4D LABS

GOLDELOX-SGC Command Set

Software Interface Specification

Document Date: 17th November 2011
Document Revision: 6.0

Note: This manual applies to the GOLDELOX-SGC Revision 17 PmmC files and above.

Table of Contents

1. Host Interface

The GOLDELOX-SGC chip is a slave 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. The protocol is simple and easy to implement.

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

1.1 Command Protocol : Flow Control

The GOLDELOX-SGC is a slave device and all communication and events must be initiated by the host. Each command 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 GOLDELOX-SGC chip receives a command that it does not understand it will reply back with a negative acknowledge called the NAK (15hex). Since a command is only identified by its position in the sequence of data bytes sending incorrect data can result in wildly incorrect operation.

1.2 Serial Set-up : Auto-Baud

The GOLDELOX-SGC 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 initialised 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 (06hex).

☞ **Auto-Bauding must be performed each time the device is powered up or reset.**

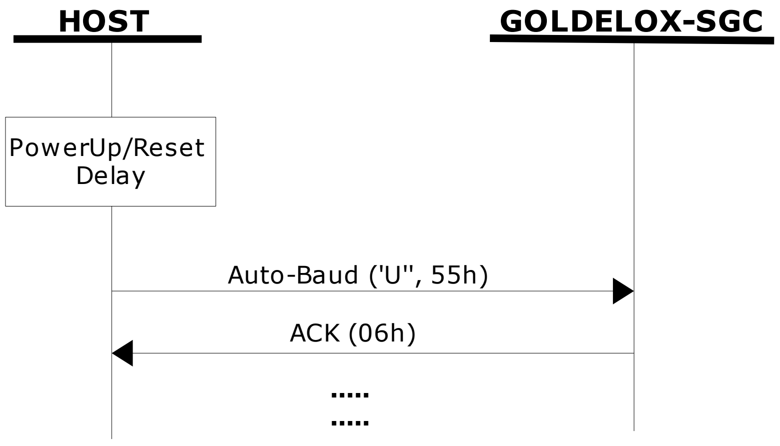
If the host needs to change the baud rate, the GOLDELOX-SGC must be power/reset cycled. The "A**uto-Baud**" command cannot be used to change the baud rate during the middle of normal usage.

1.3 Power-up and Reset

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

- Allow up to 500ms delay after power-up or reset for the module to settle without a uSD/uSDHC card inserted. If a uSD card is inserted the initialisation time of the particular card will need to be added, better quality cards tend to initialise in about 75ms or quicker, lower quality ones can take up to a second. 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.

- The host transmits the Auto-Baud character (capital **U**, **55**hex) as the first command so the device can lock onto the host's baud rate.
- Once the host receives the ACK, the GOLDELOX-SGC is now ready to accept commands from the host.



1.4 Splash Screen on Power Up

The GOLDELOX-SGC will wait up to 5 seconds with its screen blank for the host to transmit the Auto-Baud command ('U', 55hex). If the host has not transmitted the Auto-Baud command by the end of this period the module will display its 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.

1.5 Auto Run Memory Card Script Program

The GOLDELOX-SGC has a feature that will auto run a preloaded script program on power-up. If the SWITCH input (pin 27) on the GOLDELOX-SGC is connected to GND (on power-up) and if there is a script program present in the memory card then the device will auto run the script program. This is a useful feature for those stand alone applications where the device does not require a host controller to play a slide show of images, video clips, etc.

1.6 FAQs About PICASO-SGC

Note: All the Frequently asked Questions about PICASO-SGC and related Display modules are available in the FAQ section of the Support Forum.

2. 4DSL Scripting Language

The complete command set for the GOLDELOX-SGC device is listed in section 3 of this document. The command execution is not only limited to the host sending these via the serial interface. The majority of them can be composed as a script and written into memory card. A 4DSL 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.

4DSL is a Scripting language developed to provide the SGC modules, which are labelled as Slave devices, some degree of independence. The syntax of the commands is simple and easy to use. 4DSL commands can be written on the uSD card at a particular address. The script can be called from a host controller with respect to the script address. Scripts saved at Sector 0 can run automatically with Run jumper shunt installed .

For quick start and slide show scripting FAT Controller can be used. However it doesn't provide a text editor to write a detailed script. 4D Workshop3 IDE or above are set to provide complete text editor to write a detailed 4DSL script. You can also test your script using the IDE while the module is connected to the PC via suitable interface.

4DSL command syntax or keywords are unique while the arguments are mostly the same as normal serial commands. Some of the commands can be run from the PC only which are named as Macros. They can be used for testing/debugging and to copy data to and from the SGC modules to enable field updating and or customisation..

Scripts can be run on a Windows PC from within the Workshop 3 IDE, or from the command prompt, thus they can be embedded within .BAT files to enable ‘simple’ use In the field.

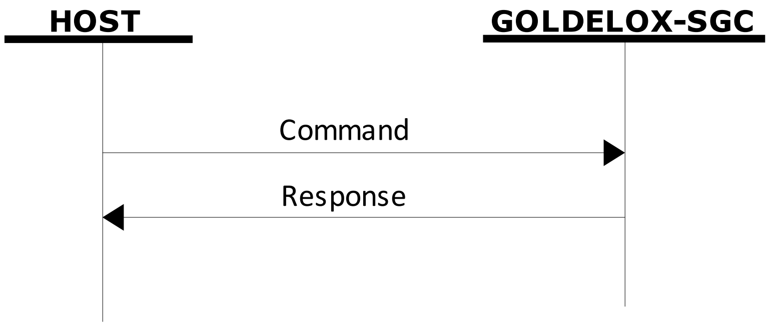
Note: Details of Testing/Debugging a 4DSL Script using the 4D Workshop3 IDE is provided on the ***“4D-Workshop3-IDE-User-Guide-rev3.pdf”*** or above.

Note: The downloaded (4D Workshop3 IDE) setup application will create the required 4D-Workshop3 folders and install all the required files. Note that in-line with current Microsoft philosophy all the 4DSL sample Scripts and demos are located in the ‘All Users\Shared Documents\4D Labs\Scripts’ folder (XP) or ‘Users\Public\Documents\4D Labs\Scripts’ folder (Vista and Windows 7).

Note: The **4DSL Example** gives reference to the 4DSL sample scripts installed with the 4D Workshop3 IDE. Each script can be found in the folders as notified above.

3. Command Set

The command interface between the GOLDELOX-SGC 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 acknowledge or data.



The command set is grouped into following sections:

- General Commands
- Graphics Commands
- Text Commands
- SD/SDHC Memory Card Commands
- 4DSL - Scripting Language Commands

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

☞ **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.**

3.1 General Commands

Summary of Commands in this section:

- AutoBaud – **55hex**
- Version-Device Info Request – **56hex**
- Replace Background Colour – **42hex**
- Clear Screen – **45hex**
- Display Control Functions – **59hex**
- Sleep– **5Ahex**
- Switch-Buttons-Joystick Status - **4Ahex**
- Switch-Buttons-Joystick Wait for Status - **6Ahex**
- Sound – **4Ehex**
- Tune – 6**Ehex**

3.1.1 AutoBaud - 55hex

Command	**cmd**	
4DSL Cmd	**AutoBaud**	
	cmd	**55**(hex) or **U**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte
Description	This must be the very first command sent to the GOLDELOX-SGC after power-up or reset. This will enable the device to lock on to the host baud rate.	
Serial Example	**Command Data:** 55hex Send Autobaud command.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.2 Version-Device Info Request - 56hex

Command	**Cmd, Output**	
4DSL Cmd	**Version(Output)**	
	cmd	**56**(hex) or **V**(ascii) : Command header byte
	Output	**00hex** : 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.
Response	**device_type, hardware_rev, firmware_rev, horizontal_res, vertical_res**	
	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. **22hex** : 220 pixels **28hex** : 128 pixels **32hex** : 320 pixels **60hex** : 160 pixels **64hex** : 64 pixels **76hex** : 176 pixels **96hex** : 96 pixels
Description	This command requests all the necessary information from the device about its characteristics and capability.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.3 Replace Background Colour - 42hex

Command	**cmd, colour**(msb:lsb)	
4DSL Cmd	**ReplaceBackground(colour)**	
	cmd	**42**(hex) or **B**(ascii) : Command header byte
	colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command changes 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.	
Serial Example	**Command Data:** 42hex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.4 Clear Screen - 45hex

Command	**cmd**	
4DSL Cmd	**Clear**	
	cmd	**45**(hex) or **E**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command clears the entire screen using the current background colour	
Serial Example	**Command Data:** 45hex (Clear the screen).	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.5 Display Control Functions - 59hex

Command	**cmd, mode, value**	
4DSL Cmd	**Control(mode, value)**	
	cmd	**59**(hex) or **Y**(ascii) : Command header byte
	mode	**00hex : NA** **01hex : Display ON/OFF** DISPLAY OFF : when value = 00hex DISPLAY ON : when value = 01hex **02hex : Contrast Adjust** CONTRAST RANGE : when value = 00hex to 0Fhex **03hex : Display PowerUp-Shutdown (low power mode)** DISPLAY SHUTDOWN : when value = 00hex DISPLAY POWERUP : when value = 01hex
	value	See mode description above.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command changes some of the display settings such as contrast and low power mode.	
Serial Example	**Command Data:** 59hex, 01hex, 00hex Turn the Display off.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.6 Sleep- 5Ahex

Command	**cmd, mode, delay**	
4DSL Cmd	**Sleep(mode, delay)**	
	cmd	**5A**(hex) or **Z**(ascii) : Command header byte
	mode	**80hex : Turn off uSD/uSDHC(must reinit manually)** **02hex : Wake-up on Joystick** **01hex : Wake-up on Serial**
	delay	N/A - Not used.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Puts GOLDELOX-SGC chip in to low power mode and optionally waits for certain conditions to wake it up. To reduce the current consumption even further “Display Control Functions – 59hex” must also be used to set the display in low power mode .	
Serial Example	**Command Data:** 5Ahex, 02hex, 00hex Sleep until Joystick state changed.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.7 Switch-Buttons-Joystick Status - 4Ahex

Command	**cmd, option**	
4DSL Cmd	**Joystick(option)**	
	cmd	**4A**(hex) or **J**(ascii) : Command header byte
	option	**08hex :** Return Buttons-Joystick Status **0Fhex :** Wait for Buttons-Joystick to be pressed and released **00hex :** Wait until any Buttons-Joystick pressed **01hex :** Wait until SW1 (UP) released. **02hex :** Wait until SW2 (LEFT) released. **03hex :** Wait until SW3 (DOWN) released. **04hex :** Wait until SW4 (RIGHT) released. **05hex :** Wait until SW5 (FIRE) released.
Response	**status**	
	status	**00hex :** No Buttons pressed (or pressed button has been released). **01hex :** SW1 (UP) pressed. **02hex :** SW2 (LEFT) pressed. **03hex :** SW3 (DOWN) pressed. **04hex :** SW4 (RIGHT) pressed. **05hex :** SW5 (FIRE) pressed.
Description	This command returns the status of the Buttons-Joystick in several options.	
Serial Example	**Command Data:** 4Ahex, 01hex Wait until SW1(UP) released.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.8 Wait for Switch-Buttons-Joystick Status - 6Ahex

