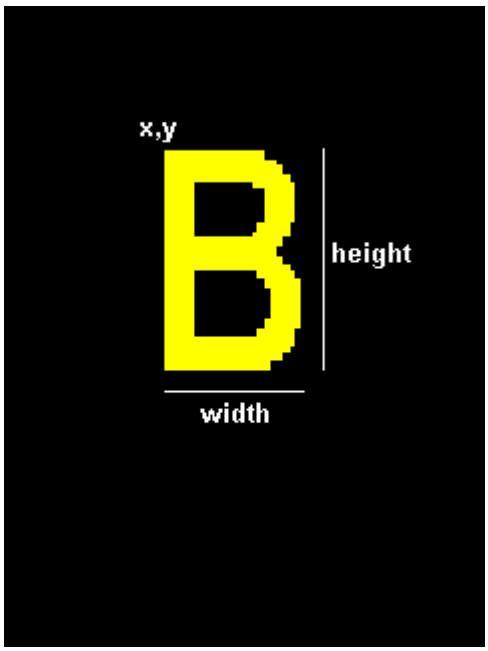
y(msb:lsb) **:** the vertical position of character (in pixel units). 2 bytes.

colour(msb:lsb) **:** 2 byte colour value of the character. 2 bytes.

width : horizontal size of the character, n x normal size

height : vertical size of the character, m x normal size

Description : This command will place a coloured built in ASCII character anywhere on the screen at a location specified by **(x, y)**. Unlike the '**T**' 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 '**Font Size**' command.



2.2.23 LCD DisplaY Control Functions (Y)

Syntax : cmd, mode, value

cmd : 59hex**, Y**ascii

mode : = **00**hex : **BACKLIGHT CONTROL**.
value = **00**hex: Backlight OFF
= **01**hex: Backlight ON

mode : = **01**hex : **DISPLAY ON/OFF**.
value = **00**hex: Display OFF
= **01**hex: Display ON

mode : = **02**hex : **LCD CONTRAST**.
value = **XX**hex: has no effect. This feature is not implemented at this stage.

mode : = **03**hex : **LCD POWER-UP/POWER-DOWN**.
value = **00**hex: LCD Power-Down
= **01**hex: LCD Power-Up

Note: It is important that the **µLCD** be issued with the Power-Down command before switching off the power. This command switches off the internal voltage boosters and current amplifiers and they need to be turned off before main power is removed. If the power is removed without issuing this command, the LCD display maybe damaged (over a period of time). This command also turns off the display. This command need not only be issued to shutdown but can be issued to conserve power by turning off the display and the backlight.
The Power-Up command does not need to be executed when applying power. If a Power-Down command has been issued and Power is not switched off, the Power-Up command can be sent to Power the display back up again.

2.2.24 Version/Device Info Request (V)

Syntax : cmd, output

Response : device_type, hardware_rev, firmware_rev, horizontal_res, vertical_res

cmd : **56**hex**, V**ascii

output : **00**hex : outputs the version and device info to the serial port only.
01hex : outputs the version and device info to the serial port as well as to the screen.

device_type : this response indicates the device type.
00hex = micro-OLED.
01hex = micro-LCD.
02hex = micro-VGA.

hardware_rev : this response indicates the device hardware version.

firmware_rev : this response indicates the device firmware version.

horizontal_res : this response indicates the horizontal resolution of the display.
22hex : 220 pixels
28hex : 128 pixels
32hex : 320 pixels
60hex : 160 pixels
64hex : 64 pixels
76hex : 176 pixels
96hex : 96 pixels

vertical_res : this response indicates the vertical resolution of the display. See horizontal_res above for resolution options.

Description : This command requests all the necessary information from the module about its characteristics and capability.

2.3 Display Specific Command Set

The following commands are specific to the µLCD-320-PMD2 module and may not be available in other 4D intelligent display systems. These are detailed in this section.

Display Specific Command Set	Live	Object	Memory
($P) Set Floating Pointer Parameters	√		
($M) Move Floating Pointer	√		
($C) Send Floating Pointer Coordinates	√		
($V) Vsync Lock	√		

2.3.1 Set Floating Pointer Parameters ($P)

Syntax : spCmd, cmd, mode, appearance, size, colour(msb:lsb)

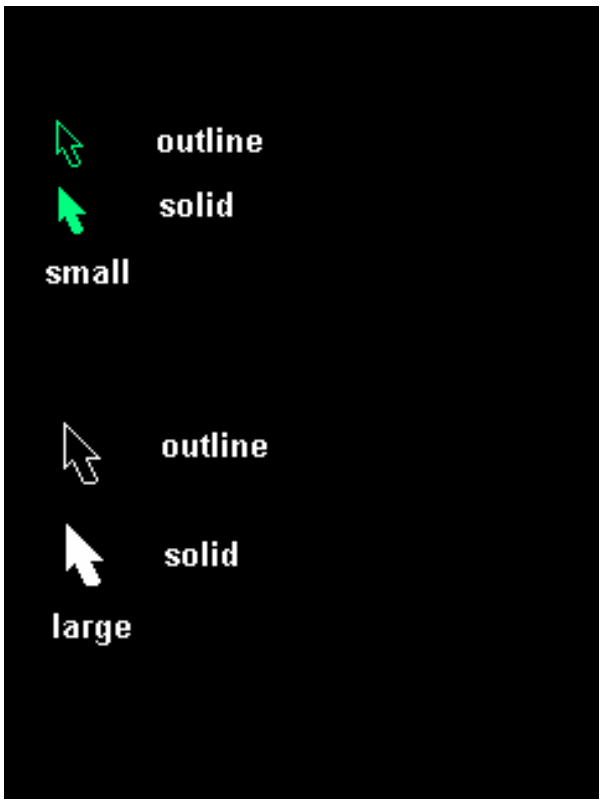
spCmd : 24hex**, $**ascii

cmd : 50hex**, P**ascii

mode :	**00**hex : hide floating pointer. **01**hex : show floating pointer.
appearance :	**00**hex : outline. **01**hex : solid.
size :	**00**hex : small 11x19 pixels. **01**hex : large 15x25 pixels.

colour(msb:lsb) **:** 2 byte colour value of the floating pointer.

