4D LABS

PICASO-SGC Command Set

Software Interface Specification

Document Date: 1st March 2011
Document Revision: 6.0

Note: This manual applies to the PICASO-SGC Revision 18 PmmC files and above.

Table of Contents

1. Host Interface

The PICASO-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

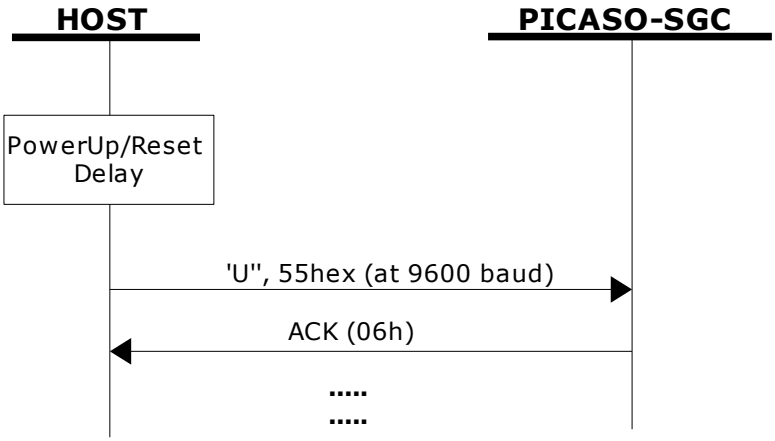
The PICASO-SGC is a slave device and all communication and events must be initiated by the host. Each command is made up of a sequence of data bytes. When a command is sent to the device and the operation is completed, it will always return a response. For a command that has no specific response the 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 PICASO-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 Power-up and Reset

When the PICASO-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 PICASO-SGC to settle. Do not attempt to communicate with the device during this period. The device may send garbage on its TX Data line during this period, the host should disable its Rx Data reception.
  Note: For applications that utilise memory cards with large capacity, allow up to 3 seconds for the card initialisation.
- The host must send the Auto-Baud command, capital '**U**' (55hex), at 9600 baud and wait for an ACK (06hex) from the PICASO-SGC. The default baud rate of the PICASO-SGC is 9600 bps and the host must communicate initially with the device at this speed. The "**Set new Baud-Rate**" command can then be used to change to a different baud-rate if desired.
  Note: Unlike the GOLDELOX-SGC, the PICASO-SGC does not have an Auto-Baud feature. To maintain compatibility, the Auto-Baud command must still be sent (at 9600 baud) for the PICASO-SGC.
- Once the host receives the ACK, the PICASO-SGC is now ready to accept commands from the host.



1.3 Splash Screen on Power Up

The PICASO-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 PICASO-SGC will display a built-in splash screen. If the host has transmitted the Auto-Baud command, the screen will remain blank. This wait period is for those customer specific applications where the splash screen is undesired.

1.4 4DSL Memory Card Script Program

The complete command set for the PICASO-SGC device is listed in section 2 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 2.8.

1.5 Auto Run Memory Card Script Program

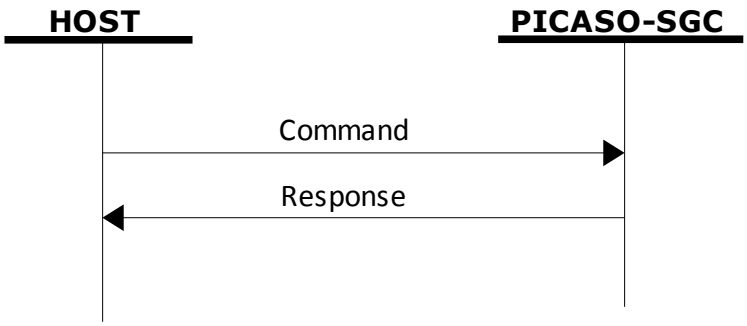
The PICASO-SGC has a feature that will auto run a preloaded script program on power-up. The PICASO-SGC device is equipped to accept memory cards and when using the FAT file system, upon power-up, if a 4DSL script program file called ‘**autoexec.4ds**’ exists on the memory card, the PICASO-SGC will automatically run this script program. This is a useful feature for those stand alone applications where the device does not require a host controller to send commands to the PICASO-SGC to play a slide show of images, video clips, etc.

The user will have to create and upload a slide show composition to the card to benefit from this auto play feature.

Refer to **'Section 4: Appendix B'** at the end of this document for a quick guide to creating scripting files using the FAT-Controller software tool available from 4D Labs.

2. Command Set

The command interface between the PICASO-SGC and the host is via the serial interface. Easy to learn commands provide complete access to all the available functions of the PICASO-SGC. 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
- Touch Screen Commands
- SD Memory Card Commands (Low-Level/RAW)
- SD Memory Card Commands (FAT-Level/DOS)
- Script Control (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.**

2.1 General Commands

Summary of Commands in this section:

- AutoBaud – **55hex**
- Set new Baud-Rate - **51hex**
- Version-Device Info Request – **56hex**
- Replace Background Colour – **42hex**
- Clear Screen – **45hex**
- Display Control Functions – **59hex**
- Set Volume - **76hex**
- Sleep – **5Ahex**
- Read GPIO Pin - **61hex**
- Write GPIO Pin - **69hex**
- Read GPIO Bus - **79hex**
- Write GPIO Bus – **57hex**

2.1.1 AutoBaud - 55hex

Serial Cmd	**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 PICASO-SGC (at 9600 baud) after power-up or reset.	
4DSL Sample	General.4DScript	

2.1.2 Set new Baud Rate - 51hex

Serial Cmd	**cmd, rate**	
4DSL Cmd	**SetBaud(rate)**	
	cmd	**51**(hex) or **Q**(ascii) : Command header byte
	rate	**00**hex : 110 Baud **01**hex : 300 Baud **02**hex : 600 Baud **03**hex : 1200 Baud **04**hex : 2400 Baud **05**hex : 4800 Baud **06**hex : 9600 Baud **07**hex : 14400 Baud **08**hex : 19200 Baud **09**hex : 31250 Baud **0A**hex : 38400 Baud **0B**hex : 56000 Baud **0C**hex : 57600 Baud **0D**hex : 115200 Baud **0E**hex : 128000 Baud **0Fhex :** 256000 Baud
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command changes the Baud-Rate on the fly.	
4DSL Sample	General.4DScript	

2.1.3 Version-Device Info Request - 56hex

Serial Cmd	**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 **FF hex** : Unknown
	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 **FF hex** : Unknown
Description	This command requests all the necessary information from the device about its characteristics and capability.	
4DSL Sample	General.4DScript	

2.1.4 Replace Background Colour - 42hex

Serial Cmd	**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.	
Example	**Command Data:** 42hex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	
4DSL Sample	General.4DScript	

2.1.5 Clear Screen - 45hex

Serial Cmd	**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	
Example	**Command Data:** 45hex (Clear the screen).	
4DSL Sample	General.4DScript	

2.1.6 Display Control Functions - 59hex

Serial Cmd	cmd, mode, value	
4DSL Cmd	Control(mode, value)	
	cmd	**59**(hex) or **Y**(ascii) : Command header byte
	mode	**00hex : Backlight Control** *(Does not apply to uVGA-II)* value = 00hex : BACKLIGHT OFF value = 01hex : BACKLIGHT ON **01hex : Display ON/OFF** *(Does not apply to uVGA-II)* value = 00hex : DISPLAY OFF value = 01hex : DISPLAY ON **02hex : Contrast Adjust** *(Does not apply to uVGA-II)* value = 00hex – 0Fhex : CONTRAST RANGE **03hex : Display PowerUp-Shutdown (low power mode)** value = 00hex : DISPLAY SHUTDOWN value = 01hex : DISPLAY POWERUP **04hex : Display Orientation** *(Does not apply to uVGA-II)* value = 01hex : LANDSCAPE value = 02hex : LANDSCAPE_R (Rotated) value = 03hex : PORTRAIT value = 04hex : PORTRAIT_R (Rotated) **Note:** The orientation command is effective for all graphics and text functions except for images and video. For example, if the natural orientation of the display is PORTRAIT and the orientation was set to LANDSCAPE, images and video will remain in the PORTRAIT mode. To achieve a true LANDSCAPE format display, the DISP software tool must be used to set the proper display hardware parameters which is then programmed into the PICASO-SGC. **05hex : Touch Control** *(Does not apply to uVGA-II)* value = 00hex : Enable Touch Screen value = 01hex : Disable Touch Screen value = 02hex : Reset the Active region to the entire Screen **06hex : Image Format (for @43hex and @49hex commands)** value = 00hex : New Format, includes header (type2) value = 01hex : Old Format, no header (type1) **08hex : Protect FAT** value = 01hex : PROTECT value = 00hex : UNPROTECT **Note:** This command protects the FAT file system (if present) on the card from being read or written to by low level (RAW) commands. If the memory card contains any FAT (FAT16 or FAT32) partition, when the initialise command is executed or the device comes out of a reset, FAT Protection is turned ON automatically. This means the host will not be able to access the card using Low-Level (RAW) read or write commands unless it subsequently turns off the FAT protection. For a 'Non Standard' card containing two partitions, one FAT and one RAW (Partition type **DA**hex), the default is "PROTECT". In this case FAT reads and writes will occur to the FAT partition and RAW reads and

		writes will be offset into the RAW partition. i.e. a write to sector 0 will write to sector 0 in the raw partition. FAT32 is currently not supported. If you mount a FAT32 formatted disk, you will not be able to access it at all, both FAT and RAW commands will fail. You can either reformat the memory card as FAT or unprotect the card and replace sector 0 with 512 hex 00s. **09hex : Display Page Select***(Applies to uVGA-II only)* value = Depends on resolution selected. **e.g**. value = 00hex to 04hex for 320x240 resolution **0Ahex : Read Page Select** *(Applies to uVGA-II only)* value = Depends on resolution selected. **e.g**. value = 00hex to 04hex for 320x240 resolution **0Bhex : Write Page Select** *(Applies to uVGA-II only)* value = Depends on resolution selected. **e.g**. value = 00hex to 04hex for 320x240 resolution **0Chex : Screen Resolution** *(Applies to uVGA-II only)* value = 00hex : 320 x 240 resolution value = 01hex : 640 x 480 resolution value = 02hex : 800 x 480 resolution value = XXhex : Custom resolution set through DISP tool **0Dhex : Screen saver startup delay** *(Applies to uVGA-II only)* value = 00hex – FFhex :Value in seconds **0Ehex : Screen Saver 'next line' delay** *(Applies to uVGA-II only)* value = 00hex – FFhex :Value in seconds **Note:** *On the uVGA-II pages start scrolling as soon as the Screen Saver Startup Delay expires since the last command sent.*
	value	See mode description above.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command has multiple features. Refer to individual notes above.	
4DSL Sample	General.4DScript	

2.1.7 Set Volume - 76hex

Serial Cmd	**cmd, volume**	
4DSL Cmd	**Volume(volume)**	
	cmd	**76**(hex) or **v**(ascii) : Command header byte
	volume	Volume is in the range of 8 – 127 (08hex - 7Fhex) **08hex :** Minimum Volume. **7Fhex :** Maximum Volume. Special Values: **00hex :** Mute. **01hex :** Volume Down 8. **03hex :** Volume Down. **FDhex :** Volume Up. **FEhex :** Volume Up 8. **FFhex :** Mute Off.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command sets the volume level of the AUDIO pin output.	
4DSL Sample	TestFAT.4DScript	

2.1.8 Sleep - 5Ahex

Serial Cmd	**cmd, options, sleep_time**	
4DSL Cmd	**Sleep(options, sleep_time)**	
	cmd	**5A**(hex) or **Z**(ascii) : Command header byte
	options	**80**hex : Turn off uSD (must reinit manually) (*Does not apply to uVGA-II)* **40**hex : Turn off Audio (*Does not apply to uVGA-II)* (Automatically turns on when used) **20**hex : Turn off Touch (*Does not apply to uVGA-II)* **08**hex : Wake-Up on P1 state change **04**hex : Wake-Up on P0 state change **02**hex : Wake-Up on Touch (*Does not apply to uVGA-II)* **01**hex : Wake-Up on Serial
	sleep_time	1 byte sleep time in seconds: **0 :** Sleep forever (wake up on P0 or P1 or RESET). **1-255 :** Seconds to Sleep.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte when wake-up on event **15**(hex) : NAK byte if unsuccessful
Description	Puts parts of the PICASO-SGC into a lower power state to conserve power and optionally waits for certain conditions to wake it up. If it is desired to turn off the display then do this first using ”Display Control Functions - 59hex” command (mode 1 option).	
4DSL Sample	General.4DScript	

2.1.9 Read GPIO Pin - 69hex

Serial Cmd	**cmd, pin**	
4DSL Cmd	**ReadPin(pin)**	
	cmd	**69**(hex) or **i**(ascii) : Command header byte
	pin	Select one of P0 – P15 pins: **0 - 15** : P0 - P15
Response	**pin_status**	
	pin_status	Returns 1 byte: **00**(hex) : Pin is Low **01**(hex) : Pin is High
Description	Reads the state of the selected GPIO pin (P0-P15). Returns 0 or 1.	
4DSL Sample	General.4DScript	

2.1.10 Write to GPIO Pin - 79hex

Serial Cmd	**cmd, pin, value**	
4DSL Cmd	**WritePin(pin, value)**	
	cmd	**79**(hex) or **y**(ascii) : Command header byte
	pin	Select one of P0 – P15 pins: **0 - 15** : P0 - P15
	value	**0** : write Low to pin **1** : write High to pin
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Writes a LOW or HIGH to the specified GPIO pin (P0-P15).	
4DSL Sample	General.4DScript	

2.1.11 Read GPIO Bus - 61hex

Serial Cmd	**cmd**	
4DSL Cmd	**ReadBus**	
	cmd	**61**(hex) or **a**(ascii) : Command header byte
Response	**bus_status**	
	bus_status	Returns 1 byte BUS0-BUS7 status.
Description	Reads the state of the GPIO Bus (BUS0-BUS7 or P8-P15).	
4DSL Sample	General.4DScript	

2.1.12 Write to GPIO Bus - 57hex

Serial Cmd	**cmd, value**	
4DSL Cmd	**WriteBus(value)**	
	cmd	**57**(hex) or **W**(ascii) : Command header byte
	value	1 byte value to write to GPIO Bus (BUS0-BUS7).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Writes a byte to the GPIO Bus (BUS0-BUS1 or P8-P15).	
4DSL Sample	General.4DScript	