Command	**cmd, option, waitTime**(msb:lsb)	
4DSL Cmd	**WaitJoystick(option,waitTime)**	
	cmd	**6A**(hex) or **j**(ascii) : Command header byte
	option	**00hex :** Wait until any Buttons-Joystick pressed. **01hex :** Wait until SW1 (UP) released. **02hex :** Wait until SW2 (LEFT) released. **03hex :** Wait until SW3 (DOWN) released. **04hex :** Wait until SW4 (RIGHT) released. **05hex :** Wait until SW5 (FIRE) released.
	waitTime	2 bytes (big endian) define the wait time (in milliseconds).
Response	**status**	
	status	**00hex :** Time-Out (or Button released). **01hex :** SW1 (UP) pressed. **02hex :** SW2 (LEFT) pressed. **03hex :** SW3 (DOWN) pressed. **04hex :** SW4 (RIGHT) pressed. **05hex :** SW5 (FIRE) pressed.
Description	This command asks for the status of the Buttons-Joystick in several options with a wait time.	
Serial Example	**Command Data:** 6Ahex, 01hex, 00hex, 3Chex Wait until 1min timed out or SW1(UP) released.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.9 Sound - 4Ehex

Command	**cmd, note**(msb:lsb)**, duration**(msb:lsb)	
4DSL Cmd	**Sound(note, duration)**	
	cmd	**4E**(hex) or **N**(ascii) : Command header byte
	note	2 bytes (big endian) define the note or frequency of the sound. **0 :** No sound, silence. **1-84 :** 5 octaves piano range + 2 more. **100-20000 :** Frequency in Hz.
	duration	2 bytes (big endian) define the duration of the note (in milliseconds).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will generate a specified note or frequency for a certain duration.	
Serial Example	**Command Data:** 4Ehex, 03hex, E8hex, EAhex, 60hex Play a 1KHz note for 1 min.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.1.10 Tune - 6Ehex

Command	**cmd, length, note1, duration1, note2, duration2, ... noteN, durationN**	
4DSL Cmd	**Tune(length, note1, duration1, note2, duration2, ... noteN, durationN)**	
	cmd	**6E**(hex) or **n**(ascii) : Command header byte
	length	1byte, Number of note/duration pairs to follow: Maximum 64.
	note	2 bytes (big endian) define the note or frequency of the sound. **0 :** No sound, silence. **1-84 :** 5 octaves piano range + 2 more. **100-20000 :** Frequency in Hz.
	duration	2 bytes (big endian) define the duration of the note (in milliseconds).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will generate a sequence of specified note or frequency for a specified duration.	
Serial Example	**Command Data:** 6Ehex, 04hex, 00hex, 20hex, 00hex, 3Chex, 00hex, 25hex, 00hex, 3Chex, 00hex, 20hex, 00hex, 3Chex, 00hex, 40hex, 00hex, 3Chex Play a tune.	
4DSL Example	Refer to the General.4DScript sample script file.	

3.2 Graphics Commands

Summary of Commands in this section:

- Add User Bitmap Character – **41hex**
- Draw Circle – **43hex**
- Draw User Bitmap Character – **44hex**
- Draw Triangle – **47hex**
- Draw Image-Icon – **49hex**
- Set Background colour **– 4Bhex**
- Draw Line – **4Chex**
- Draw Pixel – **50hex**
- Read Pixel – **52hex**
- Screen Copy-Paste – **63hex**
- Draw Polygon – **67hex**
- Replace colour **– 6Bhex**
- Set Pen Size – **70hex**
- Draw Rectangle – **72hex**

3.2.1 Add User Bitmap Character - 41hex

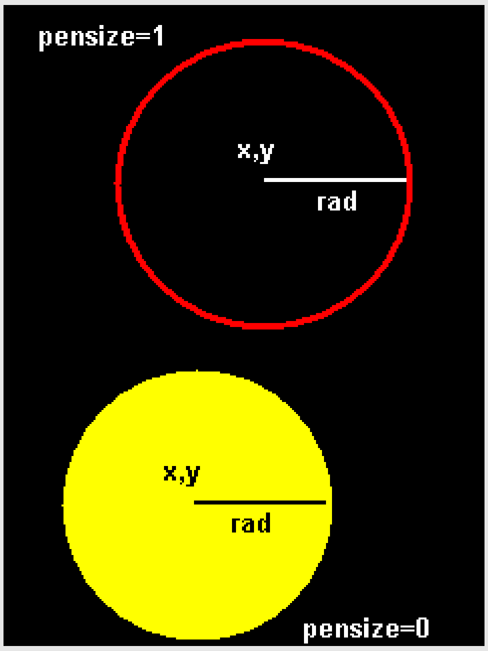
Command	**cmd, char_idx, data1, data2, .. , data8**	
4DSL Cmd	**AdduserBitmap(char_idx, data1, data2, .. , data8)**	
	cmd	**41**(hex) or **A**(ascii) : Command header byte
	char_idx	Bitmap character index to add to memory. Range is 0 to 31 (**00**h to **1F**h), 32 characters of 8x8 format.
	data1..data8	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.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will add a user defined bitmap character into the internal memory.	

b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**		**← Data Bits**
			■	■					**data1 (18**hex**)**
		■			■				**data2 (24**hex**)**
	■					■			**data3 (42**hex**)**
■							■		**data4 (81**hex**)**
■							■		**data5 (81**hex**)**
	■					■			**data6 (42**hex**)**
		■			■				**data7 (24**hex**)**
			■	■					**data8 (18**hex**)**

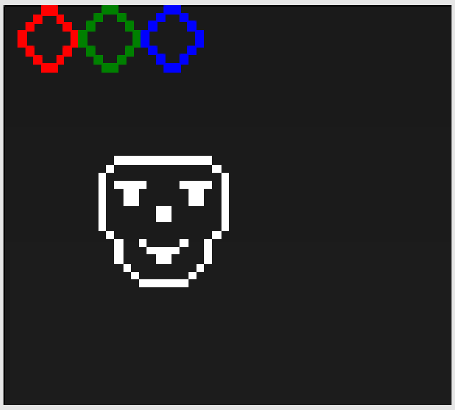
Example of 8x8 User defined bitmap

Serial Example	**Command Data:** 41hex, 01hex, 18hex, 24hex, 42hex, 81hex, 81hex, 42hex, 24hex, 18hex This example adds and saves a user defined 8x8 bitmap as character index 1 into memory.
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.

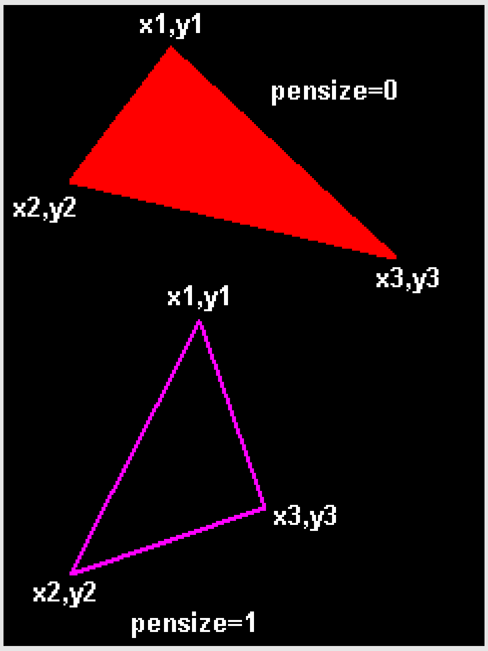
3.2.2 Draw Circle - 43hex

Command	**cmd, x, y, radius, colour**(msb:lsb)	
4DSL Cmd	**Circle(x, y, radius, colour)**	
	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.
	colour	2 bytes define the circle colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured circle centred 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). when Pen Size = 0 : circle is solid when Pen Size = 1 : circle is wire frame 	
Serial Example	**Command Data:** 43hex, 3Fhex, 3Fhex, 22hex, 00hex, 1Fhex Draws a RED circle (**001F**hex) centred at x = **63**dec (**3F**hex) and y = **63**dec (**3F**hex) with a radius of **34**dec (**22**hex).	
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.	

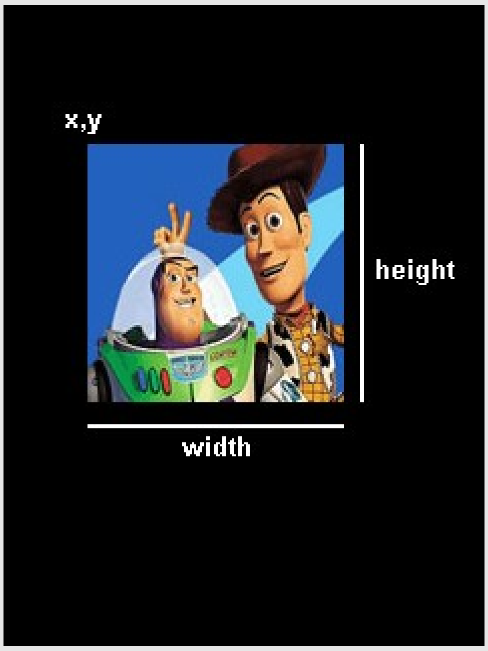
3.2.3 Draw User Bitmap Character - 44hex

Command	**cmd, char_idx, x, y, colour**(msb:lsb)	
4DSL Cmd	**DrawUserBitmap(char_idx, x, y, colour)**	
	cmd	**44**(hex) or **D**(ascii) : Command header byte
	char_idx	Bitmap character index to draw from the previously added bitmap characters into memory. Range is 0 to 31 (**00**h to **1F**h), 32 characters of 8x8 format.
	x	Horizontal display position of the bitmap character.
	y	Vertical display position of the bitmap character.
	colour	2 bytes bitmap colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command draws the previously defined user bitmap character at location **(x, y)** on the screen. User defined bitmaps allow drawing & displaying unlimited graphic patterns quickly & effectively. 	
Serial Examples	**Command Data:** 44hex, 01hex , 00hex, 00hex, F8hex, 00hex (Display 8x8 bitmap character index 1 at x = 0, y = 0, colour = RED). **Command Data:** 44hex, 02hex, 08hex, 00hex, 07hex, E0hex (Display 8x8 bitmap character index 2 at x = 8, y = 0, colour = GREEN). **Command Data:** 44hex, 03hex, 10hex, 08hex, 00hex, 1Fhex (Display 8x8 bitmap character index 3 at x = 16, y = 8, colour = BLUE).	
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.	

3.2.4 Draw Triangle - 47hex

Command	**cmd, x1, y1, x2, y2, x3, y3, colour**(msb:lsb)	
4DSL Cmd	**Triangle(x1, y1 ,x2 , y2, x3, y3, colour)**	
	cmd	**47**(hex) or **G**(ascii) : Command header byte
	x1, y1, x2, y2, x3, y3	3 vertices of the triangle. These must be specified in an anti-clockwise fashion. 1 Byte each.
	colour	2 bytes (big endian) triangle colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command draws a Solid/Wire-Frame 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. when Pen Size = 0 : triangle is solid when Pen Size = 1 : triangle is wire frame  x1,y1 pensize=0 x2,y2 x3,y3 x1,y1 x3,y3 x2,y2 pensize=1	
Serial Example	**Command Data:** 47hex, 25hex , 10hex, 05hex, 30hex, 35hex, 35hex, F8hex, 00hex (Display a RED triangle).	
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.	

