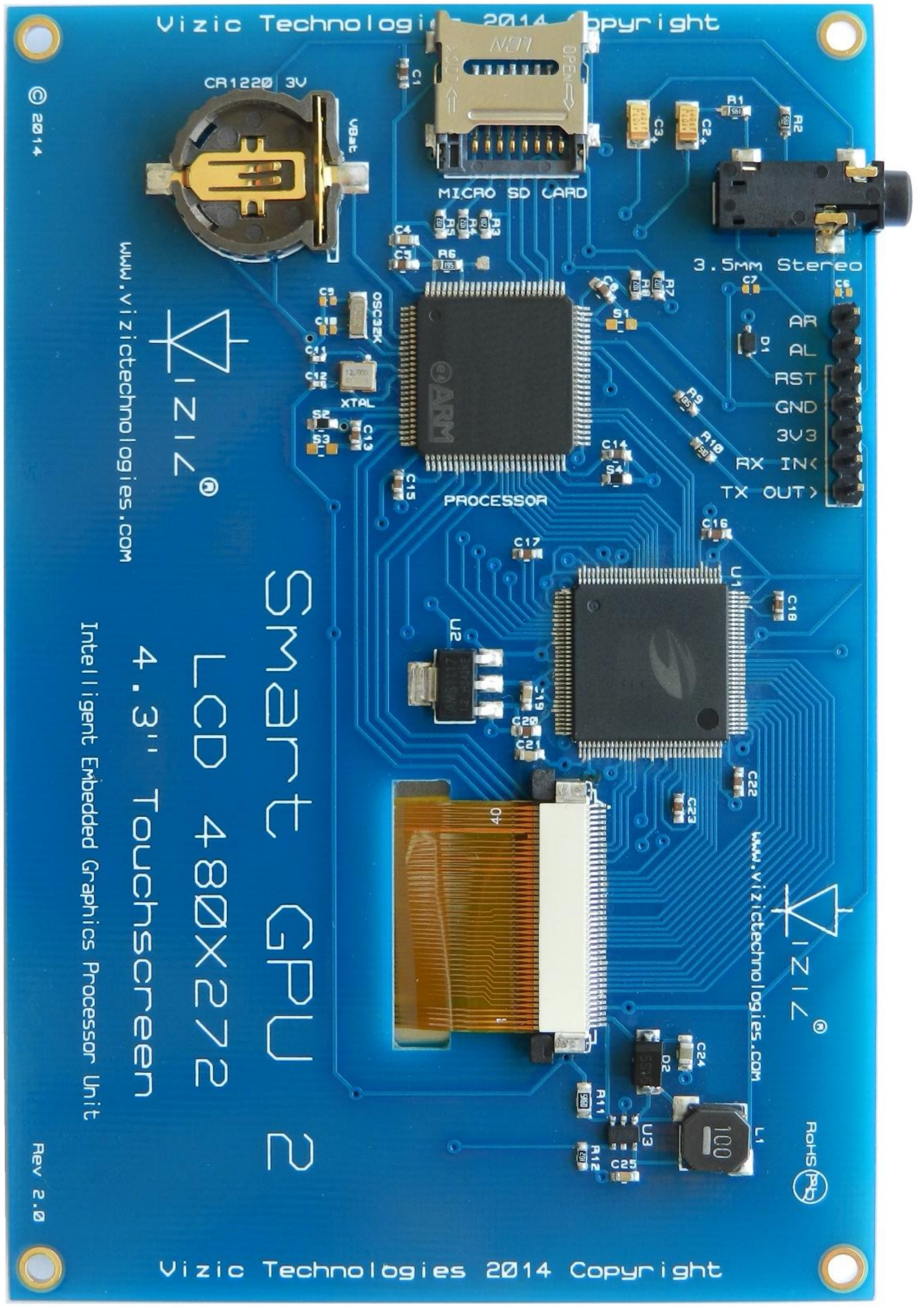
VIZIC TECHNOLOGIES

SMART GPU 2 4.3" TOUCH

COMMAND SET----Rev 1.0

SMART GPU 2– Intelligent Embedded Graphics, Audio and Touch Processor.



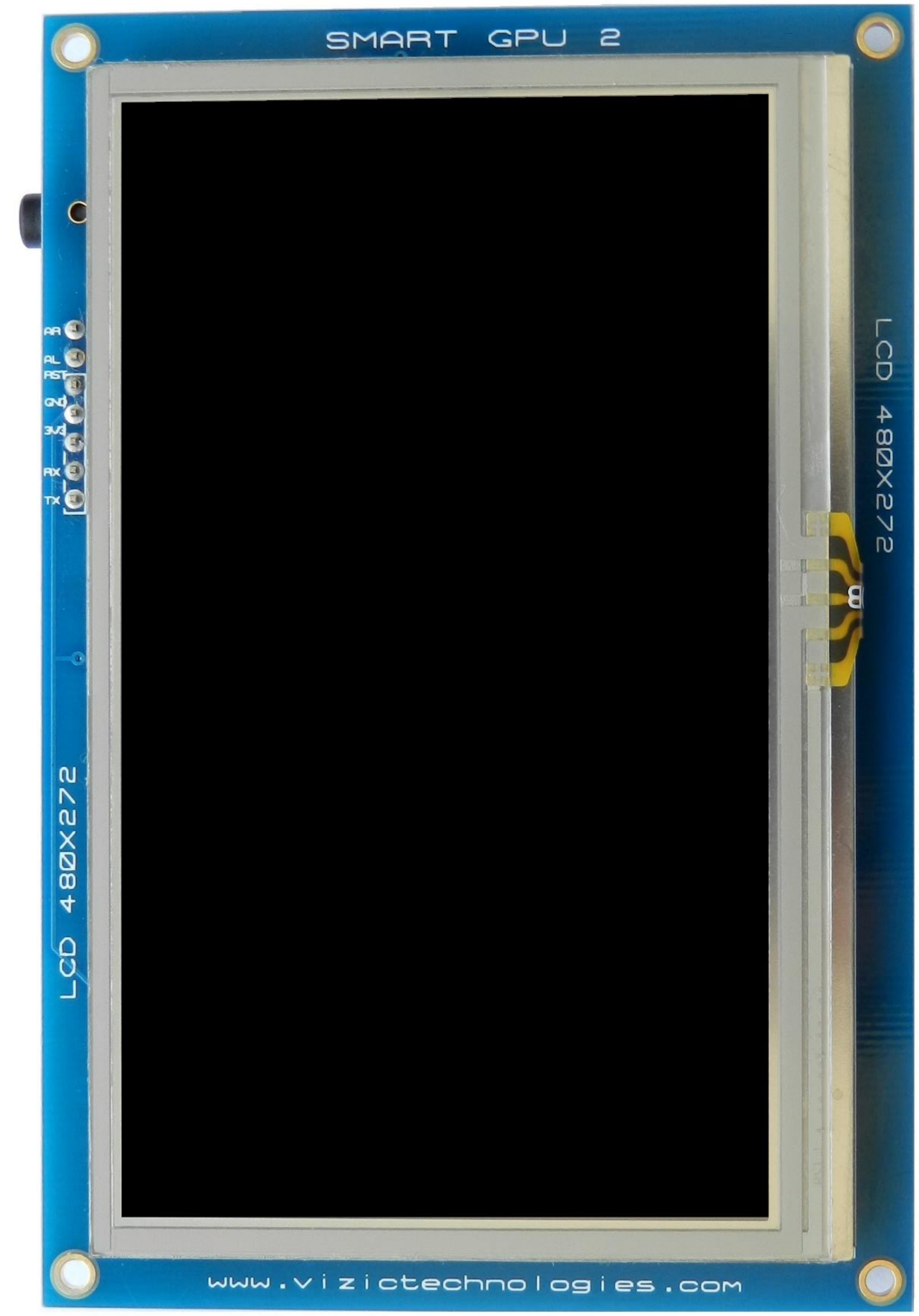
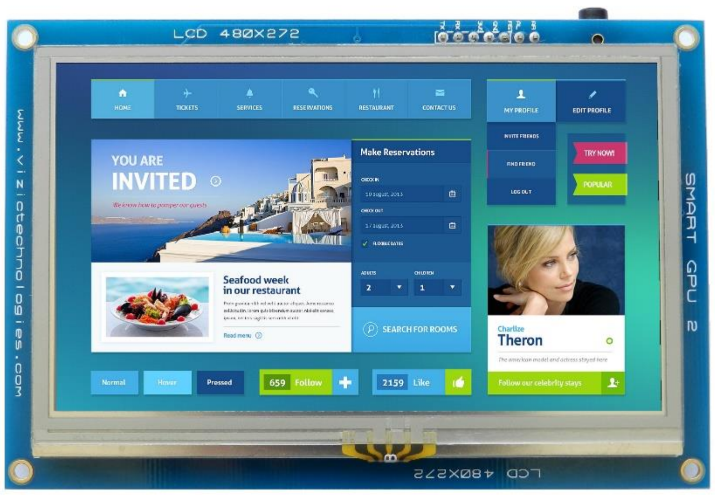

SMART GPU 2
LCD 480X272
LCD 480X272
AR
AL
RST
GND
3V3
RX
TX
www.vizictechnologies.com

Table of Contents:

Smart GPU 2:

Intelligent Embedded Graphics, Audio and Touchscreen Processor.

Description:



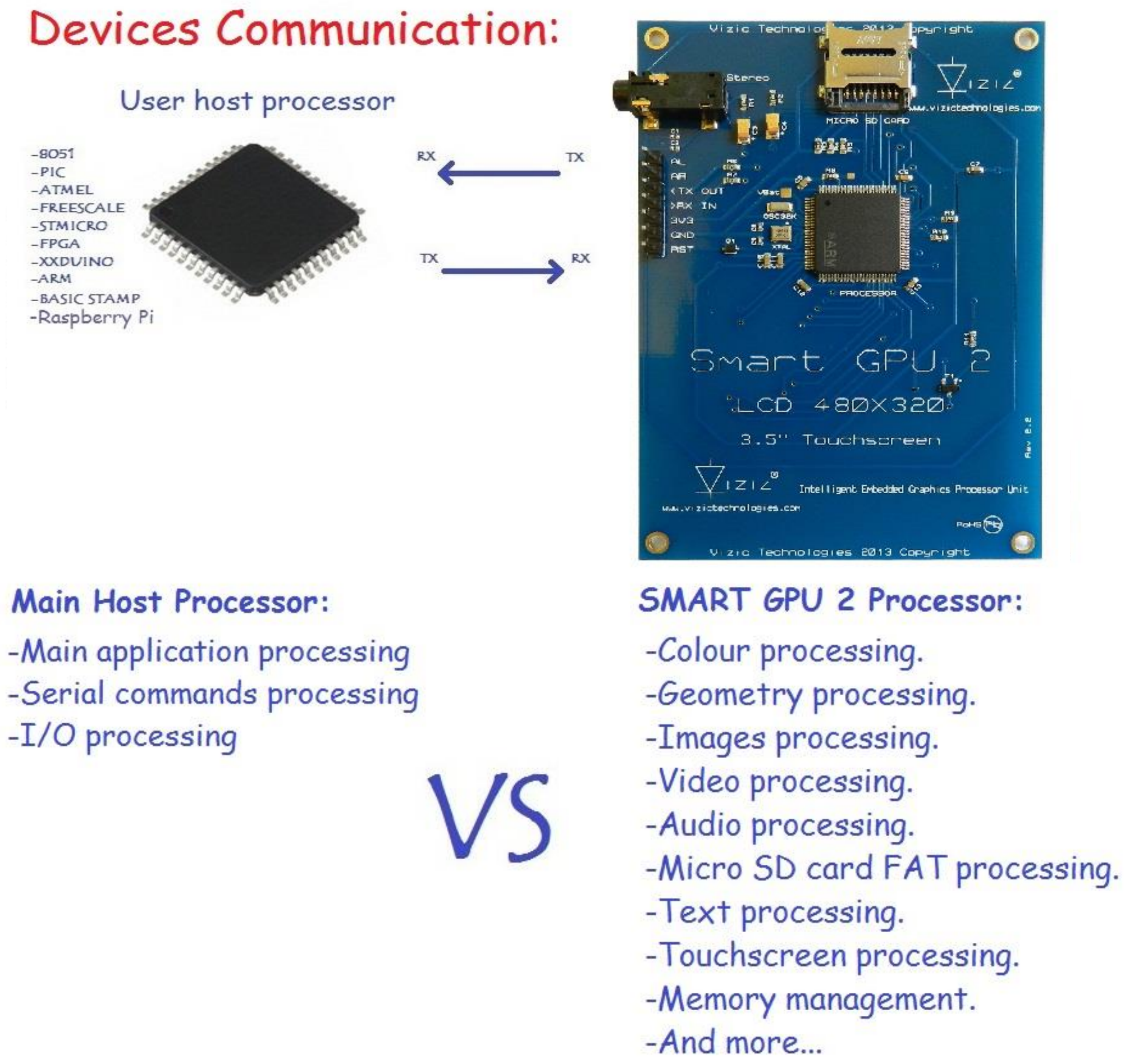
The Smart GPU 2 is a powerful easy to use, intellectual property, embedded graphics, audio and touchscreen processor in a state-of-the-art ARM Cortex-M3 chip; this is mounted on a board with a touchscreen color LCD. It's aimed to help developers to create advanced Graphical User Interfaces (GUIs) in a very easy way. It features high end FAT format data management functions (Data Logger) to create even more advanced applications in just minutes, not days. The Smart GPU 2 processor doesn't need any configuration or programming on itself, it's a slave device that only receives orders, reducing and facilitating dramatically the code size, complexity and processing load in the master host processor.

The Smart GPU 2 offers a simple yet effective serial interface UART to any host micro-controller/microprocessor that can communicate via a serial port(8051, PIC, ATMEL, FREESCALE, STMICRO, ARM, CORTEX, ARDUINO, raspberry PI, FPGA MBED, etc. even PCs(RS232)). All graphics, audio and touchscreen related functions are sent using simple commands via the serial interface.

The main goal of the Smart GPU 2 processor it's to bring a very easy way to add colour, audio and touch interfacing to any application or project without the need of having experience in handling LCDs and graphics algorithms. The Smart GPU 2 it's a low power/very high performance processor, it integrates the FAT/FAT12/FAT16 or FAT32 universal PCs file System for data storage (read/write), supporting up to 32 GB of storage with a microSD/HC card, NO special format is required.

SmartGPU2 chip is also sold as "bare chip" for high end applications and can be adapted to drive any LCD with parallel 8080 interface.

The next image clearly explains the roles played by the user host main processor and the Smart GPU 2 processor:

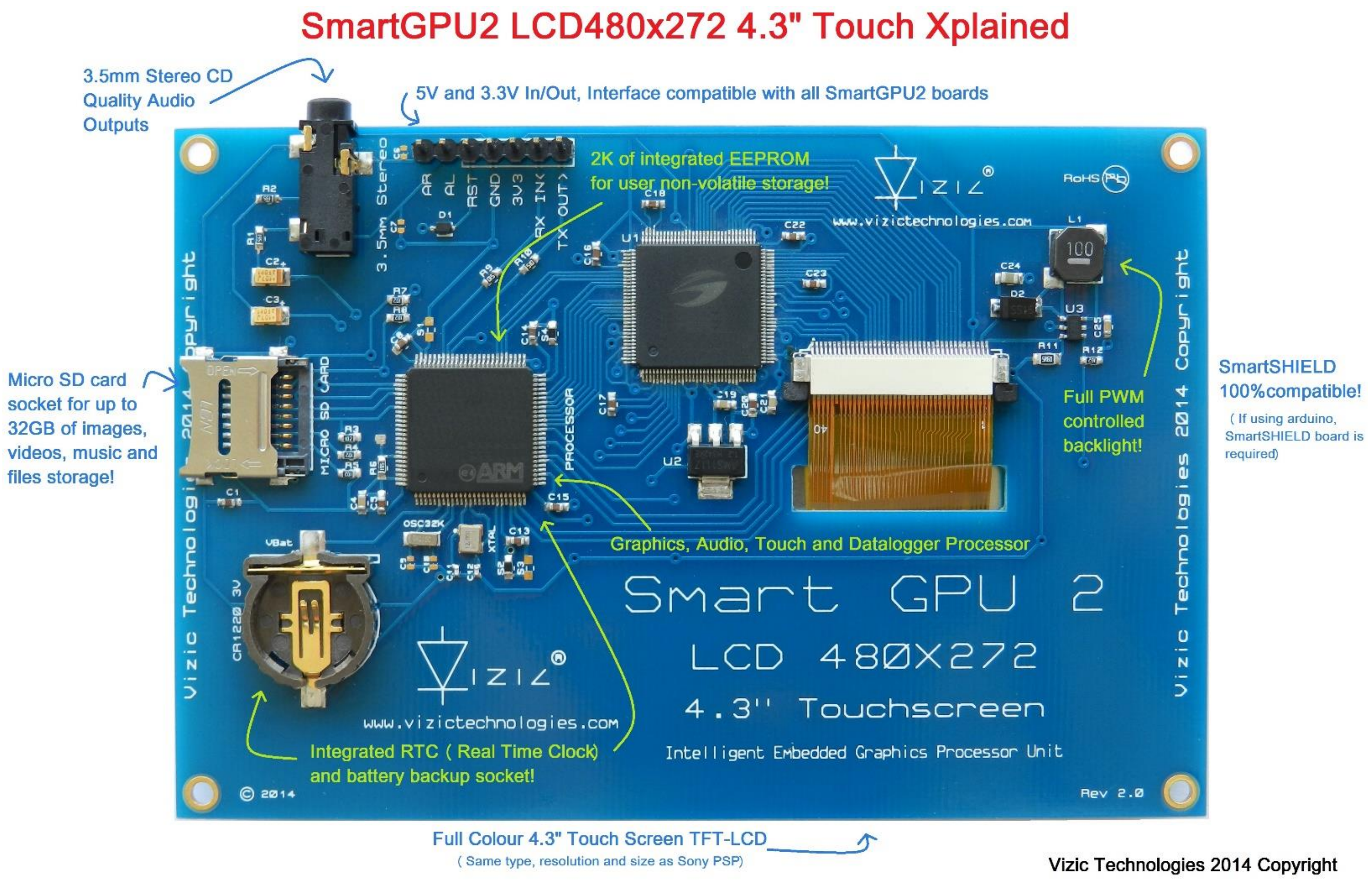


Instead of loading all the Geometry, Images, audio, video, SD FAT memory access, etc. processing to the main host processor, the Smart GPU 2 does the entire job and stuff in parallel with the user microcontroller/microprocessor by receiving simple orders or commands.