2.2 Graphics Commands

Summary of Commands in this section:

- Add User Bitmap Character – **41hex**
- Draw User Bitmap Character – **44hex**
- Draw Circle – **43hex**
- Draw Triangle – **47hex**
- Draw Image-Icon – **49hex**
- Set Background colour **– 4Bhex**
- Draw Line – **4Chex**
- Draw Polygon – **67hex**
- Draw Rectangle – **72hex**
- Draw Ellipse – **65hex**
- Draw Pixel – **50hex**
- Read Pixel – **52hex**
- Screen Copy-Paste – **63hex**
- Replace colour **– 6Bhex**
- Set Pen Size – **70hex**

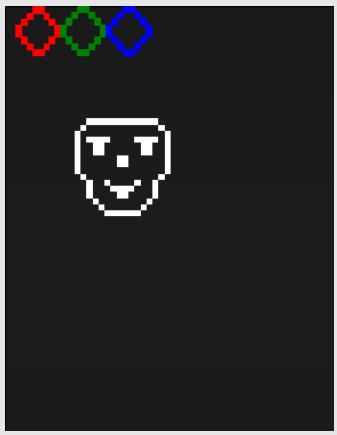
2.2.1 Add User Bitmap Character - 41hex

Serial Cmd	**cmd, group, char_idx, data1, data2, .. , dataN**	
4DSL Cmd	**AddUserBitmap8(char_idx, data1, data2, .. , dataN)** ***(For 8x8 bitmap format)*** **AddUserBitmap16(char_idx, data1, data2, .. , dataN)** (For ***16x16 bitmap format)*** **AddUserBitmap32(char_idx, data1, data2, .. , dataN)** ***(For 32x32 bitmap format)***	
	cmd	**41**(hex) or **A**(ascii) : Command header byte
	group	Selects the appropriate bitmap format **00**hex : selects the 8x8 bitmap format **01**hex : selects the 16x16 bitmap format **02**hex : selects the 32x32 bitmap format
	char_idx	Bitmap character index to add to memory. 0 to 63 (**00**hex to **3F**hex): 64 characters of 8x8 format for group = **00**hex 0 to 15 (**00**hex to **0F**hex): 16 characters of 16x16 format for group = **01**hex 0 to 7 (**00**hex to **07**hex): 4 characters of 32x32 format for group = **02**hex
	data1..dataN	Number of data bytes that make up the composition and format of the bit-mapped character. The 8x8 bitmap composition is 1 byte wide (8bits) by 8 bytes deep which makes N = 1x8 = 8. The 16x16 bitmap composition is 2 bytes wide (16bits) by 16 bytes deep which makes N = 2x16 = 32. The 32x32 bitmap composition is 4 bytes wide (32bits) by 32 bytes deep hence N = 4x32 = 128
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will add a user defined bit-mapped character into the internal memory. There are 3 different size bitmaps available, 8x8 format, 16x16 format and 32x32 format. The desired format is selected by specifying the appropriate value in 'group'.	
Example	**Command Data:** 41hex, 00hex, 01hex, 18hex, 24hex, 42hex, 81hex, 81hex, 42hex, 24hex, 18hex Adds and saves a user defined 8x8 bitmap (group 0) as character index 1 into memory.	
4DSL Sample	GraphicsPt1.4DScript	

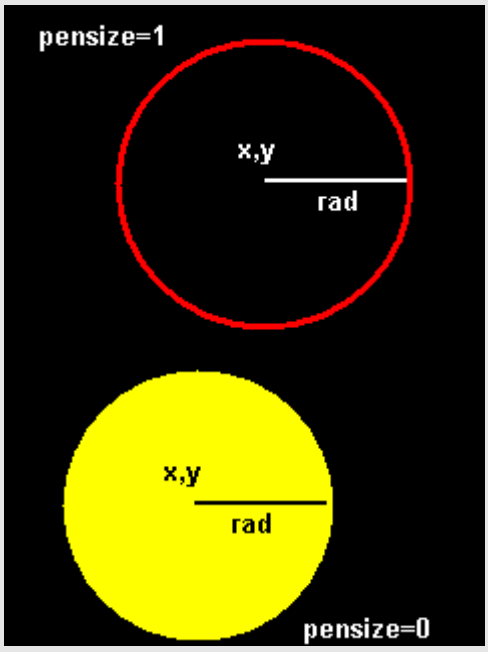
b7	b6	b5	b4	b3	b2	b1	b0		← Data Bits
			■	■					**data1 (18**hex**)**
		■			■				**data2 (24**hex**)**
	■					■			**data3 (42**hex**)**
■							■		**data4 (81**hex**)**
■							■		**data5 (81**hex**)**
	■					■			**data6 (42**hex**)**
		■			■				**data7 (24**hex**)**
			■	■					**data8 (18**hex**)**

Example of 8x8 User defined bitmap

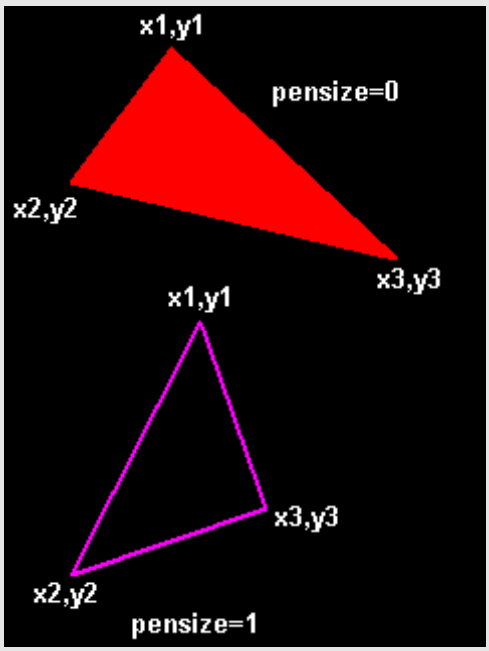
2.2.2 Draw User Bitmap Character - 44hex

Serial Cmd	**cmd, group, char_idx, x**(msb:lsb), **y**(msb:lsb), **colour**(msb:lsb)	
4DSL Cmd	**DrawUserBitmap8(char_idx, x, y, colour)** ***(8x8 bitmap)*** **DrawUserBitmap16(char_idx, x, y, colour)** ***(16x16 bitmap)*** **DrawUserBitmap32(char_idx, x, y, colour)** ***(32x32 bitmap)***	
	cmd	**44**(hex) or **D**(ascii) : Command header byte
	group	Selects the appropriate bitmap format **00**hex : selects the 8x8 bitmap format **01**hex : selects the 16x16 bitmap format **02**hex : selects the 32x32 bitmap format
	char_idx	Bitmap character index to draw from the previously added bitmap characters into memory. 0 to 63 (**00**hex to **3F**hex): 64 characters of 8x8 format when group = **00**hex 0 to 15 (**00**hex to **0F**hex): 16 characters of 16x16 format when group = **01**hex 0 to 7 (**00**hex to **07**hex): 4 characters of 32x32 format when group = **02**hex
	x	Horizontal display position of the bitmap character (2 bytes).
	y	Vertical display position of the bitmap character (2 bytes).
	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. 	
Examples	**Command Data:** 44hex, 00hex, 01hex , 00hex, 00hex, 00hex, 00hex, F8hex, 00hex (Display 8x8 bitmap character (group 0) index 1 at x = 0, y = 0, colour = RED). **Command Data:** 44hex, 00hex, 02hex, 00hex, 08hex, 00hex, 00hex, 07hex, E0hex (Display 8x8 bitmap character (group 0) index 2 at x = 8, y = 0, colour = GREEN). **Command Data:** 44hex, 00hex, 03hex, 00hex, 10hex, 00hex, 08hex, 00hex, 1Fhex (Display 8x8 bitmap character (group 0) index 3 at x = 16, y = 8, colour = BLUE).	
4DSL Sample	GraphicsPt1.4DScript	

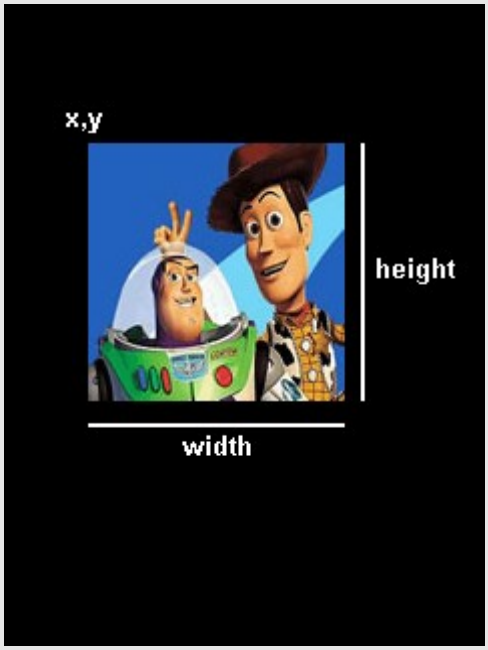
2.2.3 Draw Circle - 43hex

Serial Cmd	**cmd, x**(msb:lsb)**, y**(msb:lsb)**, radius**(msb:lsb)**, 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 (2 bytes).
	y	Vertical position of the circle centre (2 bytes).
	radius	Radius of the circle (2 bytes).
	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  	
Example	**Command Data:** 43hex, 00hex, 3Fhex, 00hex, 3Fhex, 00hex, 22hex, 00hex, 1Fhex Draws a RED circle (**001F**hex) centred at x = **63**dec (**003F**hex) and y = **63**dec (**003F**hex) with a radius of **34**dec (**0022**hex).	
4DSL Sample	GraphicsPt1.4DScript	

2.2.4 Draw Triangle - 47hex

Serial Cmd	**cmd,x1**(msb:lsb)**,y1**(msb:lsb)**,x2**(msb:lsb)**,y2**(msb:lsb)**,x3**(msb:lsb)**,y3**(msb:lsb)**,color**(msb:lsb)	
4DSL Cmd	**Triangle(x1, y1 ,x2 , y2, x3, y3, color**)	
	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 (2 bytes per each parameter).
	color	2 bytes 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  	
4DSL Sample	GraphicsPt1.4DScript	

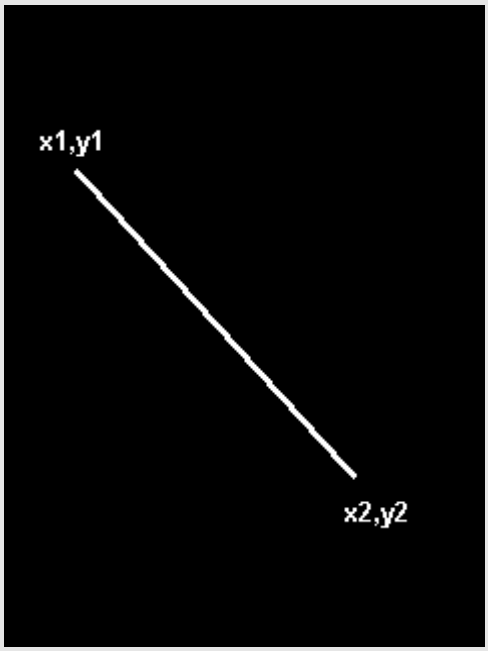
2.2.5 Draw Image-Icon - 49hex

Serial Cmd	**cmd, x**(msb:lsb), **y**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb), **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, 2 bytes).
	y	Image vertical start position (top left corner, 2 bytes).
	width	Horizontal size of the image (2 bytes).
	height	Vertical size of the image (2 bytes).
	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 Sample	GraphicsPt1.4DScript	

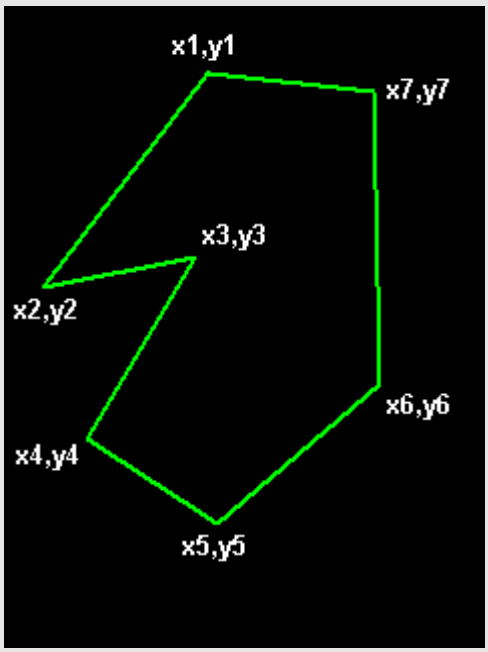
2.2.6 Set Background colour - 4Bhex

Serial Cmd	**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.	
Example	**Command Data:** 4Bhex, FFhex, FFhex This example sets the background colour value to FFFFhex (White).	
4DSL Sample	General.4DScript	

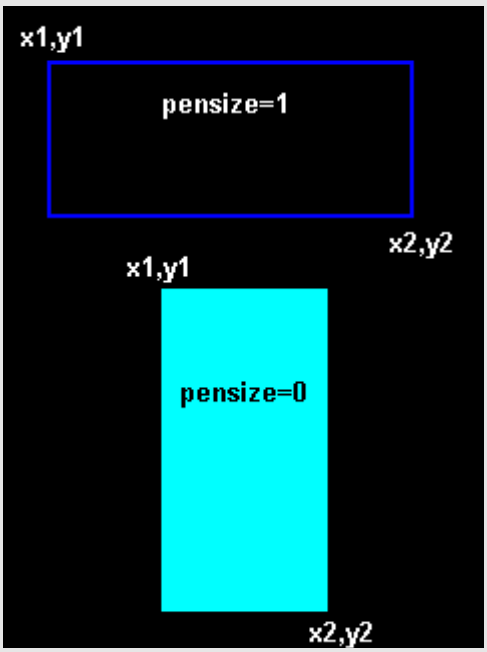
2.2.7 Draw Line – 4Chex

Serial Cmd	**cmd, x1**(msb:lsb)**, y1**(msb:lsb)**, x2**(msb:lsb)**, y2**(msb:lsb)**, 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 (2 bytes).
	y1	Top left vertical start position of line (2 bytes).
	x2	Bottom right horizontal end position of line (2 bytes).
	y2	Bottom right vertical end position of line (2 bytes).
	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.  	
Example	**Command Data:** 4Chex, 00hex, 00hex, 00hex, 00hex, 00hex, 7Fhex, 00hex, 7Fhex, FFhex, FFhex Draws a WHITE line (FFFFhex) from (x1 = 0000hex, y1 = 0000hex) to (x2 = 007Fhex, y2 = 007Fhex).	
4DSL Sample	GraphicsPt2.4DScript	