Description : This command sets up the required parameters for the built in floating pointer. A floating pointer allows easy implementation of a mouse type of an application.



2.3.2 Move Floating Pointer ($M)

Syntax : spCmd, cmd, x, **y**(msb:lsb)

spCmd : 24hex**, $**ascii

cmd : 4Dhex**, M**ascii

x : the horizontal screen position of pointer destination.

y(msb:lsb) **:** the vertical screen position of pointer destination. 2 bytes.

Description : This command moves the floating pointer to a desired location on the screen. The user must make sure the pointer parameters are first initialised and the pointer made visible using the "**Set Floating Pointer Parameters**" command.

As the pointer is moved, a set of complex internal operations are performed which are all transparent to the user. First, the area of the screen the pointer is to be moved is copied into an internal buffer. This buffer will be written back to the screen next time the pointer is made to move which will restore the original portion of the screen before the pointer was displayed. The module then waits for the next available Vsync (Vertical Sync) and displays the pointer at the given coordinates. At the same time the previously copied portion of the screen is written back to the screen which has the effect of restoring the screen as well as erasing the pointer. If the module did not synchronise displaying the pointer with the Vsync signal the effect would be of a blinking and torn pointer as it moves across the screen. The result of the above set of events gives the appearance of a smooth floating pointer that seems to glide over objects on the screen.

Note : When the pointer is only moved around the screen not much attention to detail needs to be paid by the user. However, extra care must be taken along with some experimentation when, objects underneath the pointer requires dynamic updates such as a text button that needs to be simulated for button presses. Although the module internally takes care of screen copies and restores the user must experiment when to hide/show the pointer and when to change the object under the pointer to achieve a smooth operation.

2.3.3 Send Floating Pointer Coordinates ($C)

Syntax : spCmd, cmd

spCmd : 24hex**, $**ascii

cmd : 43hex**, C**ascii

response : x, y(msb:lsb) (3 bytes of uLCD response)
The module will respond with 1 byte of x followed by 2 bytes of y co-ordinate information of the pointer position via its serial link.

Description : This command will inform the host of the **x,y** coordinates of the current location of the floating pointer via its serial link.

2.3.4 Vsync Lock ($V)

Syntax : spCmd, cmd

spCmd : 24hex**, $**ascii

cmd : 56hex**, V**ascii

response : 06hex (ACK)
The uLCD will respond with the ACK byte once it encounters the next internal Vertical Sync signal.

Description : This command allows the host to lock onto the start of the next video frame.
This is a very useful command when moving/animating objects on the screen where the object needs to be displayed and then erased so it can be redisplayed at a different location to give the effect of movement. Unless the host locks onto the start of frame, undesired blinking and tearing of the displayed object will occur. When the host sends this command it should wait for an 'ACK' from the µLCD and once the ACK is received the host can then start issuing display commands. Use this command for dynamically displayed objects such as a bouncing ball, etc. For static objects that don't move or that do not need continuos display updates, this command is not necessary. Example of a moving object is given below:

```
while() {
      send_cmd_uLCD("$V");          // send the Vsync Lock command
       wait_uLCD_ACK();             //  µLCD will respond at start of frame
       send_cmd_uLCD('D', 1, x, y, BLACK);       // erase object at old loc.
       send_cmd_uLCD ('D', 1, ++x, ++y, RED);   // display at new loc.
}  // loop around while loop
```

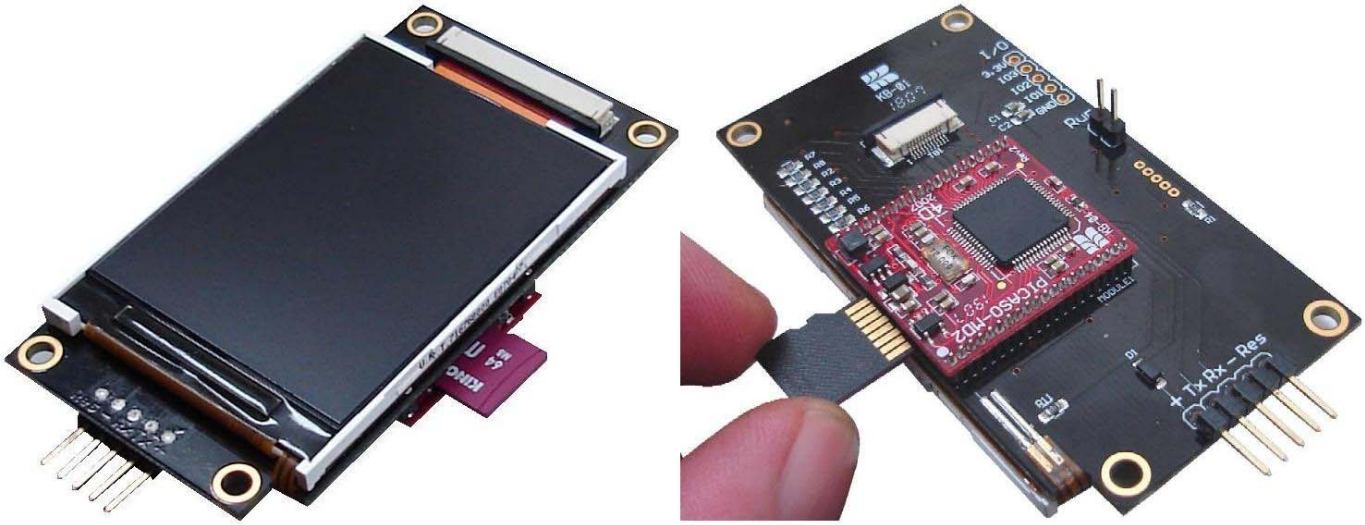
Note: A word of caution: If the serial link is slow, that is, less than 115Kbaud, the response from the µLCD will be too slow for this command to be effective. Try and make the serial link as fast as possible.

2.4 Extended Command Set

The following commands are related to the µLCD-320-PMD2 extended command set and they are described in this section. The µLCD-320-PMD2 is designed around the **PICASO-MD2** module (can be seen in the images below) which has an integrated micro-SD (**µSD**) memory card adaptor. The PICASO-MD2 module can accept memory cards of any size from 64Mb up to 2Gig for storing of text, images, icons, animations, movie clips and all other graphics objects. To utilise the Extended Command set, a **µSD** memory card must be inserted into the module since all of these commands are based around the memory card.