Board Features:

- 4.3", 480x272 pixels resistive touchscreen LCD, capable of displaying 262,144 colours.
- Easy 5 pin interface to any host device: **VCC, TX, RX, GND, RESET.**
- On-board uSD/uSDHC card adaptor, FAT(windows PC) Support up to **32GB** for storing images and text, **Data Logger functions (read-write)** and LFN(long File Names).
- BMP and JPG images support.
- Video and Audio(CD quality) capable.
- Integrated 2 channel DACs outputs can play **stereo** audio.
- Integrated Touch screen driver, 12 bit accuracy touch.
- 5 general purpose Icons on touchscreen panel.
- Integrated RTC - Real Time Clock with battery back-up.
- PWM controlled display brightness.
- Sleep mode.
- UART/USART Baud Rate speeds from 9600bps up to 2000000bps, 8 bits, no parity, 1 stop bit.
- 5V and 3V3 I/O compatible, 3V3 power supply.

Smart GPU 2 Board – EXPLAINED


SmartGPU2 LCD480x272 4.3" Touch Xplained
3.5mm Stereo CD Quality Audio Outputs
5V and 3.3V In/Out, Interface compatible with all SmartGPU2 boards
2K of integrated EEPROM for user non-volatile storage!
Micro SD card socket for up to 32GB of images, videos, music and files storage!
SmartSHIELD 100%compatible!
(If using arduino, SmartSHIELD board is required)
Full PWM controlled backlight!
Graphics, Audio, Touch and Datalogger Processor
Integrated RTC (Real Time Clock) and battery backup socket!
Full Colour 4.3" Touch Screen TFT-LCD
(Same type, resolution and size as Sony PSP)
Vizic Technologies 2014 Copyright
Smart GPU 2
LCD 480X272
4.3'' Touchscreen
Intelligent Embedded Graphics Processor Unit
www.vizictechnologies.com

1.-Host Interface

The Smart GPU 2 is a slave peripheral device and it provides a bidirectional serial interface to a host controller via its USART(Universal Serial Asynchronous Receiver - Transmitter).

Any microcontroller or processor (AVR, PIC, BASICstamp, XXDUINO, raspberry PI, 8051, MBED, FPGA, ARM, STmicro, etc) or PC(by serial interface RS232) as host, can communicate to the device over this serial interface from 9600bps up to 2000000bps.

The Smart GPU 2 doesn't need to be configured in any way; it's a plug-and-play device, could be used by students, up to industrial and professional applications, its compatible with any device and existing development board with a USART/UART.

The serial protocol is universal and very easy to implement.
Serial Data Format: 8 Bits, No Parity, 1 Stop Bit.
BaudRate: 9600 bps (default; could be changed).
Serial data is true and not inverted.

1.1 Command Protocol : Flow Control

The Smart GPU 2 Intelligent Graphics Processor Unit is a slave device and all communication and events must be initiated first by the host. Commands consist of a sequence of data bytes beginning with the command/function byte.

When a command is sent from host to the device, this process the command and when the operation is completed, it will always return a response*. The device will send back a single acknowledge byte called the ACK (4Fhex, 'O' ascii), in the case of success, or NAK (46hex, 'F' ascii), in the case of failure or not recognized command.

** 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.*

1.2 Serial Set-up

The Smart GPU 2 is configured to be always initialized at a standard **baud rate of 9600 bps**. So the first command that the host sends to the Smart GPU 2 must be at that speed.

Always after any power-up or reset, the Smart GPU 2 must be initialized by sending the uppercase ascii character '**U**' (55hex) at 9600bps. This will initialize all the processor, and when done it will respond with an ACK byte (4Fhex, 'O'ascii).

If the Smart GPU 2 respond with a NAK(46hex, 'F'ascii), Host must try to send the uppercase ascii character '**U**' (55hex) again until a valid ACK is received, meaning this that Smart GPU 2 is ready and running.

Once the chip is initialized, user can change the baud rate speed to a total of 8 different speeds up to 2Mbps.

Remember:
The SMART GPU 2 always initializes the micro SD card after a valid 'U' character is received. If a micro SD card is detected the ACK 'O' will be response almost immediately, however if no micro SD card is detected, the ACK 'O' could be delayed while the SMART GPU 2 retries to initialize a micro SD card, however if no micro SD card is detected after several tries, the SMART GPU 2 will send the ACK 'O' and the processor will function normally without the SD card functions.

1.3 Power-up and Reset

When the Smart GPU 2 device comes out of a power up or external reset, a 200ms delay before sending any command must be met, do not attempt to communicate with the module before this period.

If no valid uppercase ascii character '**U**' (55hex) is sent before 3 seconds, the Smart GPU 2 logo will automatically show up, host still can send the uppercase ascii character '**U**' (55hex) to initialize the processor even if the logo has already appeared.

Remember:
*The host transmits the upper case character ('**U'**, **55**hex) as the first command so the device to start communication.*

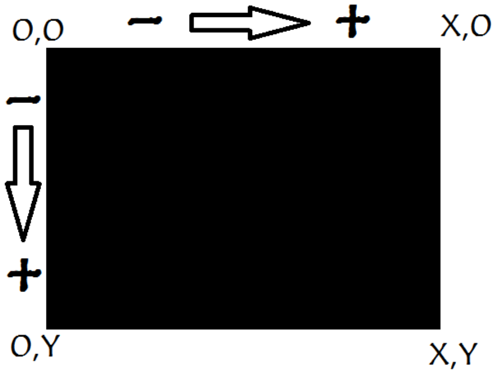
1.4 Splash Screen on Power Up

The Smart GPU 2 will wait up to 3 seconds with its screen in black, for the host to transmit the Initial command ('U', 55hex). If the host has not transmitted this initial command the module will display its splash screen. If the host has transmitted only the initial command and has received a valid ACK, the screen will remain in black. This wait period of the splash screen to appear, is to allow the user initialize the Smart GPU 2 before the welcome screen appears when it is undesired.

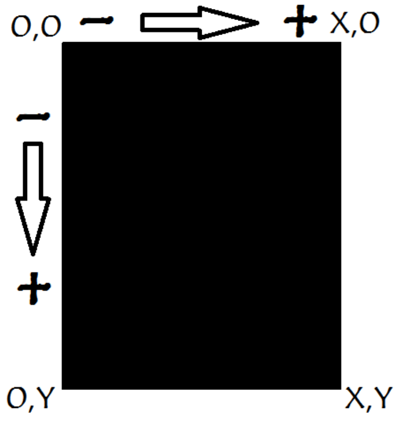
1.5 Understanding the Computer's graphic coordinate system

As well as a computer monitor's coordinate system, the Smart GPU 2 uses the same universal coordinate system, on computer's there's only one positive coordinate quadrant, and there's no negative numbers or points. This quadrant is represented as follows:

The upper left corner is 0,0 if we go right the X values increases, as we go down the Y values increase.



This image shows a LANDSCAPE orientation of the screen, the upper left corner is 0,0 (zero,zero). The maximum values of the SMART GPU 2 in LANDSCAPE mode are X:479,Y:271.



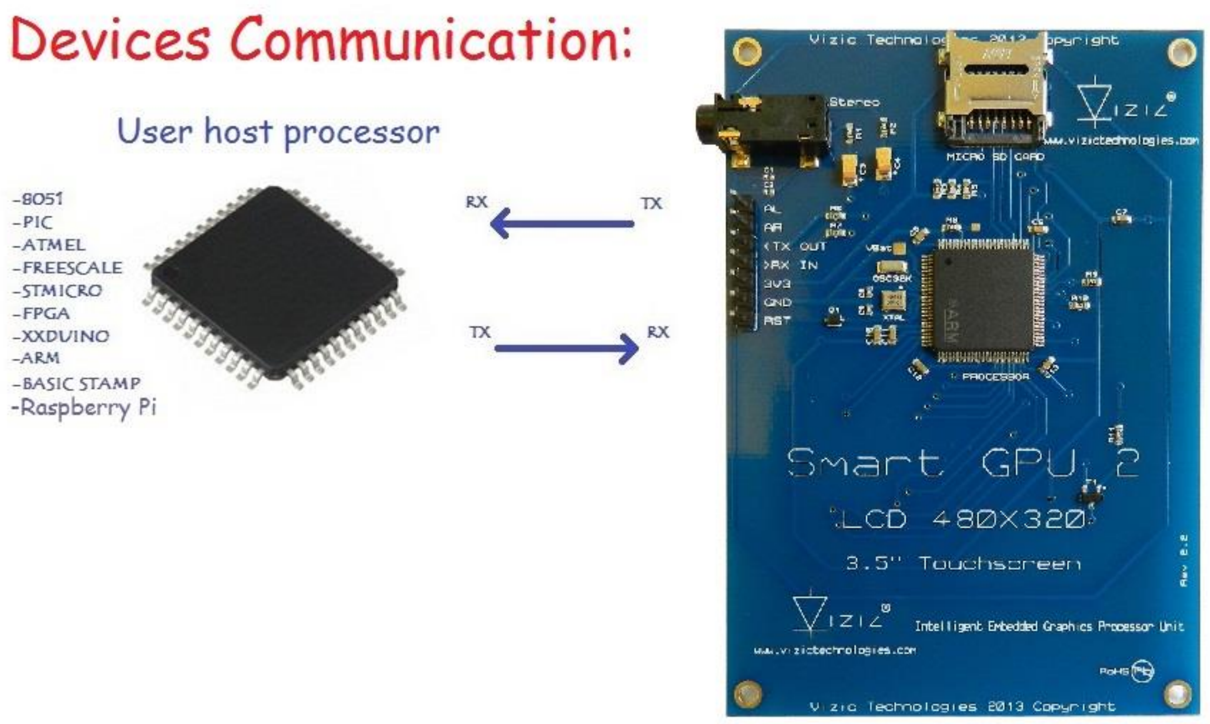
This image shows a PORTRAIT orientation of the screen, the upper left corner is 0,0 (zero,zero). The maximum values of the SMART GPU 2 in PORTRAIT mode are X:271,Y:479.

2. SMART GPU 2 Command Set - Software Interface Specification

As mentioned before the command interface between the Smart GPU 2 and the host processor is via the serial interface USART/UART.

A list of very easy to learn commands provide complete access to all the available functions. Commands and responses can be a single byte or a byte package. All commands always return a response, either a single ACK, or data followed by an ACK.

Remember all commands start with a uppercase letter (ascii).



2.1 General Commands

Briefly Summary of Commands in this section:

• Initialize SMART GPU 2 – **55hex 'U'**

2.1.1 Initialize SMART GPU 2 - 55hex - U ascii

Commands (host)	1 byte
	1.- 0x55 (hex), U (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command is needed only once to initialize communication with the SMARTGPU2 after any power-up or reset, remember to wait at least 200ms after any power up or reset before sending this command.
Example (sent commands)	*Example 1:* <55> Initializes SMART GPU 2. All data is hex.

2.2 Master Commands

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'M'-4Dhex, as they are Master commands, followed by the next parameters/bytes.**

• Erase Screen	– **45hex 'E'**
• Set Erase Background Colour	– **43hex 'C'**
• Display Orientation	– **4Fhex 'O'**
• Display Brightness	– **42hex 'B'**
• BaudRate Change	– **58hex 'X'**
• Sleep	– **5Ahex 'Z'**
• Calibrate Touch	– **54hex 'T'**
• Software Reset	– **52hex 'R'**

The colour parameter needed on the Set Erase Background Colour command,

consist of 16bits (2 bytes) RGB565:

R4R3R2R1R0G5G4G3 G2G1G0B4B3B2B1B0

That is:

5bits for red, 6 bits for green, 5bits for blue.

High byte colour: R4R3R2R1R0G5G4G3

Low byte colour: G2G1G0B4B3B2B1B0

2.2.1 Erase Screen - 45hex - E ascii

Commands (host)	2 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x45 (hex), E (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to erase the entire screen, the display will be draw with the background colour if it is set before, if not, default background colour on reset or power on is black.
Example (sent commands)	*Example 1:* <4D, 45> - Erase screen with currently set background colour. All data is in hex.

2.2.2 Set Erase Background Colour - 43hex - C ascii

Commands (host)	4 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x43 (hex), C (ascii). 3.- High byte colour. 4.- Low byte colour.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to set background, the colour consist of 16bits (2 bytes) RGB565: R4R3R2R1R0G5G4G3G2G1G0B4B3B2B1B0 That is: 5bits for red, 6 bits for green, 5bits for blue. High byte : R4R3R2R1R0G5G4G3 Low byte : G2G1G0B4B3B2B1B0 Once this command is sent, each time that the device receives the Erase Command, the screen will be draw with this background colour. Default background colour on reset or power on is black.
Example (sent commands)	*Example 1:* <4D,43,FF,FF> Sets Background Colour to white (FFFF). *Example 2:* <4D,43,F8,00> Sets Background Colour to red (F800). *Example 3:* <4D,43,00,1F> Sets Background Colour to blue (001F). All data is in hex.

2.2.3 Display Orientation – 4Fhex - O ascii

Commands (host)	3 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x4F (hex), O (ascii). 3.- HorizontalR(landscape): 00(hex) or VerticalL(portrait): 01(hex) or HorizontalL(landscape): 02(hex) or VerticalT(portrait): 03(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to set display orientation mode, landscape or portrait. Default display orientation mode on reset or power on is HorizontalR 00(hex).
Example (sent commands)	*Example 1:* <4D,4F,01> Set VerticalL mode. *Example 2:* <4D,4F,02> Set HorizontalL mode. All data is in hex.

2.2.4 Display Brightness - 42hex - B ascii

Commands (host)	3 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x42 (hex), B (ascii). 3.- Brightness value (0-100) (0 hex-64 hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to adjust the display brightness, 0(0hex) stands for none, 100(64hex) stands for maximum brightness. Default brightness on reset or power on is 100(64hex).
Example (sent commands)	*Example 1:* <4D,42,00> Set minimum brightness (leds off). *Example 2:* <4D,42,64> Set maximum brightness. *Example 3:* <4D,42,32> Set half brightness. All data is in hex.

2.2.5 BaudRate Change – 58hex - X ascii

Commands (host)	6 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x58 (hex), X (ascii). 3.- Baud Rate High Byte. 4.- Baud Rate Medium High Byte. 5.- Baud Rate Medium Low Byte. 6.- Baud Rate Low Byte.
Responses (device)	**1 byte when fail, 2 bytes when success**
	1.- 0x4F (hex), O (ascii) – success ACK at actual baud rate. --Delay of 500ms. 2.- 0x4F (hex), O (ascii) – success ACK at new baudrate. or 1.- 0x46 (hex), F (ascii) – fail NAK at actual baud rate.
Description	Command needed to set a different baud rate, the command sent by the host must be at the actual baud rate that is being used; if the command is invalid a NAK will be responded by the SMART GPU and the baud rate will not be modified. If the command is accepted an ACK will be received at the actual baud rate then SMART GPU will change to the new baud rate during 500ms and then it will send another ACK at the new baud rate selected. Only when an ACK has been received by the host, the next commands must be sent at the new baud rate defined. Default reset - power on baud rate is 9600.
Example (sent commands)	*Example 1:* <4D,58,00,01,C2,00> Set 115200 bps baud. *Example 2:* <4D,58,00,1E,84,80> Set 2000000 bps baud. All data is in hex.