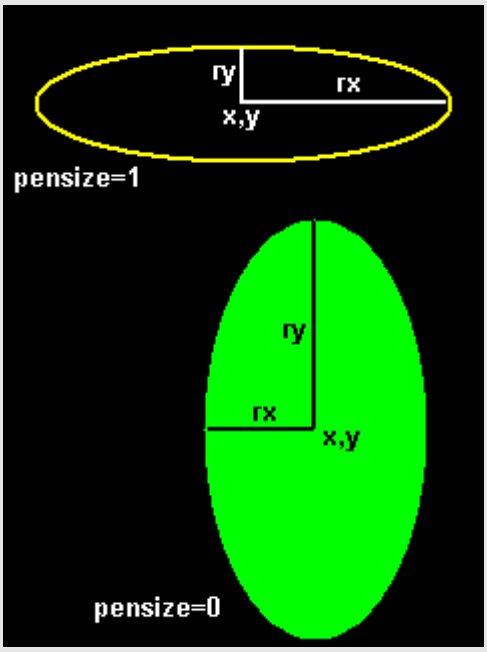
2.2.8 Draw Polygon - 67hex

Serial Cmd	**cmd, vertices, x1**(msb:lsb)**, y1**(msb:lsb)**, .. , xn**(msb:lsb)**, yn**(msb:lsb)**, 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 triangle. These can be specified in any fashion. Each parameter is 2 bytes.
	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.  	
4DSL Sample	GraphicsPt2.4DScript	

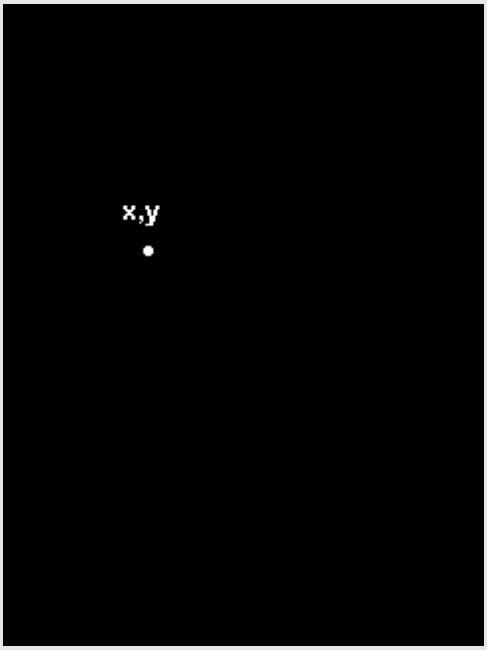
2.2.9 Draw Rectangle - 72hex

Serial Cmd	**cmd, x1**(msb:lsb)**, y1**(msb:lsb)**, x2**(msb:lsb)**, y2**(msb:lsb)**, 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 (2 bytes).
	y1	Top left vertical start position of rectangle (2 bytes).
	x2	Bottom right horizontal end position of rectangle (2 bytes).
	y2	Bottom right vertical end position of rectangle (2 bytes).
	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.  	
4DSL Sample	GraphicsPt2.4DScript	

2.2.10 Draw Ellipse - 65hex

Serial Cmd	**cmd, x**(msb:lsb)**, y**(msb:lsb)**, rx**(msb:lsb)**, ry**(msb:lsb)**, colour**(msb:lsb)	
4DSL Cmd	**Ellipse(x, y, rx, ry, colour)**	
	cmd	**65**(hex) or **e**(ascii) : Command header byte
	x	Horizontal position of the ellipse centre (2 bytes).
	y	Vertical position of the ellipse centre (2 bytes).
	rx	Radius in the x-axis (2 bytes).
	ry	Radius in the y-axis (2 bytes).
	colour	2 bytes define the ellipse colour.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command will draw a coloured ellipse centred at **(x, y)** with its shape determined by the values of **rx** (the x radius) and **ry** (the y radius). The ellipse can be either solid or wire frame (empty) depending on the value of the Pen Size (see **Set Pen Size** command). Pen Size = 0 : ellipse is solid Pen Size = 1 : ellipse is wire frame.  	
4DSL Sample	GraphicsPt2.4DScript	

2.2.11 Draw Pixel - 50hex

Serial Cmd	**cmd, x**(msb:lsb)**, y**(msb:lsb)**, colour**(msb:lsb)	
4DSL Cmd	**Pixel(x, y, colour)**	
	cmd	**50**(hex) or **P**(ascii) : Command header byte
	x	Horizontal position of the pixel (2 bytes).
	y	Vertical position of the pixel (2 bytes).
	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. 	
Example	**Command Data:** 50hex, 00hex, 01hex, 00hex, 0Ahex, FFhex, FFhex Draw a WHITE pixel (FFFFhex) at location (x = 0001hex, y = 000Ahex).	
4DSL Sample	GraphicsPt2.4DScript	

2.2.12 Read Pixel - 52hex

Serial Cmd	**cmd, x**(msb:lsb)**, y**(msb:lsb)	
4DSL Cmd	**ReadPixel(x, y)**	
	cmd	**52**(hex) or **R**(ascii) : Command header byte
	x	Horizontal position of the pixel (2 bytes).
	y	Vertical position of the pixel (2 bytes).
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.	
Example	**Command Data:** 52hex, 00hex, 01hex, 00hex, 0Ahex **PICASO-SGC Response:** 00hex, 1Fhex Reads a BLUE pixel (001Fhex) at location (x = 0001hex, y = 000Ahex).	
4DSL Sample	GraphicsPt2.4DScript	

2.2.13 Screen Copy-Paste - 63hex

Serial Cmd	**cmd, xs**(msb:lsb), **ys**(msb:lsb), **xd**(msb:lsb), **yd**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb)	
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, 2bytes).
	ys	Top left vertical start position of screen area to be copied (source, 2 bytes).
	xd	Top left horizontal start position of where copied area is to be pasted (destination, 2 bytes).
	yd	Top left vertical start position of where copied area is to be pasted (destination, 2 bytes).
	width	Width of screen area to be copied (source, 2 bytes).
	height	Height of screen area to be copied (source, 2 bytes).
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.	
4DSL Sample	GraphicsPt2.4DScript	

2.2.14 Replace Colour - 6Bhex

Serial Cmd	**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..	
4DSL Sample	General.4DScript	

2.2.15 Set Pen Size - 70hex

Serial Cmd	**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.	
Examples	**Command Data:** 70hex, 00hex (All objects will be drawn solid). **Command Data:** 70hex, 01hex (All objects will be drawn wire-frame).	
4DSL Sample	GraphicsPt2.4DScript	

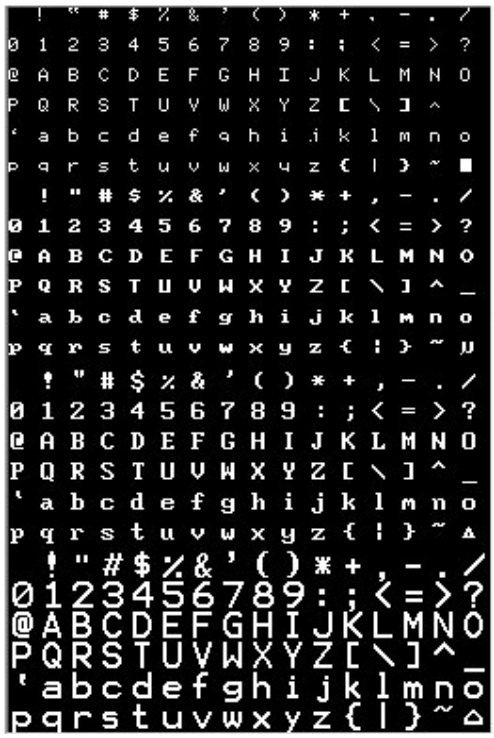
2.3 Text Commands

The PICASO-SGC is shipped with 4 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 PICASO-SGC. The converted font set can then be exported into the **DISP-Tool** utility which can then be downloaded into the PICASO-SGC on-chip flash memory. Both the FONT-Tool and the DISP-Tool are available free from www.4dsystems.com.au

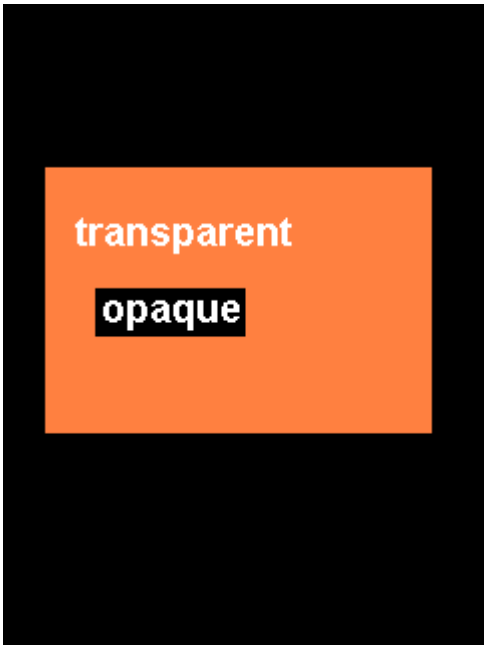
Summary of Commands in this section:

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

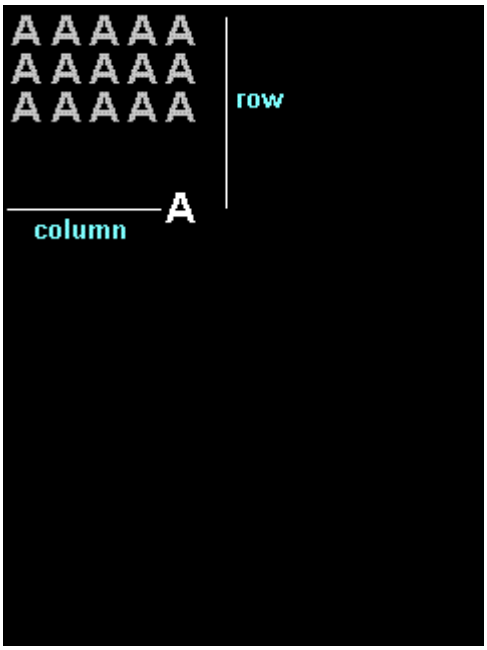
2.3.1 Set Font - 46hex

Serial Cmd	**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 **03**hex : 12x16 largest size font set These fonts can be altered and other fonts can be added with the aid of the FONT-Tool and the DISP-Tool software tools.
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 PICASO-SGC is shipped with 4 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').	
Examples	**Command Data:** 46hex, 00hex (Select small 5x7 font). **Command Data:** 46hex, 01hex (Select medium 8x8 font). **Command Data:** 46hex, 02hex (Select large 8x12 font). **Command Data:** 46hex, 03hex (Select largest 12x16 font).	
4DSL Sample	Text.4DScript	

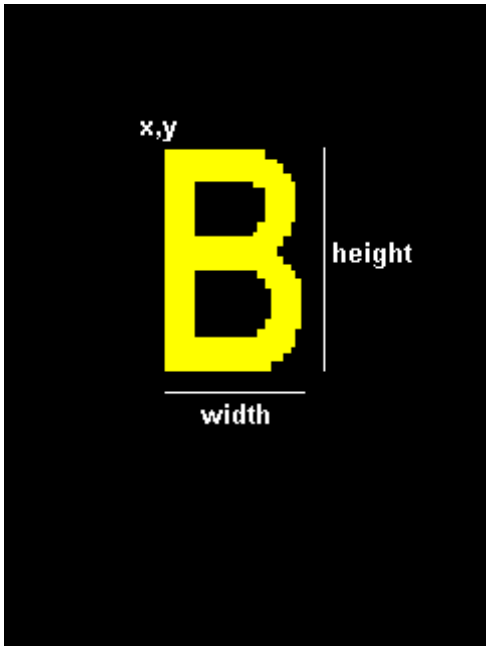
2.3.2 Set Transparent-Opaque Text - 4Fhex

Serial Cmd	**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.	
Examples	**Command Data:** 4Fhex, 00hex (Transparent text mode). **Command Data:** 4Fhex, 01hex (Opaque text mode).  	
4DSL Sample	Text.4DScript	

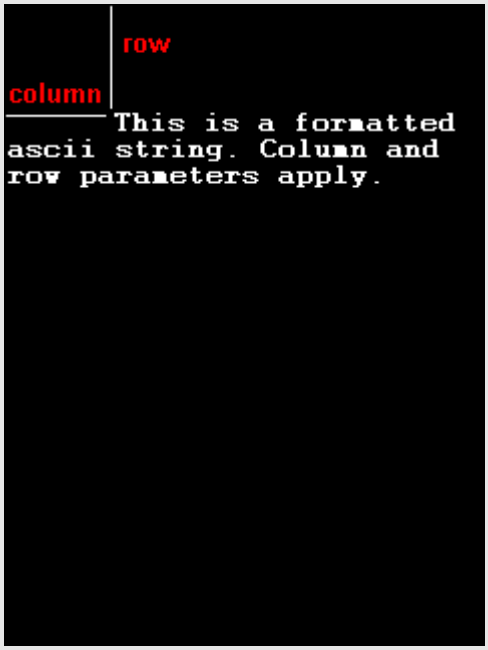
2.3.3 Draw ASCII Character (text format) - 54hex

Serial Cmd	**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).
	row	Vertical position of the character (character units).
	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. Use the ”Set Font - 46hex” command to select the desired font.	
Example	**Command Data:** 54hex, 41hex, 00hex, 00hex, FFhex, FFhex Draw/Display character ‘A’ (41hex) at column = 0, row = 0, colour = white (FFFFhex).  	
4DSL Sample	Text.4DScript	

2.3.4 Draw ASCII Character (graphics format) - 74hex

Serial Cmd	**cmd, char, x**(msb:lsb)**, y**(msb:lsb)**, 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, 2 byte parameter).
	y	Vertical position of the character (pixel units, 2 byte parameter).
	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.  	
4DSL Sample	Text.4DScript	

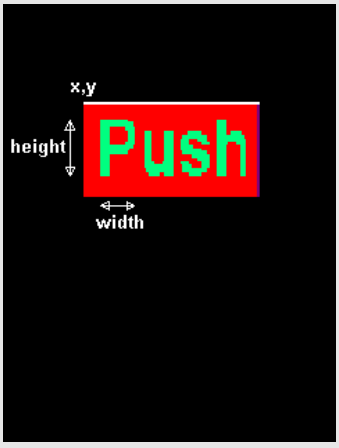
2.3.5 Draw “String” of ASCII Text (text format) - 73hex

Serial Cmd	**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).
	row	Vertical start position of the string (character 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 **3** : 12x16 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**.	
4DSL Sample	Text.4DScript	

2.3.6 Draw "String" of ASCII Text (graphics format) - 53hex

Serial Cmd	**cmd, x**(msb:lsb)**, y**(msb:lsb)**, 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). 2 byte parameter.
	y	Top left vertical start position of the string (pixel units). 2 byte parameter.
	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 **3** : 12x16 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. 1 byte parameter.
	height	This byte defines the height or vertical size multiplier of the character in the string. Effects the total height of the string. 1 byte parameter.
	"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 **512 bytes**.	
4DSL Sample	Text.4DScript	

2.3.7 Draw Text Button - 62hex

Serial Cmd	**cmd, state, x**(msb:lsb)**, y**(msb:lsb)**, 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 (2 bytes).
	y	Top left vertical start position of the button (2 bytes).
	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 **3** : 12x16 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 Sample	Text.4DScript	

2.4 Touch Screen Commands

The following commands are related to Touch Screen support of the PICASO-SGC and they are described in this section.