You will find references being made to "**Objects**" throughout this section. An object can be simply defined as those commands that reside inside the memory card (programmed/downloaded previously) and can be displayed on the screen by the **"Display Object from Memory Card"** command. The idea of programming objects into the memory card is so that they can be automatically replayed back like a slide show without any host processor intervention.

There are also some commands that can only reside inside the card and must be executed from there. These commands will return a NAK if executed live from the serial link.



Extended Command Set	Live	Object	Memory
(@i) initialise uSD Memory Card	√		
(@R) Read Sector	√		
(@W) Write Sector	√		
(@r) read Byte	√		
(@w) write Byte	√		
(@A) Set Address	√		
(@C) Copy Screen to Memory Card	√		
(@I) Display Image/Icon from Memory Card	√	√	√
(@O) Display Object from Memory Card	√		
(@P) Run Program from Memory Card	√		
(07hex) Delay (in milliseconds)			√
(08hex) Set Counter			√
(09hex) Decrement Counter			√
(0Ahex) Jump to Address if Counter not Zero			√
(0Bhex) Jump to Address			√
(0Chex) Exit Program from Memory Card	√		√

NOTES:

Live : **Those commands that can be sent via the serial link and executed by the uLCD module.**

Object : **Those commands that can be recalled from the memory card at any time by the host and displayed on the screen using the "Display Object from Memory Card" command.**

Memory: **Those commands that can reside and be executed from inside the memory card.**

2.4.1 initialise Memory Card (@i)

Syntax : extCmd, cmd

extCmd : 40hex**,** @ascii

cmd : 69hex**, i** ascii

Description : This command initialises the **µSD** memory card. The memory card is always initialised 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 initialise the card.

2.4.2 Read Sector Data from Memory Card (@R)

Syntax : **extCmd, cmd, SectorAddress**(hi:mid:lo)

extCmd : 40hex**,** @ascii

cmd : 52hex**, R**ascii

SectorAddress(hi:mid:lo)**:** A 3 byte sector address. Sector Address 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.

Description : This command provides a means of reading data back from the memory card in lengths of 512 bytes. It maybe useful in validating the data that was stored previously using the Write Sector command. Once this command is sent, the µLCD will return 512 bytes of data relating to that particular sector.

2.4.3 Write Sector Data to Memory Card (@W)

Syntax : **extCmd, cmd, SectorAddress**(hi:mid:lo), **data**(1)**, .. , data**(512)

extCmd : 40hex**,** @ascii

cmd : 57hex**, W**ascii

SectorAddress(hi:mid:lo)**:** A 3 byte sector address. Sector Address 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.

data(1)**, .. , data**(512)**:** 512 bytes of sector data. The data length must be 512 bytes long. Unused bytes must be padded even if not all are used.

Description : This command allows downloading of objects such as images and other commands for storage that can be retrieved and used later on. It can also be used as general purpose storage for user specific data. Downloads must always be limited to 512 bytes in length. For large objects 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 into the card. 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 00hex or FFhex (it can be anything).

If only few bytes of data are to be written then the **Write Byte** command can be used.

Once this command message is sent, the µLCD will take a few milliseconds to write the data into its memory card and at the end of which it will reply back with an **ACK**(06hex) if the write cycle was successful. If there was a problem in writing the data to the card a **NAK**(15hex) will be sent back without any write attempts.

Only **data**(1) to **data**(512) are stored in the card. Other bytes in the command message such as Sector Address are not stored.

2.4.4 read Byte Data from Memory Card (@r)

Syntax : extCmd, cmd

extCmd : 40hex**,** @ascii

cmd : 72hex**, r** ascii

Description : This command provides a means of reading a single byte of data back from the memory card. Before this command can be used the card memory address location must be set using the **Set Memory Address** command. Once this command is sent, the µLCD 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 address location.

2.4.5 write Byte Data to Memory Card (@w)

Syntax : extCmd, cmd, data

extCmd : 40hex**,** @ascii

cmd : 77hex**, w** ascii

data : 1 byte of memory card data.

Description : This command allows writing single bytes of data to the memory card. This is useful for writing small chunks of data relating to graphics objects or user application specific data for general purpose storage. For large data blocks it is more efficient to use the **Write Sector Data** command described in the previous section.

Before this command can be used the card memory address location must be set using the **Set Memory Address** command. Once this command is sent, the µLCD will write 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 address location.

Only the **data** byte is stored in the card. Other bytes in the command message are not stored.

2.4.6 Set Memory Address (@A)

Syntax : **extCmd, cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)

extCmd : 40hex**,** @ascii

cmd : 41hex**, A**ascii

Address(Umsb:Ulsb:Lmsb:Llsb)**:** A 4 byte memory card address for byte wise access.

Description : This command sets the card memory Address pointer for byte wise reads and writes. After a byte read or write the Address pointer is automatically incremented internally to the next Address location.

2.4.7 Copy Screen to Memory Card (@C)

Syntax : extCmd, cmd, x, y(msb:lsb)**, width, height**(msb:lsb)**,**
SectorAddress(hi:mid:lo)

extCmd : 40hex**,** @ascii

cmd : 43hex**, C**ascii

x : Screen horizontal start position (top left corner)

y(msb:lsb) **:** Screen vertical start position (top left corner). 2 bytes.

width : horizontal size of the screen area to be copied

height(msb:lsb) **:** vertical size of the screen area to be copied. 2 bytes.

SectorAddress(hi:mid:lo)**:** A 3 byte sector address where the copied screen area is to be stored.

Description :
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. This is similar the **Block Copy and 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.
This is a very powerful feature for animating objects, smooth scrolling, or implementing a windowing system.