2.2.6 Sleep – 5Ahex - Z ascii

Commands (host)	3 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x5A (hex), Z (ascii). 3.- Sleep On: 01(hex). Sleep Off: 00(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to set sleep mode. It takes 150ms to get in/out sleep mode after command is accepted. During sleep mode the screen turns completely white, the LCD oscillator and voltage generator turns off, but the memory is conserved. To save more power combine this command with Display Brightness set to zero.
Example (sent commands)	*Example 1:* <4D,5A,01> Set sleep mode On. *Example 2:* <4D,5A,00> Set sleep mode Off. All data is in hex.

2.2.7 Calibrate Touch – 54hex - T ascii

Commands (host)	3 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x54 (hex), T (ascii). 3.- 0x43 (hex), C (ascii). Calibrate. 0x53 (hex), S (ascii). Save Values. 0x44 (hex), D (ascii). Delete Values.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command helps user to Calibrate touch, save calibration values to flash, and delete-restore factory touch calibration values from flash. It's recommended to perform any calibration process with a stylus, not fingers. The SmartGPU 2 contains saved in flash 4 values that are the factory calibration values to get a correct touch reading on screen when user press the panel. User can update those values by saving new calibration values, or can restore factory values by calling the Delete Values command. When the user calls the command Calibrate(M,T,C) , the SmartGPU will execute the next process to calibrate touchscreen: 1.-Screen will turn red.(means calibration process begin). 2.-Screen will turn black. 3.-A white dot will be printed on one corner of the screen, user must touch this point and hold while the screen turns green, release point Immediately after the screen turns green. 4.-Screen turns green. (means calibration of First point is done). 5.-Screen will turn black. 6.-A second white dot will be printed on one corner of the screen, user must touch this point and hold while the screen turns blue, release point immediately after the screen turns blue. 7.-Screen turns blue. (means calibration of Second point is done). 8.-Screen will turn black.(means end of process).

	9.-Finally the SmartGPU will store on **RAM** the new 4 working calibration values obtained with touch on points pressing. User can now call some drawing with touch routine to check If the calibration was successfully and the values are correct. When the above calibration process is executed, calibration values are **only stored on RAM**, this means that if a reset or power off is performed, those values will be lost, so once that user performs a calibration process and its sure that the values are correct, values can be saved with the command Save Values(M,T,S) from RAM to **non-volatile memory FLASH**, This way even if a reset or power off is performed, values will remain saved. Last command Delete-restore factory calibration values(M,T,D) deletes any saved values by user to **non-volatile memory FLASH**, and restores the factory calibration values to SmartGPU2 touchscreen. A common touchscreen calibration procedure will be: 1.-Delete Values(M,T,D).(get default factory values). 2.-Perform calibration process(M,T,C). 3.-Test updated values with some routine of drawing points on screen with the received touch coordinates(points). 4.-If user is happy with the new updated working calibration values go to 5, if not, go to 2. 5.-Save to flash working calibration values(M,T,S)
Example (sent commands)	*Example 1:* <4D,54,43> Run calibration process. *Example 2:* <4D,54,53> Save Values to Flash. All data is in hex.

2.2.8 Software Reset – 52hex - R ascii

Commands (host)	2 bytes
	1.- 0x4D (hex), M (ascii). *Master command 2.- 0x52 (hex), R (ascii).
Responses (device)	None
Description	This command performs total software RESET on the SmartGPU2, it has the same effect as a hardware RESET by the RST pin. After this command is called, user must have to initialize the SmartGPU2 with the initialize SmartGPU command ('U').
Example (sent commands)	*Example 1:* <4D,52> Reset SmartGPU 2. All data is in hex.

2.3 Geometry Commands

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'G'-47hex, as they are Geometry commands, followed by the next parameters/bytes.**

• Put Pixel	– **50hex 'P'**
• Draw Line	– **4Chex 'L'**
• Draw Rectangle	– **52hex 'R'**
• Draw Round Rectangle	– **4Fhex 'O'**
• Draw Gradient Rectangle	– **47hex 'G'**
• Draw Arc	– **41hex 'A'**
• Draw Circle	– **43hex 'C'**
• Draw Ellipse	– **45hex 'E'**
• Draw Triangle	– **54hex 'T'**

The colour parameter needed on all of those commands, consist of 16bits (2 bytes) RGB565:

R4R3R2R1R0G5G4G3 G2G1G0B4B3B2B1B0

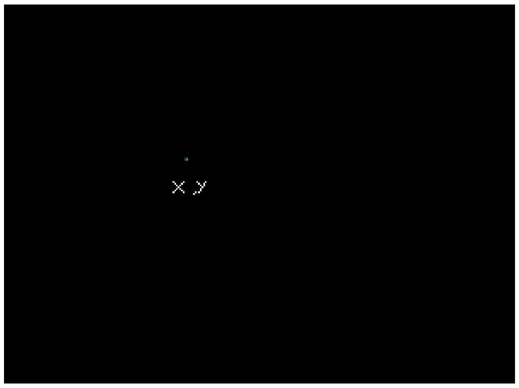
That is:

5bits for red, 6 bits for green, 5bits for blue.

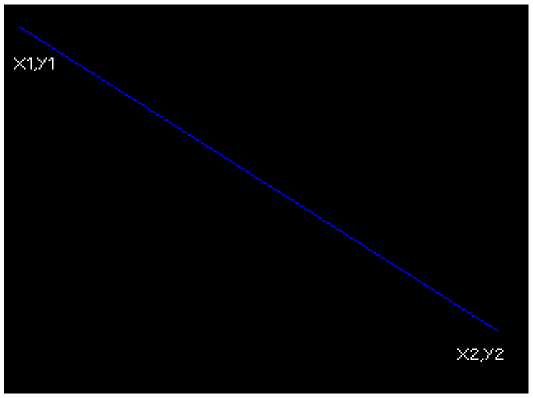
High byte colour: R4R3R2R1R0G5G4G3

Low byte colour: G2G1G0B4B3B2B1B0

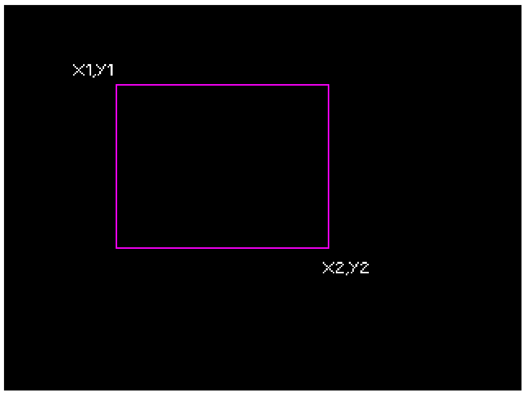
2.3.1 Put Pixel – 50hex - P ascii

Commands (host)	8 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x50 (hex), P (ascii). 3.- X high byte. 4.- X low byte. 5.- Y high byte. 6.- Y low byte. 7.- High byte colour. 8.- Low byte colour.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple dot on the screen at the given X(16bit) and Y(16bit) coordinates. Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <47,50,00,32,00,3C,07,E0> Put a Green Pixel at X:50(dec), Y:60(dec).  All data is in hex. **Note:** Maximum X or Y acceptable size value depend on display orientation.

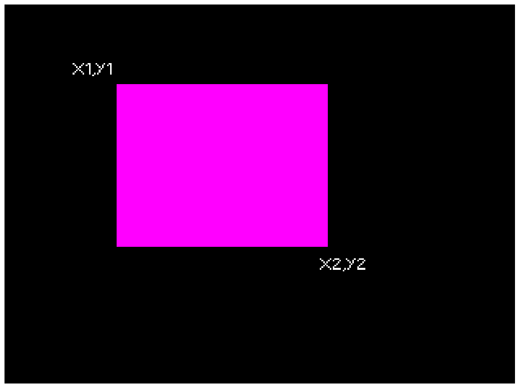
2.3.2 Draw Line – 4Chex - L ascii

Commands (host)	12bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x4C (hex), L (ascii). 3.- X1 high byte. 4.- X1 low byte. 5.- Y1 high byte. 6.- Y1 low byte. 7.- X2 high byte. 8.- X2 low byte. 9.- Y2 high byte. 10.- Y2 low byte. 11.- High byte colour. 12.- Low byte colour.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple line on the screen with the two given points: X1(16bit), Y1(16bit) and X2(16bit),Y2(16bit). Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <47,4C,00,0A,00,0F,01,2C,00,C8,00,1F> Draws a Blue line from X1:10(dec),Y1:15(dec) to X2:300(dec),Y2:200(dec).  All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.3.3 Draw Rectangle – 52hex - R ascii

Commands (host)	13 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x52 (hex), R (ascii). 3.- X1 high byte. 4.- X1 low byte. 5.- Y1 high byte. 6.- Y1 low byte. 7.- X2 high byte. 8.- X2 low byte. 9.- Y2 high byte. 10.- Y2 low byte. 11.- High byte colour. 12.- Low byte colour. 13.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple rectangle on the screen with the two given points: X1(16bit), Y1(16bit) and X2(16bit),Y2(16bit). Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <47,52,00,46,00,32,00,C8,00,96,F8,1F,00> Draws a no filled purple rectangle from X1:70(dec),Y1:50(dec) to X2:200(dec), Y2:150(dec). 

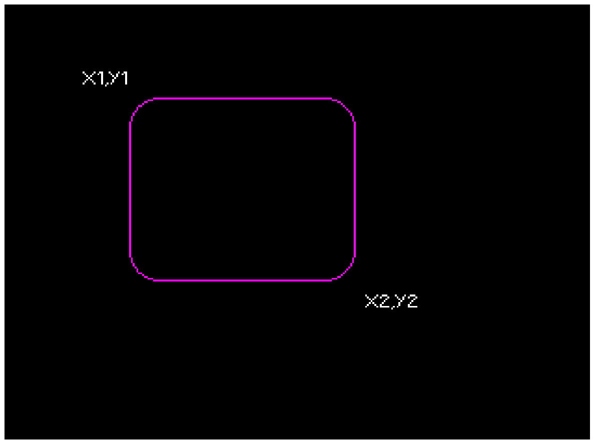
Example 2:
<47,52,00,46,00,32,00,C8,00,96,F8,1F,01>
Draws a filled purple rectangle from X1:70(dec),Y1:50(dec) to X2:200(dec), Y2:150(dec).



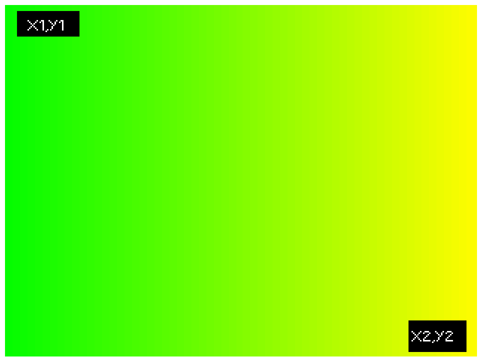
All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.3.4 Draw Round Rectangle – 4Fhex - O ascii

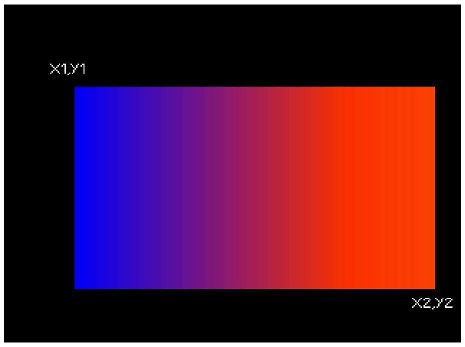
Commands (host)	15 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x4F (hex), O (ascii). 3.- X1 high byte. 4.- X1 low byte. 5.- Y1 high byte. 6.- Y1 low byte. 7.- X2 high byte. 8.- X2 low byte. 9.- Y2 high byte. 10.- Y2 low byte. 11.- Radius high byte. 12.- Radius low byte. 13.- High byte colour. 14.- Low byte colour. 15.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple round rectangle on the screen with the two given points: X1(16bit), Y1(16bit) and X2(16bit), Y2(16bit), with given Radius(16bit). Colour format is the same RGB565. **Note that the next conditions must meet to command success:* *1.- (Radius * 2) < (y2-y1)* + *2.- (Radius * 2) < (x2-x1)* *If above conditions are not meet, command will fail with NAK 'F'*
Example (sent commands)	*Example 1:* <47,52,00,46,00,32,00,C8,00,96,00,14,F8,1F,00> Draws a no filled rounded purple rectangle from X1:70(dec), Y1:50(dec) to X2:200(dec), Y2:150(dec) and Radius:20(dec).

	 All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

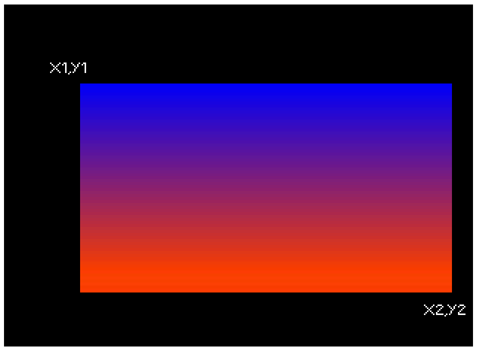
2.3.5 Draw Gradient Rectangle – 47hex - G ascii

Commands (host)	15 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x47 (hex), G (ascii). 3.- X1 high byte. 4.- X1 low byte. 5.- Y1 high byte. 6.- Y1 low byte. 7.- X2 high byte. 8.- X2 low byte. 9.- Y2 high byte. 10.- Y2 low byte. 11.- High byte first colour. 12.- Low byte first colour. 13.- High byte second colour. 14.- Low byte second colour. 15.- Direction: 0x00(hex) Horizontal. 0x01(hex) Vertical.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple gradient rectangle on the screen with the two given points: X1(16bit), Y1(16bit) and X2(16bit),Y2(16bit). The first colour and second colour are mix as they approach the middle of the rectangle. Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <47,47,00,00,00,00,01,3F,00,EF,07,E0,FF,E0,00> Draws a horizontal green + yellow gradient rectangle from X1:00(dec), Y1:00(dec) to X2:319(dec), Y2:239(dec). 

Example 2:
<47,47,00,32,00,3C,01,2C,00,C8,00,1F,F9,E0,00>
Draws a horizontal blue + red gradient rectangle from X1:50(dec), Y1:60(dec) to X2:300(dec), Y2:200(dec).

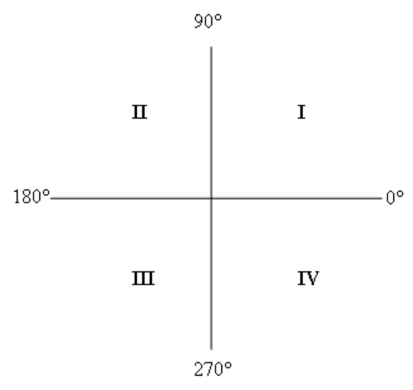


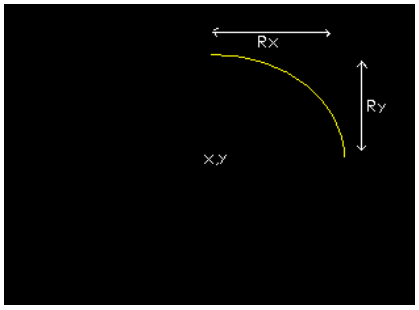
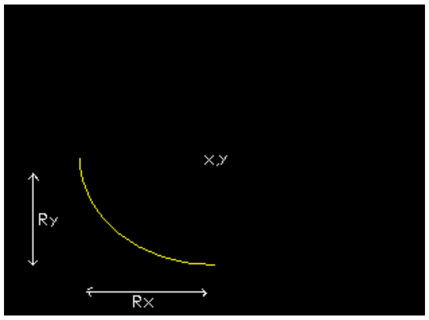
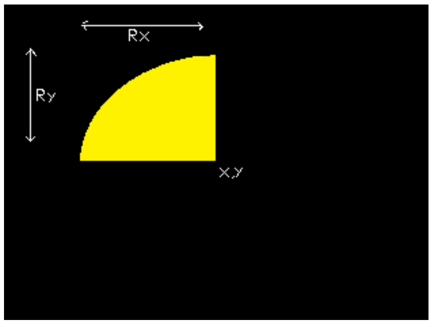
Example 3:
<47,47,00,32,00,3C,01,2C,00,C8,00,1F,F9,E0,01>
Draws a vertical blue + red gradient rectangle from X1:50(dec), Y1:60(dec) to X2:300(dec), Y2:200(dec).