Summary of Commands in this section:

- Get Touch Coordinates - **6Fhex**
- Wait Until Touch - **77hex**
- Detect Touch Region - **75hex**

Touch screen must be enabled to be able to use the touch commands. To enable touch screen , See section 2.1.6.

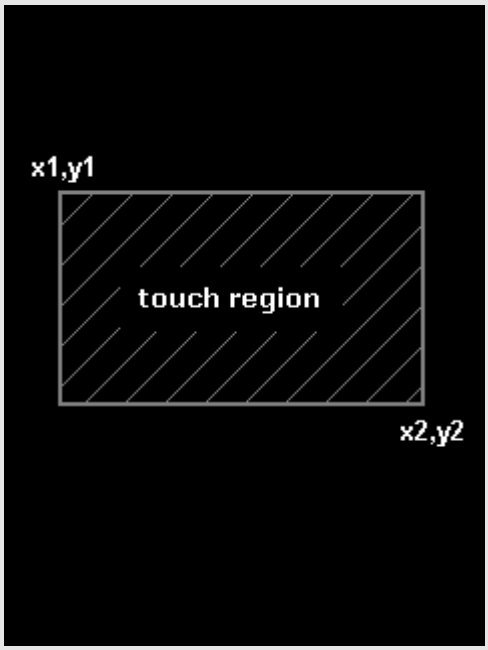
2.4.1 Get Touch Coordinates - 6Fhex

Serial Cmd	**cmd, mode**	
4DSL Cmd	**GetTouch(mode)**	
	cmd	**6F**(hex) or **o**(ascii) : Command header byte
	mode	**00hex :** Wait until any touch activity **01hex :** Wait until Touch press **02hex :** Wait until Touch release **03hex :** Wait until Touch Moving **04hex :** Get Touch status **05hex :** Get Touch coordinates
Response	**x_coord**(msb:lsb)**, y_coord**(msb:lsb)	
	x_coord	2 byte value for x-coordinates of the touch screen. For mode = 04hex only, returns the following: **0** : No Touch Activity **1** : Touch Press **2** : Touch Release **3** : Touch Moving
	y_coord	2 byte value for y-coordinates of the touch screen.
Description	This command returns the x and y coordinate of a touch to the screen for **00hex to 03hex** and **05hex**. Note that for "**mode = 05hex**" the coordinates relate to the last touch activity i.e. they will be same as the coordinates returned to the touch command with mode **00hex - 03hex** or coordinates detected for the touch command with "**mode = 04hex**" when touch activity occurred.	
4DSL Sample	Touch.4DScript	

2.4.2 Wait Until Touch - 77hex

Serial Cmd	**cmd, value**(msb:lsb)	
4DSL Cmd	**WaitTouch(value)**	
	cmd	**77**(hex) or **w**(ascii) : Command header byte
	value	2 byte wait time in milliseconds. Maximum value of 65,535 msec or 65.5 seconds.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	When objects from the memory card such as images are displayed sequentially, a delay can be inserted between subsequent objects. A delay basically has the same effect as a NOP (No Operation) which can be used to determine how long the object stays on the screen before the next object is displayed. If the user touches the display during the delay period, the delay will end immediately. The touch region, if used, is taken into account.	
4DSL Sample	Touch.4DScript	

2.4.3 Detect Touch Region – 75hex

Serial Cmd	**cmd, x1**(msb:lsb), **y1**(msb:lsb), **x2**(msb:lsb), **y2**(msb:lsb)	
4DSL Cmd	**DetectRegion(x1, y1, x2, y2)**	
	cmd	**75**(hex) or **u**(ascii) : Command header byte
	x1	Top left horizontal start position of the region (2 bytes).
	y1	Top left vertical start position of the region (2 bytes).
	x2	Bottom right horizontal end position of the region (2 bytes).
	y2	Bottom right vertical end position of the region (2 bytes).
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command specifies a touch detect region on the screen. This setting will filter out any touch activity outside the region and only touch activity within that region will be returned by "Get Touch Coordinates – 6Fhex" command. 	
4DSL Sample	Touch.4DScript	

2.5 SD Memory Card Commands (Low-Level/RAW)

The commands detailed in this section utilise the SD/microSD memory card which must be connected to the SPI port of the PICASO-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:

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

2.5.1 Initialise Memory Card - @69hex

Serial Cmd	**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.	
4DSL Sample	Raw.4DScript	

2.5.2 Set Address Pointer of Card (RAW) - @41hex

Serial Cmd	**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.	
4DSL Sample	Raw.4DScript	

2.5.3 Read Byte Data from Card (RAW) - @72hex

Serial Cmd	**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.	
4DSL Sample	Raw.4DScript	

2.5.4 Write Byte Data to Card (RAW) - @77hex

Serial Cmd	**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.	
4DSL Sample	Raw.4DScript	

2.5.5 Read Sector Block Data from Card (RAW) - @52hex

Serial Cmd	**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 Sample	Raw.4DScript	

2.5.6 Write Sector Block Data to Card (RAW) - @57hex

Serial Cmd	**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 Card - @77hex"** 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 Sample	Raw.4DScript	

2.5.7 Screen Copy – Save to Card (RAW) - @43hex

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb)**, 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 (2 bytes).
	y	Top left vertical start position of screen area to be copied (2 bytes).
	width	Width of screen area to be copied (source, 2 bytes).
	height	Height of screen area to be copied (source, 2 bytes).
	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 Card"** command. This is a very powerful feature for animating objects, smooth scrolling, or implementing a windowing system. **Notes:** • The **"Screen Copy-Save to 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.	
4DSL Sample	Raw.4DScript	

2.5.8 Display Image-Icon from Card (RAW) - @49hex

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **SectorAdd**(hi:mid:lo)	
4DSL Cmd	**NewUsdImage(x, y, 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, 2 bytes).
	y	Image vertical start position (top left corner, 2 bytes).
	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	Image should be loaded to the SD card using **"uSD RAW"** under **4DGL Platform** option on the **Graphics Composer**. This command displays an image 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)**. Display Control Function (*Syntax: cmd, mode, value*) should be used to set "Image Format" to "New format" i.e mode = 06hex. Value = 00hex. Refer to Sec 2.1.6. Image format is "Old Format" by default. **Note:** To create the images, use the 'uSD Raw' option and a sector offset that is greater than the start of the Raw partition, because Graphics Composer currently does not recognise multiple partitions on a uSD card. To issue the Display Image command the Sector Address is the offset into the raw partition, not the offset you specified in Graphics Composer. The image parameters such as width, height and colour-mode are built into the image header file and do not need to be specified by the host.	
Example1 **Example2**	Partition a 4GB card such that first NonFs sector is 3872257. Set the offset address, under "uSD Raw", to 3872260. Sector address to display the image will be 0000(MSW), 0003(LSW). With "Unprotect FAT" command (not recommended) executed, the Sector addresses to be used with above command will be the same as the sector offset specified in GC file. Partition a 4GB card such that first NonFs sector is 3872257. Set the offset address, under "uSD Raw", to 3872260. Sector address to display the Image will be 003B(MSW), 1604(LSW).	
4DSL Sample	Raw.4DScript	

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb), **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, 2 bytes).
	y	Image vertical start position (top left corner, 2 bytes).
	width	Horizontal size of the image (2 bytes).
	height	Vertical size of the image (2 bytes).
	colourMode	**08**(hex) : 256 colour mode, 8bits/1byte per pixel. **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	Image should be loaded to the SD card using **Serial Command Platform** option on the **Graphics Composer**. 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 using **"Screen Copy-Save to Card"** or an image saved through the Graphics Composer. Position of the image to be displayed is specified by **(x, y).** Other parameters can be extracted from the text file created on the Graphics Composer after burning the slide show on the SD card under serial platform option. If the previously stored image was in 8 bit colour format (1 byte per pixel) or 16 bits (2 bytes per pixel) then this must be specified in the **colourMode** byte parameter. Do not store an image/icon in one colour format then display it in another colour format, this will result in a corrupted image. Display Control Function (*Syntax: cmd*, *mode*, *value*) should be used to set "Image Format" to "Old format" i.e mode = 06hex. Value = 01hex. Refer to Sec 2.1.6. Image format is "Old Format" by default. **Notes:** • The **"Screen Copy-Save to 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.	
4DSL Sample	Raw.4DScript	

2.5.9 Display Object from Card (RAW) - @4Fhex

Serial Cmd	**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 Sample	Raw.4DScript	

2.5.10 Display Video-Animation Clip from Card (RAW) - @56hex

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **delay, SectorAdd**(hi:mid:lo)	
4DSL Cmd	**NewVideo(x, y, delay, 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, 2 bytes).
	y	Video vertical start position (top left corner, 2 bytes).
	delay	1 byte inter-frame delay in milliseconds. This parameter can be used to determine the speed of the video playback.
	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	Video/Animation should be loaded to the SD card using **"uSD RAW"** under **4DGL Platform** option on the **Graphics Composer**. 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. **Note:** To create the video, use the 'uSD Raw' option and a sector offset that is greater than the start of the Raw partition, because Graphics Composer currently does not recognise multiple partitions on a uSD card. To issue the Display Video command the Sector Address is the offset into the raw partition, not the offset you specified in Graphics Composer. The video format used by this command is different from the video format used in the GOLDELOX-SGC format. The video parameters such as width, height and colour-mode are built into the video header file and do not need to be specified by the host.	
Example1 **Example2**	Partition a 4GB card such that first NonFs sector is 3872257. Set the offset address, under "uSD Raw", to 3872260. Sector address to display the video will be 0000(MSW), 0003(LSW). With "Unprotect FAT" command (not recommended), the Sector addresses to be used will be the same as the sector offset specified in GC file. Partition a 4GB card such that first NonFs sector is 3872257. Set the offset address, under "uSD Raw", to 3872260. Sector address to display the video will be 003B(MSW), 1604(LSW).	
4DSL Sample	Raw.4DScript	

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb), **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, 2 bytes).
	y	Video vertical start position (top left corner, 2 bytes).
	width	Width of the video/animation, 2 bytes
	height	Height of the video/animation, 2 bytes
	colourMode	**08**(hex) : 256 colour mode, 8bits/1byte per pixel. **10**(hex) : 65K colour mode, 16bits/2bytes per pixel .
	delay	1 byte inter-frame delay in milliseconds. This parameter can be used to determine the speed of the video playback.
	frames	Number of frames to be displayed, 2 bytes
	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	Video/Animation should be loaded to the SD card using **Serial Command Platform** option on the **Graphics Composer**. 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. Position of the clip to be played is specified by **(x, y).** Other parameters can be extracted from the text file created on the Graphics Composer after burning the slide show on the SD card under serial platform option. Display Control Function (*Syntax: cmd, mode, value*) can be used to set “Image Format” to “Old format” i.e mode = 06hex. Value = 01hex. Refer to Sec 2.1.6. Image format is “Old Format” by default. **Note:** Make sure colourMode is same as set on the Graphics Composer while setting the video/animation properties.	
Example	Set Display Control Function 59hex, 06hex, 01hex Display video/Animation from card 40hex, 56hex, 00hex, 00hex, 00hex, 00hex, 01hex, 40hex, 00hex, F0hex, 10hex, 00hex, 02hex, 5Fhex, 00hex, 10hex, 00hex	
4DSL Sample	Raw.4DScript	

2.5.11 Run Script (4DSL) Program from Card (RAW) - @50hex

Serial Cmd	**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 2.8. 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.	
Example	**A sample script program inside the memory card:**	
	Address	**Command** / **Comment**
	00000000	45 — Erase Screen
	00000001	43 00 64 00 32 00 14 00 1F — Draw Circle
	0000000A	07 03 E8 — Delay(1second)
	0000000D	72 00 00 00 00 00 3C 00 3C 07 E0 — Draw Rectangle
	00000018	40 56 00 00 00 00 00 46 00 32 10 0A 02 5F 00 10 00 — Play video from card
	00000029	0B 00 00 00 00 — Goto Address 00000000
4DSL Sample	Raw.4DScript	

2.6 SD Memory Card Commands (FAT-Level/DOS)

The commands detailed in this section utilise the SD/microSD memory card which must be connected to the SPI port of the PICASO-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 FAT-Level memory card operations and they are described in this section. Currently only the FAT (FAT16) format is supported. FAT32 is not supported and if a FAT32 formatted card is used then all access (both RAW and FAT16 commands) to the card will fail.