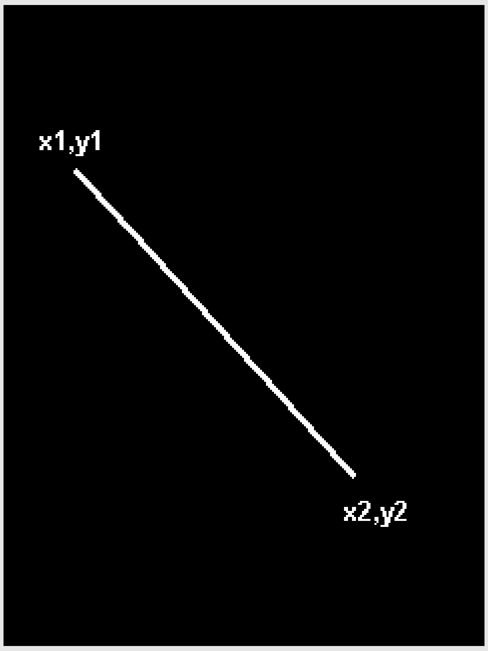
3.2.5 Draw Image-Icon - 49hex

Command	**cmd, x, y, width, height, colourMode, pixel1, .. pixelN**	
4DSL Cmd	**Image(x, y, width, height, colourMode, pixel1, .. pixelN)**	
	cmd	**49**(hex) or **I**(ascii) : Command header byte
	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	**08**(hex) : 256 colour mode, 8bits/1byte per pixel. **10**(hex) : 65K colour mode, 16bits/2bytes per pixel .
	pixel1..pixelN	Image pixel data where N is the total number of pixels. N = width x height (when colourMode = 08hex) N = 2 x width x height (when colourMode = 10hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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. 	
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.	

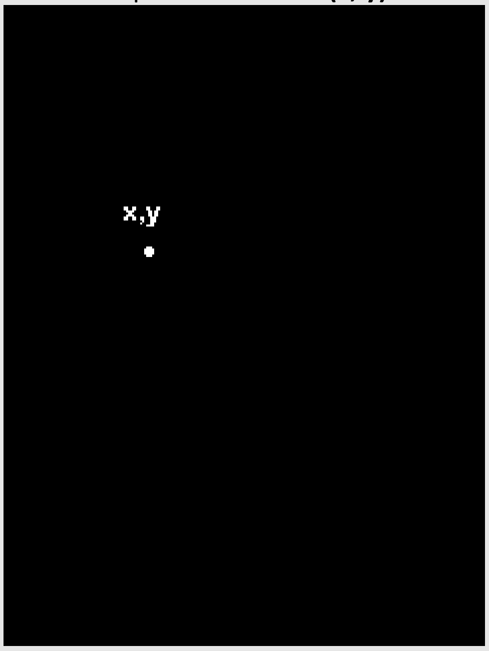
3.2.6 Set Background colour - 4Bhex

Command	**cmd, colour**(msb:lsb)	
4DSL Cmd	**SetBackground(colour)**	
	cmd	**4B**(hex) or **K**(ascii) : Command header byte
	colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command sets the background colour for the next erase and draw(refers to opaque mode text in Set Transparent-Opaque Text – 4Fhex) commands to be sent. Once this command is sent, the background colour will only change when it is rewritten. Nothing on the screen will be affected.	
Serial Example	**Command Data:** 4Bhex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	
4DSL Example	Refer to the General.4DScript sample script file.	

3.2.7 Draw Line – 4Chex

Command	**cmd, x1, y1, x2, y2, colour**(msb:lsb)	
4DSL Cmd	**Line(x1, y1, x2, y2, colour)**	
	cmd	**4C**(hex) or **L**(ascii) : Command header byte
	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.
	colour	2 bytes define the Line colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured line from point **(x1, y1)** to point **(x2, y2)** on the screen. 	
Serial Example	**Command Data:** 4Chex, 00hex, 00hex, 7Fhex, 7Fhex, Ffhex, FFhex Draws a WHITE line (FFFFhex) from (x1 = 00hex, y1 = 00hex) to (x2 = 7Fhex, y2 = 7Fhex).	
4DSL Example	Refer to the GraphicsPt1.4DScript sample script file.	

3.2.8 Draw Pixel - 50hex

Command	**cmd, x, y, colour**(msb:lsb)	
4DSL Cmd	**Pixel(x, y, colour)**	
	cmd	**50**(hex) or **P**(ascii) : Command header byte
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
	colour	2 bytes (16 bits) define the pixel colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured pixel at location **(x, y)** on the screen. 	
Serial Example	**Command Data:** 50hex, 01hex, 0Ahex, FFhex, FFhex Draw a WHITE pixel (FFFFhex) at location (x = 01hex, y = 0Ahex).	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

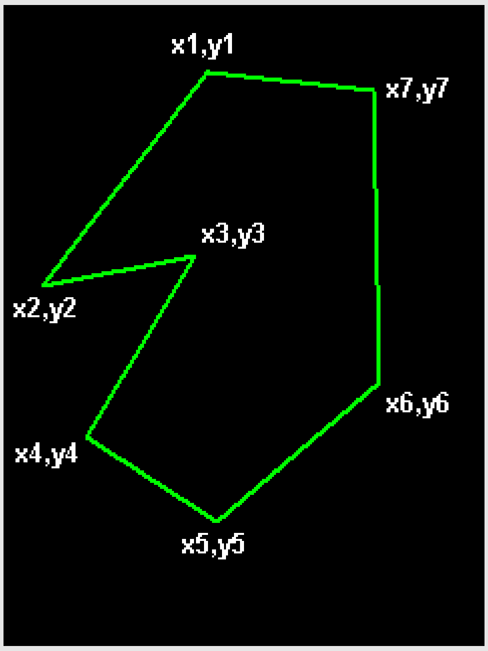
3.2.9 Read Pixel - 52hex

Command	**cmd, x, y**	
4DSL Cmd	**ReadPixel(x, y)**	
	cmd	**52**(hex) or **R**(ascii) : Command header byte
	x	Horizontal position of the pixel.
	y	Vertical position of the pixel.
Response	**colour(**msb:lsb)	
	colour	Returns back 2 bytes (16 bits) pixel colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 (msb is 1st byte) lsb : G2G1G0B4B3B2B1B0 (lsb is 2nd byte)
Description	This command will read the colour 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 colour on the screen and switch the colour of the pointer when it's on top of a light coloured area.	
Serial Example	**Command Data:** 52hex, 01hex, 0Ahex **GOLDELOX-SGC Response:** 00hex, 1Fhex Reads a BLUE pixel (001Fhex) at location (x = 01hex, y = 0Ahex).	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

3.2.10 Screen Copy-Paste - 63hex

Command	**cmd, xs, ys, xd, yd, width, height**	
4DSL Cmd	**ScreenCopyPaste(xs, ys, xd, yd, width, height)**	
	cmd	**63**(hex) or **c**(ascii) : Command header byte
	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).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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). 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.	
Serial Example	**Command Data:** 63hex, 00hex, 00hex, 50hex, 50hex, 28hex, 28hex Copy 40x40 area from 0,0 location to 80,80 location.	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

3.2.11 Draw Polygon - 67hex

Command	**cmd, vertices, x1, y1, .. , xn, yn, colour**(msb:lsb)	
4DSL Cmd	**Polygon(vertices, x1, y1, .. , xn, yn, colour)**	
	cmd	**67**(hex) or **g**(ascii) : Command header byte
	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 Polygon. These can be specified in any fashion. 1 Byte each.
	colour	2 bytes triangle colour value.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
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. 	
Serial Example	**Command Data:** 67hex, 20hex, 05hex, 05hex, 20hex, 40hex, 28hex, 80hex, 10hex Draw a polygon.	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

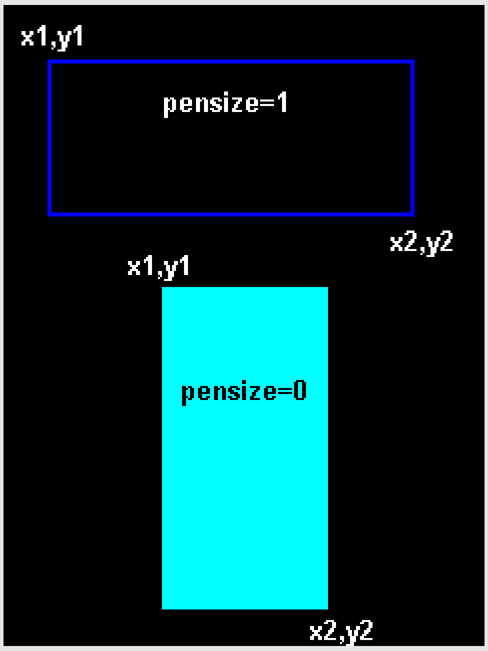
3.2.12 Replace Colour - 6Bhex

Command	**cmd, x1, y1, x2, y2, old colour**(msb:lsb)**, new colour**(msb:lsb)	
4DSL Cmd	**ReplaceColor(x1, y1, x2, y2, old colour, new colour)**	
	cmd	**6B**(hex) or **k**(ascii) : Command header byte
	x1	Top left horizontal start position.
	y1	Top left vertical start position.
	x2	Bottom right horizontal end position.
	y2	Bottom right vertical end position.
	old colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
	new colour	2 bytes (16 bits) define the background colour in RGB format: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 where: msb : R4R3R2R1R0G5G4G3 lsb : G2G1G0B4B3B2B1B0
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if operation successful **15**(hex) : NAK byte if unsuccessful
Description	This command replaces the old colour of the selected rectangular region to the new specified colour..	
Serial Example	**Command Data:** 6Bhex, 00hex, 00hex, 50hex, 50hex, 00hex, 00hex, FFhex, FFhex Change the colour of 80x80 rectangular region from BLACK (0000hex) to WHITE (FFFFhex).	
4DSL Example	Refer to the General.4DScript sample script file.	

3.2.13 Set Pen Size - 70hex

Command	**cmd, size**	
4DSL Cmd	**Pen(size)**	
	cmd	**70**(hex) or **p**(ascii) : Command header byte
	size	Selects one of the 2 options: **00**hex : All graphics objects are drawn solid **01**hex : All graphics objects are drawn wire-frame Note: Does not apply to polygon command.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command determines if certain graphics objects are drawn in solid or wire frame fashion.	
Serial Examples	**Command Data:** 70hex, 00hex (All objects will be drawn solid). **Command Data:** 70hex, 01hex (All objects will be drawn wire-frame).	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

3.2.14 Draw Rectangle - 72hex

Command	**cmd, x1, y1, x2, y2, colour**(msb:lsb)	
4DSL Cmd	**Rectangle(x1, y1, x2, y2, colour)**	
	cmd	**72**(hex) or **r**(ascii) : Command header byte
	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.
	colour	2 bytes define the rectangle colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured rectangle from point **(x1, y1)** to point **(x2, y2)** 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, the rectangle will be solid, otherwise it will be wire-frame if value was 1. 	
Serial Example	**Command Data:** 72hex, 00hex, 00hex, 50hex, 50hex, 00hex, 1Fhex Draw a blue rectangle.	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

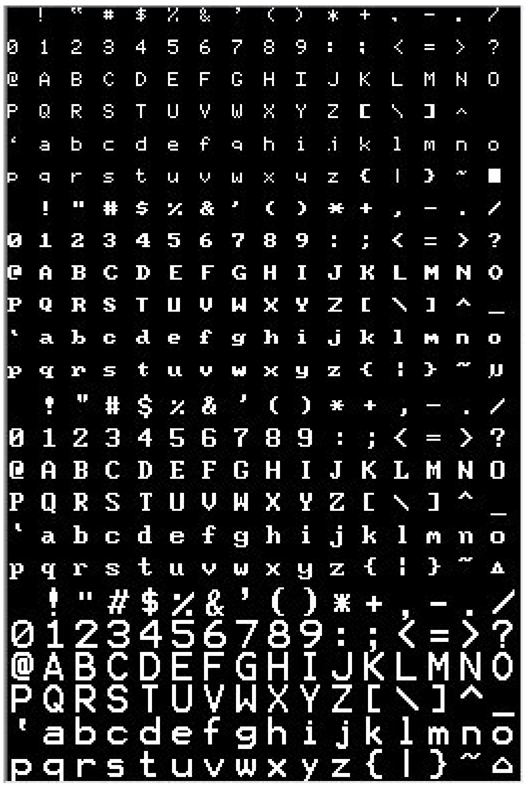
3.3 Text Commands

The GOLDELOX-SGC is shipped with 3 internal fonts. These fonts can be altered, deleted and replaced with new fonts. The **FONT-Tool** is a free software tool that can assist in the conversion of any Windows fonts into the bitmap format that can be used by the GOLDELOX-SGC. The converted font set can then be exported into the **DISP-Tool** utility which can then be downloaded into the GOLDELOX-SGC on-chip flash memory. Both the FONT-Tool and the DISP-Tool are available free from www.4dsystems.com.au