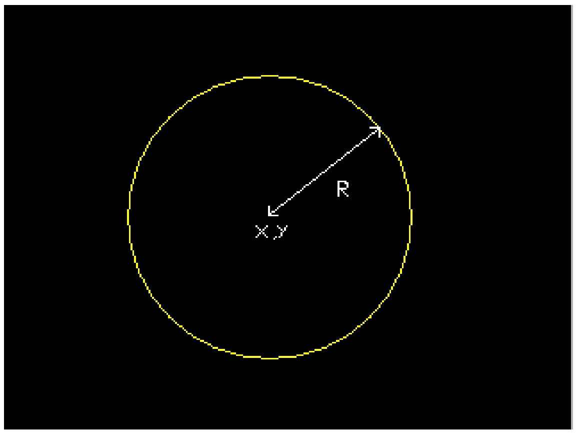
All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

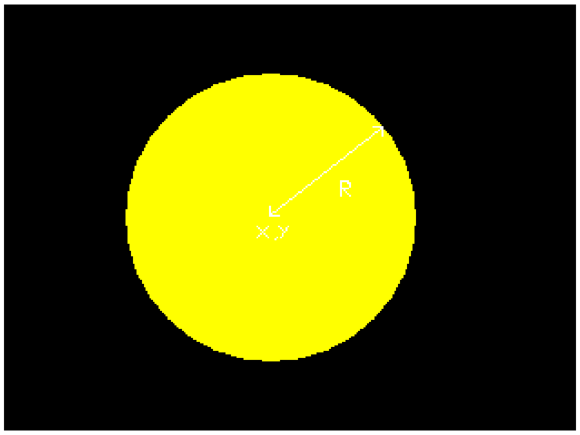
2.3.6 Draw Arc – 41hex - A ascii

Commands (host)	14 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x41 (hex), A (ascii). 3.- X high byte. 4.- X low byte. 5.- Y high byte. 6.- Y low byte. 7.- Radius in X high byte. 8.- Radius in X low byte. 9.- Radius in Y high byte. 10.- Radius in Y low byte. 11.- Arc quadrant to draw. 12.- High byte colour. 13.- Low byte colour. 14.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	**1 byte**
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple arc quadrant on the screen with center on the given point: X(16bit), Y(16bit), RADIUSX(16bit) and RADIUSY(16bit) values. Colour format is the same RGB565. The arc quadrants are:  *Radius values must be always different than zero.*

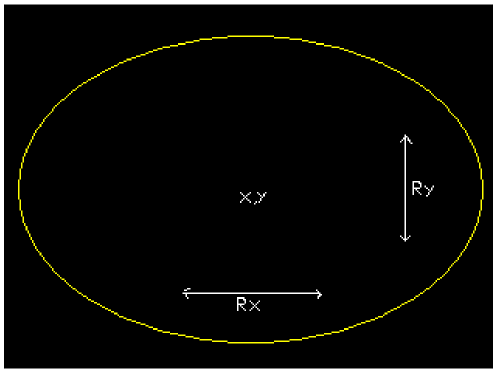
Example (sent commands)	*Example 1:* <47,41,00,A0,00,78,00,64,00,50,01,FF,E0,00> Draws a no filled yellow Arc in quadrant 1, with center in X:160(dec), Y:120(dec), RADIUSX:100(dec) and RADIUSY:80(dec).  *Example 2:* <47,41,00,A0,00,78,00,64,00,50,03,FF,E0,00> Draws a no filled yellow Arc in quadrant 3, with center in X:160(dec), Y:120(dec), RADIUSX:100(dec) and RADIUSY:80(dec).  *Example 3:* <47,41,00,A0,00,78,00,64,00,50,02,FF,E0,01> Draws a filled yellow Arc in quadrant 2, with center in X:160(dec), Y:120(dec), RADIUSX:100(dec) and RADIUSY:80(dec).  All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.3.7 Draw Circle – 43hex - C ascii

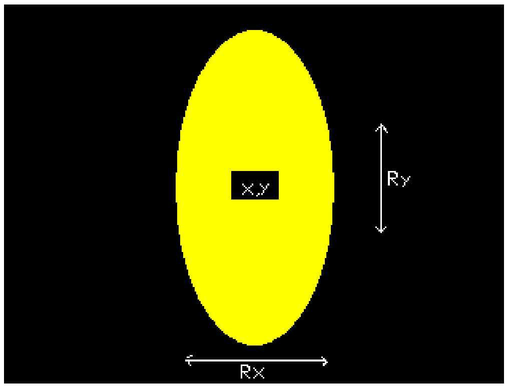
Commands (host)	11 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x43 (hex), C (ascii). 3.- X high byte. 4.- X low byte. 5.- Y high byte. 6.- Y low byte. 7.- Radius high byte. 8.- Radius low byte. 9.- High byte colour. 10.- Low byte colour. 11.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple circle on the screen with center on the given point: X(16bit),Y(16bit) and RADIUS(16bit) value. Colour format is the same RGB565. *Radius value must be always different than zero.*
Example (sent commands)	*Example 1:* <47,43,00,96,00,78,00,50,FF,E0,00> Draws a no filled yellow circle with center X:150(dec), Y:120(dec) and RADIUS:80(dec). 

	Example 2: <47,43,00,96,00,78,00,50,FF,E0,01> Draws a filled yellow circle with center X:150(dec), Y:120(dec) and RADIUS:80(dec).  All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.3.8 Draw Ellipse – 45hex - E ascii

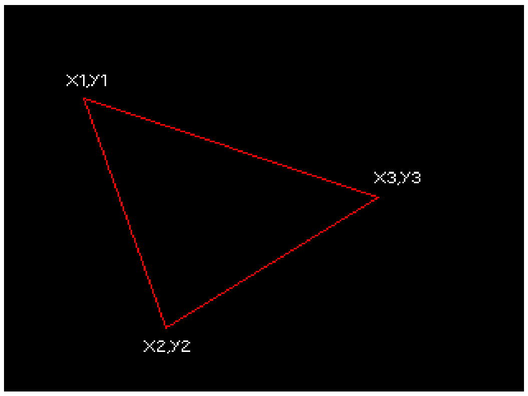
Commands (host)	13 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x45 (hex), E (ascii). 3.- X high byte. 4.- X low byte. 5.- Y high byte. 6.- Y low byte. 7.- Radius in X high byte. 8.- Radius in X low byte. 9.- Radius in Y high byte. 10.- Radius in Y low byte. 11.- High byte colour. 12.- Low byte colour. 13.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple ellipse on the screen with center on the given point: X(16bit), Y(16bit), RADIUSX(16bit) and RADIUSY(16bit) values. Colour format is the same RGB565. *Radius values must be always different than zero.*
Example (sent commands)	*Example 1:* <47,45,00,A0,00,78,00,96,00,64,FF,E0,00> Draws a no filled yellow ellipse, with center in X:160(dec), Y:120(dec), RADIUSX:150(dec) and RADIUSY:100(dec). 

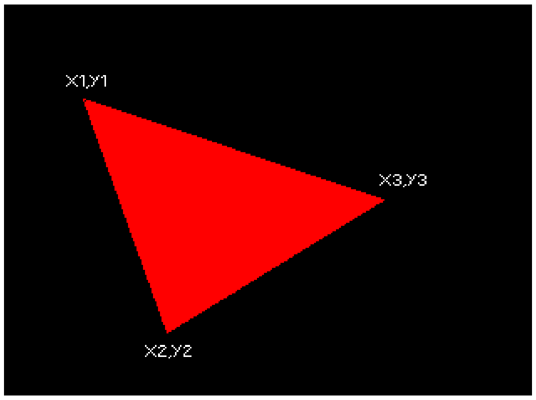
Example 2:
<47,45,00,A0,00,78,00,32,00,64,FF,E0,01>
Draws a filled yellow ellipse, with center in X:160(dec), Y:120(dec), RADIUSX:50(dec) and RADIUSY:100(dec).



All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.3.9 Draw Triangle – 54hex - T ascii

Commands (host)	17 bytes
	1.- 0x47 (hex), G (ascii). *Geometry Command. 2.- 0x54 (hex), T (ascii). 3.- X1 high byte. 4.- X1 low byte. 5.- Y1 high byte. 6.- Y1 low byte. 7.- X2 high byte. 8.- X2 low byte. 9.- Y2 high byte. 10.- Y2 low byte. 11.- X3 high byte. 12.- X3 low byte. 13.- Y3 high byte. 14.- Y3 low byte. 15.- High byte colour. 16.- Low byte colour. 17.- Fill: 0x00(hex) No Fill Geometry or 0x01(hex) Fill Geometry.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command draws a simple triangle on the screen with the given points: X1(16bit),Y1(16bit), X2(16bit),Y2(16bit) and X3(16bit),Y3(16bit). Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <47,54,00,32,00,3C,00,64,00,C8,00,E6,00,78,F8,00,00> Draws a no filled red triangle with given points: X1:50(dec), Y1:60(dec), X2:100(dec), Y2:200(dec) and X3:230(dec), Y3:120(dec). 

	Example 2: <47,54,00,32,00,3C,00,64,00,C8,00,E6,00,78,F8,00,01> Draws a filled red triangle with given points: X1:50(dec), Y1:60(dec), X2:100(dec), Y2:200(dec) and X3:230(dec), Y3:120(dec).  All data is in hex. **Note:** Maximum Xs or Ys acceptable size values depend on display orientation.

2.4 Text/String Commands

The SMART GPU 2 is capable of generate 13 different sizes of fonts, with transparent or colour background, font type is Helvetica Neue.

The command Display String requires two points that forms an imaginary text box that helps delimit the text writing to certain area; it is helpful when the user only wants to display text only on some area of the screen.

Smart GPU 2 can also manage text files, so any file with .txt extension can be easily opened and displayed with selected size and colour.

A maximum of 32GBs micro SD memory card is supported, allowing storing thousands of text files.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'S'-53hex, as they are String commands, followed by the next parameters/bytes.**

• Put Letter	– **4Chex 'L'**
• Print Number	– **4Ehex 'N'**
• Display String	– **53hex 'S'**
• Display String SD	– **46hex 'F'**
• Strings Configuration	– **43hex 'C'**

The colour parameter needed on all those commands, consist of 16bits (2 bytes) RGB565:

R4R3R2R1R0G5G4G3 G2G1G0B4B3B2B1B0

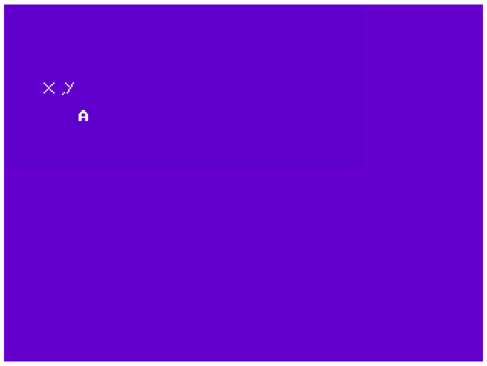
That is:

5bits for red, 6 bits for green, 5bits for blue.

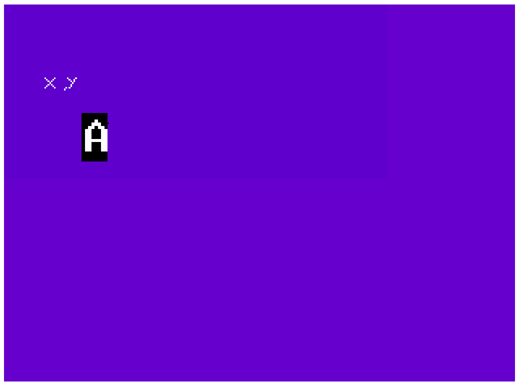
High byte colour: R4R3R2R1R0G5G4G3

Low byte colour: G2G1G0B4B3B2B1B0

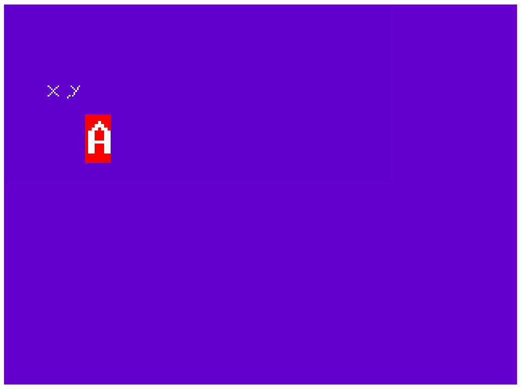
2.4.1 Put Letter – 4Chex - L ascii

Commands (host)	7 bytes
	1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x4C (hex), L (ascii). 3.- X coord high byte. 4.- X coord low byte. 5.- Y coord high byte. 6.- Y coord low byte. 7.- letter to display in (0xXX) hex (ascii equivalent).
Responses (device)	Up X (2 bytes) + 1 byte ACK
	1.- Up X high byte 2.- Up X low byte 3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **Up X stands for Updated X position.*
Description	This command displays a given char or letter on the screen with the given point: X(16bit), Y(16bit) as top left corner. It returns the Updated X, this is useful to know where X position to print the next characters and avoid characters overlapping. *Text size, text colour, text background colour, and filled of unfilled background must be set with the **Strings Config Command**, Defaults are Size 0, White text colour, Black background colour and unfilled background.* Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <53,4C,00,14,00,1E,41> Draws an 'A' letter, with top left corner at X:20(dec),Y:30(dec). With Default settings. 

Example 2:
<53,4C,00,14,00,1E,41> Draw an 'A' letter, with top left corner at X:20(dec),Y:30(dec). Filled Background colour is Set and Font Size was changed to Size 3.

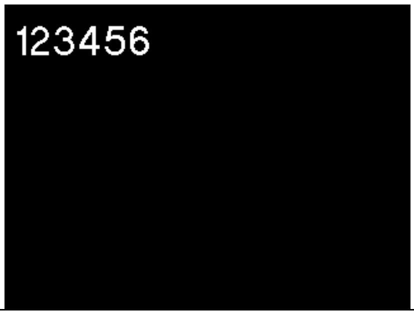


Example 3:
<53,4C,00,14,00,1E,41> Draw an 'A' letter, with top left corner at X:20(dec),Y:30(dec), Background colour was changed to RED, Filled Background colour is Set and Font Size was changed to Size 3.

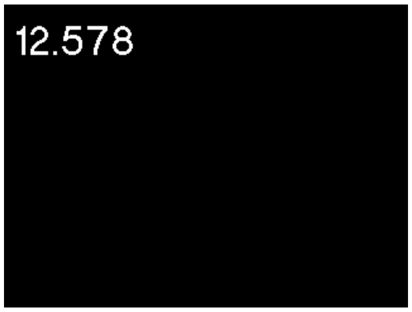


All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

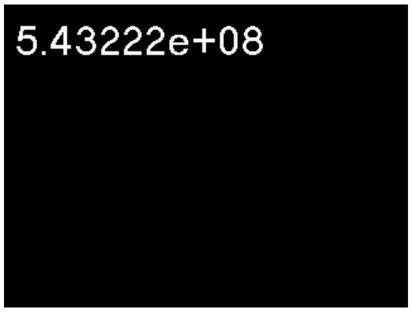
2.4.2 Print Number – 4Ehex - N ascii

Commands (host)	10 bytes
	1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x4E (hex), N (ascii). 3.- X coord high byte. 4.- X coord low byte. 5.- Y coord high byte. 6.- Y coord low byte. 7.- Float Number High Byte. 8.- Float Number Medium High Byte. 9.- Float Number Medium Low Byte. 10.-Float Number Low Byte.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This command displays/prints a received **float number**(4 bytes), at the received top left X(16bit) and Y(16bit) coordinates, the number is printed with the minimum notation possible, this means that zeros will be cut, as an example a zero(0x00000000) will be printed as "0" instead of "0.000000". If number to print doesn't fit on the screen with the given X, Y coordinates or current font size, number will be discarded and command will fail with NAK 'F'. *Text size, text colour, text background colour, and filled of unfilled background must be set with the **Strings Config Command**, Defaults are Size 0, White text colour, Black background colour and unfilled background.* Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <53,4E,00,0A,00,0F,47,F1,20,00> Draws an "123456" number, with top left corner at X:10(dec),Y:15(dec). With Font4 settings. 

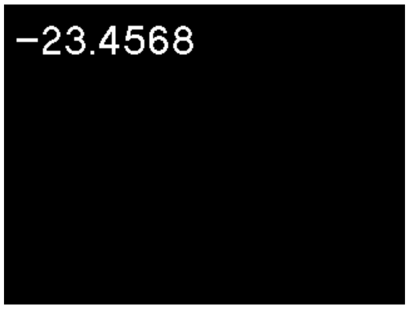
Example 2:
<53,4E,00,0A,00,0F,41,49,3F,7D> Draws an "12.578" number, with top left corner at X:10(dec),Y:15(dec). With Font4 settings.


12.578

Example 3:
<53,4E,00,0A,00,0F,4E,01,83,9D> Draws an "543221568" number, with top left corner at X:10(dec),Y:15(dec). With Font4 settings.


5.43222e+08

Example 4:
<53,4E,00,0A,00,0F,C1,BB,A7,87> Draws an "-23.4568" number, with top left corner at X:10(dec),Y:15(dec). With Font4 settings.