2.4.8 Display Image/Icon from Memory Card (@I)

Syntax : extCmd, cmd, x, y(msb:lsb)**, width, height**(msb:lsb)**, colourMode, SectorAddress**(hi:mid:lo)

extCmd : 40hex**,** @ascii

cmd : 49hex**, I**ascii

x : Screen horizontal start position (top left corner)

y(msb:lsb) **:** Screen vertical start position (top left corner). 2 bytes.

width : horizontal size of the Image/Icon

height(msb:lsb) **:** vertical size of the Image/Icon. 2 bytes.

colourMode : 8dec = 256 colour mode, 8bits/1byte per pixel
16dec = 65K colour mode, 16bits/2bytes per pixel

SectorAddress(hi:mid:lo)**:** A 3 byte memory card sector address of a previously stored Image or an Icon that is about to be displayed.

Description : This command displays a bitmap image or an icon on to the screen that has been previously stored at a particular sector address in the memory card. 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.

If the previously stored image was in 8 bit colour format (1 byte per pixel) or 16 bits (2 bytes per pixel) then this must be specified in the **colourMode** byte parameter. Do not store an image/icon in one colour format then display it in another colour format, this will result in a corrupted image display.

Notes:

- The **Copy Screen 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.

2.4.9 Display Object from Memory Card(@O)

Syntax : extCmd, cmd, Address(Umsb:Ulsb:Lmsb:Llsb)

extCmd : 40hex**,** @ascii

cmd : 4Fhex**, O**ascii

Address(Umsb:Ulsb:Lmsb:Llsb)**:** A 4 byte (32 bit) memory address of a previously stored Object that is about to be displayed.

Description: 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. The user must make sure the 32 bit address of each stored command/object is known before using this feature.
For example, a series of images can be stored as icons and later displayed as the application requires them. The table at the end of this section lists all of the commands that can be stored as objects within the memory card.

2.4.10 Run Program from Memory Card (@P)

Syntax : extCmd, cmd, Address(Umsb:Ulsb:Lmsb:Llsb)

extCmd : 40hex**,** @ascii

cmd : 50hex**, P**ascii

Address(Umsb:Ulsb:Lmsb:Llsb)**:** A 4 byte memory card address for the internal command execution.

Description : The Run command forces the 32bit internal memory pointer to jump to the specified address and automatically start executing commands, from the memory card without any further interaction by the host processor. It will sequentially execute any valid memory related commands and display objects until it gets to the end of the memory. It is advisable to have the **Exit Program** or the **Jump to Address** commands at the end of the user composed program so that the pointer does not run off so to speak.

2.4.11 Delay (07hex) (memory card command only)

Syntax : cmd, value(msb:lsb)

cmd : 07hex

value(msb:lsb) **:** A 2 byte delay value in milliseconds. Maximum value of 65,535 milliseconds or 65.5 seconds.

Description : When objects from the memory card such as images are displayed sequentially, a delay can be inserted between subsequent objects. A delay basically has the same effect as a NOP (No Operation) which can be used to determine how long the object stays on the screen before the next object is displayed.

2.4.12 Set Counter (08hex) (memory card command only)

Syntax : cmd, value

cmd : 08hex

value : A 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.

Description : A 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.

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. The following demonstrates how this maybe used:

Address (dec)	Command
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)

Note : 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 offcourse be the series of hex codes.

2.4.13 Decrement Counter (09hex) (memory card command only)

Syntax : cmd, value

cmd : 08hex

Description : Decrements the counter. See detailed description on how this command can be used effectively in the **Set Counter** command section.

2.4.14 Jump to Address If Counter Not Zero (0Ahex)
(memory card command only)

Syntax : **cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)

cmd : 0Ahex

Address(Umsb:Ulsb:Lmsb:Llsb)**:** A 4 byte (32 bit) memory jump address if the counter is not zero.

Description : 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 command. Please see detailed description on how this command can be used effectively in the **Set Counter** command section.

2.4.15 Jump to Address (0Bhex) (memory card command only)

Syntax : **cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)

cmd : 0Bhex

Address(Umsb:Ulsb:Lmsb:Llsb)**:** A 4 byte (32 bit) memory jump address.

Description : This command will force the internal 32 bit program memory pointer to jump unconditionally to the specified address and start executing commands from there.

2.4.16 Exit Program from Memory Card (0Chex)

Syntax : cmd

cmd : 0Chex

Description : 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 stop. It can also be sent via the serial port while the program is running and commands are being executed from the memory card.

Summary of Commands Executable from Memory Card		
Command	**Object**	**Memory**
(**B**) Set **B**ackground Colour	√	√
(**b**) Place Text **b**utton	√	√
(**C**) Draw **C**ircle	√	√
(**e**) Draw **e**llipse	√	√
(**E**) **E**rase Screen	√	√
(**F**) **F**ont Size	√	√
(**G**) Draw Trian**G**le	√	√
(**L**) Draw **L**ine	√	√
(**O**) **O**paque or Transparent Text	√	√
(**p**) Set **p**en Size	√	√
(**r**) Draw **r**ectangle	√	√
(**S**) Place **S**tring of ASCII Text (unformatted)	√	√
(**s**) Place **s**tring of ASCII Text (formatted)	√	√
(**T**) Place **T**ext Character (formatted)	√	√
(**t**) Place **t**ext Character (unformatted)	√	√
(**Y**) LCD Displa**Y** Control functions		√
(**@I**) Display **I**mage/**I**con from Memory Card	√	√
(**07**hex) Delay (in milliseconds)		√
(**08**hex) Set Counter		√
(**09**hex) Decrement Counter		√
(**0A**hex) Jump to Address if Counter not Zero		√
(**0B**hex) Jump to Address		√
(**0C**hex) Exit Program from Memory Card		√

NOTES:

Object : Those commands that can be recalled from the memory card at any time by the host and displayed on the screen using the "Display Object from Memory Card" command.

Memory: Those commands that can reside and be executed from inside the memory card.

2.5 Serial Interface (TTL)

The **uLCD-320-PMD2** module communicates with the outside world via its serial link. The serial link is always used as a means to upload a **PmmC** file for a new platform change or to update the module with enhancements. If the module is running the "**Serial Command**" platform, it can be used by a host processor to send serial commands to the module so that characters and graphics can be displayed on the screen. If the module is running the "**4DGL High Level Language**" platform then user application code is downloaded into the PICASO processor via the same serial link. The serial link is also used to upload the onboard micro-SD memory card with graphics objects such as images as well as slide shows with animations and movie clips using the Graphics Composer PC tool.