Note: The PICASO-SGC also supports high capacity HC memory cards (4Gb and above). The available capacity of SD-HC cards varies according to the way the card is partitioned and the commands used to access it.

The FAT partition is always first (if it exists) and can be up to the maximum size permitted by FAT16. Windows will format FAT16 up to 2Gb and the Windows command prompt will format FAT16 up to 4Gb.

For the RAW partition, byte reads and writes can access 2^32 (i.e. 4gb) of the card, Sector reads and writes can access 2^24 sectors (of 512 bytes, i.e. 8gb). The total amount of the card usable is the sum of the FAT and RAW partitions.

Summary of Commands in this section:

- Initialise Memory Card - **@69hex**
- Read File from Card (FAT) - **@61hex**
- Write File to Card (FAT) - **@74hex**
- Erase file from Card (FAT) - **@65hex**
- List Directory from Card (FAT) - **@64hex**
- Screen Copy-Save to Card (FAT) - **@63hex**
- Display Image-Icon from Card (FAT) - **@6Dhex**
- Play Audio WAV file from Card (FAT) - **@6Chex**
- Run Script (4DSL) Program from Card (FAT) - **@70hex**

2.6.1 Initialise Memory Card - @69hex

Serial Cmd	**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.	
4DSL Sample	RAW.4DScript	

2.6.2 Read File from Card (FAT) - @61hex

Serial Cmd	**ext_cmd, cmd, handshaking, "file_name", terminator**	
4DSL Cmd	**See $ReadFile in Sec. 3.3.6**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**61**(hex) or **a**(ascii) : Command header byte
	handshaking	How often the host sends an ACK(06hex) to request more data during transmission:- 00 – No handshaking, Use only for small files (<= 512 bytes) 01 – Once for each byte 02 – Once for each two bytes ... 50(**32**hex), maximum allowed
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified in the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex).
Response	**file_size**(Umsb:Ulsb:Lmsb:Llsb)**, file_data**(1...N)**, acknowledge**	
	file_size	4 bytes of file size (big endian format).
	file_data	Complete file data block : No Handshaking Block of file data : block size determined by value set in handshaking.
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	Using this command, the host can read a DOS compatible (FAT) file from the memory card. Because the time taken to process the read bytes varies, a technique is required to ensure that the host communications buffer does not overflow and data is not lost. This is implemented by a simple *handshaking* protocol where the PICASO-SGC will break up the file into smaller data blocks. When the host receives a block, it sends an ACK(06hex) to request the next block of data. The size of the data block is initially set by the host in the command packet, specified by the value in the "handshaking" byte. The larger the value the better, as long as the host system can buffer the incoming block size. Setting this value too low will slow the transfer. The first 4 bytes sent by the device, after receiving the command packet, represent the size of the requested file (Umsb:Ulsb:Lmsb:Llsb). The host then responds with an ACK(06hex) to indicate it wants the file, or NAK(15hex) if it wishes to terminate the receive. The first block of data bytes (block size set by the handshaking value) of the file are then sent from the device, the host then responds with another ACK(06hex) to receive the next block of bytes. This process continues until all of the file data has been received. The device responds with a final ACK if the transfer completes successfully, otherwise it responds with a NAK. The final ACK is not part of the handshaking. **Note: Refer to Diagram1 and Diagram2 on the next page.**	
4DSL Sample	**See $ReadFile in Sec. 3.3.6**	

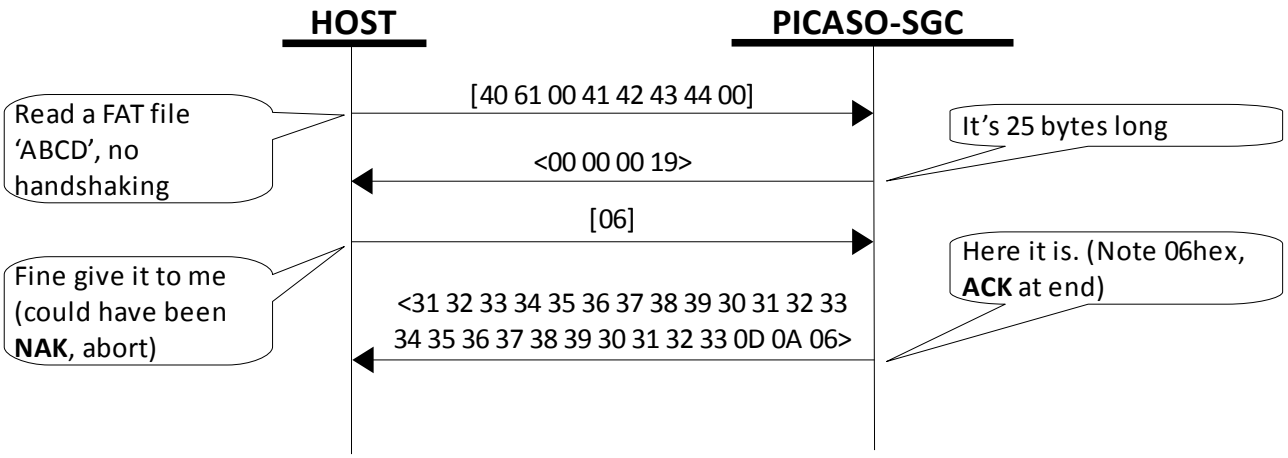


Diagram 1: Read File – No Handshaking

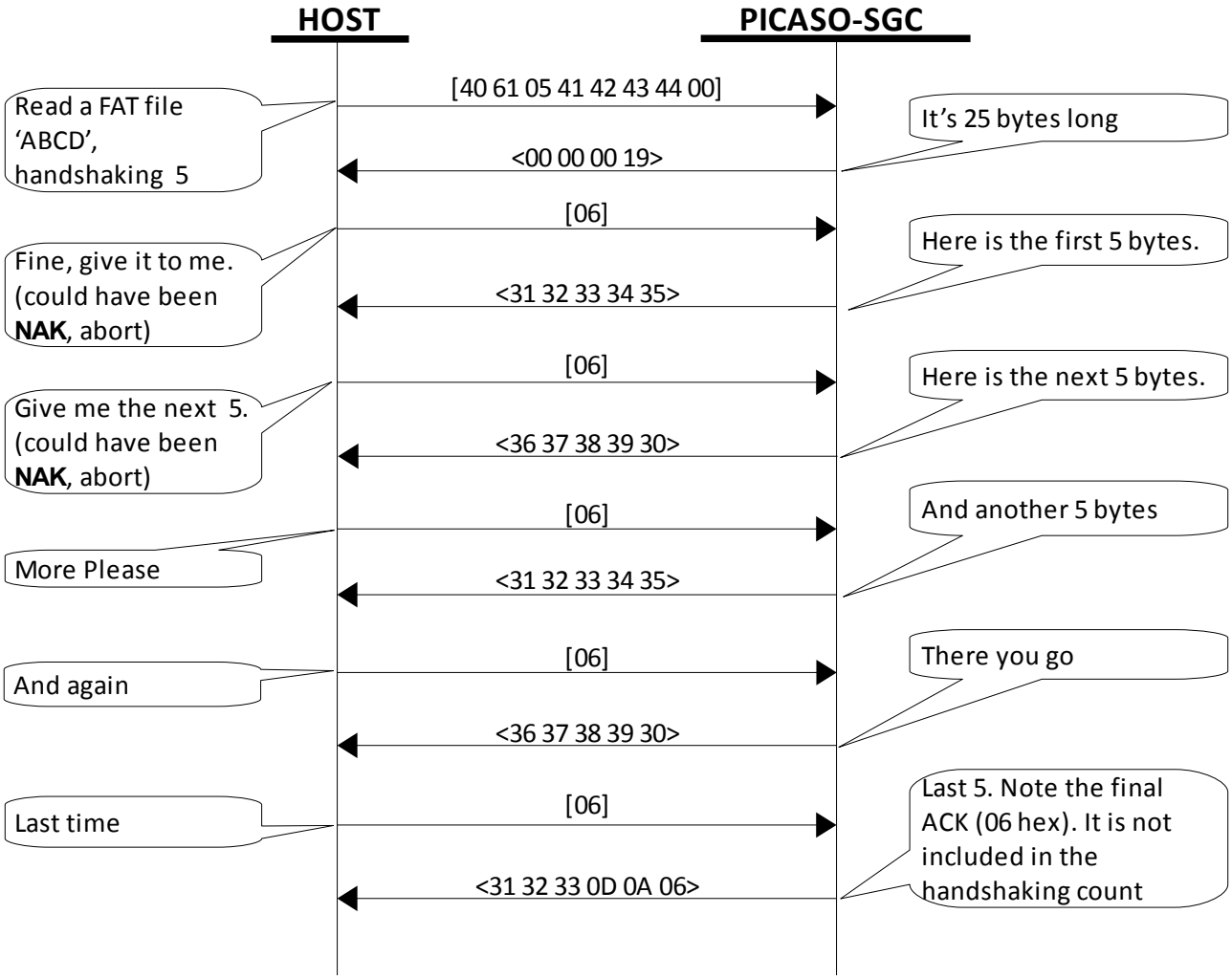


Diagram 2: Read File - Handshaking other than 0

2.6.3 Write File to Card (FAT) - @74hex

Serial Cmd	**ext_cmd, cmd, options, "file_name", terminator, filesize**(Umsb:Ulsb:Lmsb:Llsb), **file_data**(1.. N)	
4DSL Cmd	**See $WriteFile in Sec. 3.3.10**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**74**(hex) or **t**(ascii) : Command header byte
	options	Controls handshaking (how often the device sends an ACK to request more data from the host) and whether an existing file is appended to. **Handshaking:** 00 – No handshaking, limit this to small files (<= 100 bytes) 01 – Once for each byte 02 – Once for each two bytes ... 50(**32**hex), maximum allowed **Append Mode:** **00**(hex)–No Append, file will be created (or overwritten if it exists). **80**(hex)–Append mode, file will be appended to (or created if it doesn't exist). **Note:** The two option are added or **OR**ed together to produce the final options, e.g. **82**(hex) would indicate handshaking for every two bytes and Append mode.
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified in the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex).
	file_size	4 bytes of file size (big endian format).
	file_data	Complete file data block : No Handshaking Block of file data : block size determined by value set in handshaking.
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command allows the host to write a DOS compatible (FAT) file to the memory card. The PICASO-SGC device serial port (UART), has a buffer size of 512 bytes for capturing incoming data from the host. If this buffer fills up and overflows, data will be lost. Therefore the host must allow the device enough time to write its buffer to the memory card so it can then receive further file data from the host. For small files (less than 100 bytes in the append mode) the host can send the complete file data in one attempt. However, for larger files a simple *handshaking* protocol is implemented where the host sends the file data in small blocks. Using this handshaking method, the host always waits for an ACK from the device before sending the next block of data. The size of the data block is initially set by the host in the command packet, specified by the handshaking value in the "options" byte. The larger the value the better. Setting it too low will slow the transfer. The first ACK is always sent by the device, after the filesize parameter is transmitted by the host, or this could also be a NAK in which case one of the parameters is invalid or a file system error occurred. **Note: Do not set handshaking to zero if the file size is larger than 512 bytes.**	

	Note: Refer to Diagram3 and Diagram4 on the next page.
4DSL Sample	**See $WriteFile in Sec. 3.3.10**