-23.4568

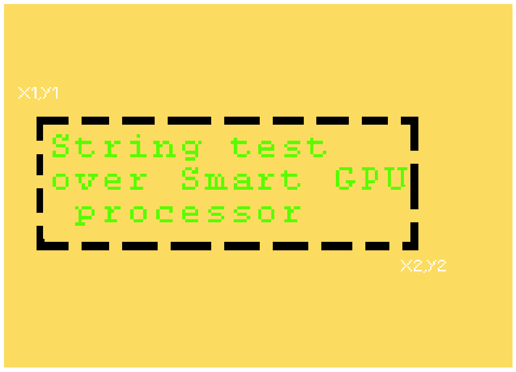
All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

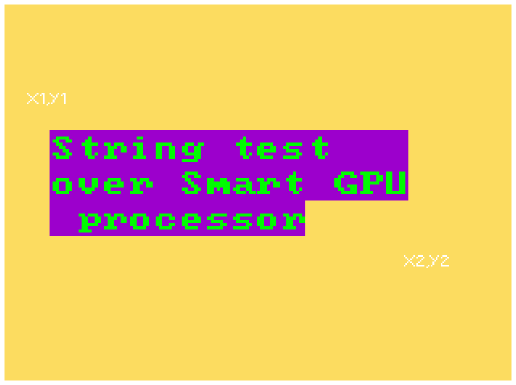
2.4.3 Display String – 53hex - S ascii

Commands (host)	10 bytes + string + 1byte(NULL)
	1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x53 (hex), S (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11 up to N.- string text (hex) ascii equivalent. N+1.- 0x00 (hex) NULL ascii.
Responses (device)	SPC(2 bytes) + 1 byte ACK
	1.- SPC high byte 2.- SPC low byte 3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **SPC stands for Successfully Printed Chars.*
Description	This command displays a string on the screen on an imaginary text box, with the given points: X1(16bit), Y1(16bit) as top left corner, and X2(16bit), Y2(16bit) as bottom right corner. The text or string is automatically adjusted to fit in this defined text box, if the text is longer to fit in the text box, it will stop discarding all the rest of the String. The command always return the SPC(successfully printed bytes), this way user can determine how many of the chars fitted and were printed on the received text box. The maximum size of string to receive in one same call is 5000 bytes + NULL character. The host must always send the NULL(0x00 hex) character after the string to indicate end of string. *Text size, text colour, text background colour, and filled of unfilled background must be set with the **Strings Config Command**, Default are Size 0, White text colour, Black background colour and unfilled background.* Colour format is the same RGB565.

| **Example (sent commands)** | *Example 1:*
<53,53,00,14,00,28,00,87,00,64,text...,00> Draws a string/text with top left corner at X1:20(dec), Y1:40(dec), and bottom right corner X2:135(dec), Y2:100(dec). Font Size was changed to Size 3.

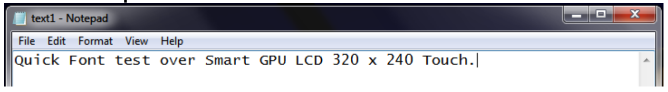
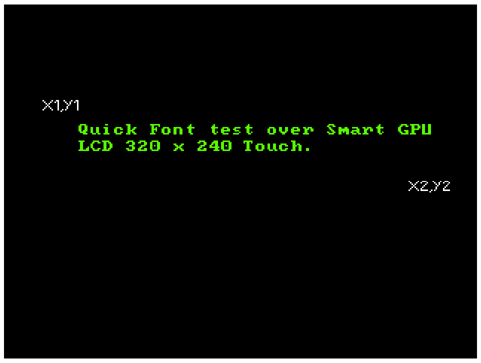
text= String test over Smart GPU processor

Note: the black dotted box is just to exemplify the imaginary defined text box, it is not actually drawn.

Example 2:
<53,53,00,14,00,28,00,87,00,64,text...,00> Draws a string/text with top left corner at X1:20(dec), Y1:40(dec), and bottom right corner X2:135(dec), Y2:100(dec). Background colour was changed to MAGENTA, Filled Background colour is Set and Font Size was changed to Size 3.

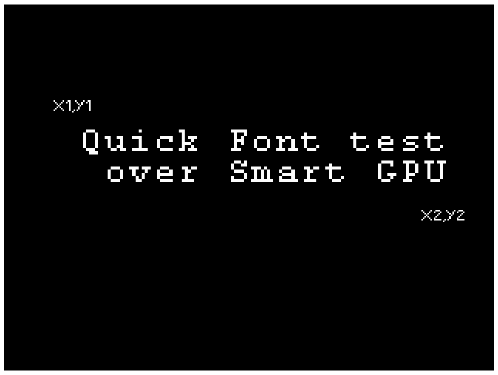
text= String over Smart GPU processor

All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation. |

2.4.4 Display String SD – 46hex - F ascii

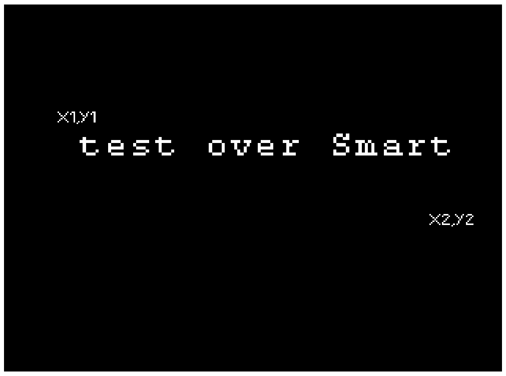
Commands (host)	14 bytes + text file name + 1byte(NULL)
	1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x46 (hex), F (ascii). –from SD .txt File 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Start Character (position) high byte. 12.- Start Character (position) low byte. 13.- Characters to Read (from position) high byte. 14.- Characters to Read (from position) low byte. 11 up to N (file name). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	SPC(2 bytes) + 1 byte ACK
	1.- SPC high byte 2.- SPC low byte 3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **SPC stands for Successfully Printed Chars.*
Description	This command calls a text file stored on the micro SD card and displays the contained text on an imaginary text box created with the given points: X1(16bit), Y1(16bit) as top left corner and X2(16bit), Y2(16bit) as bottom right corner. The text or string is automatically adjusted to fit in this defined text box, if the text is longer to fit in the text box, it will stop discarding all the rest of the String. The command always return the SPC(successfully printed bytes), this way user can determine how many of the chars fitted and were printed on the received text box. The **Start Character** parameter means, the number of bytes that will be jumped from the start of the file, including spaces. If this parameter is zero (0x0000)hex, the file will be read and displayed from the beginning. The **Characters to Read** parameter means: the number of bytes that will be read up from the **Start character** position. When this parameter is zero (0x0000)hex, mini SmartGPU will try to read all

	the file contents up from the given **Start Character** position and display the text while it fits on the given text box. The maximum number of characters to read from a file in one call is 5000 bytes (characters). **The file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the file name, in order to indicate to mini SMART GPU the end of this file name, **the name to receive must not include the .txt extension, just bare file name.** *Text size, text colour, text background colour, and filled of unfilled background must be set with the* ***Strings Config Command****, Default are Size 0, White text colour, Black background colour and unfilled background.* Colour format is the same RGB565.
Example (sent commands)	**The next examples consider a .txt file previously stored on the micro SD memory card, named "text1.txt".** Here's a picture of the contents of the .txt file:  Quick Font test over Smart GPU LCD 320 x 240 Touch. *Example 1:* <53,46,00,14,00,32,00,8C,00,A0,00,00,00,00,74,65,78,74,31,00> Opens the "text1.txt" file of the SDcard with top left corner at X1:20(dec), Y1:50(dec) and bottom right corner at X2:140(dec), Y2:160(dec), Start Character 0x0000, Characters to Read 0x0000(means all contents). Text colour was changed to GREEN.  Quick Font test over Smart GPU LCD 320 x 240 Touch.

Example 2:
<53,46,00,14,00,32,00,8C,00,A0,00,00,00,1E,74,65,78,74,31,00>
Opens the same "text1.txt" file of the SDcard with top left corner at X1:20(dec), Y1:50(dec) and bottom right corner at X2:140(dec), Y2:160(dec), Start Character 0x0000, Characters to Read 30(dec)0x001E(hex). Text colour was changed to WHITE and Font Size was changed to Size 3.



Example 3:
<53,46,00,14,00,32,00,8C,00,A0,00,0B,00,0E,74,65,78,74,31,00>
Opens the same "text1.txt" file of the SDcard with top left corner at X1:20(dec), Y1:50(dec) and bottom right corner at X2:140(dec), Y2:160(dec), Start Character 11(dec) 0x000B(hex), Characters to Read 14(dec) 0x000E(hex). Text colour was changed to WHITE and Font Size was changed to Size 3.



Just Remember: the "Start Character" is the pointer start position of the .txt file, the "Characters to Read" is the number of positions that will be read from the Start Character pointer position of the .txt file.

All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

2.4.5 Strings Configuration - 43hex - C ascii

***This command is divided in 4 sub-commands, each one is explained next:**

Commands (host)	X bytes
	Set Text Colour 1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x43 (hex), C (ascii). – Config. 3.- 0x54 (hex), T (ascii). 4.- High byte text colour. 5.- Low byte text colour. **Set Text Background Colour** 1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x43 (hex), C (ascii). – Config. 3.- 0x42 (hex), B (ascii). 4.- High byte background colour. 5.- Low byte background colour. **Set Text Font Size** 1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x43 (hex), C (ascii). - Config 3.- 0x53 (hex), S (ascii). 4.- 0xXX – Text Size: 0x00(hex) to 0x0C(hex). **Set Text Background FILLED/UNFILLED** 1.- 0x53 (hex), S (ascii). *String Command. 2.- 0x43 (hex), C (ascii). – Config. 3.- 0x46 (hex), F (ascii). 4.- 0x00 (hex) – Unfilled or 0x01 (hex) – Filled.
Responses (device)	1 byte – all commands.
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Up above are described the needed commands/parameters to set each one of the strings configurations, those are: Text Colour, Background Colour, Text Size and Filled/Unfilled Background. Once one of those commands is sent, next calls to **Put Letter, Display String** and **Display String SD** will be displayed at the new set parameters by the user with this commands.

	Defaults at reset/power on are: -WHITE Text Colour. -BLACK Background Colour. -FONT0 Text Size. -UNFILLED Background fill/unfill. Colour format is the same RGB565.
Example (sent commands)	*Example 1:* <53,43,54,FF,E0> Set Text colour to YELLOW (0xFFE0). *Example 2:* <53,43,42,06,E0> Set Text Background colour to GREEN(0x06E0). *Example 3:* <53,43,53,02> Set Text Size to FONT2. *Example 4:* <53,43,46,01> Set Text Background fill to FILLED. All data is in hex.

2.5 Image Commands

The SMART GPU 2, unlike other graphic development tools in the market, is the only embedded graphic processor capable of managing files directly in FAT/FAT12/FAT16 or FAT32 file systems without any special program/interface or micro SD rare formats. It is fully compatible with any PC.

Smart GPU 2 can manage images, so any file with **.bmp** or **.jpg** extension will be easily opened and displayed.

A maximum of 32GBs micro SD memory card is supported, allowing storing thousands of full screen images.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'I'-49hex, as they are Image commands, followed by the next parameters/bytes.**

• Draw Image/Icon	– **49hex 'I'**
• Image BMP SD	– **42hex 'B'**
• Image JPG SD	– **4Ahex 'J'**
• Memory Read	– **4Dhex 'M'**
• Screenshot BMP	– **53hex 'S'**

A complete tutorial on how to load images to the SD card is explained on the "SmartGPU2LCD320x240Datasheet.pdf" file, at the SD card file management section.

*The colour parameter needed on **Draw Image/Icon** command consist of 16bits (2 bytes) RGB565:*

R4R3R2R1R0G5G4G3 G2G1G0B4B3B2B1B0

That is:

5bits for red, 6 bits for green, 5bits for blue.

High byte colour: R4R3R2R1R0G5G4G3

Low byte colour: G2G1G0B4B3B2B1B0

The SMART GPU 2 contains a 230,400bytes (1,843,200bits) internal memory, that stores the current display image on the screen, the command **Memory Read**, allows the host to be able to read this internal memory in order to know the value of the pixels or what is being displayed in some area or the full screen at that particular moment.

The colour convention for this memory read command is RGB888, instead of the RGB565, this is in order to avoid quality and colour depth loss, and also the images that are read from the SD card are in RGB888 convention.

*The colour parameter needed on **Memory Read** command consist of 24bits (3 bytes) RGB888:*

R7R6R5R4R3R2R1R0 G7G6G5G4G3G2G1G0 B7B6B5B4B3B2B1B0

That is:

8bits for red, 8 bits for green, 8bits for blue.

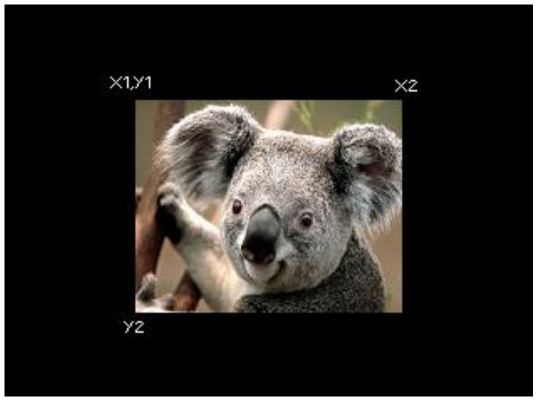
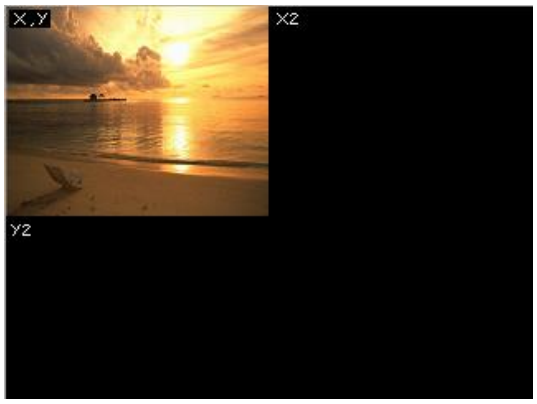
High byte colour: R7R6R5R4R3R2R1R0

Medium byte colour: G7G6G5G4G3G2G1G0

Low byte colour: B7B6B5B4B3B2B1B0

2.5.1 Draw Image/Icon – 49hex - I ascii

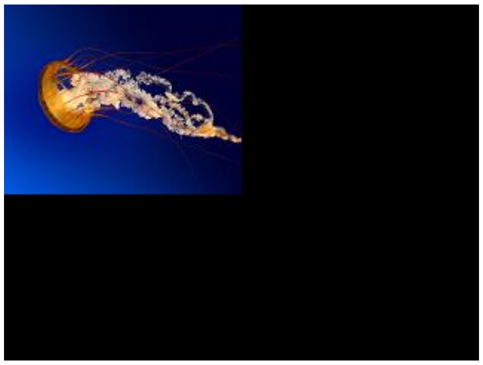
Commands (host)	10 bytes + pixel number x 2
	1.- 0x49 (hex), I (ascii). *Image Command. 2.- 0x49 (hex), I (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. Now send pixel by pixel: 11.- High byte colour pixel 1. 12.- Low byte colour pixel 1. 13.- High byte colour pixel 2. 14.- Low byte colour pixel 2. 15.- High byte colour pixel 3. 16.- Low byte colour pixel 3. …. Last- High byte colour pixel n((X2-X1+1)*(Y2-Y1+1)). Last- Low byte colour pixel n((X2-X1+1)*(Y2-Y1+1)). *Remember: each pixel is composed by two bytes following the same RGB565 convention.*
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK. *Any of those two commands will not be sent until the host finish to send all the width(X2-X1+1)*height(Y2-Y1+1) pixels of the image or icon.*
Description	This command draws an icon/ image on the screen starting at the given points(top left corner): X1(16bit), Y1(16bit) and ending on the (bottom right corner)X2(16bit), Y2(16bit). Colour format for each pixel is the same RGB565.