NOTE!

The RX and the TX signals are at 3.3V levels. If interfacing to a host system running at voltages greater than 3.6V levels, say 5.0Volts, then a 100 to 220 Ohms series resistor must be inserted between the Host Tx and the µLCD-320-PMD2 Rx signals.

Serial Data Format: 8 Bits, No Parity, 1 Stop Bit.

Auto Baud Detect:

As previously mentioned, the µLCD core has an auto-baud detect function which can operate from **2400 baud to 512K baud**. Prior to any graphical formatting and commands being sent to the core, it must first be initialized by sending the ASCII character **'U' (55h)** after power-up. This will allow the core to determine and lock on to the baud rate of the host automatically without needing any further setup.
This must be done every time the core is powered up.

If the host needs to change the baud rate, the µLCD must be powered down and powered back up again. The "U" command cannot be used to change the baud rate during the middle of normal usage.

Serial Timing:

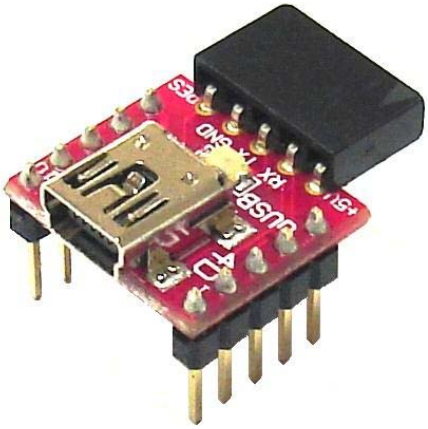
Each µLCD command is made up of a sequence of data bytes. Some commands are a single byte and others are multiple bytes. When a command is sent to the µLCD and the operation is completed, the µLCD will reply back with a single acknowledge byte called the **ACK** (06h). This tells the host that the command was understood and the operation is completed. It will take the µLCD anywhere between 1 to several milliseconds to reply back with an **ACK**, depending on the command and the operation the µLCD has to perform. If the µLCD receives a command that it does not understand it will reply back with a negative acknowledge called the **NAK** (15h).

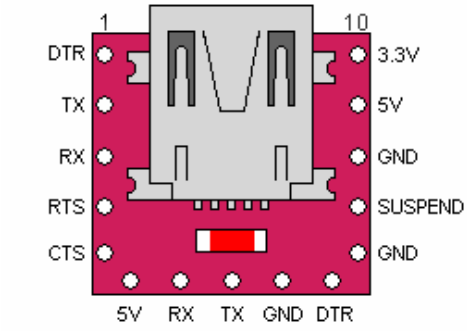
For example, if a command has 5 bytes but only 4 bytes are sent, the command will not be executed and when the next following command bytes are sent the µLCD will reply back with a **NAK** for each and every byte it receives. For correct operation make sure the command bytes are sent in the correct sequence.

Note: No termination character is to be sent at the end of the command sequence. i.e. don't send any CR, or Null, or any other end of command bytes.

2.6 USB Interface

The µLCD can be interfaced to a PC using a standard USB cable and the 4D Systems micro-USB module (uUSB-MB5) as shown below. The micro-USB module (optional extra), simply connects to the µLCD 5 pin header and captures the USB data and converts it into serial TTL data. The microUSB modules and drivers are available from your local 4D distributor. This is an optional extra product and is not included with the µLCD module.





2.7 Personality Module Micro Code (PmmC)

One of the important features of the **PICASO-MD2** module that drives the **uLCD-320-PMD2** is the ability to upload its onboard **PICASO** processor with a micro-Code firmware which allows the module to take on a new personality. This is referred to as Personality Module Micro Code (**PmmC**). The benefits of this are as follows:

- Allows the module to be easily upgraded by the user at any time with PmmC files as further enhancements are made in the future. This allows the user to benefit from those latest features.
- Allows the user to upload a new Operating System to change the device from a serial command driven platform into a high level language platform such as 4DGL. A built-in higher level language such as 4DGL allows user programs to be run directly within the PICASO processor where the graphics operations can be performed much faster than sending the commands serially. This avoids serial bottle necks for those graphics intensive applications as well as allows the user to take complete control of all of the internal resources of the module.

The latest **PmmC** system file for the **uLCD-320-PMD2** can be downloaded from:
www.4dsystems.com.au/downloads/micro-LCD/uLCD-320-PMD2/PmmC/

The latest version of **PmmCLoader.exe** PC software tool can be downloaded from:
www.4dsystems.com.au/downloads/PmmC-Loader/Software/Windows/
and the User Guide can be found here:
www.4dsystems.com.au/downloads/PmmC-Loader/Docs/Pdf/