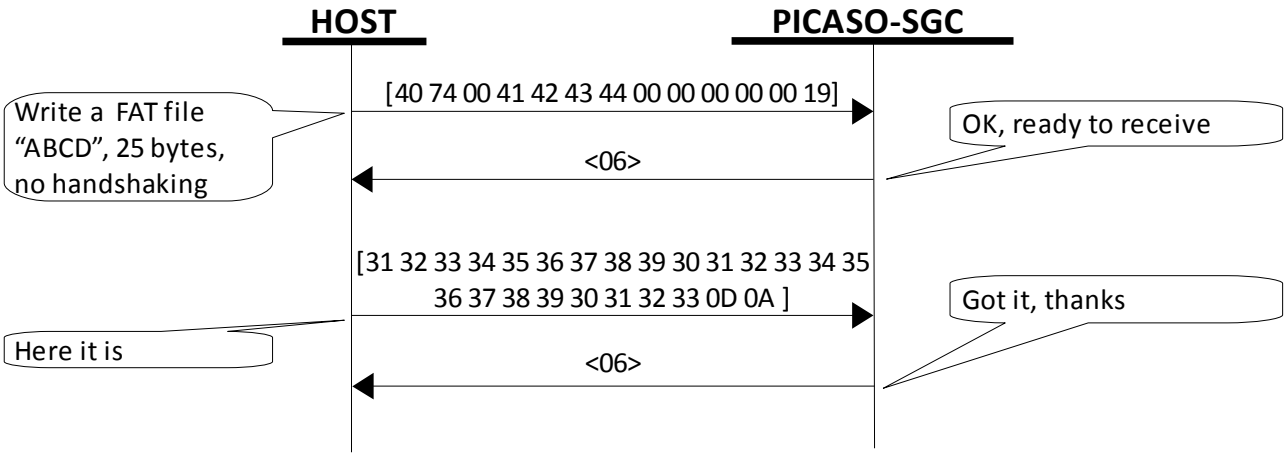


Diagram 3: Write File - No Handshaking

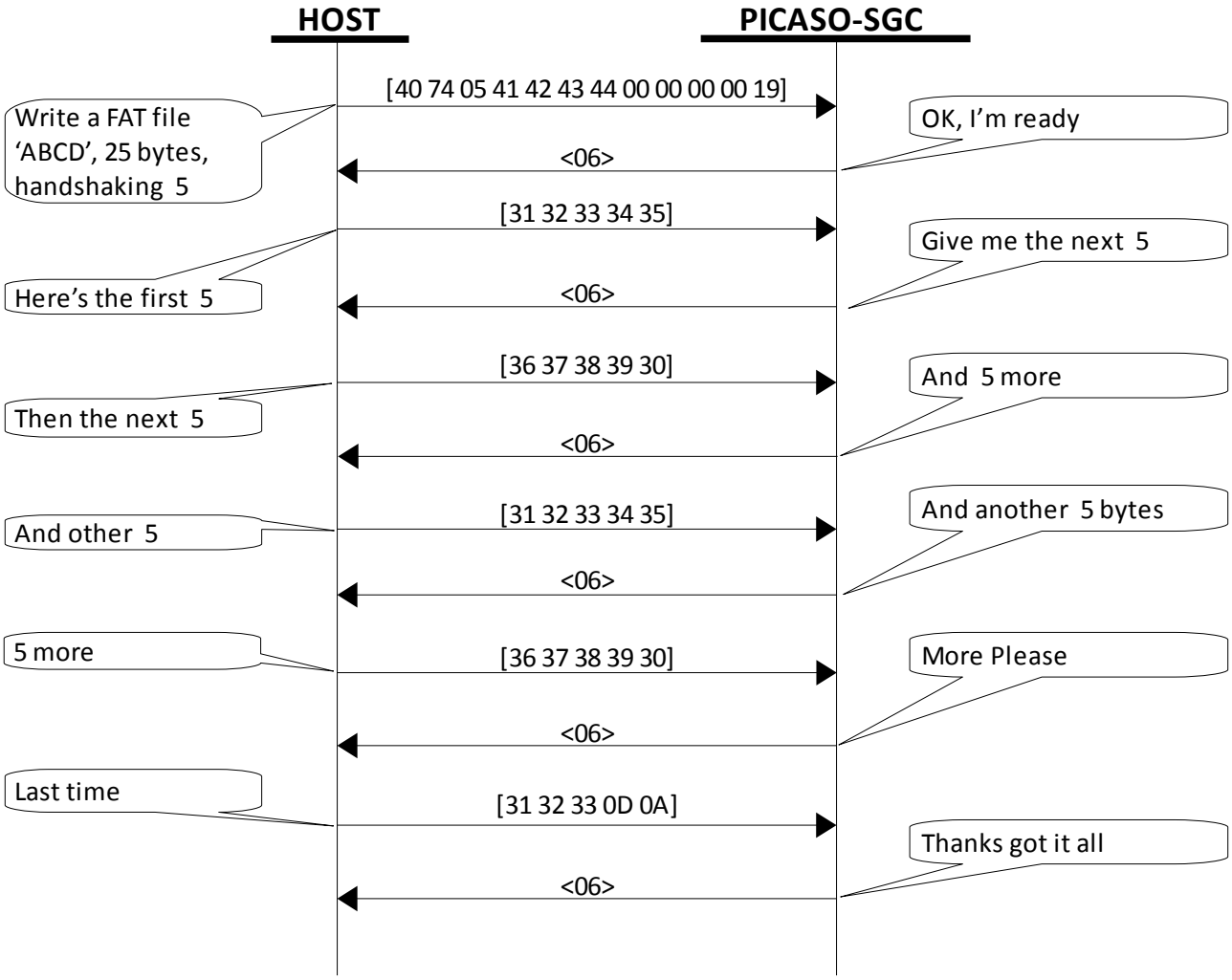


Diagram 4: Write File - Handshaking Other than 0

2.6.4 Erase File from Card (FAT) - @65hex

Serial Cmd	**ext_cmd, cmd, “file_name”, terminator**	
4DSL Cmd	**EraseFile('file_name')**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd_hdr	**65**(hex) or **e**(ascii) : Command header byte
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if file deleted successfully **15**(hex) : NAK byte if file not found or an error has occurred
Description	Erases the file specified in the “file_name”.	
4DSL Sample	FAT.4DScript	

2.6.5 List Directory of Card (FAT) - @64hex

Serial Cmd	**ext_cmd, cmd, “file_name”, terminator**	
4DSL Cmd	**Dir('file_name')**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd_hdr	**64**(hex) or **d**(ascii) : Command header byte
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename. Wild cards such as '*' and '?' are allowed.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**“fileName1”, delimiter, .. , “fileNameN”, delimiter, acknowledge**	
	fileName	Character string of the file name in the memory card. Maximum of 12 character bytes (including '.' separator) are returned in the string for the file name. This will be repeated for all files in the directory. An empty directory with no files or the result of an unsuccessful file name or wild card search will only return an ACK. **Note:** At present the PICASO-SGC chip only supports a single directory structure. Future enhancements will support nested directories.
	delimiter	**0A**(hex) : newline
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if an error has occurred
Description	Returns a directory listing (stream of characters) consisting of the files names matching the “file_name” delimited by a Newline(**0A**hex) character. Always responds with an ACK at the completion of a listing. Responds with a NAK if a file error occurs. An empty directory with no files or the result of an unsuccessful file name or wild card search will only return an ACK.	
4DSL Sample	FAT.4DScript	

2.6.6 Screen Copy – Save to Card (FAT) - @63hex

Serial Cmd	**ext_cmd, cmd, x**(msb:lsb), **y**(msb:lsb), **width**(msb:lsb), **height**(msb:lsb)**, "file_name", terminator**	
4DSL Cmd	**ScreenCopyFAT(x, y, width, height, 'file_name')**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**63**(hex) or **c**(ascii) : Command header byte
	x	Top left horizontal start position of screen area to be copied (2 bytes).
	y	Top left vertical start position of screen area to be copied (2 bytes).
	width	Width of screen area to be copied (source, 2 bytes).
	height	Height of screen area to be copied (source, 2 bytes).
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
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 in FAT format with the specified **filename**. 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 Card (FAT)"** command. This is a very powerful feature for animating objects, smooth scrolling, or implementing a windowing system. **Note:** This command always stores that part of the screen as a 16 bit image, i.e. 2 bytes per pixel.	
4DSL Sample	FAT.4DScript	

2.6.7 Display Image-Icon from Card (FAT) - @6Dhex

Serial Cmd	**ext_cmd, cmd, "file_name", terminator, x**(msb:lsb)**, y**(msb:lsb)**, imagePos**(msw:lsw)	
4DSL Cmd	**FATImage('file_name', x, y, imagePos)**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**6D**(hex) or **m**(ascii) : Command header byte
	file_name	The filename is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
	x	Image horizontal start position (top left corner).
	y	Image vertical start position (top left corner).
	imagePos	4 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 to the screen that has been previously stored in the FAT file (specified by " **filename**") on the memory card. The screen position of the image to be displayed is specified by **(x, y)**. **Note:** The **"Screen Copy-Save to Card (FAT)"** command always stores that part of the screen as a 16 bit image, i.e. 2 bytes per pixel. This command can also be used to display multimedia objects in the GCI file, created from the Graphics Composer. The video parameters such as width, height and colour-mode are built into the GCI file; they don't need to be specified by the host controller. All you need is GCI filename which is loaded to the SD card, and the pointer to the multimedia object which can be found from the .DAT file created by the Graphics Composer along with the GCI file. **Note: Y**ou should not attempt to use "Display Image-Icon from Card (FAT) - @6Dhex" command to display video, it may work, but it will be extremely slow.	
4DSL Sample	FAT.4DScript	

2.6.8 Play Audio WAV file from Card (FAT) - @6Chex

Serial Cmd	**ext_cmd, cmd, option, "file_name", terminator**	
4DSL Cmd	**PlayWav(option, 'file_name')**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**6C**(hex) or **l**(ascii) : Command header byte
	option	This byte specifies the available options for the WAV file: **00hex** : Return when playing complete **01hex** : Return immediately **02hex** : STOP currently playing WAV file **03hex** : PAUSE currently playing WAV file **04hex** : RESUME currently playing WAV file **05hex** : LOOP playing until stopped
	file_name	The name of the WAV file in the memory card. The file name is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This command plays a WAV file from the memory card. Wave files should be mono to keep data bandwidth to a minimum, and should be '**canonic**' format. Lots of windows formats wont work. Use something like '**Cool Edit**' or similar to tailor the wav files to a suitable format. The ideal sample rate of the WAV file is 16Khz-Mono and the maximum should be 22Khz. Any higher sample rate will extremely slow down the system. Sample rates below 12Khz, the PWM will cause aliasing (filtering is a bare minimum). If you only hear noise or random snippets of sound remember, the Speed and Capacity of the memory card are important, most 2Gb cards should be fine, 64mb cards fail all but the most simple sounds.	
4DSL Sample	FAT.4DScript	

2.6.9 Run Script (4DSL) Program from Card (FAT) - @70hex

Serial Cmd	**ext_cmd, cmd, "file_name", terminator**	
4DSL Cmd	**RunScriptFAT('file_name')**	
	ext_cmd	**40**(hex) or **@**(ascii) : Extended Command header byte
	cmd	**70**(hex) or **p**(ascii) : Command header byte
	file_name	The file name of the scripting file in the memory card. The file name is 1-12 chars long with an assumed '.' between chars 8 and 9, if there is not one specified the filename.
	terminator	The file_name string must be terminated with a NULL, **00**(hex)
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful or card not present.
Description	The Run command executes the specified file name from the memory card without any further interaction by the host processor. It will sequentially execute all valid commands and display objects until it gets to the end of the file. It is advisable to have the **Exit Program** or the **Jump to Address** commands at the end of the user composed program to ensure there are no undesirable results at the end of the file. The PICASO-SGC device is equipped to accept memory cards and has a feature that will auto run a preloaded script program on power-up. When using the FAT file system, upon power-up, if a file called '**autoexec.4ds**' exists on the memory card, the PICASO-SGC will automatically execute this script program from the memory card. This is a useful feature for those stand alone applications where the device does not require a host controller to send commands to the PICASO-SGC to play a slide show of images, video clips, etc. The user will have to create and upload a slide show composition to the card to benefit from this auto play feature. Refer to **Appendix B** at the end of this document for a quick guide to creating scripting files using the **FAT-Controller** software tool available from 4D Labs.	
4DSL Sample	FAT.4DScript	

3. 4DSL Scripting Language

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 saved on the uSD card in the form of a File called 4DSL scripting file. The script files can be called from a host controller or they can be saved as autoexec.4DS file to run automatically on power up.

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. 4DGL 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 writing a 4DSL Script on the 4DGL Workshop3 IDE is provided on the ***“4DGL-Workshop3-IDE-User-Guide-rev3.pdf”*** or above.

3.1 Script Control 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.

Summary of Commands in this section:

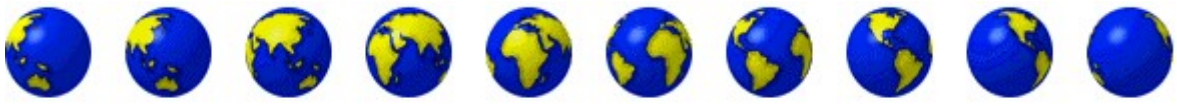
- Delay – **07hex**
- Set Counter – **08hex**
- Decrement Counter – **09hex**
- Jump to Card Address If Counter Not Zero – **0Ahex**
- Jump to Card Address – **0Bhex**
- Exit-Terminate Script Program – **0Chex**

3.1.1 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 Sample	ABC.4DScript	

3.1.2 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 Sample	ABC.4DScript	



3.1.3 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 Sample	ABC.4DScript

3.1.4 Jump to Card 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 card 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 Sample	ABC.4DScript	

3.1.5 Jump to Card 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 card address and start executing commands from there.	
4DSL Sample	ABC.4DScript	

3.1.6 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 Sample	ABC.4DScript

3.2 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

3.2.1 #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.	
Example	#Compile(Picaso,COM4,9600,5,Wrap)	
4DSL Sample	Most of the 4DS Script Samples.	

3.2.2 #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	
Example	#define red 0xf800 OR #define file "C:\test.fle"	
4DSL Sample	4DScript_16bitColours.inc	

3.2.3 #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..	
Example	#include "4DScript_16bitColours.inc"	
4DSL Sample	Most of the 4DS Script Samples.	

3.2.4 #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. Or the filename when the file is written in FAT mode	
Example	#Origin 0x400 // start on sector 2	
4DSL Sample	ABC400.4DScript	

3.2.5 #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.	
Example	#run(Picaso,COM4,9600,5,Wrap); //Line 1 set script run and options	
4DSL Sample	Most of the 4DS Script Samples.	

3.3 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
- $ReadFATImage
- $ReadFile
- $ReadSectors
- $ReaduSDImage
- $StartSave
- $WriteFile
- $WriteSectors

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

3.3.1 $4DGLLoadprogram

4DSL Cmd	$4DGLLoadprogram(“4DGLprogram”, ram|flash)	
	4DGLprogram	The filename of a compiled 4DGL program to be loaded onto a display
	ram|flash	**Ram :**Target is RAM on the processor. **Flash :**Target is Flash on the processor.
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.	
Example	$4DGLLoadProgram("c:\Documents and Settings\All Users\Documents\4D Labs\PICASO GFX2\Picaso – Graphics\worm",Ram)	
4DSL Sample	Load4DGLProgram .4DScript	

3.3.2 $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.	
Example	$LoadPmmC("c:\Documents and Settings\All Users\Documents\4D Labs\PICASO GFX2\Picaso – Graphics\worm",Ram)	
4DSL Sample	LoadPmmC .4DScript	

3.3.3 $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.	
Example	$Message('About to halve Volume')	
4DSL Sample	FAT.4DScript	

3.3.4 $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.	
Example	$Readbytes('50bytes.hex',50)	
4DSL Sample	RAW.4DScript	

3.3.5 $ReadFATImage

4DSL Cmd	$ReadFATImage("uSDFile", "PCFile")	
	uSDFile	The image file on the uSD card.
	PcFile	The name of the output BMP file on the PC.
Description	Reads an image file from the uSD card and converts it to a Bitmap file and saves it on the PC.	
Example	$ReadFATImage('scrncopy','scrncopy.bmp')	
4DSL Sample	FAT.4DScript	

3.3.6 $ReadFile

4DSL Cmd	$ReadFile("Handshaking", "uSDFile", "StartAcks", "pcFile")	
	Handshaking	The handshaking to use 0-50.
	uSDFile	The name of the file on the uSD Card to read.
	StartAcks	The number of ACKS to send at the start, Double or Single, Double gives the best performance, but will not work on all controllers.
	pcFile	The destination file on the PC
Description	This reads a FAT file from the uSD card and saves it on the PC.	
Example	$ReadFile(50,'scrncopy',double,'scrncopy.raw')	
4DSL Sample	FAT.4DScript	

3.3.7 $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.	
Example	$readsectors('Sector0.hex',1,1)	
4DSL Sample	RAW.4DScript	

3.3.8 $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.	
Example	$ReaduSDImage(0,-1,-1,'ImageNew.bmp')	
4DSL Sample	RAW.4DScript	