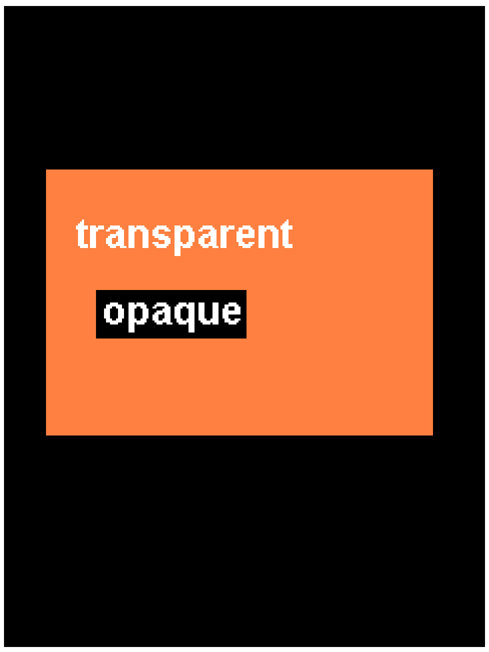
Summary of Commands in this section:

- Set Font – **46hex**
- Set Transparent-Opaque Text – **4Fhex**
- Draw “String” of ASCII Text (graphics format) – **53hex**
- Draw ASCII Character (text format) – **54hex**
- Draw Text Button – **62hex**
- Draw “String” of ASCII Text (text format) – **73hex**
- Draw ASCII Character (graphics format) – **74hex**

3.3.1 Set Font - 46hex

Command	**cmd, fontSet**	
4DSL Cmd	**Font(fontSet)**	
	cmd	**46**(hex) or **F**(ascii) : Command header byte
	fontSet	Selects one of internal fonts. The supplied 3 fonts are: **00**hex : 5x7 small size font set **01**hex : 8x8 medium size font set **02**hex : 8x12 large size font set These fonts can be altered and other fonts can be added.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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. **NOTE:** The GOLDELOX-SGX is shipped with three fonts displaying the characters 0x20 to 0x7f'. i.e. Space to the character after the tilde. 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').	
Serial Examples	**Command Data:** 46hex, 00hex (Select small 5x7 font). **Command Data:** 46hex, 00hex (Select medium 8x8 font). **Command Data:** 46hex, 00hex (Select large 8x12 font). 	
4DSL Example	Refer to the Text.4DScript sample script file.	

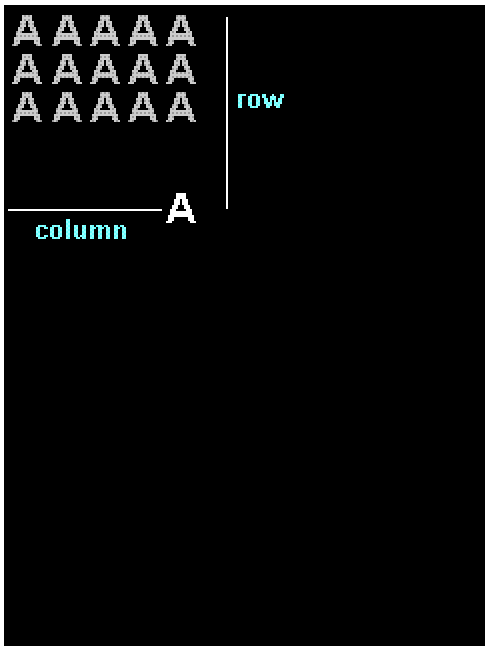
3.3.2 Set Transparent-Opaque Text - 4Fhex

Command	**cmd, mode**	
4DSL Cmd	**Opacity(mode)**	
	cmd	**4F**(hex) or **O**(ascii) : Command header byte
	mode	Select one of the following options for text appearance: **00**hex : Transparent, objects behind text are visible. **01**hex : Opaque, objects behind text blocked by background.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
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.	
Serial Examples	**Command Data:** 4Fhex, 00hex (Transparent text mode). **Command Data:** 4Fhex, 01hex (Opaque text mode).	
4DSL Example	Refer to the Text.4DScript sample script file.	

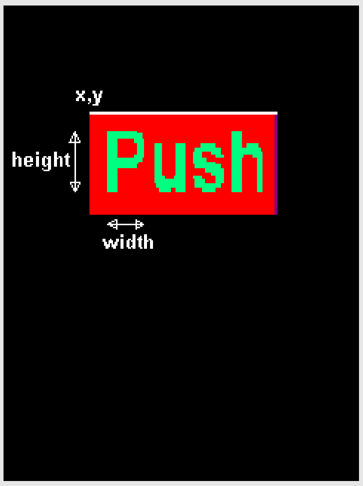
3.3.3 Draw “String” of ASCII Text (graphics format) - 53hex

Command	**cmd, x, y, font, stringColour**(msb:lsb)**, width, height, “string”, terminator**	
4DSL Cmd	**StringG(x, y, font, stringColour, width, height, 'string')**	
	cmd	**53**(hex) or **S**(ascii) : Command header byte
	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: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added. **OR**ing the fonts with 0x10 will cause the string to be displayed in a proportional manner (eg 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text colour.
	width	This byte defines the width or horizontal size multiplier of the character in the string. Effects the total width of the string.
	height	This byte defines the height or vertical size multiplier of the character in the string. Effects 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**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw/display a string of ASCII text anywhere on the screen in pixel coordinates specified by **x** and **y** parameters. 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 size 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, a wrap around will occur on to the next line. Maximum string length is **256 bytes**.	
Serial Example	**Command Data:** 53hex, 14hex, 14hex, 01hex, FFhex, FFhex, 01, 01, 48hex, 65hex, 6Chex, 6Chex, 6Fhex, 00hex Draw/Display String ‘Hello’ at x = 20, y = 20 in WHITE (FFFFhex) colour .	
4DSL Example	Refer to the Text.4DScript sample script file.	

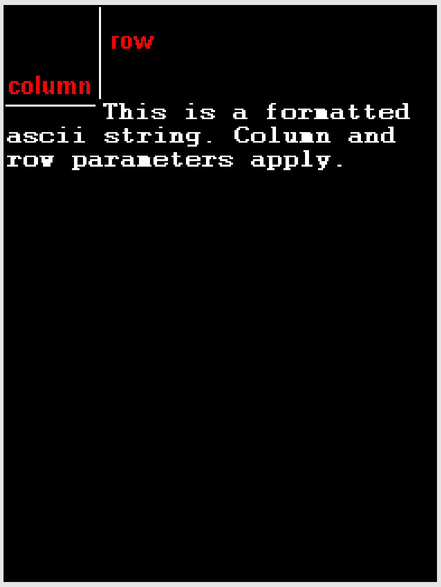
3.3.4 Draw ASCII Character (text format) - 54hex

Command	**cmd, char, column, row, charColour**(msb:lsb)	
4DSL Cmd	**AsciiChar(char, column, row, charColour)**	
	cmd	**54**(hex) or **T**(ascii) : Command header byte
	char	Inbuilt standard ASCII character. range : 32dec – 127dec (20hex - 7Fhex).
	column	Horizontal position of the character (character units). range : **0 - 20** for 5x7 font. range : **0 - 15** for 8x8 and 8x12 fonts.
	row	Vertical position of the character (character units). range : **0 - 15** for 5x7 and 8x8 fonts. range : **0 - 9** for 8x12 font.
	charColour	2 bytes define the character colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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.	
Serial Example	**Command Data:** 54hex, 41hex, 00hex, 00hex, FFhex, FFhex Draw/Display character 'A' (41hex) at column = 0, row = 0, colour = white (FFFFhex). 	
4DSL Example	Refer to the Text.4DScript sample script file.	

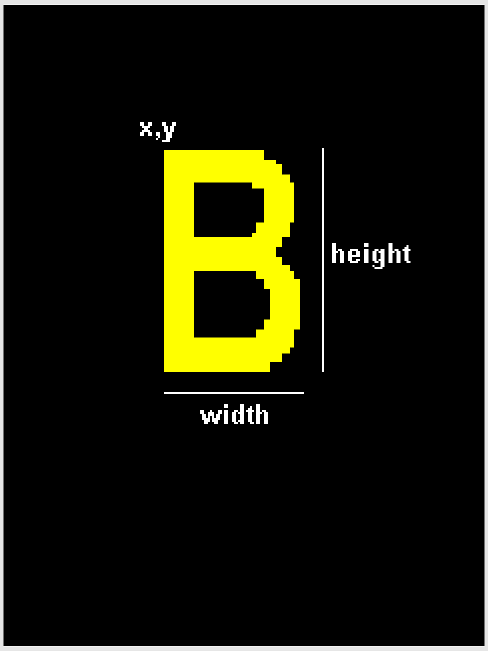
3.3.5 Draw Text Button - 62hex

Command	**cmd, state, x, y, buttonColour**(msb:lsb)**, font, stringColour**(msb:lsb)**, width, height, "string", terminator**	
4DSL Cmd	**Button(state, x, y, buttonColour, font, stringColour, width, height, 'string')**	
	cmd	**62**(hex) or **b**(ascii) : Command header byte
	state	This byte specifies whether the displayed button is drawn **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	Top left vertical start position of the button.
	buttonColour	2 bytes define the button colour.
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added.
	stringColour	2 bytes define the string text colour.
	width	This byte defines the width or horizontal size (x magnification) of the character in the string. Effects 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. Effects 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**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will place a Text button similar to the ones used in a PC Windows environment. 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. 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.	
4DSL Example	Refer to the Text.4DScript sample script file.	

3.3.6 Draw "String" of ASCII Text (text format) - 73hex

Command	**cmd, column, row, font, stringColour**(msb:lsb), **"string", terminator**	
4DSL Cmd	**String(column, row, font, stringColour, 'string')**	
	cmd	**73**(hex) or **s**(ascii) : Command header byte
	column	Horizontal start position of the string (character units). range : **0 - 20** for 5x7 font. range : **0 - 15** for 8x8 and 8x12 fonts.
	row	Vertical start position of the string (character units). range : **0 - 15** for 5x7 and 8x8 fonts. range : **0 - 9** for 8x12 font.
	font	This byte specifies which internal font set to use for the string. The supplied fonts are: **0** : 5x7 internal font **1** : 8x8 internal font **2** : 8x12 internal font These fonts can be altered and other fonts can be added. **OR**ing the fonts with 0x10 will cause the string to be displayed in a proportional manner (eg 0x10 is font 0 proportional, 0x11 is font 1 proportional, etc).
	stringColour	2 bytes define the string text colour.
	"string"	String of ASCII characters to be displayed (max. 256 characters).
	terminator	The string must be terminated with **00**hex.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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**hex. 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**.	 This is a formatted ascii string. Column and row parameters apply.
Serial Example	**Command Data:** 73hex, 07hex, 08hex, 01, FFhex, FFhex, 48hex, 65hex, 6Chex, 6Chex, 6Fhex, 00hex Display "Hello" at 7th column and 8th row in WHITE (FFFFhex) colour with 8x8 Fonts.	
4DSL Example	Refer to the Text.4DScript sample script file.	