Example (sent commands)	*Example 1:* <49,49,00,1E,00,28,00,BD,00,9F, Pix1H,Pix1L,Pix2H,Pix2L,…,PixNH,PixNL> Draws an image of 160x120 pixels with top left corner: X1:30(dec),Y1:40(dec) and bottom right corner: X2:189(dec), Y2:159(dec).  *Example 2:* <49,49,00,00,00,00,00,9F,00,77, Pix1H,Pix1L,Pix2H,Pix2L,…,PixNH,PixNL> Draws an image of 160x120 pixels with top left corner: X1:0(dec),Y1:0(dec) and bottom right corner: X2:159(dec), Y2:119(dec).  All data is in hex. **Note: See that X2 is the sum of X1+(width in pixels - 1), the -1 in the pixels is because pixel number 0 also count. Also Y2 is the sum of Y2+(height in pixels – 1) because the same reason as X2.** **User must be careful not to exceed the Maximum X1, Y1, x2, y2 acceptable parameters(depending orientation) of the screen in order to avoid unwanted distorted images.**

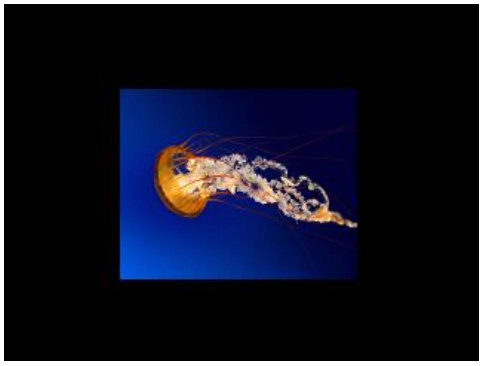
2.5.2 Image BMP SD – 42hex - B ascii

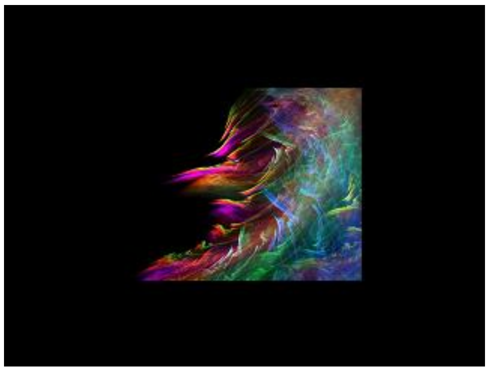
Commands (host)	6 bytes + image name + 1byte(NULL)
	1.- 0x49 (hex), I (ascii). *Image Command. 2.- 0x42 (hex), B (ascii). 3.- X coord high byte (left corner). 4.- X coord low byte. 5.- Y coord high byte (top corner). 6.- Y coord low byte. 7 up to N (file name). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command calls an image stored on the micro SD card and displays it with the given point: X(16bit), Y(16bit) as top left corner. If the image is 320x240 pixels this point must be 0,0 if not, the image won't fit on the screen and command will fail(depends orientation). **Any size of image could be called, however user is responsible that the image fits on the screen with the X,Y top left corner adjustment.** **The file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the file name, in order to indicate to SMART GPU the end of this file name, **the name to receive must not include the .bmp extension.**
Example (sent commands)	*Example 1:* <49,42,00,00,00,00,4B,4F,41,4C,41,00> Opens the 320X240pixels "KOALA.bmp" file with top left corner at X:00(dec), Y:00(dec) and displays it on the screen.

Example 2:
<49,42,00,00,00,00,4A,65,6C,6C,79,38,30,00>
Opens the 160x120 pixels "Jelly80.bmp" file with top left corner at X:00(dec), Y:00(dec) and displays it on the screen.



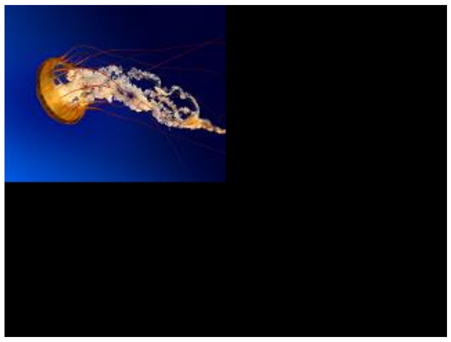
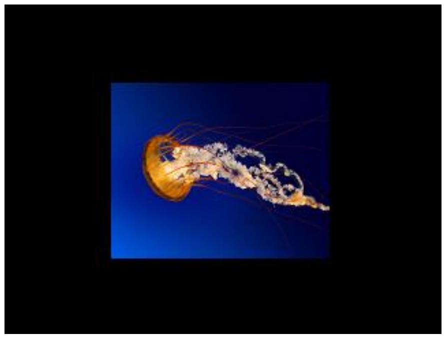
Example 3:
<49,42,00,28,00,1E,4A,65,6C,6C,79,38,30,00>
Opens the 160x120 pixels "Jelly80.bmp" file with top left corner at X:40(dec), Y:30(dec) and displays it on the screen.



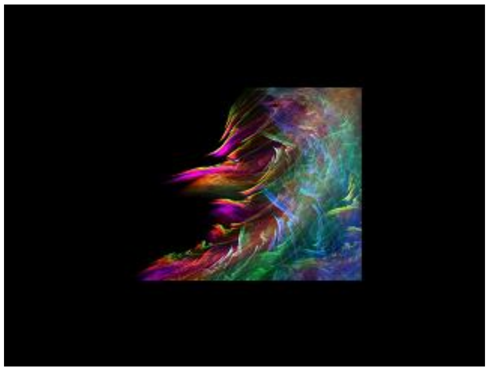
	Example 4: <49,42,00,28,00,1E,66,72,61,63,74,61,6C,00> Opens the 160x120 pixels "fractal.bmp" file with top left corner at X:40(dec), Y:30(dec) and displays it on the screen.  **For example: An Image of size 160x120, could not be called to be displayed on X>160(dec) and Y>120(dec), because it won't fit on the screen, and command will fail. As mentioned before, user is responsible of calling images with top left corners that ensures that the image will fit on the display.** All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation and image file size on pixels.

2.5.3 Image JPG SD – 4Ahex - J ascii

Commands (host)	7 bytes + image name + 1byte(NULL)
	1.- 0x49 (hex), I (ascii). *Image Command. 2.- 0x4A (hex), J (ascii). 3.- X coord high byte (left corner). 4.- X coord low byte. 5.- Y coord high byte (top corner). 6.- Y coord low byte. 7.- Scale Factor(1/1, 1/2, 1/4, 1/8). 8 up to N (file name). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command calls an image stored on the micro SD card and displays it with the given point: X(16bit), Y(16bit) as top left corner. This command is very similar to the BMP SD command, differences are that this command calls JPG images and accepts a scale factor of 1/1(0x00), 1/2(0x01), 1/4(0x02), 1/8(0x03). The scale factor characteristic is suitable to generate image thumb nails in a Graphic User Interface. **Any size of image could be called, however user is responsible that the image fits on the screen with the X,Y top left corner adjustment.** **The file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the file name, in order to indicate to SMART GPU the end of this file name, **the name to receive must not include the .jpg extension.**

Example (sent commands)	*Example 1:* <49,4A,00,00,00,00,00,4B,4F,41,4C,41,00> Opens the 320X240pixels "KOALA.bmp" file with top left corner at X:00(dec), Y:00(dec), scale factor 1/1(0x00) and displays it on the screen. *Example 2:* <49,4A,00,00,00,00,01,4A,65,6C,6C,79,38,30,00> Opens the 320x240 pixels "Jelly80.bmp" file with top left corner at X:00(dec), Y:00(dec), scale factor 1/2(0x01) and displays it on the screen.  *Example 3:* <49,4A,00,28,00,1E,01,4A,65,6C,6C,79,38,30,00> Opens the 320x240 pixels "Jelly80.bmp" file with top left corner at X:40(dec), Y:30(dec), scale factor 1/2(0x01) and displays it on the screen. 

Example 4:
<49,4A,00,28,00,1E,01,66,72,61,63,74,61,6C,00>
Opens the 320x240 pixels "fractal.bmp" file with top left corner at X:40(dec), Y:30(dec) , scale factor 1/2(0x01) and displays it on the screen.



For example: An Image of size 160x120(scale factor 1/1), could not be called to be displayed on X>160(dec) and Y>120(dec), because it won't fit on the screen, and command will fail, however if user select scale factor 1/2, image will fit. As mentioned before, user is responsible of calling images with top left corners and scale factors that ensures that the image will fit on the display.

All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation, image file size on pixels and selected scale factor.

2.5.4 Memory Read – 4Dhex - M ascii

Commands (host)	10 bytes
	1.- 0x49 (hex), I (ascii). *Image Command. 2.- 0x4D (hex), M (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte.
Responses (device)	pixel number multiplied by 3(R,G,B)
	1-N- pixels defined by the X1(16bit), Y1(16bit), and X2(16bit), Y2(16bit) window. N+1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK. *Remember: each pixel is composed by three bytes following the RGB888 convention.*
Description	This command reads the internal memory of the SMART GPU 2 that is currently displayed, defined by the given points(top left corner): X1(16bit), Y1(16bit) and (bottom right corner): X2(16bit), Y2(16bit).
Example (sent commands)	**The next examples are considered to be executed when the next full screen image is being displayed on the SMART GPU screen.**

Example 1:
<49,4D,00,00,00,00,00,4F,00,3B> Reads the contents of the SMART GPU memory from the top left corner: X1:00(dec),Y1:00(dec) and bottom right corner: X2:79(dec), Y2:59(dec), and returns the next image of 80x60 pixels, sent as RGB each pixel.

Example 2:
<49,4D,00,28,00,1E,00,77,00,59> Reads the contents of the Mini SMART GPU memory from the top left corner: X1:40(dec), Y1:30(dec) and bottom right corner: X2:109(dec), Y2:89(dec), and returns the next image of 80x60 pixels, sent as RGB each pixel.

Example 3:
<49,4D,00,50,00,00,00,77,00,3B> Reads the contents of the Mini SMART GPU memory from the top left corner: X1:80(dec), Y1:00(dec) and bottom right corner: X2:119(dec), Y2:59(dec), and returns the next image of 40x60 pixels, sent as RGB each pixel.

All data is in hex. **NOTE:** Maximum X1, Y1, X2, Y2 acceptable size values depend on display orientation.

2.5.5 Screenshot BMP – 53hex - S ascii

Commands (host)	2 bytes
	1.- 0x49 (hex), I (ascii). *Image Command. 2.- 0x53 (hex), S (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command reads the internal memory of the SMART GPU 2 that is currently displayed, creates a file named "ScreenshotXXX.bmp" on the microSD card, and appends the read memory data from screen to the file, creating a .BMP format file. In other words, this command creates a .bmp file with the current contents displayed on SmartGPU 2 screen. Once the command is received, the SmartGPU2 will attempt to create a consecutive file names: "Screenshot000.bmp", "Screenshot0001.bmp", etc. The file will be created on the current directory path, if a "ScreenshotXXX.bmp" file already exists, the SmartGPU will add 1 to the file name and create the new screenshot file. User can later call the created images with the IMAGE BMP SD command.
Example (sent commands)	*Example 1:* <49,53> Take a screenshot of SmartGPU 2 display and store as a .bmp file called "Screenshot000.bmp" All data is in hex.

2.6 Video Commands

The SMART GPU 2, unlike other graphic development tools in the market, is the only embedded graphic processor capable of managing files directly in FAT/FAT12/FAT16 or FAT32 file systems without any special program/interface or micro SD rare formats. It is fully compatible with any PC.

Smart GPU 2 can manage video; files with custom extension **.vid** will be easily opened and played.

A maximum of 32GBs micro SD memory card is supported, allowing storing several frames of full screen videos.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'V'-56hex, as they are Video commands, followed by the next parameters/bytes.**

• Allocate Video SD	– **41hex 'A'**
• Play Video SD	– **50hex 'P'**
• Set Frame Video SD	– **46hex 'F'**
• De-allocate Video SD	– **44hex 'D'**

A complete tutorial on how to load Videos to the SD card is explained on the "SmartGPU2VideoCreation.pdf" file, at the SD card file management section.

The SmartGPU 2 uses a special file format to display videos, the extension of the file must be *.vid, however this isn't a Microsoft generic standard *.vid file, the .vid video files that SmartGPU 2 can play must contain and be created as follows:

-Header of the Video File 12 bytes.
-Frames data (videoWidth x videoHeight x totalVideoFrames x 2) bytes.

The **12 bytes Header** of a .vid file that SmartGPU 2 can play is composed as next:

-Video Width in pixels	– 2 bytes
-Video Height in pixels	– 2 bytes
-Frames per Second	– 2 bytes
-Total Video Frames	– 2 bytes
-Reserved for future use	– 2 bytes
-Capital Letter 'V'	– 1 byte
-Capital Letter 'I'	– 1 byte

The **Frames data** of a .vid file that SmartGPU 2 can play is composed of VideoWidth x VideoHeight x totalVideoFrames x 2 bytes, that is:

-Each pixel of video data is composed by 2 bytes with the RGB 565 convention:

R4R3R2R1R0G5G4G3 G2G1G0B4B3B2B1B0

That is:

5bits for red, 6 bits for green, 5bits for blue.

High byte colour: R4R3R2R1R0G5G4G3

Low byte colour: G2G1G0B4B3B2B1B0

-The size of each video frame is the multiplication of VideoWidth x Video Height x 2.

As an example:

A video file of a resolution of 320 x 240 pixels, 23 frames per second and duration of 3 minutes, will be in size: 635,904,012bytes, ~635Mbytes:

-Header	= 12 bytes
-Frame Size	= 320x240x2 = 153600 bytes
-Frames per second	= 23 frames
-Video duration	= 3 minutes x 60 seconds = 180 seconds
-Video Total Frames	= 23 x 180 frames

So if we multiply (320x240x2 x 180 x 23) + 12 = 635,904,012 bytes.

Is recommended to have blocks of videos of a maximum size of 6 minutes, if more minutes are needed, user can create a video file sequence and call the videos one after each other: "video1.vid", "video2.vid", etc.

2.6.1 Allocate Video SD – 41hex - A ascii

Commands (host)	2 bytes + Video name + 1byte(NULL)
	1.- 0x56 (hex), V (ascii). *Video Command. 2.- 0x41 (hex), A (ascii). 3 up to N (file name). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	**8 bytes + 1 byte ACK/NAK**
	1.- Video Width high byte. 2.- Video Width low byte. 3.- Video Height high byte. 4.- Video Height low byte. 5.- Video Frames per Second high byte. 6.- Video Frames per Second low byte. 7.- Video Total Frames high byte. 8.- Video Total Frames low byte. 9.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command allocates a Video stored in the microSD card on the SmartGPU 2 buffer to later perform play frames and advance/set frame operations, this command can also be seen as an "open a file for reading operations", **This command must always be called prior to any "Play Video" / "Set Frame Video" command, as those commands require an already allocated video to be performed successfully.** The response of this command is very useful as once the video is successfully allocated/opened, user can know the width, height, frames per second, and total frames of the video for next video command calls. **The file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the file name, in order to indicate to SMART GPU the end of this file name, **the name to receive must not include the .vid extension.**

Example (sent commands)	*Example 1:* <56,41,77,69,6C,64,6C,69,66,65,00> Allocates the "wildlife.vid" video file in the SmartGPU 2 buffer. *Example 2:* <56,41,6E,61,74,75,72,65,00> Allocates the "nature.vid" video file in the SmartGPU 2 buffer. All data is in hex.

2.6.2 Play Video SD – 4Ahex - J ascii

Commands (host)	8 bytes
	1.- 0x56 (hex), V (ascii). *Video Command. 2.- 0x50 (hex), P (ascii). 3.- X coord high byte (left corner). 4.- X coord low byte. 5.- Y coord high byte (top corner). 6.- Y coord low byte. 7.- Frames to Play high byte. 8.- Frames to Play low byte.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command calls a video previously allocated on the SmartGPU 2 and displays it on the screen with the given point: X(16bit), Y(16bit) as top left corner, If the video is 320x240 pixels this point must be 0,0 if not, the video won't fit on the screen and command will fail(depends orientation). User can determine the maximum acceptable X and Y values with the information obtained when the video was allocated, the next conditions must always be met: *In Horizontal orientations:* *X + VideoWidth <= 320* *Y + VideoHeight <= 240* *In Vertical orientations:* *X + VideoWidth <= 240* *Y + VideoHeight <= 320* **Any size of video could be called, however user is responsible that the video fits on the screen with the X,Y top left corner adjustment.** The parameter **Frames to Play,** means the desired frames to play on the screen up from the current video position, each time this "Play Video SD" command is called, the current video frame position will advance in frames the **Frames to Play** parameter. If the **Frames to Play** parameter exceed the Total Video Frames obtained with "Allocate Video SD", the SmartGPU2 will play only the remaining frames and will return a NAK as response.

Example (sent commands)	*Example 1:* <56,50,00,00,00,00,00,64> Plays 100(dec) frames from a video allocated on the SmartGPU 2 buffer, with top left corner at X:00(dec), Y:00(dec). *Example 2:* <56,50,00,32,00,50,00,64> Plays 100(dec) frames from a video allocated on the SmartGPU 2 buffer, with top left corner at X:50(dec), Y:80(dec). **For example: An Video of size 160x120, could not be called to be displayed on X>160(dec) and Y>120(dec), because it won't fit on the screen, and command will fail. As mentioned before, user is responsible of calling Videos with top left corners that ensures that the Video will fit on the display.** All data is in hex.

2.6.3 Set Frame Video SD – 46hex - F ascii

Commands (host)	4 bytes
	1.- 0x56 (hex), V (ascii). *Video Command. 2.- 0x46 (hex), F (ascii). 3.- Frame to Set high byte. 8.- Frame to Set low byte.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command advances-rewinds a video previously allocated on the SmartGPU 2 buffer, the parameter **Frame to Set** will set the desired frame to start when a "Play Video SD" command is called. If **Frame to set** parameter is sent as 0(dec), it means the video will play from the frame 0 or beginning. The maximum acceptable value of **Frame to Set** parameter is the **total video frames - 1**. If the **Frame to Set** parameter exceeds **Total Video Frames – 1** obtained with "Allocate Video SD", the command will fail with NAK as response.
Example (sent commands)	*Example 1:* <56,46,00,64> Set the frame start position to 100(dec) for the next "play Video SD" command (advances video to frame 100). *Example 2:* <56,46,00,00> Set the frame start position to 0(dec) for the next "play Video SD" command (rewinds video to frame 0). All data is in hex.