3.3.9 $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.	
Example	$startsave('pixel.hex') : : $endsave	
4DSL Sample	Graphics Pt2.4DScript	

3.3.10 $WriteFile

4DSL Cmd	$WriteFile("Handshaking", "AppRep", "startacks", "uSDFile", "pcFile")	
	Handshaking	The handshaking to use 0-50.
	apprep	Append or Replace
	StartAcks	The number of ACKS to send at the start, Double or Single, Double gives the best performance, but will not work on all controllers.
	uSDFile	The name of the file on the uSD Card to read.
	pcFile	The destination file on the PC
Response	**acknowledge**	
	acknowledge	**06**(hex) : ACK byte if successful **15**(hex) : NAK byte if unsuccessful
Description	This writes a FAT file to the uSD card.	
Example	$WriteFile(50,Replace,double,'test1.gci','..\..\resources\copy to usd\test1.gci') ; OR $WriteFile(50,Replace,double,'abc.4ds','abc.4DScrpObj') ;	
4DSL Sample	FAT.4DScript	

3.3.11 $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.	
Example	$WriteSectors("..\..\resources\Misc\NemoNewFmt.hex",500)	
4DSL Sample	Most of the 4DSL Scripts	

3.3.12 Extended Macros

$4DGLAttn	**Use this to get attention of the 4DGL hypervisor to enable the use of the read and write sectors commands.**	
	4DSL Sample	Load4DGLuSD.4DScript
$4DGLExit	**Use this to exit from the 4DGL hypervisor**	
	4DSL Sample	Load4DGLuSD.4DScript
$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 Sample	General.4DScript
$CloseComPort	**This closes the com port, if it is open.**	
	4DSL Sample	General.4DScript
$EndSave	**This closes the file previously openned with $StartSave.**	
	4DSL Sample	Graphics Pt2.4DScript
$FlushBuffer	**This flushes the Comms buffer.**	
	4DSL Sample	RAW.4DScript
$IgnoreErrors	**This causes the script to continue if a command resturns a NAK or other unexpected result. The opposite is $AbortonError**	
	4DSL Sample	General.4DScript
$OpenComport	**This Opens the com port, no checking is done. See $OpenInit for a verified open**	
	4DSL Sample	General.4DScript
$OpenInit	**This Opens the com port and sends autobaud('U') and checks for an ACK. Up to 10 retries are performed.**	
	4DSL Sample	Most of the 4DSL Scripts
$ReadCSD	**This reads CSD record from the uSD card and displays the card's capacity.**	
	4DSL Sample	RAW.4DScript
$TimeOff	**This turns off logging of times for each command (default).**	
	4DSL Sample	General.4DScript
$TimeOn	**This turns on logging of times for each command.**	
	4DSL Sample	General.4DScript

3.4 4DSL Keywords

#run, #compile option

'Picaso'	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

PlayWav Option

WaitComplete	Return when playing complete
Return	Return immediately
Stop	STOP currently playing WAV file
Pause	PAUSE currently playing WAV file
Resume	RESUME currently playing WAV file
Loop	LOOP playing until stopped

Volume Option

'min'	Minimum Volume.
'max'	Maximum Volume.
'mute'	Mute
'down'	Volume Down
'down8'	Volume Down 8

'up'	Volume Up
'up8'	Volume Up 8
'muteoff'	Mute Off

WriteFile option

Append	Append mode, file will be appended to (or created if it doesn't exist)
Replace	No Append, file will be created (or overwritten if it exists).

WriteFile, ReadFile option

Single	The number of ACKS to send at the start or ReadFile/WriteFile, Double or Single, Double gives the best performance, but will not work on all controllers.
Double	The number of ACKS to send at the start or ReadFile/WriteFile, Double or Single, Double gives the best performance, but will not work on all controllers.

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.

3.5 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.
  - Set Volume - 76hex
  - Sleep - 5Ahex
  - Read GPIO Pin - 69hex
  - Write to GPIO Pin - 79hex
  - Read GPIO Bus - 61hex
  - Write to GPIO Bus - 57hex
  - 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.
  - Draw Ellipse - 65hex
- **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.

- **Touch Screen Commands**
  - Get Touch Coordinates - 6Fhex
  - Wait Until Touch - 77hex
  - Detect Touch Region – 75hex
- **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.
- **SD Memory Card Commands (FAT-Level/DOS)**
  - Erase File from Card (FAT) - @65hex.
  - List Directory of Card (FAT) - @64hex.
  - Screen Copy – Save to Card (FAT) - @63hex.
  - Display Image-Icon from Card (FAT) - @6Dhex.
  - Play Audio WAV file from Card (FAT) - @6Chex.
  - Run Script (4DSL) Program from Card (FAT) - @70hex.
- **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
- $ReadFATImage
- $ReadSectors
- $ReaduSDImage
- $StartSave
- $TimeOff
- $TimeOn
- $WriteSectors

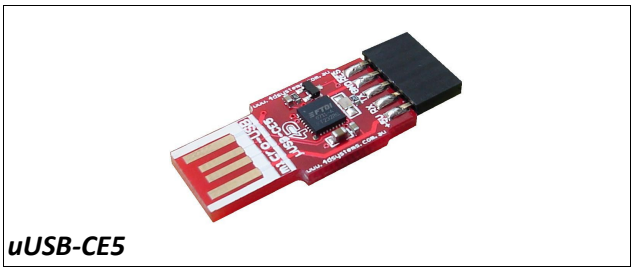
4. Appendix A : Development and Support Tools

4.1 PmmC Loader – PmmC File Programming Software Tool

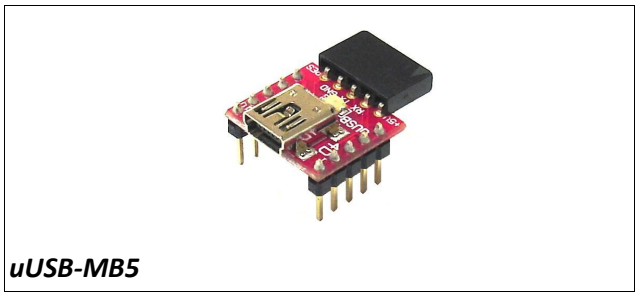
The 'PmmC Loader' is a free software tool for Windows based PC platforms. Use this tool to program the latest PmmC file into the PICASO-SGC chip embedded in your application board. It is available for download from the 4D Systems website, www.4dsystems.com.au

4.2 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 PICASO-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 PICASO-SGC chip.



uUSB-CE5



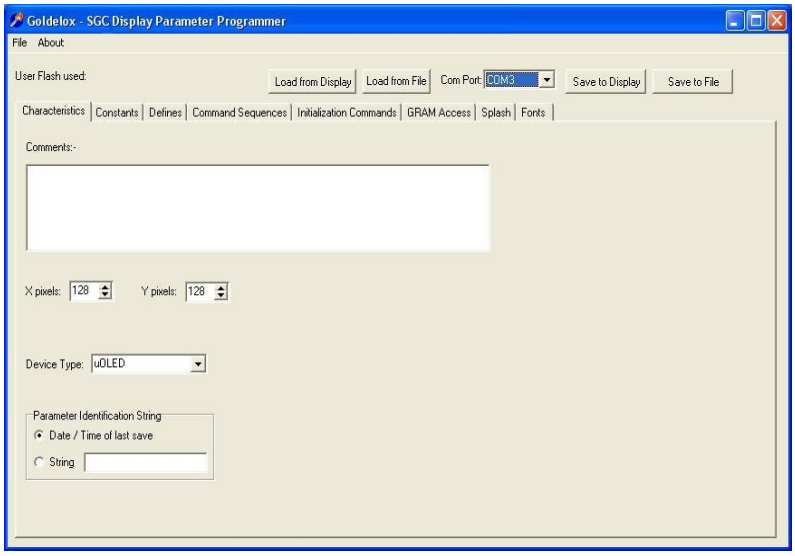
uUSB-MB5

4.3 Display Initialisation Setup Personality (DISP) – Software Tool

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

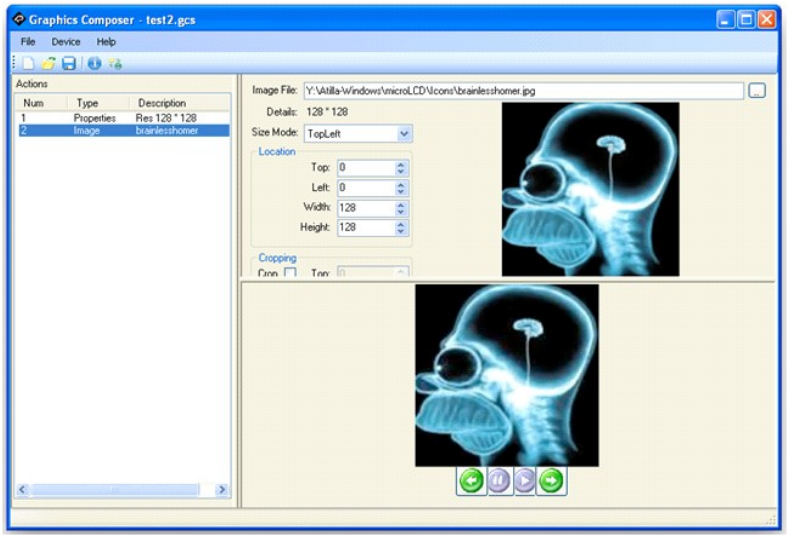
- Configure the PICASO-SGC chip to work with a specific display.
- Modify the way the chip initially sets up the display, e.g. screen saver, brightness, etc.
- Construct the splash screens.
- Replace or modify the embedded fonts.

It is available for download from the 4D Systems website, [www.4dsystems.com.au.](http://www.4dsystems.com.au)



4.4 Graphics Composer – Software Tool

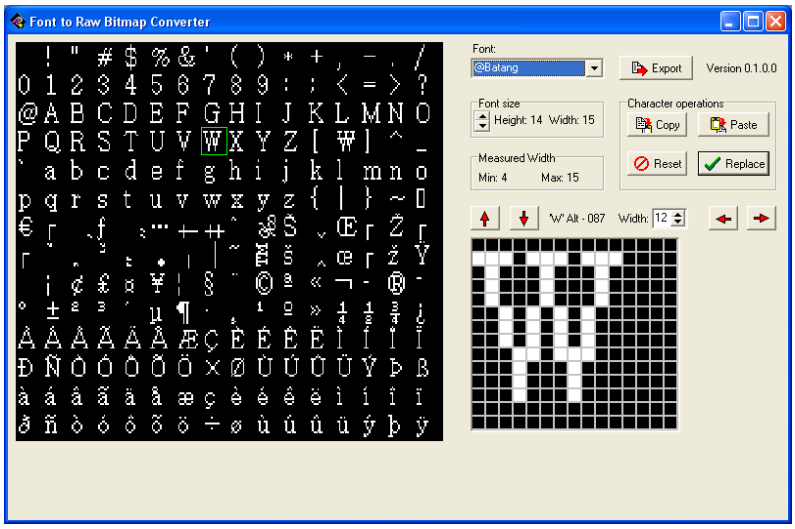
The Graphics Composer is a free software tool for Windows. This software tool is an aid to composing a slide show of images/animations/movie-clips (multi-media objects) which can then be downloaded into the SDHC/SD/uSD/MMC memory card that is supported by the PICASO-SGC. The host simply sends commands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



4.5 FONT Tool – Software Tool

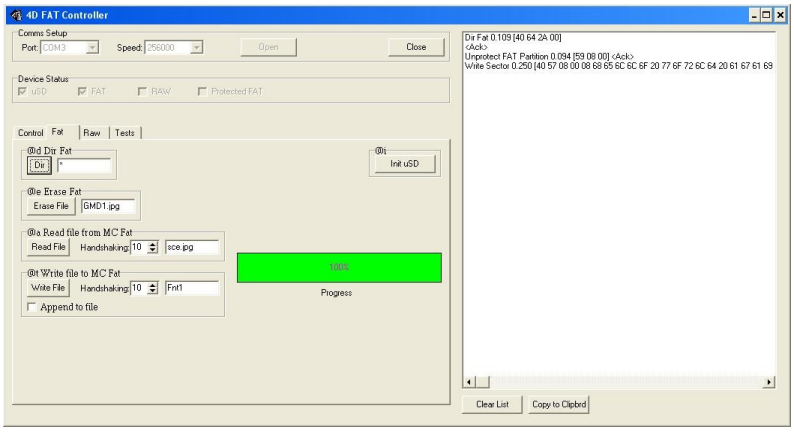
Font-Tool is a free software utility for Windows based PC platforms. This tool can be used to assist in the conversion of standard Windows fonts (including True Type) into the bitmap fonts used by the PICASO-SGC chip. It is available for download from the 4D Systems website, www.4dsystems.com.au.

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



4.6 FAT Controller – Software Test Tool

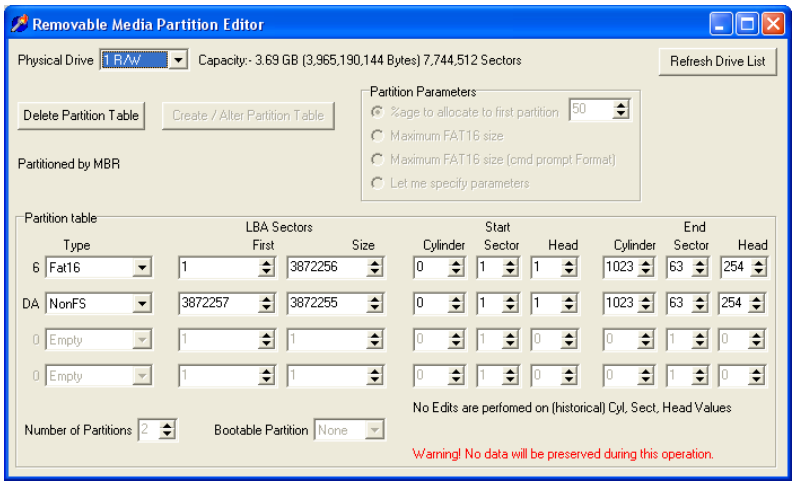
The 4D FAT Controller is a free software tool to test all of the functionality of the GOLDELOX-DOS, GOLDELOX-SGC and the PICASO-SGC devices and their respective modules. It is useful in learning about how to communicate with the chips and the modules. For the GOLDELOX-SGC and the PICASO-SGC it can also 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.