3.3.7 Draw ASCII Character (graphics format) - 74hex

Command	**cmd, char, x, y, charColour**(msb:lsb)**, width, height**	
4DSL Cmd	**AsciiCharG(char, x, y, charColour, width, height)**	
	cmd	**74**(hex) or **t**(ascii) : Command header byte
	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 colour.
	width	This byte defines the width or horizontal size (multiplier) of the character.
	height	This byte defines the height or vertical size (multiplier) of the character.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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 (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. 	
Serial Example	**Command Data:** 74hex, 24hex, 30hex, 30hex, FF, FFhex, 03hex, 03hex Display a '$' sign at 48, 48 location in WHITE (FFFFhex) colour.	
4DSL Example	Refer to the Text.4DScript sample script file.	

3.4 SD/SDHC Memory Card Commands

The commands detailed in this section utilise the SDHC/SD/microSD memory card which must be connected to the SPI port of the GOLDELOX-SGC. 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 such as data logging applications.

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

Summary of Commands in this section:

- Set Address Pointer of Memory Card – **@41hex**
- Screen Copy-Save to Memory Card - **@43hex**
- Display Image-Icon from Memory Card - **@49hex**
- Display Object from Memory Card - **@4Fhex**
- Run Script (4DSL) Program from Memory Card - **@50hex**
- Read Sector Block Data from Memory Card - **@52hex**
- Display Video-Animation Clip from Memory Card - **@56hex**
- Write Sector Block Data to Memory Card - **@57hex**
- Initialise Memory Card - **@69hex**
- Read Byte Data from Memory Card - **@72hex**
- Write Byte Data to Memory Card - **@77hex**

3.4.1 Set Address Pointer of Memory Card - @41hex

Command	**ext_cmd, cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)	
4DSL Cmd	**SetAddress(Address)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**41**(hex) or **A**(ascii) : Command header byte
	Address	A 4 byte card memory address (big endian) for byte wise access.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
Description	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.	
Serial Example	**Command Data:** 40hex, 41hex, 00hex, 00hex, 04hex, 00hex Set Internal memory address pointer to 000400hex.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.2 Screen Copy – Save to Memory Card - @43hex

Command	**ext_cmd, cmd, x, y, width, height, SectorAdd**(hi:mid:lo)	
4DSL Cmd	**ScreenCopyuSD(x,y,width,height, SectorAdd)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**43**(hex) or **C**(ascii) : Command header byte
	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	3 bytes (big endian) sector address where the copied screen area is to be saved.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
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 **“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. This is a very powerful feature for animating objects, smooth scrolling, or implementing a windowing system. **Notes:** • 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.	
Serial Example	**Command Data:** 40hex, 43hex, 00hex, 00hex, 80hex, 80hex, 00hex, 04hex, 00hex Save 128x128 screen area on the uSD card at sector address 000400hex.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.3 Display Image-Icon from Memory Card - @49hex

Command	**ext_cmd, cmd, x, y, width, height, colourMode, SectorAdd**(hi:mid:lo)	
4DSL Cmd	**UsdImage(x, y, width, height, colourMode, SectorAdd)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**49**(hex) or **I**(ascii) : Command header byte
	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	**10**(hex) : 65K colour mode, 16bits/2bytes per pixel .
	SectorAdd	3 bytes (big endian) sector address of a previously stored Image-Icon that is about to be displayed.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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. 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. The **colourMode** byte parameter can only be set to 10hex, i.e. the previously stored image can only be 16 bit colour format (2 bytes per pixel). **Notes:** • 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. • Do not store an image/icon in 8 bit colour format, this will result in a corrupted image. • 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.	
Serial Example	**Command Data:** 40hex, 49hex, 00hex, 00hex, 80hex, A0hex, 10hex, 3Bhex, 16hex, 04hex Display the image at (0, 0) from sector address 3B1604hex.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.4 Display Object from Memory Card - @4Fhex

Command	**ext_cmd, cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)	
4DSL Cmd	**Object(Address)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**4F**(hex) or **O**(ascii) : Command header byte
	Address	A 4 byte card memory address (big endian) of a previously stored Object that is about to be displayed.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
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.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.5 Run Script (4DSL) Program from Memory Card - @50hex

Command	**ext_cmd, cmd, Address**(Umsb:Ulsb:Lmsb:Llsb)	
4DSL Cmd	**RunScript(Address)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**50**(hex) or **P**(ascii) : Command header byte
	Address	A 4 byte card memory start address (big endian) of a 4DSL (4D Scripting Language) program.
Response	**acknowledge**	
	acknowledge	There is no response to a successful command, as potentially the command may never end. **15**(hex) : NAK byte if unsuccessful or card not present.
Description	The majority of the commands can be composed as a script and written into memory card. A 4DSL 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 2.6. This command forces the 32bit internal memory pointer to jump to the specified address and automatically start executing a 4DSL script program, from the memory card without any further interaction by the host processor. It will sequentially execute any valid 4DSL instruction and commands until it gets to the end of the program.	
Serial Example	**A sample script program inside the memory card:**	
4DSL Example	Refer to the Raw.4DScript sample script file.	

Serial Example — **A sample script program inside the memory card:**

Address	**Command**	**Comment**
00000000	45	Erase Screen
00000001	43 64 32 14 00 1F	Draw Circle
0000000A	07 03 E8	Delay(1second)
0000000D	72 00 00 3C 3C 07 E0	Draw Rectangle
00000018	40 56 00 00 46 32 10 0A 02 5F 00 10 00	Play video from card
00000029	0B 00 00 00 00	Goto Address 00000000

3.4.6 Read Sector Block Data from Memory Card - @52hex

Command	**ext_cmd, cmd, SectorAdd**(hi:mid:lo)	
4DSL Cmd	**ReadSector(SectorAdd)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**52**(hex) or **R**(ascii) : Command header byte
	SectorAdd	3 bytes (big endian) 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.
Response	**data(1..512)**	
	data	512 bytes of sector data
Description	This command will return 512 bytes of data relating to a sector.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.7 Display Video-Animation Clip from Memory Card - @56hex

Command	**ext_cmd, cmd, x,y,width, height, colourMode, delay, frames**(msb:lsb), **SectorAdd**(hi:mid:lo)	
4DSL Cmd	**Video(x, y, width, height, colourMode, delay, frames, SectorAdd)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**56**(hex) or **V**(ascii) : Command header byte
	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	**10**(hex) : 65K colour 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.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	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. The **colourMode** byte parameter can only be set to 10hex, i.e. the previously stored video/animation can only be 16 bit colour format (2 bytes per pixel). **Notes:** • Do not store a video/animation in 8 bit colour format, this will result in a corrupted result.	
Serial Example	**Command Data:** 40hex, 56hex, 00hex, 00hex, 80hex, A0hex, 10hex, 01hex, 64hex, 00hex, 00hex, 04hex Display the video/animation at (0, 0) from sector address 000004hex.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.8 Write Sector Block Data to Memory Card - @57hex

Command	**ext_cmd, cmd, SectorAdd**(hi:mid:lo), data(1..512)	
4DSL Cmd	**WriteSector(SectorAdd, data**(1..512)**)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**57**(hex) or **W**(ascii) : Command header byte
	SectorAdd	3 bytes (big endian) sector address.
	data	512 bytes of sector data. Data length must be 512 bytes.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
Description	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 00hex or FFhex (it can be anything). If only few bytes of data are to be written then the **"Write Byte Data to Memory Card"** command can be used. 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. Only **data**(1..512) are written to the sector. Other bytes in the command message do not get written.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.9 Initialise Memory Card - @69hex

Command	**ext_cmd, cmd**	
4DSL Cmd	**InituSD**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**69**(hex) or **i**(ascii) : Command header byte
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
Description	This command initialises the 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. **Note!** There is no card insert/remove auto detect facility.	
Serial Example	**Command Data:** 40hex, 69hex Initialise the uSD card.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.10 Read Byte Data from Memory Card - @72hex

Command	**ext_cmd, cmd**	
4DSL Cmd	**ReadByte**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**72**(hex) or **r**(ascii) : Command header byte
Response	**data_byte**	
	data_byte	1 byte of card data
Description	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.	
Serial Example	**Command Data:** 40hex, 41hex, 00hex, 00hex, 04hex, 00hex Set Internal memory address pointer to 000400hex. 40hex, 72hex Read 1 byte from the 000400hex address on theuSD card.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

3.4.11 Write Byte Data to Memory Card - @77hex

Command	**ext_cmd, cmd, data**	
4DSL Cmd	**WriteByte(Data)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**77**(hex) or **w**(ascii) : Command header byte
	data	1 byte of card data
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
Description	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.	
Serial Example	**Command Data:** 40hex, 41hex, 00hex, 00hex, 00hex, A0hex Set Internal memory address pointer to 0000A0hex. 40hex, 77hex, 4Dhex Write 1 byte ('M') at 0000A0hex address on the uSD card.	
4DSL Example	Refer to the Raw.4DScript sample script file.	

4. 4DSL Scripting Language

4.1 Script Commands (4DSL - Script Language)

The commands detailed in this section must reside in the SDHC/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. Additional commands are under development to expand the scripting language and these will be released in due course.

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

Summary of Commands in this section:

- Delay – **07hex**
- Set Counter – **08hex**
- Decrement Counter – **09hex**
- Jump to Address If Counter Not Zero – **0Ahex**
- Jump to Address – **0Bhex**
- Exit-Terminate Script Program – **0Chex**

4.1.1 Delay - 07hex

4DSL Cmd	**Delay(value)**	
	value	2 byte (big endian) delay value in milliseconds.
Description	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.	
4DSL Example	Refer to the ABC400.4DScript sample script file.	

4.1.2 Set Counter - 08hex

4DSL Cmd	**SetCounter(value)**	
	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.
Description	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** — **Comment** 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 of course be the series of hex codes.	
4DSL Example	Refer to the ABC400.4DScript sample script file.	

4.1.3 Decrement Counter - 09hex

4DSL Cmd	**Decrement**
Description	Decrements the Counter. See detailed description on how this command can be used effectively in the **“Set Counter”** command section.
4DSL Example	Refer to the ABC400.4DScript sample script file.

4.1.4 Jump to Address If Counter Not Zero - 0Ahex

4DSL Cmd	**JumpNotZero(Address)**	
	Address	A 4 byte (big endian) card memory jump address if 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 script command. Please see detailed description on how this command can be used effectively in the **”Set Counter”** command section.	
4DSL Example	Refer to the ABC400.4DScript sample script file.	

4.1.5 Jump to Address - 0Bhex

4DSL Cmd	**GoTo(Address)**	
	Address	A 4 byte (big endian) card 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.	
4DSL Example	Refer to the ABC400.4DScript sample script file.	

4.1.6 Exit-Terminate Script Program - 0Chex

4DSL Cmd	**Exit**
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 terminate. It can also be sent, by the host, via the serial link to terminate a program currently executing from the memory card.
4DSL Example	Refer to the ABC400.4DScript sample script file.

4.2 Directives (4DSL - Script Language)

Directives are lines included in the program but are not program statements. These lines are always preceded by a hash sign (#). They are executed before the actual compilation of code begins.

They extend only across a single line of code. As soon as a newline character is found, the directive is considered to end. No semicolon ";" is expected at the end of the directive..