2.6.4 De-Allocate Video SD – 44hex - D ascii

Commands (host)	2 bytes
	1.- 0x56 (hex), V (ascii). *Video Command. 2.- 0x44 (hex), D (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command de-allocates a previously allocated video on the SmartGPU 2 buffer, this command can also be seen as "close an open video file" and frees the buffer. **This command must always be called when the previously allocated video isn't needed anymore and prior to allocate another video on the buffer.**
Example (sent commands)	*Example 1:* <56,44> De-Allocates a video file in the SmartGPU 2 buffer. All data is in hex.

2.7 Audio Commands

The SMART GPU 2, unlike other graphic development tools in the market, is the only embedded graphic processor capable of managing files directly in FAT/FAT12/FAT16 or FAT32 file systems without any special program/interface or micro SD rare formats. It is fully compatible with any PC.

Smart GPU 2 can manage Audio files, so any file with **.wav** extension will be easily opened and played.

A maximum of 32GBs micro SD memory card is supported, allowing storing thousands of songs/audio files.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'A'-41hex, as they are Audio commands, followed by the next parameters/bytes.**

• Initialize/De-initialize DACs/Audio	– **49hex 'I'**
• Play WAV File	– **50hex 'P'**
• Pause WAV File	– **57hex 'W'**
• Advance WAV File	– **41hex 'A'**
• Stop WAV File	– **53hex 'S'**
• Set Volume WAV	– **56hex 'V'**
• Get Playing State	– **47hex 'G'**
• Audio Boost	– **42hex 'B'**

A complete tutorial on how to load Audio Files to the SD card is explained on the "SmartGPU2LCD320x240Datasheet.pdf" file, at the SD card file management section.

The SMART GPU 2 can play any RIFF-WAVE format sound files known as Microsoft wave file in LPCM: 8/16 bits, Mono or Stereo sound, and up to 48KHz sampling rate (CD Quality). Any other format of sound files (mp3,acc,wma,etc) must be converted to the .wav format.

2.7.1 Initialize/De-initialize DACs/Audio – 49hex - I ascii

Commands (host)	3 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x49 (hex), I (ascii). 3.- DACs state: OFF 0x00(hex) or ON 0x01(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to turn ON or turn OFF the audio DACs outputs (refer to datasheet to learn more about audio DACs). This command must always be called prior to any Audio Command, as this command initializes and turns ON the entire audio engine. If no more Audio operations are needed, user can turn OFF DACs audio outputs to save power. Default reset or power on state of the Audio DACs outputs is 0x00(hex) DACs OFF.
Example (sent commands)	*Example 1:* <41,49,01> Turn ON audio DACs outputs. *Example 2:* <41,49,00> Turn OFF audio DACs outputs. All data is in hex.

2.7.2 Play WAV File – 50hex - P ascii

Commands (host)	2 bytes + file name + 1byte(NULL)
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x50 (hex), P (ascii). 3 up to N (file name). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	2 bytes + 1 byte ACK/NAK
	1.- File duration in seconds high byte. 2.- File duration in seconds low byte. 3.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	When this command is sent and successfully executed(ACK 'O' received) a Wav file stored in the microSD card will begin playing audio on the AL(Audio Left channel) and AR(Audio Right channel), it's highly recommended to place two 100uF-400uF capacitors in each output to decouple output voltage(refer to datasheet for recommended circuitry). The SmartGPU 2 automatically will read and sample the audio file(22100Hz, 44100Hz, etc.) according to the WAV header file. If the file is MONO, both channels will have the same output, if file is STEREO, channels will have different output. The response of this command is very useful as once the audio file begins playing, user can know the total **file duration in seconds** or the time that the SmartGPU 2 will be playing the audio file. The file will continue playing until it finish, user can send any other command meanwhile the audio file is being played. If this same "Play WAV File" command is sent while the SmartGPU 2 is already playing audio, command will be just ignored(NAK) and audio will continue playing. **The file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the file name, in order to indicate to SMART GPU the end of this file name, **the name to receive must not include the .wav extension.**

Example (sent commands)	*Example 1:* <41,50,73,6F,75,6E,64,00> Play the audio file "sound.wav" stored on the microSD card. *Example 2:* <41,50,72,6F,63,6B,00> Play the audio file "rock.wav" stored on the microSD card. All data is in hex.

2.7.3 Pause WAV File – 57hex - W ascii

Commands (host)	2 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x57 (hex), W (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	When command is successfully executed (ACK), it simply toggles between **pause** and **play** audio states. *This command is only valid when an audio file is being played, otherwise will always return NAK.*
Example (sent commands)	If File is playing: *Example 1:* <41,57> Pause Audio File. *Example 2:* <41,57> Resume/Play Audio File. If File isn't playing: *Example 1:* <41,57> Command ignored - NAK. All data is in hex.

2.7.4 Advance WAV File – 41hex - A ascii

Commands (host)	4 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x41 (hex), A (ascii). 3.- File second high byte. 4.- File second low byte.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command advances/rewinds the currently playing audio file, to the desired **File Second** parameter. If **File Second** parameter is sent as 0(dec), it means the audio file will play from the second 0 or beginning. The maximum acceptable value of **File Second** parameter is the **file duration in seconds**. If the **File Second** parameter exceeds **file duration in seconds** obtained with "Play WAV File", the command will fail with NAK as response and audio file will stop playing. To avoid strange/undesired sound output while the audio file is being advanced or rewind, user can set first "Pause WAV File" command to toggle pause, then advance/rewind file, and finally set again "Pause WAV FIle" to resume file playing. *This command is only valid when an audio file is being played, otherwise will always return NAK.*
Example (sent commands)	*Example 1:* <41,41,00,64> Advance Audio File to second 100(dec). *Example 2:* <41,41,00,00> Rewind Audio File to second 00(dec). All data is in hex.

2.7.5 Stop WAV File – 53hex - S ascii

Commands (host)	2 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x53 (hex), S (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	When command is successfully executed (ACK), it simply **stops** a currently playing audio file.
Example (sent commands)	*Example 1:* <41,53> Stop Audio File. All data is in hex.

2.7.6 Set Volume WAV – 56hex - V ascii

Commands (host)	3 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x56 (hex), V (ascii). 3.- Volume to set 0-100(dec) 0x00-0x64(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to adjust the audio volume output, 0(0hex) stands for none, 100(64hex) stands for maximum audio volume. Default audio volume on reset or power on is 100(64hex).
Example (sent commands)	*Example 1:* <41,56,32> Set Audio Volume to 50(dec) Half power. *Example 2:* <41,56,00> Set Audio Volume to 00(dec) No sound. All data is in hex.

2.7.7 Get Playing State – 47hex - G ascii

Commands (host)	2 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x47 (hex), G (ascii).
Responses (device)	1 byte + 1 byte ACK/NAK
	1.- Playing State: 0x00(hex) Active or 0x01(hex) Not Active. 2.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This is one of the most simple yet effective commands on SmartGPU 2, this command replies the currently playing audio state, 0x00(hex) for not playing, and 0x01(hex) for active playing. With this command user can know if a previously called audio file with "Play WAV File" command, is still being playing or already ended. *Note that the "Pause WAV File" is the unique command that can stop audio sound and this "Get Playing State" command will still return an Active Playing State.*
Example (sent and received commands)	*Example 1:* <41,47> 00,4F. No audio file is playing. *Example 2:* <41,47> 01,4F. An audio file is being played. All data is in hex.

2.7.8 Audio Boost WAV File – 42hex - B ascii

Commands (host)	3 bytes
	1.- 0x41 (hex), A (ascii). *Audio Command. 2.- 0x42 (hex), B (ascii). 3.- Boost State: OFF 0x00(hex) or ON 0x01(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	Command needed to turn ON or turn OFF the audio output boost (refer to datasheet to learn more about this feature). The OFF boost state is recommended for headphones, and the ON boost state is recommended for external amplifiers or speakers with amplifier. Default reset or power on state of the Audio Boost is 0x00(hex) boost OFF. *This command could be called anytime, however is recommended to be called in a non-playing state to avoid clicks in the audio outputs.*
Example (sent commands)	*Example 1:* <41,42,01> Turn ON audio boost. *Example 2:* <41,42,00> Turn OFF audio boost. All data is in hex.

2.8 Touch Commands

The integrated touch controller on the SMART GPU 2 chip, it's an accurate 12 bit ADC reader, very accurate and perfectly debugged to avoid unwanted touch points.

This touch controller manages a resistive touch screen that is capable of handling finger touch and stylus pen touch, however for a precision drawing it's recommended the use of stylus.

The touch screen also contain 5 general purpose Icons drawn on the bottom corner, any touch on those Icons are processed on the touch controller and sent by the SMART GPU 2 as touch on Icon, instead of just another coordinate touch.

Never operate the touch with sharp objects or pens that are not designed for touch screens.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'T'-54hex, as they are Touch commands, followed by the next parameters/bytes.**

- Get Touchscreen – **53hex 'S'**
- Get Touch icons – **49hex 'I'**

To calibrate the touchscreen, please refer to the command:

- *Calibrate Touch* – ***54hex 'T'***

This command is under Master Commands section.

2.8.1 Get Touchscreen – 53hex - S ascii

Commands (host)	1 byte
	1.- 0x54 (hex), T (ascii). *Touch command. 2.- 0x53 (hex), S (ascii).
Responses (device)	**4 bytes + 1 byte ACK/NAK**
	1.- X coord high byte. 2.- X coord low byte. 3.- Y coord high byte. 4.- Y coord low byte. 5.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command performs a reading on the touchscreen panel and responds immediately the current reading in that specific time. When the command is called and a valid touch point on the touchscreen is read/obtained, the SmartGPU 2 will respond with the X(16bit) and Y(16bit) coordinates of the touch point followed by an ACK(means valid touchscreen point). If during the command call, no valid touch on the touchscreen panel is present, SmartGPU 2 will respond X and Y coordinates as zeros: 00,00,00,00(hex), followed by a NAK(means no valid touchscreen point and coordinates values must be discarded). Unlike other touch devices/systems that wait until a touchscreen point is obtained, Smart GPU 2 responses immediately in order to release the main processor and not freeze the system with a getchar()/reply loop, this way the application program can perform other tasks and ask periodically for a touchscreen point. If a simple wait until valid point / touch on screen is needed, user can call this command repeatedly until an ACK with valid coordinates is obtained.

Example (sent and received commands)	*Example 1:* <54,53> 01,15,00,56,4F – A valid touch point is obtained with X:277(dec), Y:86(dec) coordinates. *Example 2:* <54,53> 00,96,00,32,4F – A valid touch point is obtained with X:150(dec), Y:50(dec) coordinates. *Example 3:* <54,53> 00,00,00,00,46 – Not valid touch point is obtained(means no touch on screen), X:00(dec) and Y:00(dec) coordinates must be discarded. All data is in hex.

2.8.2 Get Touch Icons – 49hex - I ascii

Commands (host)	1 byte
	1.- 0x54 (hex), T (ascii). *Touch command. 2.- 0x49 (hex), I (ascii).
Responses (device)	1 byte + 1 byte ACK/NAK
	1.- Icon Name First Capital Letter. 2.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command performs a reading on the touchscreen panel and responds immediately the current reading in that specific time. When the command is called and a valid touch on the touchscreen Icons is read/obtained, the SmartGPU 2 will respond with the Name First Capital Letter of the Icon followed by an ACK(means valid touch on Icon). If during the command call, no valid touch on the touchscreen Icons is present, SmartGPU 2 will respond 'N'(ascii) followed by a NAK(means no valid touch on Icons). Possible responses: **If no touch:** 'N' - None – 4E, 46 **Touch on Home Icon:** 'H' - Home – 48, 4F **Touch on Message Icon:** 'M' - Message – 4D, 4F **Touch on Book Icon:** 'B' - Book – 42, 4F **Touch on Phone Icon:** 'P' – Phone – 50, 4F **Touch on Song/Note Icon:** 'S' - Song – 53, 4F If a simple wait until valid point / touch on screen is needed, user can call this command repeatedly until an ACK with a different than 'N' None Icon is obtained.

Example (sent and received commands)	*Example 1:* <54,49> 48,4F – A valid touch on "Home" Icon is obtained. *Example 2:* <54,49> 50,4F – A valid touch on "Phone" Icon is obtained. *Example 3:* <54,49> 4E,46 – Not valid touch on Icon is obtained(means no touch on screen panel), Icon letter must be discarded. All data is in hex.

2.9 FAT Data Management/Data Logger Commands

The new Smart GPU 2 includes full Data management functions, create files/dirs, open files/dirs, read files, write files, etc.

Those full Data Logger functions enable full possibilities with the Smart GPU 2, as it is now more complete than ever. User can create easy and advanced Data Logger + graphic applications.

The SmartGPU 2 also support nested folders management and up to **4 simultaneous open files for read-write operations**, those open files can be allocated to **Workspace Blocks**, the SmartGPU 2 contains workspace block 0 to workspace block 3, the command "Open File" is used to assign a file to a workspace block#, only one object/file can be allocated at the same time in the same workspace block, the command "Close File" frees the workspace block.

Note that all commands of this section always respond 2 types of ACKs, first one is FAT access execution(File ACK/NAK List) and second one is ACK 'O' or NAK 'F'.

Never remove micro SD card during "write" operations on micro SD card, data could be corrupted and damaged.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'F'-46hex, as they are FAT commands, followed by the next parameters/byes.**

• List Dirs and Files	**–4Chex 'L'**
• Get name Item#	**–47hex 'G'**
• Get Dir Path	**–48hex 'H'**
• New Dir/File	**–4Ehex 'N'**
• Open Dir	**–44hex 'D'**
• Open File	**–4Fhex 'O'**
• Read File	**–52hex 'R'**
• Write File	**–57hex 'W'**
• Set/Get Pointer	**–50hex 'P'**
• Sync File	**–53hex 'S'**
• Test Error-EOF	**–51hex 'Q'**
• Close File	**–43hex 'C'**
• Truncate File	**–56hex 'V'**
• Erase Dir/File	**–45hex 'E'**
• Dir/File Rename/move	**–4Dhex 'M'**
• Set/Get Time and Date Dir/File	**–54hex 'T'**
• Get Dir/File Info	**–49hex 'I'**
• Get Free and Total Space	**–46hex 'F'**

File ACK List:

As Mentioned before, those FAT Data Management commands always respond 2 ACKs, first ACK is for File Operation, and second ACK is for command success.

The next list of bytes, are the possible ACKs that could be obtained from any File Operation:

Byte received	Meaning	Description
0x00 (hex)	OK	Success.
0x01 (hex)	DISK ERROR	Hard error in low level disk I/O layer.
0x02 (hex)	INTERNAL ERROR	Assertion failed.
0x03 (hex)	NOT READY	Physical drive cannot work.
0x04 (hex)	NO FILE	Could not find the file.
0x05 (hex)	NO PATH	Could not find the path.
0x06 (hex)	INVALID NAME	Path name invalid format.
0x07 (hex)	DENIED	Access denied or full directory.
0x08 (hex)	EXIST	Access denied to prohibited access.
0x09 (hex)	INVALID OBJECT	File object is invalid.
0x0A (hex)	WRITE PROTECTED	Physical drive is write protected.
0x0B (hex)	INVALID DRIVE	No Drive/microSD card is inserted.
0x0C (hex)	NOT ENABLED	The volume has no work area.
0x0D (hex)	NO FILESYSTEM	There's no valid FAT volume.
0x11 (hex)	NOT ENOUGH CORE	The LFN working buffer could not be allocated.
0x12 (hex)	TOO MANY OPEN FILES	No more files can be opened.
0x13 (hex)	INVALID PARAMETER	Given parameters are invalid.

2.9.1 List Dirs and Files –4Chex – 'L' ascii

Commands (host)	2 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4C (hex), L (ascii). (List Dirs and Files)
Responses (device)	Number of Dirs(2 bytes) + Number of Files(2 bytes) + 1 byte File ACK + 1 byte Command ACK
	1.- Number of Directories (high byte). 2.- Number of Directories (low byte). 3.- Number of Files (high byte). 4.- Number of Files (low byte). 5.- 0xXX (hex) - Refer to "File ACK List". 6.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **List Dirs/Files** command read and count all directories and files under the current microSD card directory path and returns the number of found Items. Each time the User changes the directory path is recommended to call this command to know the items under the new path. This is one of the simplest yet effective commands available on the Smart GPU 2 FAT Data Management functions.
Example (sent and received commands)	*Example 1:* <46,4C> 00,0B,00,20,00,4F –List Items, we got 11(dec) Dirs, and 32(dec) Files, this means we have those 43 Items files on the current microSD path. All data is in hex.