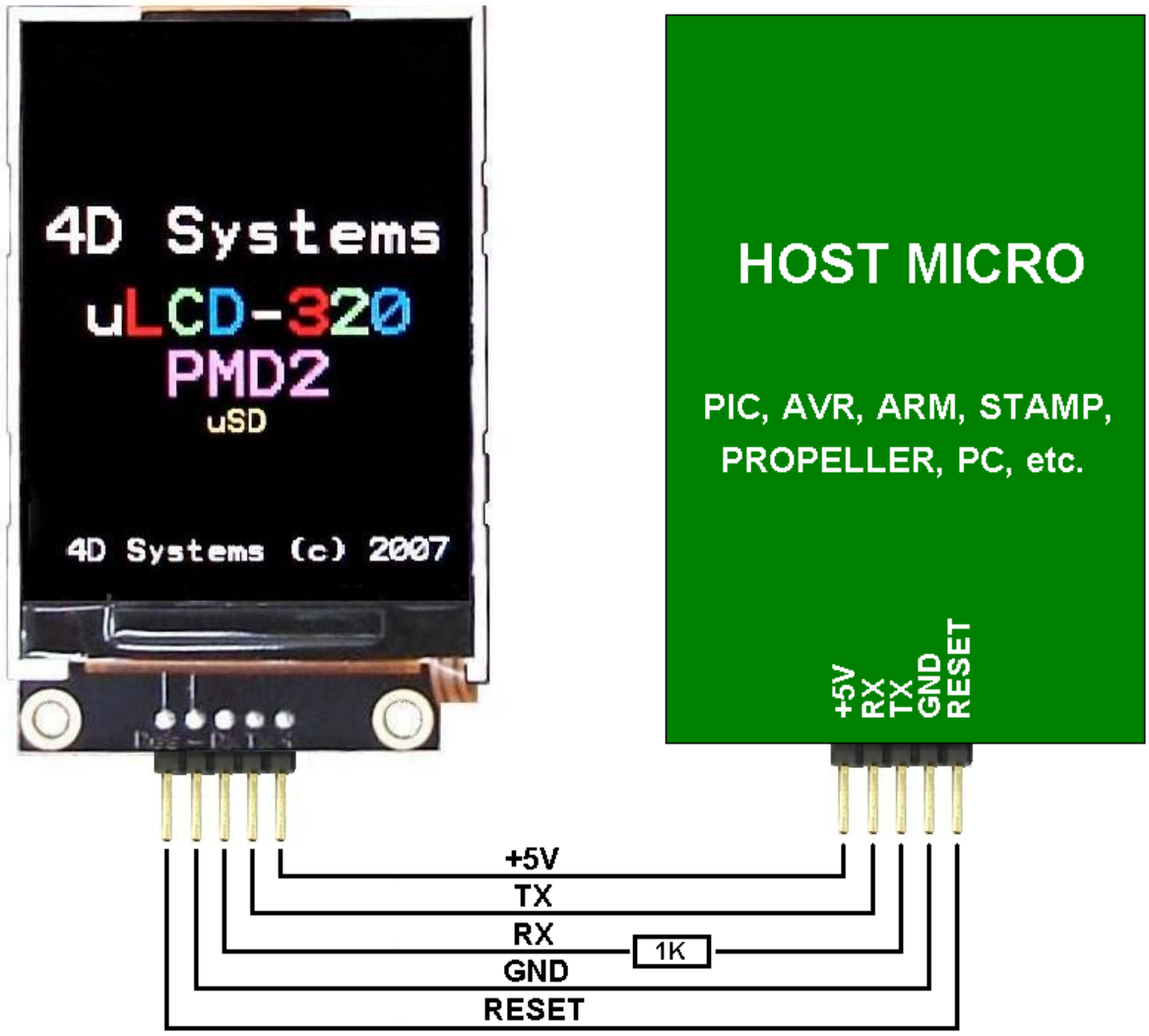
3.Specifications

The µLCD has the following electrical specifications which must be adhered to at all times to prevent damage to the device.

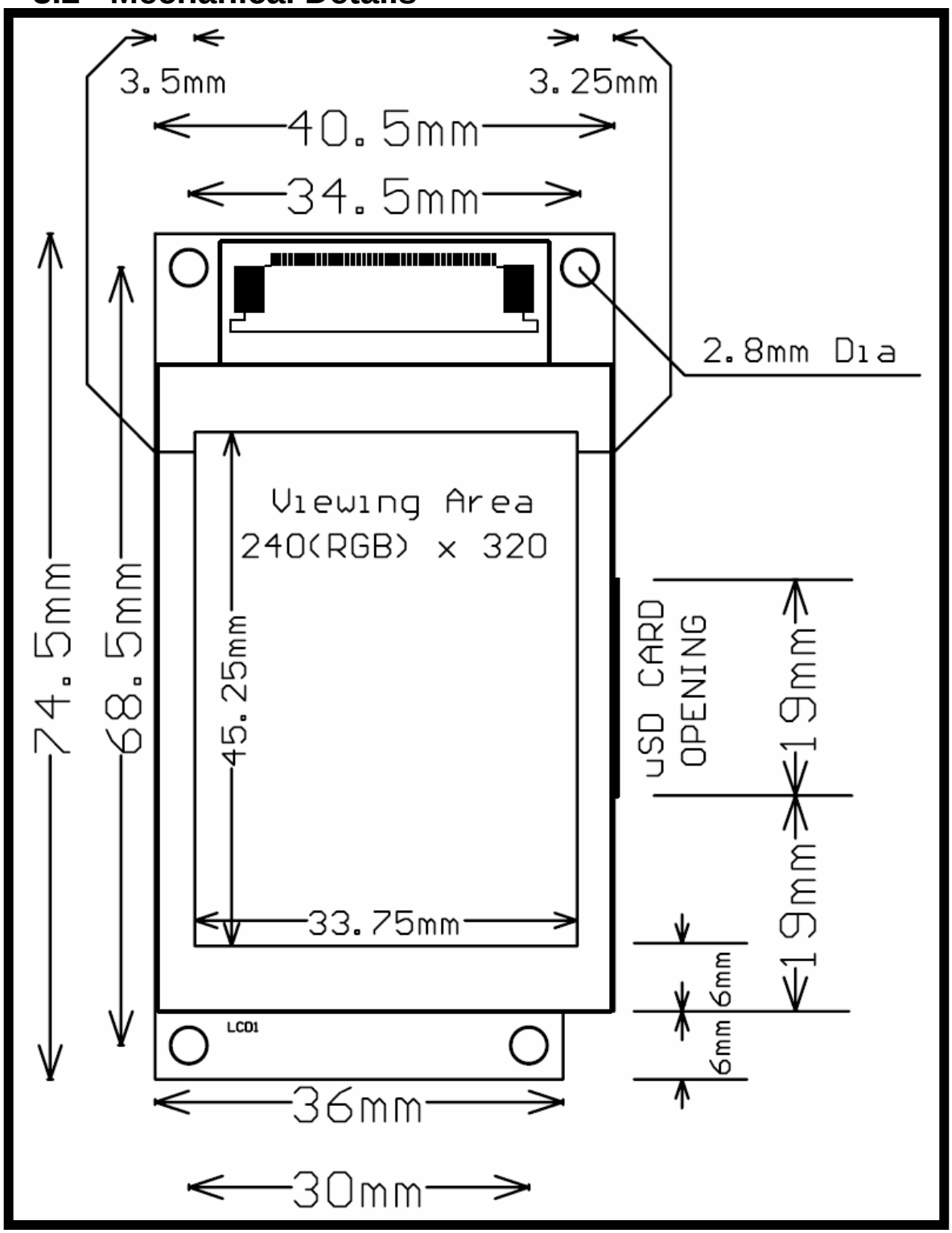
Symbol	Characteristic	Min	Typ	Max	Units
Vdd	Supply voltage	3.6*	5.0	6.0	**V**
I	Current	--	60	100	**mA**
Deg C	Operating temp	0	--	70	**C**
Tpu	Power-up delay	1500		2000	**mS**

***** Due to characteristics of certain micro-SD memory cards, the module may require supply voltages greater than 4.0 Volts.