Summary of Commands in this section:

- #Compile
- #Define
- #Include
- #Origin
- #Run

4.2.1 #Compile

4DSL Cmd	**#compile("Platform", "Comport", "Speed", "WrapCol", "WrapTrunc")**	
	Platform	Picaso or Goldelox
	Comport	The comm port to use
	Speed	The maximum speed of the Comm port, used during downloads, 9600 is used normally.
	WrapCol	The number of bytes after which wrapping or truncation occcurs in the compile listing
	WrapTrunc	Wrap or Trunc. Specifies whether the compile listing is wrapped or truncated when Wrapcol is reached
Description	Set script compile and options. This must be the first line of a script. it can be changed using the buttons in the workshop window. The comm port may be set manually.	
Serial Example	#Compile(Goldelox,COM4,9600,5,Wrap)	
4DSL Example	Most of the 4DS Script Samples.	

4.2.2 #define

4DSL Cmd	**#define ("Name", "Substitution")**	
	Name	Source to be substituted
	Substitution	The replacement text or value
Description	This can be used to define replacement for parameters so that they can be set from the command line	
Serial Example	#define red 0xf800 OR #define file "C:\test.fle"	
4DSL Example	Refer to the 4DScript_16bitColours.inc sample file.	

4.2.3 #Include

4DSL Cmd	**#include("Filename")**	
	Filename	Name of the file to be included
Description	This can be used to include other files into the script..	
Serial Example	#include "4DScript_16bitColours.inc"	
4DSL Example	Most of the 4DS Script Samples.	

4.2.4 #origin

4DSL Cmd	**#origin("Origin")**	
	Origin	The start address of this script
Description	Use this to specify the start address of a script. Only one #origin statement is permitted. This defines the start of the script when it is written to a uSD card in RAW mode.	
Serial Example	#Origin 0x400 // start on sector 2	
4DSL Example	Refer to the ABC400.4DScript sample script file.	

4.2.5 #run

4DSL Cmd	**#run("Platform", "Comport", "Speed", "WrapCol", "WrapTrunc")**	
	Platform	Picaso or Goldelox
	Comport	The comm port to use
	Speed	The maximum speed of the Comm port, used during downloads, 9600 is used normally.
	WrapCol	The number of bytes after which wrapping or truncation occcurs in the compile listing
	WrapTrunc	Wrap or Trunc. Specifies whether the compile listing is wrapped or truncated when Wrapcol is reached
Description	Set script run and options. This must be the first line of a script. it can be changed using the buttons in the workshop window. The comm port may be set manually.	
Serial Example	#run(Goldelox,COM4,9600,5,Wrap); //Line 1 set script run and options	
4DSL Example	Most of the 4DS Script Samples.	

4.3 Macros (4DSL - Script Language)

Given below is the detailed command set for Macros that are executed from the PC while the display module is connected to it. These commands begin with a $ sign. They also include some of the general serial commands that can be executed with PC acting as a host controller such as $ReadFile and $WriteSectors etc.

Summary of Commands in this section:

- $4DGLLoadprogram
- $LoadPmmC
- $Message
- $ReadBytes
- $ReadSectors
- $ReaduSDImage
- $StartSave
- $WriteSectors

- $4DGLAttn
- $4DGLExit
- $AbortOnError
- $CloseComPort
- $EndSave
- $FlushBuffer
- $IgnoreErrors
- $OpenComport
- $OpenInit
- $ReadCSD
- $TimeOff
- $TimeOn

4.3.1 $4DGLLoadprogram

4DSL Cmd	**$4DGLLoadprogram(“4DGLprogram”, ram|flash)**	
	4DGLprogram	The filename of a compiled 4DGL program to be loaded onto a display
	ram|flash	ram/flash is ignored for GOLDELOX (it's always flash).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Loads the specified program onto the display. The file extension must not be specified.	
Serial Example	$4DGLLoadProgram("c:\Documents and Settings\All Users\Documents\4D Labs\PICASO GFX2\Picaso – Graphics\worm",Ram)	
4DSL Example	Refer to the Load4DGLProgram.4DScript sample script file.	

4.3.2 $LoadPmmC

4DSL Cmd	**$LoadPmmC("PmmCName")**	
	PmmCName	The filename of the PmmC to be loaded.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Loads a PmmC file on the processor.	
Serial Example	$LoadPmmC("c:\Documents and Settings\All Users\Documents\4D Labs\PICASO GFX2\Picaso – Graphics\worm",Ram)	
4DSL Example	Refer to the LoadPmmC .4DScript sample script file.	

4.3.3 $Message

4DSL Cmd	**$Message('String')**	
	String	The string of text to be displayed .
Description	Displays a message to the user. If run from the command line the message is displayed on the console and the console waits for a key to be pressed, If run from workshop a message box is displayed.	
Serial Example	$Message('About to halve Volume')	
4DSL Example	Refer to the General.4DScript sample script file.	

4.3.4 $ReadBytes

4DSL Cmd	**$ReadBytes("Filename", "#Bytes")**	
	Filename	The file to write the bytes to
	#Bytes	The number of bytes to read
Response	**data_byte**	
	data_byte	1 byte of card data
Description	Reads number of from the uSD card at the current address and saves them to the file Filename.	
Serial Example	$Readbytes('50bytes.hex',50)	
4DSL Example	Refer to the Raw.4DScript sample script file.	

4.3.5 $ReadSectors

4DSL Cmd	**$ReadSectors("pcFile", "StartSector", "#Sectors")**	
	pcFile	The handshaking to use 0-50.
	startsector	The name of the file on the uSD Card to read.
	#sectors	The number of ACKS to send at the start, Double or Single, Double gives the best performance, but will not work on all controllers.
Description	This reads #sectors from the uSD card starting at the specified sector and saves them to the pc with the specified Filename.	
Serial Example	$readsectors('Sector0.hex',1,1)	
4DSL Example	Refer to the Raw.4DScript sample script file.	

4.3.6 $ReaduSDImage

4DSL Cmd	**$ReaduSDImage("startSector", "width", "height","pcFile")**	
	startsectors	The starting sector of the image on the uSD card.
	width	The width of the image on the uSD card or -1 for New Format, includes header (type2) images.
	height	The height of the image on the uSD card or -1 for New Format, includes header (type2) images.
	PcFile	The name of the output BMP file on the PC.
Description	This reads an image file from the uSD card and converts it to a Bitmap file and saves it on the PC.	
Serial Example	$ReaduSDImage(0,-1,-1,'ImageNew.bmp')	
4DSL Example	Refer to the Raw.4DScript sample script file.	

4.3.7 $StartSave

4DSL Cmd	**$StartSave("pcFile")**	
	PcFile	The name of the output file on the PC
Description	This causes the results of all subsequent commands to be written to the specified file until the $EndSave command is used.	
Serial Example	$startsave('pixel.hex') : : $endsave	
4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.	

4.3.8 $WriteSectors

4DSL Cmd	**$WriteSectors("pcFile", "StartSector")**	
	pcFile	The file to read the sectors from
	startsector	TThe starting sector to Write to
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This reads a FAT file from the uSD card and saves it on the PC.This reads #sectors from the uSD card starting at the specified sector and saves them to the pc with the specified Filename.	
Serial Example	$WriteSectors("..\..\resources\Misc\NemoNewFmt.hex",500)	
4DSL Example	Most of the 4DSL Scripts	

4.3.9 Extended Macros

$4DGLAttn	**Use this to get attention of the 4DGL hypervisor to enable the use of the read and write sectors commands.**	
	4DSL Example	Refer to the Load4DGLuSD.4DScript sample script file.
$4DGLExit	**Use this to exit from the 4DGL hypervisor**	
	4DSL Example	Refer to the Load4DGLuSD.4DScript sample script file.
$AbortOnError	**This causes the script to abort if a command resturns a NAK or other unexpected result. This is the default. The opposite is $IgnoreErrors**	
	4DSL Example	Refer to the General.4DScript sample script file.
$CloseComPort	**This closes the com port, if it is open.**	
	4DSL Example	Refer to the General.4DScript sample script file.
$EndSave	**This closes the file previously openned with $StartSave.**	
	4DSL Example	Refer to the GraphicsPt2.4DScript sample script file.
$FlushBuffer	**This flushes the Comms buffer.**	
	4DSL Example	Refer to the Raw.4DScript sample script file.
$IgnoreErrors	**This causes the script to continue if a command resturns a NAK or other unexpected result. The opposite is $AbortonError**	
	4DSL Example	Refer to the General.4DScript sample script file.
$OpenComport	**This Opens the com port, no checking is done. See $OpenInit for a verified open**	
	4DSL Example	Refer to the General.4DScript sample script file.
$OpenInit	**This Opens the com port and sends autobaud('U') and checks for an ACK. Up to 10 retries are performed.**	
	4DSL Example	Most of the 4DSL Scripts
$ReadCSD	**This reads CSD record from the uSD card and displays the card's capacity.**	
	4DSL Example	Refer to the Raw.4DScript sample script file.
$TimeOff	**This turns off logging of times for each command (default).**	
	4DSL Example	Refer to the General.4DScript sample script file.
$TimeOn	**This turns on logging of times for each command.**	
	4DSL Example	Refer to the General.4DScript sample script file.

4.4 4DSL Keywords

#run, #compile option

'Goldelox'	The target platform for 4DSL Script
' Wrap'	Sets the compile listing to be wrapped when Wrapcol is reached.
'Trunc'	Sets the compile listing to be truncated when Wrapcol is reached.

4DGL program Load option

'Ram'	Program is loaded to RAM.
'Flash'	Program is loaded to Flash.

Button option

'Down'	Button Down (pressed)
'Up'	Button Up (not pressed)

Font size option

'small'	5x7 small size font set
'medium'	8x8 medium size font set
'large'	8x12 large size font set
'xlarge'	12x16 largest size font set

Font magnification option

'Mag1'	Multiply the width by 1.
'Mag2'	Multiply the width by 2.
'Mag3'	Multiply the width by 3.

Opacity option

'Opaque'	Opaque, objects behind text blocked by background.
'Transparent'	Transparent, objects behind text are visible.

Pen Size Option

Solid	All graphics objects are drawn solid
Outline	All graphics objects are drawn wire-frame

16 bit Colours keywords

'AQUA'	0x07FF
'BLACK'	0x0000
'GREEN'	0x0400

Refer the 4DScript_16bitColours.inc file for complete list.

Note: Refer to the 4DSL Script Samples for details on usage of above keywords.

4.5 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.

- **General Commands**
    - AutoBaud - 55hex.
    - Set new Baud Rate - 51hex
    - Version-Device Info Request - 56hex.
    - Replace Background Colour - 42hex.
    - Clear Screen - 45hex.
    - Display Control Functions - 59hex.
    - Sleep - 5Ahex
    - Switch-Buttons-Joystick Status - 4Ahex.
    - Wait for Switch-Buttons-Joystick Status - 6Ahex.
    - Sound - 4Ehex.
    - Tune - 6Ehex
- **Graphics Commands**
    - Add User Bitmap Character - 41hex.
    - Draw Circle - 43hex.
    - Draw User Bitmap Character - 44hex.
    - Draw Triangle - 47hex.
    - Draw Image-Icon - 49hex.
    - Set Background colour - 4Bhex
    - Draw Line - 4Chex.
    - Draw Pixel - 50hex.
    - Read Pixel - 52hex.
    - Screen Copy-Paste - 63hex.
    - Draw Polygon - 67hex.
    - Replace Colour - 6Bhex
    - Set Pen Size - 70hex.
    - Draw Rectangle - 72hex.
- **Text Commands**
    - Set Font - 46hex.
    - Set Transparent-Opaque Text - 4Fhex.
    - Draw “String” of ASCII Text (graphics format) - 53hex.
    - Draw ASCII Character (text format) - 54hex.
    - Draw Text Button - 62hex.
    - Draw “String” of ASCII Text (text format) - 73hex.
    - Draw ASCII Character (graphics format) - 74hex.
- **SD Memory Card Commands (Low-Level/RAW)**
    - Set Address Pointer of Memory Card - @41hex.
    - Screen Copy – Save to Memory Card - @43hex.
    - Display Image-Icon from Memory Card - @49hex.
    - Display Object from Memory Card - @4Fhex.
    - Run Script (4DSL) Program from Memory Card - @50hex.