4.7 RMPET – Software Tool

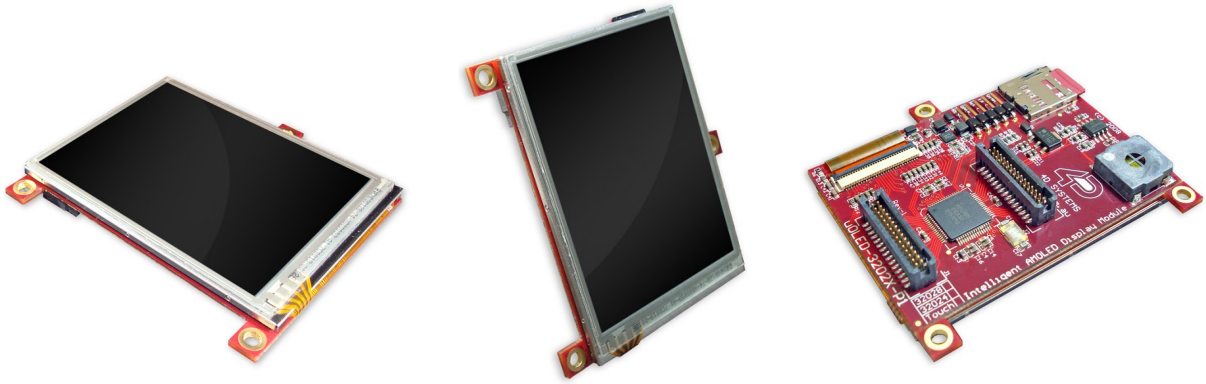
uSD/SD/SDHC memory cards nearly always come pre-partitioned with a single partition. Windows only accesses the first partition on the card and ignores any other partitions. **R**emovable **M**edia **P**artition **E**dit **T**ool (RMPET) can split a large card into two partitions, the first partition for use as a FAT16 partition and the second partition for use as a RAW partition. RMPET allows setting of the first partition to a percentage of the card, the 2Gb maximum of the FAT16 Windows format program, or the 4Gb maximum of FAT16 when the command prompt format command is used.

It is available for download from the 4D Systems website, [www.4dsystems.com.au.](http://www.4dsystems.com.au)



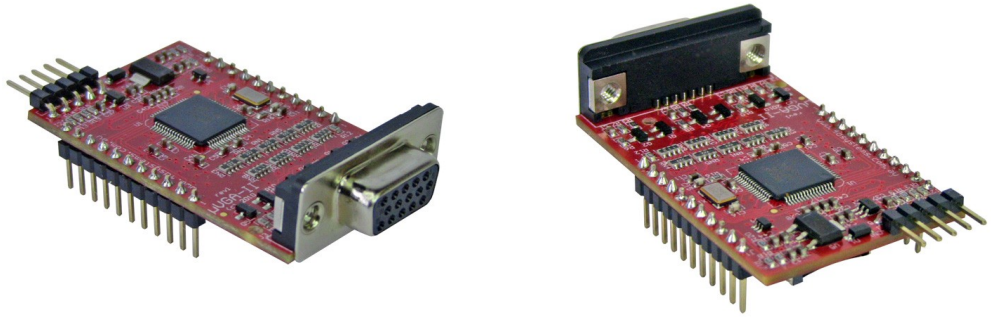
4.8 Evaluation Display Modules

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



uOLED-32028-P1T(SGC): 2.8" AMOLED, 65K Colour, Serial Display Module

uLCD-32PT(SGC): 3.2" TFT, 65K Colour, Serial Display Module



uVGA-II(SGC): Serial VGA Graphics Engine

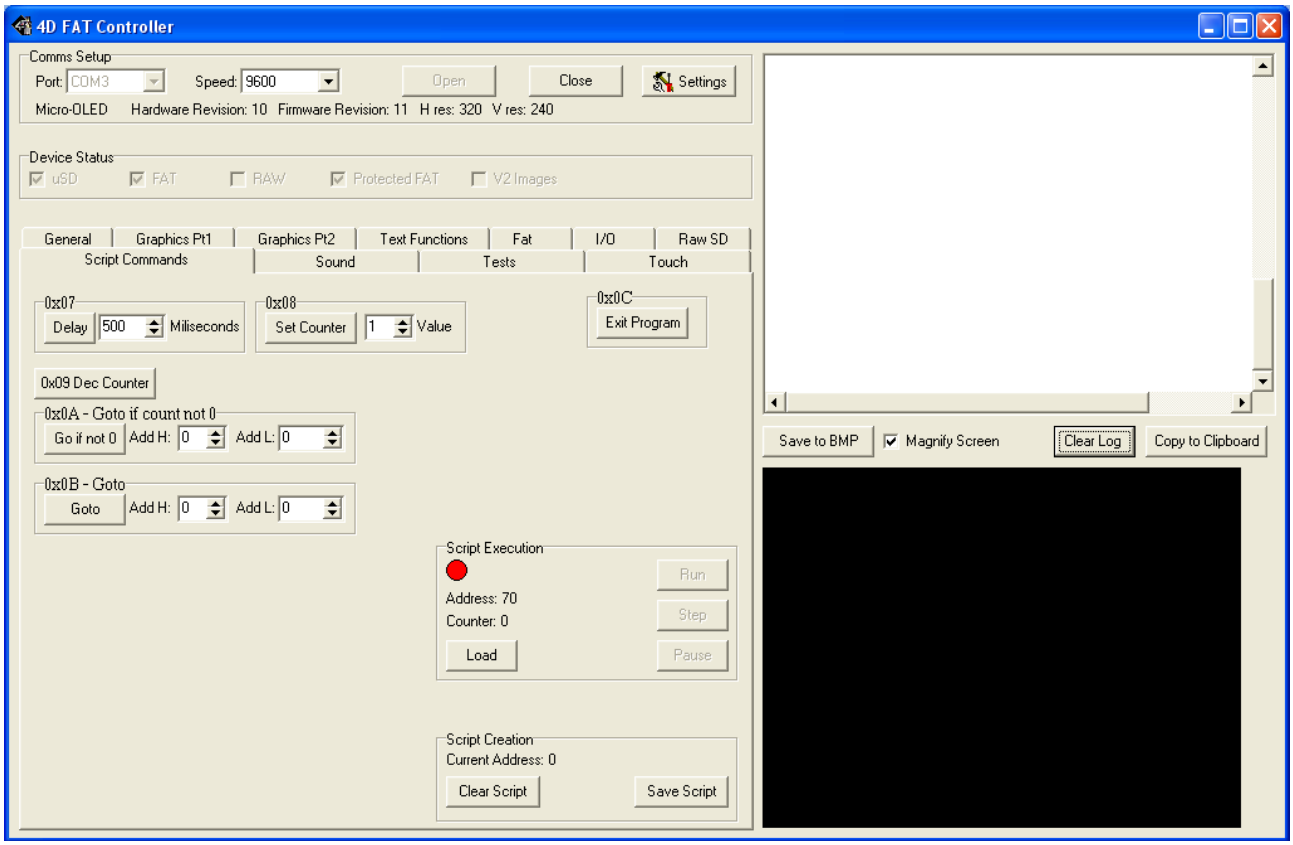
5. Appendix B: Using the FAT-Controller to Compose Script Programs

5.1 Getting Started

- Download and run the **FAT-Controller** software (for PC Windows) from the www.4dsystems.com.au website.
- You will need a USB to Serial converter module to provide a serial link between the FAT-Controller running on your PC and the PICASO-SGC application display module. 4D Systems has a range of appropriate modules such uUSB-CE5 and uUSB-MB5.

5.2 Starting a Script

- Before starting to construct your script, click the "**Clear Script**" command button, otherwise the script log will start from the time Com port is opened.
- Run all the appropriate commands you need for the script composition and click "**Save Script**" to save the script with **.4dScpObj** extension.



5.3 Running the Script

- Click Load button under the "**Script Execution Section**" under the "**Script Commands**" tab. A dialogue box will open up.
- Select your saved script file and click Run to run the script.

5.4 Running the .4ds Script Program from the Memory Card

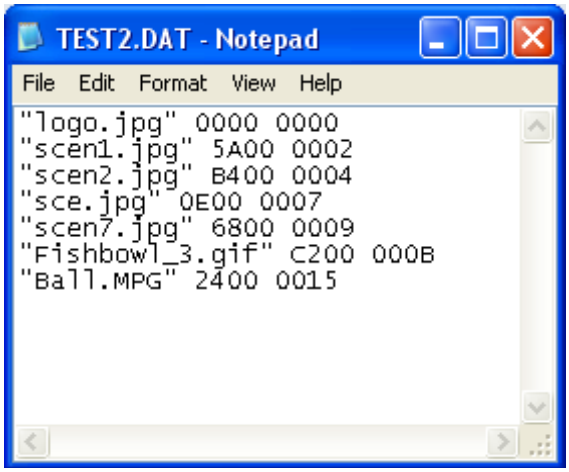
- Under the "**Fat**" tab look for the "**@p Run Program as from MC FAT**" option.
- Insert the memory card into the PICASO-SGC based application display module with. Make sure the .**4ds** script file is saved on to the memory card.
- Now click Execute Program. This command will run the slide show with the command sent to the PICASO-SGC via the serial port.

5.5 Running the Auto-Run Script Slide Show

- To make an Auto-Run script file, rename the created script file (described previously) to **autoexec.4ds** and save it to the memory card.
- Insert the memory card into the application display module and power up the module. The autoexec.4ds script file will now run automatically.

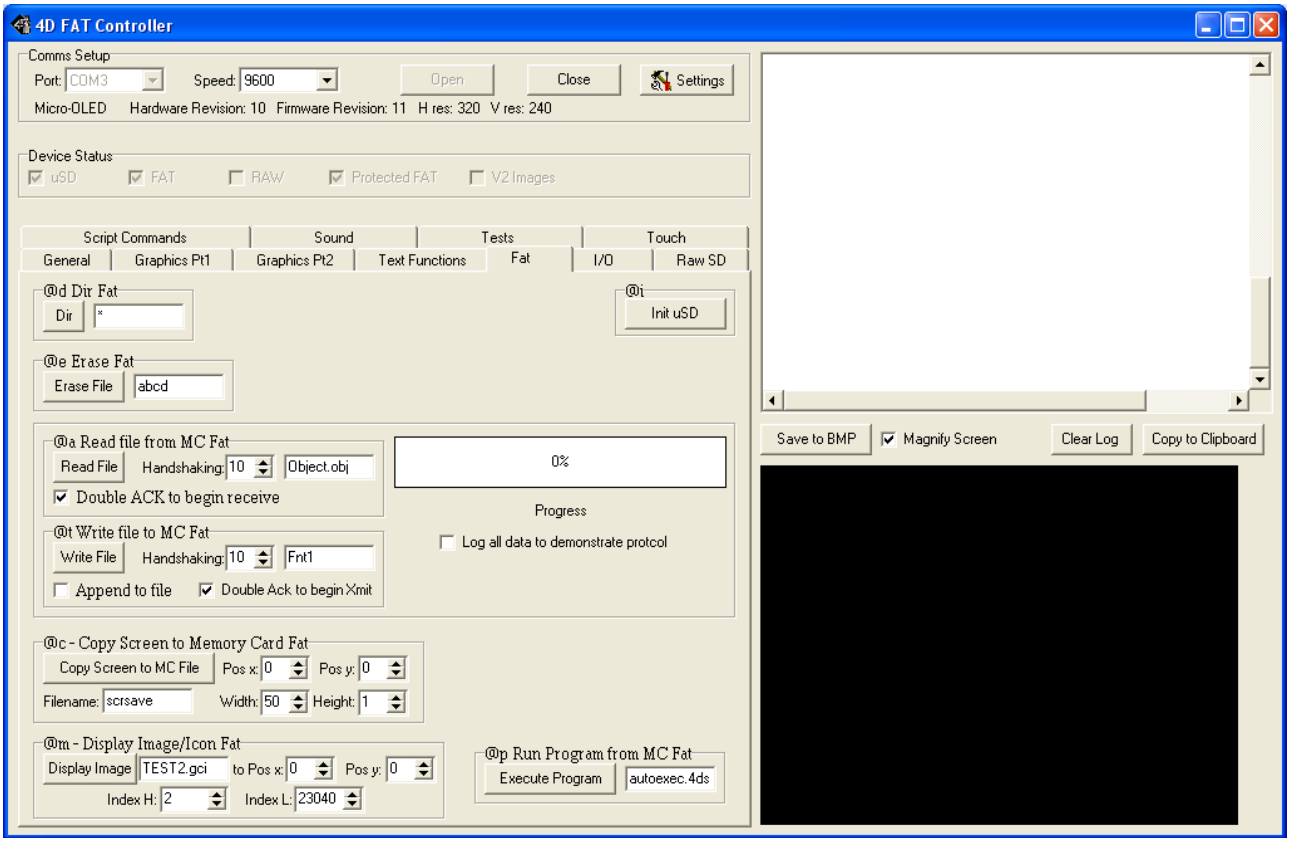
5.6 Running Graphics Composer GCI files from the Memory Card

- The Graphics Composer is a separate software tool that will convert all standard image and video files into a format that can be used by the PICASO-SGC device. It can be downloaded from www.4dsystems.com.au
- The Graphics Composer will create bot a GCI and DAT file. Load both of these files into the memory card.
- Open the DAT file with a text editor and note the the image/video file names and their corresponding addresses in the memory card. See snapshot below:



- Note the addresses on the DAT file are LSW (Least Significant Word), MSW (Most Significant Word) e.g. for "scen1.jpg" the LSW = 5A00(hex) and the MSW = 0002(hex).
- Under the "**Fat**" tab look for the "**Display Image - @m**" command. Enter the GCI file name along with its extension.
- Enter the screen position you wish to display the image.
- Enter the address location of the image in the GCI file. Note that the entry is in decimal. For example for "scen1.jpg" Index H = 2 (0002hex), Index L = 23040 (5A00hex).

- Click the “Display Image” button. The image will now be displayed on the target display module. Make sure the memory card is first inserted into the display module.



Proprietary Information

Disclaimer of Warranties & Limitation of Liability