3.1 Host Interface pin-outs

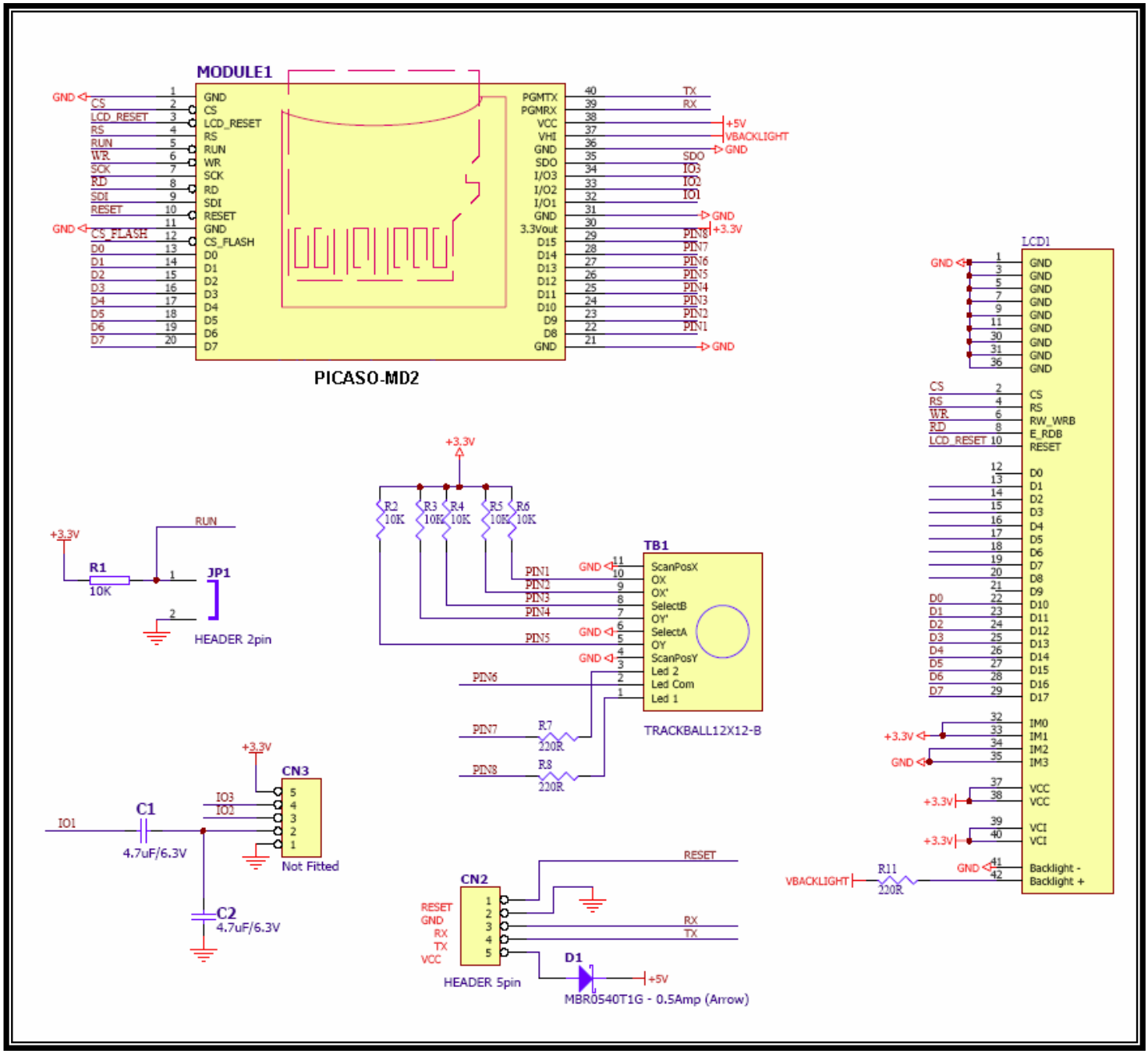


3.2 Mechanical Details



The µLCD module footprint is **74.5**mm x **40.5**mm x **10.3**mm.

3.3 Circuit Diagram



3.4 65,536 Colour Bitmap Organisation

The µLCD 65K colour byte is organised as **5** bits for **Red**(**D11, D12, D13, D14, D15**), **6** bits for **Green**(**D5, D6, D7, D8, D9, D10**) and **5** bits for **Blue**(**D0, D1, D2, D3, D4**). This will give a combination of 32x64x32 = 65,536 colours. Each colour is not limited to 32/64 shades. For example a lighter shade of Red can be obtained by adding a little bit of the Green and a little bit of the Blue. Full Red and full Green will result in Yellow. Some experimentation will be needed to obtain the desired colour.

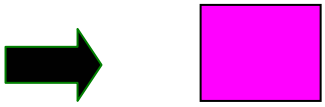
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RED	RED	RED	RED	RED	GREEN	GREEN	GREEN	GREEN	GREEN	GREEN	BLUE	BLUE	BLUE	BLUE	BLUE
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Example: To Obtain the Colour Yellow

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RED	RED	RED	RED	RED	GREEN	GREEN	GREEN	GREEN	GREEN	GREEN	BLUE	BLUE	BLUE	BLUE	BLUE
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

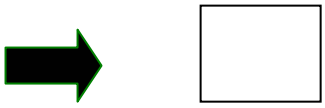
Example: To Obtain the Colour Magenta

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RED	RED	RED	RED	RED	GREEN	GREEN	GREEN	GREEN	GREEN	GREEN	BLUE	BLUE	BLUE	BLUE	BLUE
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1



Example: To Obtain the Colour White

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RED	RED	RED	RED	RED	GREEN	GREEN	GREEN	GREEN	GREEN	GREEN	BLUE	BLUE	BLUE	BLUE	BLUE
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



3.5 256 Colour Bitmap Organisation

The µLCC 256 colour byte is organised as 3 bits for **Red** (D5, D6, D7), 3 bits for **Green** (D2, D3, D4) and 2 bits for **Blue** (D0, D1). This will give a combination of 8x8x4 = 256 colours. Each colour is not limited to 4/8 shades. For example a lighter shade of Red can be obtained by adding a little bit of the Green and a little bit of the Blue. Full Red and full Green will result in Yellow. Some experimentation will be needed to obtain the desired colour.