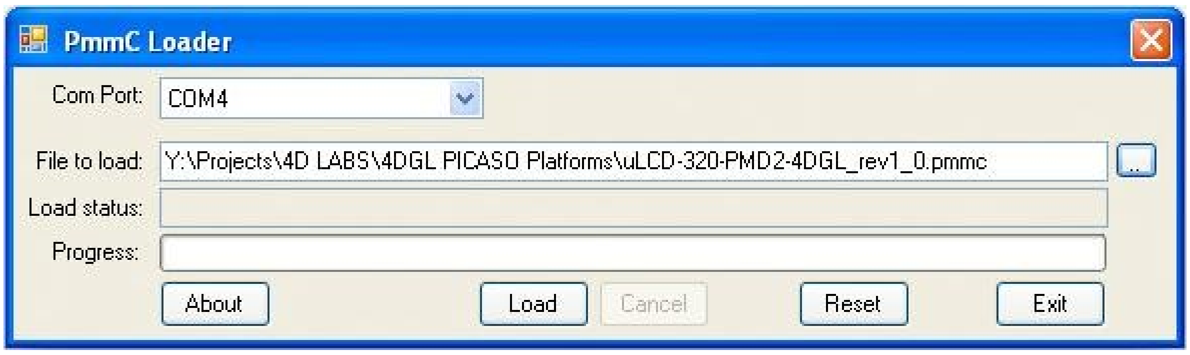
- Read Sector Block Data from Memory Card - @52hex.
- Display Video-Animation Clip from Memory Card - @56hex.
- Write Sector Block Data to Memory Card - @57hex.
- Initialise Memory Card - @69hex.
- Read Byte Data from Memory Card - @72hex.
- Write Byte Data to Memory Card - @77hex.

- **Script Control Commands (4DSL - Script Language)**
    - Delay - 07hex.
    - Set Counter - 08hex.
    - Decrement Counter - 09hex.
    - Jump to Address If Counter Not Zero - 0Ahex.
    - Jump to Address - 0Bhex.
    - Exit-Terminate Script Program - 0Chex.
- **Directives**
    - #compile
    - #define
    - #include
    - #origin
    - #run
- **Macros (4DSL - Script Language)**
    - $4DGLAttn
    - $4DGLExit
    - $4DGLLoadprogram
    - $AbortOnError
    - $CloseComPort
    - $EndSave
    - $FlushBuffer
    - $IgnoreErrors
    - $LoadPmmC
    - $Message
    - $OpenComport
    - $OpenInit
    - $ReadBytes
    - $ReadCSD
    - $ReadSectors
    - $ReaduSDImage
    - $StartSave
    - $TimeOff
    - $TimeOn
    - $WriteSectors

5. Appendix A : Development and Support Tools

5.1 PmmC Loader – PmmC File Programming Software Tool

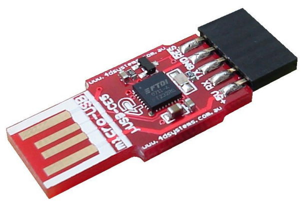
The ‘PmmC Loader’ is a free software tool for Windows based PC platforms. Use this tool to program the latest PmmC file into the GOLDELOX-SGC chip embedded in your application board. It is available for download from the 4D Systems website, www.4dsystems.com.au

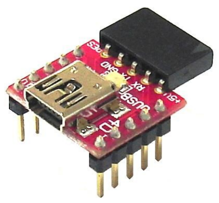


Com Port: COM4
File to load: Y:\Projects\4D LABS\4DGL PICASO Platforms\uLCD-320-PMD2-4DGL_rev1_0.pmmc
Load status:
Progress:

5.2 microUSB – PmmC Programming Hardware Tool

The micro-USB module is a USB to Serial bridge adaptor that provides a convenient physical link between the PC and the GOLDELOX-SGC device. A range of custom made micro-USB devices such as the uUSB-MB5 and the uUSB-CE5 are available from 4D Systems www.4dsystems.com.au. The micro-USB module is an essential hardware tool for all the relevant software support tools to program, customise and test the GOLDELOX-SGC chip.


uUSB-CE5


uUSB-MB5

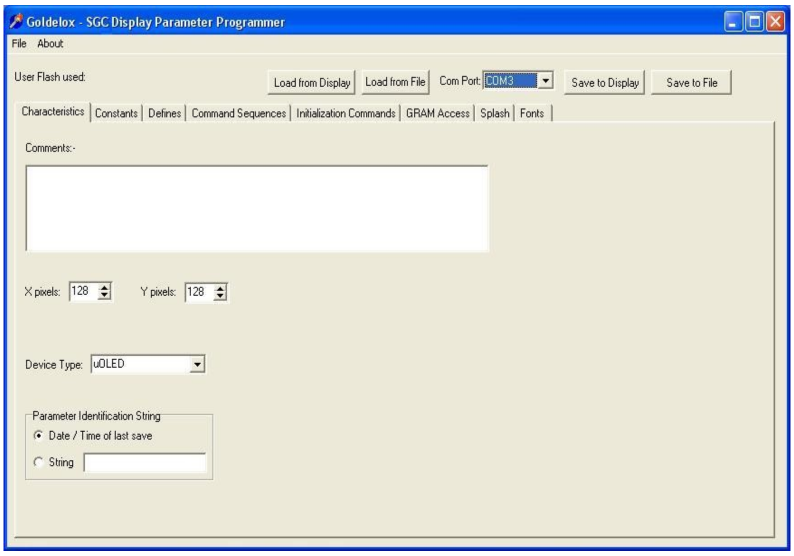
4D Programming Cable

5.3 Display Initialisation Setup Personality (DISP) – Software Tool

DISP is a free software tool for Windows based PC platforms. Use this tool to:-

- Configure the GOLDELOX-SGC chip to work with a specific display.
- Modify the way the chip initially sets up the display, e.g. screen saver, brightness, etc.
- Construct the splash screens.
- Replace or modify the embedded fonts.

It is available for download from the 4D Systems website, www.4dsystems.com.au.



Com Port: COM3

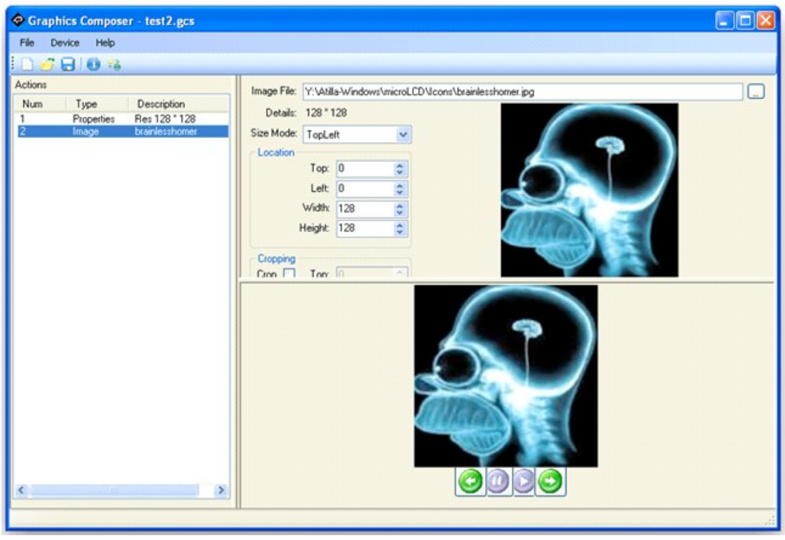
X pixels: 128 Y pixels: 128

Device Type: uOLED

(●) Date / Time of last save
( ) String

5.4 Graphics Composer – Software Tool

The Graphics Composer is a free software tool for Windows. This software tool is an aid to composing a slide show of images/animations/movie-clips (multi-media objects) which can then be downloaded into the SDHC/SD/uSD/MMC memory card that is supported by the GOLDELOX-SGC. The host simply sends commands to the GOLDELOX-SGC to display the multimedia objects.



Num	Type	Description
1	Properties	Res 128 * 128
2	Image	brainlesshomer

Image File: Y:\Atilla-Windows\microLCD\Icons\brainlesshomer.jpg

Details: 128 * 128

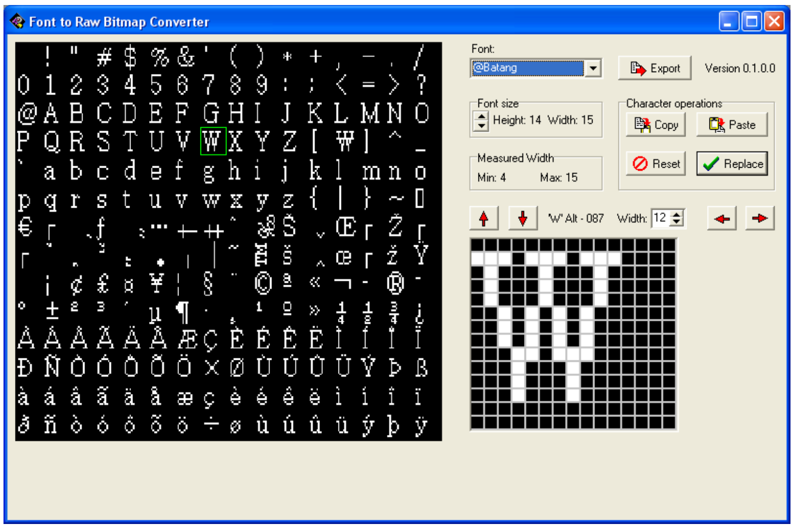
Size Mode: TopLeft

Top: 0
Left: 0
Width: 128
Height: 128

5.5 FONT Tool – Software Tool

Font-Tool is a free software utility for Windows based PC platforms. This tool can be used to assist in the conversion of standard Windows fonts (including True Type) into the bitmap fonts used by the GOLDELOX-SGC chip. It is available for download from the 4D Systems website, www.4dsystems.com.au.

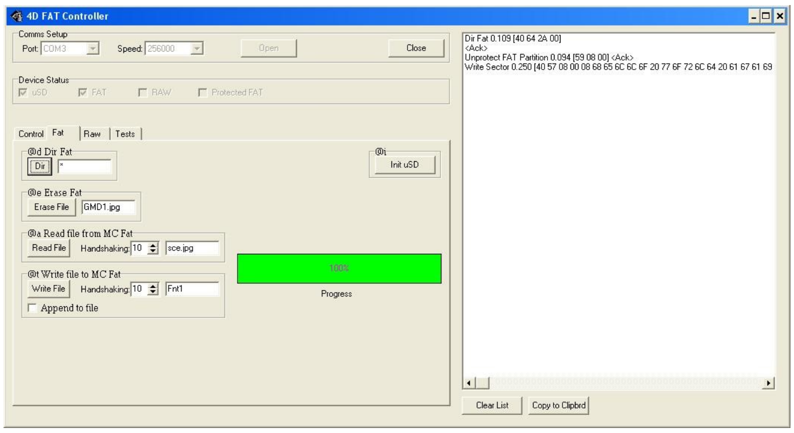
Disclaimer: Windows fonts may be protected by copyright laws. This software is provided for experimental purposes only.



Font: @Batang
Font size: Height: 14 Width: 15
Measured Width: Min: 4 Max: 15
'W' Alt - 087 Width: 12

5.6 FAT Controller – Software Test Tool

The 4D FAT Controller is a free software tool to test all of the functionality of the GOLDELOX-DOS, GOLDELOX-SGC and the GOLDELOX-SGC devices and their respective modules. It is useful in learning about how to communicate with the chips and the modules. For the GOLDELOX-SGC and the GOLDELOX-SGC it can also simulate most of the operation of the device and assist in the creation of simple scripts, either simulating the execution of those scripts and / or downloading them into a uSD/uSDHC card for execution on the display.

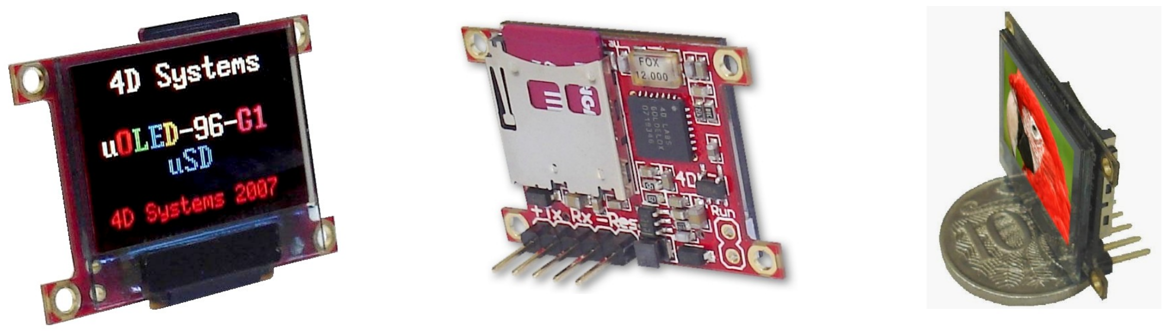


Comms Setup: Port: COM3 Speed: 256000
Device Status: [✓ uSD] [✓ FAT] RAW Protected FAT
@d Dir Fat: *
@e Erase Fat: GMD1.jpg
@a Read file from MC Fat: Handshaking: 10 sce.jpg
@t Write file to MC Fat: Handshaking: 10 Fnt1
Append to file
@i

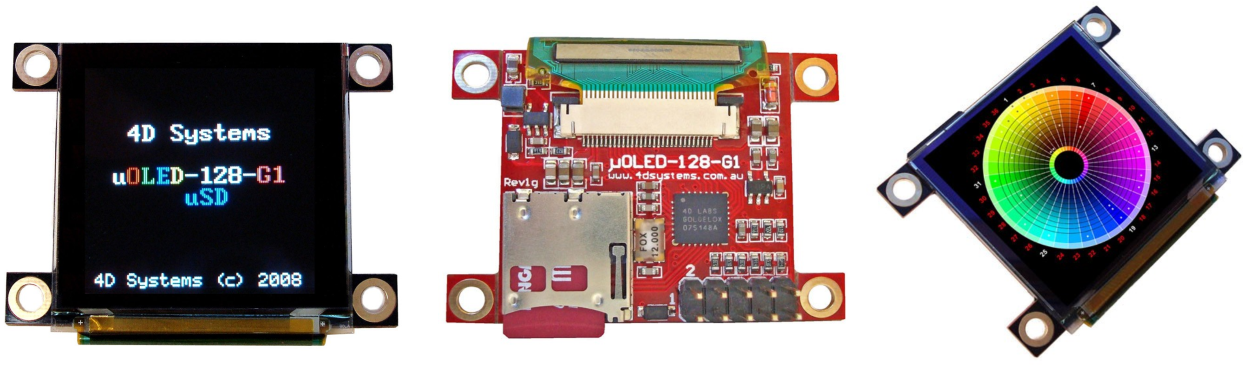
```
Dir Fat 0.109 [40 64 2A 00]
<Ack>
Unprotect FAT Partition 0.094 [59 08 00] <Ack>
Write Sector 0.250 [40 57 08 00 08 68 65 6C 6C 6F 20 77 6F 72 6C 64 20 61 67 61 69
```

5.7 Evaluation Display Modules

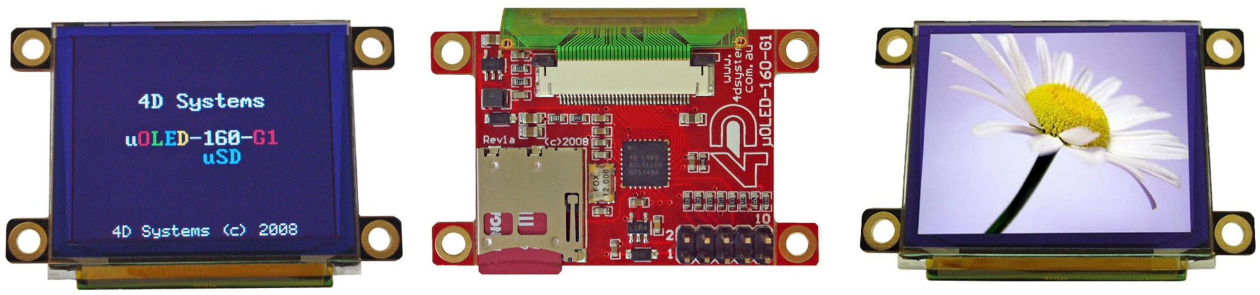
The following modules, available from 4D Systems, can be used for evaluation purposes to discover what the GOLDELOX-SGC processor has to offer.



uOLED-96-G1(SGC): 0.96" PMOLED, 65K Colour, Serial Display Module



uOLED-128-G1(SGC): 1.5" PMOLED, 65K Colour, Serial Display Module



uOLED-160-G1(SGC): 1.7" PMOLED, 65K Colour, Serial Display Module

6. Appendix B : GSGCdef.h

```
/********************************************************************************
4D LABS PTY. LTD. COPYRIGHT 2009.

THIS SOFTWARE IS PROVIDED "AS IS."  4D LABS EXPRESSLY DISCLAIM ANY WARRANTY OF
ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES  OF  MERCHANTABILITY,  FITNESS  FOR  A  PARTICULAR  PURPOSE,  OR  NON-
INFRINGEMENT. IN NO EVENT SHALL 4D LABS BE LIABLE FOR ANY INCIDENTAL, SPECIAL,
INDIRECT  OR  CONSEQUENTIAL  DAMAGES,  LOST  PROFITS  OR  LOST  DATA,  HARM  TO  YOUR
EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY
CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENCE THEREOF), ANY
CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
********************************************************************************/

/********************************************************************************
Name: GOLDELOX-SGC Host Serial Commands Definitions
File Name: GSGCdef.h
Description: Host Serial Interface Commands Definitions
********************************************************************************/

#ifndef GSGC_DEF_H
#define GSGC_DEF_H

// GSGC PROTOCOL CONSTANTS
#define ACK 0x06                    // Acknowledge
#define NAK 0x15                    // Not Acknowledge

// GSGC SWITCH-JOYSTICK CONSTANTS
#define SW1_UP 0x10                 // SW1 or Joystick UP
#define SW2_LEFT 0x20               // SW2 or Joystick LEFT
#define SW3_DOWN 0x30               // SW3 or Joystick DOWN
#define SW4_RIGHT 0x40              // SW4 or Joystick RIGHT
#define SW5_FIRE 0x50               // SW5 or Joystick FIRE

// GSGC GRAPHICS CONSTANTS
#define COLOR8 0x08                 // 8 bit Colour Mode
#define COLOR16 0x10                // 16 bit Colour Mode
#define BUTTONUP 0x01               // Button Up Mode
#define BUTTONDOWN 0x00             // Button Down Mode
#define RED 0xF800                  // RED
#define GREEN 0x07E0                // GREEN
#define BLUE 0x001F                 // BLUE
#define BLACK 0x0000                // BLACK
#define WHITE 0xFFFF                // WHITE

// GSGC TEXT CONSTANTS
#define FONT1 0x00                  // 5x7 Internal Font
#define FONT2 0x01                  // 8x8 Internal Font
#define FONT3 0x02                  // 8x12 Internal Font

// GSGC GENERAL COMMANDS DEFINITIONS
#define GSGC_AUTOBAUD 0x55          // Auto Baud Command
#define GSGC_VERSION 0x56           // Device Info Request
#define GSGC_BACKGND 0x42           // Change Background Colour
#define GSGC_CLS 0x45               // Clear Screen
```

```
#define GSGC_DISPCONT 0x59          // Display Control Functions
#define GSGC_SWITCHSTAT 0x4A        // Get Switch-Buttons Status
#define GSGC_SWITCHSTATWAIT 0x6A    // Get Switch-Buttons Status with Timeout
#define GSGC_SOUND 0x4E             // Generate a Tone

// GSGC GRAPHICS COMMANDS DEFINITIONS
#define GSGC_ADDBM 0x41             // Add User Bitmap
#define GSGC_CIRCLE 0x43            // Draw Circle
#define GSGC_BM 0x44                // Draw User Bitmap
#define GSGC_TRIANGLE 0x47          // Draw Triangle
#define GSGC_IMAGE 0x49             // Draw Image-Icon
#define GSGC_LINE 0x4C              // Draw Line
#define GSGC_PIXEL 0x50             // Draw Pixel
#define GSGC_RDPIXEL 0x52           // Read Pixel
#define GSGC_SCRNCOPYPASTE 0x63     // Screen Copy-Paste
#define GSGC_POLYGON 0x67           // Draw Polygon
#define GSGC_SETPEN 0x70            // Set Pen Size
#define GSGC_RECTANGLE 0x72         // Draw Rectangle

// GSGC TEXT COMMANDS DEFINITIONS
#define GSGC_SETFONT 0x46           // Set Font
#define GSGC_SETOPAQUE 0x4F         // Set Transparent-Opaque Text
#define GSGC_STRINGGFX 0x53         // "String" of ASCII Text (graphics format)
#define GSGC_CHARTXT 0x54           // ASCII Character (text format)
#define GSGC_BUTTONTXT 0x62         // Text Button
#define GSGC_STRINGTXT 0x73         // "String" of ASCII Text (text format)
#define GSGC_CHARGFX 0x74           // ASCII Character (graphics format)

// GSGC EXTENDED COMMANDS HEADER DEFINITION
#define GSGC_EXTCMD 0x40            // Extended Command Header

// GSGC MEMORY CARD COMMANDS DEFINITIONS
#define GSGC_MCAP 0x41              // Set Address Pointer of Memory Card
#define GSGC_MCCOPYSAVE 0x43        // Screen Copy-Save to Memory Card
#define GSGC_MCIMAGE 0x49           // Display Image-Icon from Memory Card
#define GSGC_MCOBJ 0x4F             // Display Object from Memory Card
#define GSGC_MCRUN 0x50             // Run Script (4DSL) Program from Card
#define GSGC_MCRDSECTOR 0x52        // Read Sector Block Data from Memory Card
#define GSGC_MCVIDEO 0x56           // Display Video Clip from Memory Card
#define GSGC_MCWRSECTOR 0x57        // Write Sector Block Data to Memory Card
#define GSGC_MCINIT 0x69            // Initialise Memory Card
#define GSGC_MCRDBYTE 0x72          // Read Byte Data from Memory Card
#define GSGC_MCWRBYTE 0x77          // Write Byte Data to Memory Card

// GSGC SCRIPTING COMMANDS DEFINITIONS
#define GSGC_MCAP 0x41              // Set Address Pointer of Memory Card
#define GSGC_DELAY 0x07             // Delay
#define GSGC_SETCNTR 0x08           // Set Counter
#define GSGC_DECCNTR 0x09           // Decrement Counter
#define GSGC_JMPNZ 0x0A             // Jump to Address If Counter Not Zero
#define GSGC_JMP 0x0B               // Jump to Address
#define GSGC_EXIT 0x0C              // Exit-Terminate Script Program

#endif
```

Proprietary Information

Disclaimer of Warranties & Limitation of Liability