D7	D6	D5	D4	D3	D2	D1	D0
RED2	RED1	RED0	GREEN2	GREEN1	GREEN0	BLUE2	BLUE1

Example: To Obtain the Colour Yellow

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	1	1	0	0

Example: To Obtain the Colour Magenta

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	0	1	1

Example: To Obtain the Colour White

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	1	1	1	1

3.6 Power-Up Reset

When the µLCD comes out of a power up reset it initialises the Graphics RAM and the internal Display registers. Allow up to 1500ms before attempting to communicate with the µLCD. The power up sequence of events should be as follows:

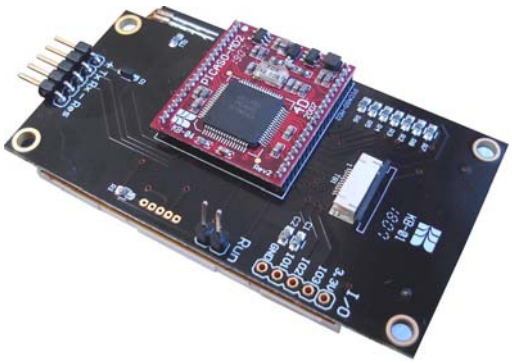
- Allow 1500ms after power-up for µLCD to settle. Do not attempt to communicate with the µLCD during this period. The µLCD may send garbage on its Tx Data line during this period, the host should disable its Rx Data reception.
- Within 100ms of powering up, the host should make sure it has its Tx line pulled HIGH. If the host Tx (µLCD Rx) is LOW or floating after the 100ms period, the µLCD may misinterpret this as the START bit and lock onto some unknown Baud Rate. If the host has a slow wake up time, i.e. less than 100ms, its Tx line maybe floating. This can be easily resolved by adding a pull up resistor on the host Tx line which will ensure the µLCD does not encounter a false START bit. The pull up resistor can be any value within 10K to 100K.
- The host transmits the ASCII 'U' (capital **U**, **55**hex) as the first command so the µLCD can lock onto the host's serial baud rate. This is called "**Auto Bauding**". The µLCD will respond with an 'ACK' (06h). See section 2.5
- The µLCD is now ready to accept screen function commands from the host.
- Note: The µLCD will wait up to 5 seconds with its screen blank for the host to transmit the Auto-Baud character. If the host has not transmitted the Auto Baud character by the end of this period the µLCD will then display its splash screen. If the host has transmitted the Auto Baud character the screen will remain blank. This wait period is for those customer specific applications where the splash screen is undesired.

4. Appendix

4.1 Available Models:

- **uLCD-320-PMD2** (with uSD memory card adaptor)

Please check stock availability with your local supplier.



4.2 Related Products:

- **uUSB-MB5**

  - microUSB module, USB to Serial Bridge
  - Standard USB miniB connector
  - 10 pin header provides the following signals:
    - 5V, 3.3V, GND, Tx, Rx, Suspend,
    - DTR, CTS, RTS, GND
  - 5 Volts supply @ 500mA, 3.3 Volts supply @ 100mA
  - Additional flow control signals, DTR, CTS, RTS
  - Available with an additional 5 pin header for the µLCD interface

- **uSD-64Mb**

  - 64Mb micro-SD Memory Card
  - Extremely small footprint.
  - Measuring only 15mm x 11mm x 0.8mm
  - The uSD-64Mb memory card can be used to store images, animations, text or any graphics objects.

- **PmmC System File for the uLCD-320-PMD2**
  - The latest PmmC system file for the **uLCD-320-PMD2** can be downloaded from: www.4dsystems.com.au/prod.php?id=2

- **PmmC Loader PC Software Tool**
  - Latest version of **PmmC-Loader** software tool can be downloaded from: www.4dsystems.com.au/downloads/PmmC-Loader/Software/Windows/ and the User Guide can be found here: www.4dsystems.com.au/downloads/PmmC-Loader/Docs/Pdf/

- **Software Utility Tools (free download)**
  - Range of PC based software utility tools for Windows
  - Download images/text/animations into the uLCD-320-GMD1 micro-SD memory card.
  - For available software tools user guides please visit the **µLCD** web-page of your local distributor or visit the 4D Systems website www.4dsystems.com.au

4.3 Auto Demo/Slide Show:

The **uLCD-320-PMD2** modules are equipped to accept memory cards. There is a **2 pin jumper** at the back of the unit (on the component side). Upon power-up, if the shunt is inserted and there are preloaded objects in the uSD memory card such as images/text/animations, the **uLCD-320-PMD2** module will automatically play/display these from the memory card. The memory cards are supplied as blank separate products and as such the user will have to upload a slide show composition to the card to benefit from this auto play feature.

For normal usage this jumper must be **removed**.

4.4 Precautions:

- Avoid having to display the same image/object on the screen for lengthy periods of time. This will cause a burn-in which is a common problem with all types display technologies. Blank the screen after a while or dim it very low by adjusting the contrast. This can be achieved via the **"LCD Display Control Functions"** command (section 2.2.23). Better still, implement a screen saver feature.
- Observe the Power-Down procedure (section 2.2.23). The µLCD module automatically takes care of the proper Power-Up sequence.

4.5 Help and Other Information:

- Assistance with latest information and downloads visit the **µLCD** product web-page of your distributor.
- Questions and technical support please email support@4dsystems.com.au
- All related product information can be downloaded from www.4dsystems.com.au/prod.php?id=2