2.9.2 Get Name of Item Number# –47hex – 'G' ascii

Commands (host)	5 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x47 (hex), G (ascii). (Get Name) 3.- Dir/File Item: 0x44 (hex), D (ascii) Dir Item or 0x46 (hex), F (ascii) File Item. 4.- Item Number (high byte). 5.- Item Number (low byte).
Responses (device)	Dir/File Name + 1 byte NULL + 1 byte File ACK + 1 byte Command ACK
	1 up to N.- Dir/File Name. N+1.- 0x00(hex), NULL(ascii). N+2.- 0xXX (hex) - Refer to "File ACK List". N+3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This command returns the file name of the received **item Number** could be a Directory Item or a File Item; the **item Number** parameter must be less than the previously found **Number of Dirs/Files** with the "List Files" Command. By using the "List Files" command combined with this "Get Name Item#" command, the user can have a detailed list of all the item and names contained under the current microSD directory path.
Example (sent commands)	*Example 1:* <46,47,46,00,01> –Get name of File item 01(dec). *Example 2:* <46,47,44,00,0A> –Get name of Dir item 10(dec). All data is in hex.

2.9.3 Get Dir Path –48hex – 'H' ascii

Commands (host)	2 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x48 (hex), H (ascii). (Get Path)
Responses (device)	Directory path name + 1 byte NULL + 1 byte File ACK + 1 byte Command ACK
	1 up to N.- Directory path name. N+1.- 0x00(hex), NULL(ascii). N+2.- 0xXX (hex) - Refer to "File ACK List". N+3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This command returns the current microSD card folder/directory path address. This command let the user know under which nested folder or the root path is performing operations. This command returns the path name in the following convention, ending with the '/' character, followed by the 0x00(hex) NULL character: **0:/directory1/directory2/** Is recommended to call this command each time that the user opens/enter a new directory.
Example (sent and received commands)	*Example 1:* <46,48> 30,3A,2F,00,00,4F –Get current directory path name: the current directory path name is root path " 0:/ ". All data is in hex.

2.9.4 New Dir/File – 4Ehex - 'N' ascii

Commands (host)	3 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4E (hex), N (ascii). (New Dir/File) 3.- Dir/File Item: 0x44 (hex), D (ascii) Dir Item or 0x46 (hex), F (ascii) File Item. 4 up to N (file name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This Command creates a new Directory or File based on the received **Dir/File Item parameter**, command fails with EXIST(0x08) if the Directory or File already exists. After Directory is created, its contents are empty, to open the new created directory, use the "Open Dir" command. After file is created, contents of the new file will be 0 bytes. **The Dir/file name must be up to 250 characters. Only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.** Always a NULL character (0x00)hex must follow the last character of the Dir/File name, in order to indicate to SMART GPU the end of this name, **In case of new File Item, the name to receive must include the .xxx desired extension.**
Example (sent commands)	*Example 1:* <46,4E,46,30,31,32,33,2E,74,78,74,00> Creates new file named "0123.txt", that doesn't exist. *Example 2:* <46,4E,44,41,42,43,00> Create a new directory named "ABC", that doesn't exist. All data is in hex.

2.9.5 Open Directory/Folder –44hex – 'D' ascii

Commands (host)	2 bytes + Dir name + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x44 (hex), D (ascii). (Open Dir) 3 up to N (file name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Opens a Directory/Folder under current directory path on microSD card. After **Open Dir** command is executed and succeeds (ACK), the current directory path name will add the new open directory name. To go inside a Directory, just call this command and give the "directory name" as **Dir name** parameter. To go outside a Directory or to parent folder(1 level), just call this command and give the ".." as **Dir name** parameter. SmartGPU 2 can also go directly inside or outside a nested Directory by giving the full path name "0:/Folder1/Folder2/directory name" as **Dir name** parameter. Once the user has changed the directory path by going inside or outside folders, is recommended to call the command "Get Dir Path" to the exactly current directory path.
Example (sent commands)	*Example 1:* **Current directory path is "0:/" root path.* <46,44,30,31,32,33,00> Open a Directory called "0123", under the current directory path. **After command succeeds Current directory path is now "0:/0123 ".* *Example 2:* **Current directory path is "0:/ABC".* <46,44,30,31,32,33,00> Open a Directory called "0123" under the current directory path. **After command succeeds Current directory path is now "0:/ABC/0123".*

	Example 3: **Current directory path is "0:/ABC".* <46,44,2E,2E,00> Go to parent Directory ".." (goes up one level). **After command succeeds Current directory path is now "0:/" root path.* *Example 4:* **Current directory path is "0:/ABC/0123".* <46,44,2E,2E,00> Go to parent Directory ".."(goes up one level). **After command succeeds Current directory path is now "0:/ABC".* *Example 5:* **Current directory path is "0:/ABC/0123".* <46,44,30,3A,2F,00> Go **directly** to root "0:/". **After command succeeds Current directory path is now "0:/".* *Example 6:* **Current directory path is any path.* <46,44,30,3A,2F,41,42,43,2F,30,31,32,33,2F,58,59,5A,00> Go **directly** to "0:/ABC/0123/XYZ" directory. **After command succeeds Current directory path is now "0:/ABC/0123/XYZ".* All data is in hex.

2.9.6 Open File –4Fhex – 'O' ascii

Commands (host)	4 bytes + file name with ".ext" + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4F (hex), O (ascii). (Open File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex). 4.- Open Mode: 0x01 (hex)- Read Only 0x02 (hex)- Write Only 0x03 (hex)- Read +Write 5 up to N (file name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Opens/Allocates a file on the Workspace Block# parameter, the file access depends on the "Open Mode" parameter: - **Read Only**: Specifies read access to the object, file data can only be read, not write. - **Write Only**: Specifies write access to the object, file data can only be write, not read. - **Read+Write**: Specifies read+write access to the object, file data can be write or read. After **Open File** command is executed and succeeds(ACK), the file object workspace block is valid. The file object workspace block is used for subsequent read/write operations to identify the file. Only **one file** can be open at the same time in the same Workspace block, always be sure close a Workspace block file object before opening a new one in it, if another file is already allocated in the Workspace block and a call to "Open File" is executed in that same Workspace block, the old file is discarded and replaced by the new one, any unsaved changes on the old file will be lost. *To save changes to file without closing it use "Sync File", to save and close an open file object use "Close File" function. If the modified/written file is not saved or closed, the file data can be collapsed.*

Example (sent commands)	*Example 1:* <46,4F,**01**,**02**,30,31,32,33,2E,74,78,74,00> Open file "0123.txt" in Workspace block 0x01, for **write only** access. *Example 2:* <46,4F,**00**,**01**,41,42,43,2E,77,78,6C,00> Open file "ABC.wxl" in Workspace block 0x00 for **read only** access. *Example 2:* <46,4F,**03**,**03**,30,31,32,33,2E,74,78,74,00> Open file "0123.txt" in Workspace block 0x03 for **read+write** access. All data is in hex.

2.9.7 Read File –52hex – 'R' ascii

Commands (host)	5 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x52 (hex), R (ascii). (Read File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex) 4.- **Bytes to Read** (High byte). 5.- **Bytes to Read** (Low byte).
Responses (device)	N Data bytes + 2 bytes (Successfully Read Bytes) + 1 File ACK + 1 Command ACK
	1 up to N.- File Data bytes.(*N= Bytes to Read*) N+1.- **Successfully Read bytes** (High byte). N+2.- **Successfully Read bytes** (Low byte). N+3.- 0xXX (hex) - Refer to "File ACK List". N+4.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The "Read File" command reads binary data from a previously allocated File("Open File") in a Workspace block, set with access for "Read Only" or "Read+Write" mode. After the command succeeded, **Successfully Read bytes** should be checked to detect the end of file. In case of **Successfully Read bytes < Bytes to Read**, it means the read/write pointer reached end of the file during read operation. This command always will try to read the **Bytes to Read** parameter, however note that even **Successfully Read bytes < Bytes to Read,** this command will always return the requested **Byte to Read** bytes, that is: if **Successfully Read bytes < Bytes to Read** then the SmartGPU 2 will complete/fill requested data bytes with 0x00(hex). **Successfully Read bytes** parameter will be the number of returned valid read bytes, the rest will be just 0x00(hex). Always after calling this command, the file pointer will increase the number of **Successfully Read bytes** from the last pointer position.

	If no File is allocated in the Workspace block# received parameter during a "Read File" command, this command will fail with INVALID OBJECT, as an attempt to read data from an empty Workspace block was done.
Example (sent commands)	*Example 1:* <46,52,**01**,00,0A> Read 10(dec) bytes from the Workspace block 0x01 file object. (file pointer will increase 10 positions after this command). *Example 2:* <46,52,**02**,13,88> Read 5000(dec) bytes from the Workspace block 0x02 file object. (file pointer will increase 5000 positions after this command). *Example 3:* <46,52,**00**,FF,FF> Read 65535(dec) bytes from the Workspace block 0x00 file object. (file pointer will increase 65535 positions after this command). All data is in hex.

2.9.8 Write File –57hex – 'W' ascii

Commands (host)	5 bytes + N Data bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x57 (hex), W (ascii). (Write File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex). 4.- **Bytes to Write** (High byte). 5.- **Bytes to Write** (Low byte). 6 up to N.- File Data bytes. *(N= Bytes to Write*). **Max Bytes to Write parameter is 512 (dec).*
Responses (device)	2 bytes (Successfully Written Bytes) + 1 File ACK + 1 Command ACK
	1.- **Successfully Written bytes** (High byte). 2.- **Successfully Written bytes** (Low byte). 3.- 0xXX (hex) - Refer to "File ACK List". 4.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Write File** command writes data to a previously allocated File("Open File") in a Workspace block, set with access "Write Only" or "Read+Write" mode. After the command succeeded, **Successfully Written bytes** should be checked to detect disk full. In case of **Successfully Written bytes < Bytes to Write**, it means the volume get full during write operation. Be sure to perform a **Sync File** or **Close File** command periodically after a write cycle to save data and avoid data corruption. Always after calling this command, the file pointer will increase the number of **Successfully Written bytes** from the last pointer position. The maximum accepted **Bytes to Write** parameter in one call is 512 bytes, if this number is exceed, the command will fail with INVALID PARAMETER. If no File is allocated in the Workspace block# received parameter during a "Write File" command, this command will fail with INVALID OBJECT, as an attempt to Write data to an empty Workspace block was done.

Example (sent commands)	*Example 1:* <46,57,**02**,00,0A,(data to write)> Write 10(dec) bytes to the Workspace block 0x02 file object. (file pointer will increase 10 positions after this command). *Example 2:* <46,57,**01**,02,00,(data to write)> Write 512(dec) (MAX in one call) bytes to the Workspace block 0x01 file object. (file pointer will 512 increase positions after this command). All data is in hex.

2.9.9 Set/Get Pointer –50hex – 'P' ascii

***This command is divided in 2 sub-commands, each one is explained next:**

Commands (host)	X bytes
	Set File Pointer position: 1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x50 (hex), P (ascii). (File Pointer) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex) 4.- 0x53 (hex), S (ascii). (Set) 5.- Position high byte. 6.- Position medium high byte. 7.- Position medium low byte. 8.- Position low byte. **Get File Pointer position:** 1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x50 (hex), P (ascii). (File Pointer) 3.- **Workspace Block** 0x00(hex) – 0x03(hex). 4.- 0x47 (hex), G (ascii). (Get) **Note that the Position parameter Set or Get is an **Unsigned Long** data type (4 bytes).*
Responses (device)	**X bytes**
	Set File Pointer position: 1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **Get File Pointer position:** 1.- Position high byte. 2.- Position medium high byte. 3.- Position medium low byte. 4.- Position low byte. 5.- 0xXX (hex) - Refer to "File ACK List". 6.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Set/Get Pointer** command moves the file read/write pointer of a Workspace block file object, or returns the current file pointer of the Workspace block file object. It can also be used to increase the file size when writing data(cluster pre-allocation). **When the pointer is Set to a given number parameter, it moves the file pointer counting from file origin to the received number, doesn't care current position.**

	When an offset above the file size is specified in write mode, the file size is increased to the offset(cluster pre-allocation); data contained in the expanded area is undefined. This is suitable to append data to a file in a quick manner, for fast write operation. **Note that Each time a "Read File" or "Write File" operation is performed; the file pointer advances the read or written bytes.** **To know if the pointer of a Workspace block file object is at the end when reading or writing, user can call the "Test EOF" command.** If no File is allocated in the Workspace block# received parameter during a "Set Pointer" command, this command will fail with INVALID OBJECT, as an attempt to Set/Get pointer from an empty Workspace block was done.
Example (sent and received commands)	*Example 1:* <46,50,**01**,53,00,00,00,0A> 00,4F - Set Workspace block 0x01 contained File pointer to 10(dec) position (origin to number). *Example 2:* <46,50,**02**,53,00,04,E6,7C> 00,4F - Set Workspace block 0x02 contained File pointer to 321148(dec) position (origin to number). *Example 3:* <46,50,**00**,47> 00,00,00,0A,00,4F - Get Workspace block 0x00 contained File pointer position, obtained pointer position is 10(dec). All data is in hex.

2.9.10 Sync File –53hex – 'S' ascii

Commands (host)	3 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x53 (hex), S (ascii). (Sync File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex)
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Sync File** command is similar to a give a click on a classic "**save changes"** button in a PC software, this command save any changes in the Workspace block file object without closing the file, it is left opened and user can continue Read/Write operations to the Workspace block file object. This command is suitable for applications that require open files for a long time in write mode, such as data logger, and this avoids data corruption. Performing **Sync File** or data **save** of periodic or immediately after file write can minimize the risk of data loss due to a sudden blackout or an unintentional disk removal. However a call to **Sync File** command immediately before **Close File** command has no advantage because **Close File** performs **Sync File** in it. In other words, the difference between those functions is that the Workspace block file object is invalidated/closed or not. If no File is allocated in the Workspace block# received parameter during a "Sync File" command, this command will fail with INVALID OBJECT, as an attempt to sync data to an empty Workspace block was done.
Example (sent commands)	*Example 1:* <46,53,**01**> Sync/Save changes to Workspace block 0x01 file object. All data is in hex.

2.9.11 Test Error-EOF File –51hex – 'Q' ascii

Commands (host)	4 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x51 (hex), Q (ascii). (Test File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex) 4.- Test Type: 0x52, R (ascii) Error test or 0x45, E (ascii) End Of File test.
Responses (device)	1 byte + 1 byte File ACK + 1 byte Command ACK
	1.- Test result 0x00 (hex) or 0x01 (hex). 2.- 0xXX (hex) - Refer to "File ACK List". 3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Test Error - EOF File** command checks for any error in a file within the given Workspace block#, in the other side, it checks for an End of File in the Workspace block# received parameter. The **Error** Test helps user to know about any write error operation. The **End of File** Test helps the user to know if the Workspace block file object pointer is at the end of the file when writing or reading from a file. If no File is allocated in the Workspace block# received parameter during a "Test Error-EOF File" command, this command will fail with INVALID OBJECT, as an attempt to test an empty Workspace block was done.
Example (sent commands)	*Example 1:* <46,51,**01**,52> Test for an Error on the Workspace block 0x01 file object. *Example 2:* <46,51,**01**,45> Test for an End of File on the Workspace block 0x01 file object. All data is in hex.

2.9.12 Close File –43hex – 'C' ascii

Commands (host)	3 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x43 (hex), C (ascii). (Close File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex)
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Close File** command closes a Workspace block# file object. If any data has been written to the file, the cached information of the file is written back to the disk. After the command succeeded, the Workspace block file object is no longer valid and it can be discarded. If the modified file is not closed, the file data can be collapsed. Be sure to always close a Workspace block file object before opening a new one in the same Workspace block to avoid losing data. If no File is allocated in the Workspace block# received parameter during a "Close File" command, this command will fail with INVALID OBJECT, as an attempt to Close an empty Workspace block was done.
Example (sent commands)	*Example 1:* <46,43,**02**> Save and Close the file contained under the Workspace block 0x02 file object. All data is in hex.

2.9.13 Truncate File –56hex – 'V' ascii

Commands (host)	3 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x56 (hex), V (ascii). (Truncate File) 3.- **Workspace Block#** 0x00(hex) – 0x03(hex)
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Truncate File** command cuts data contents from the received Workspace block# file object up from the current **pointer position**, it means that up from the current pointer position, all data in the file will be truncated and **file size will be reduced**. If no File is allocated in the Workspace block# received parameter during a "Truncate File" command, this command will fail with INVALID OBJECT, as an attempt to Truncate data from an empty Workspace block was done.
Example (sent commands)	*Example 1:* <46,56,**01**> Truncate file data contents of the file contained under the Workspace block 0x01, data up from the current file pointer position will be cut, file size will be reduced after this command. All data is in hex.

2.9.14 Erase Dir/File –45hex – 'E' ascii

Commands (host)	3 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x45 (hex), E (ascii). (Erase Dir/File) 3.- 0x4F Security un-lock byte. 3 up to N (Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Erases/Deletes an existing Directory or File from the microSD card current directory path. The Security un-lock byte, it's a simple security byte that avoids unwanted delete operations. When deleting a Directory, it must be **empty** or command will fail. When deleting a File, it must be **closed** and not allocated under any Workspace block to avoid any data corruption. If Directory or File to erase doesn't exist during an **Erase Dir/File** command, this will fail with INVALID NAME.
Example (sent commands)	*Example 1:* <46,45,4F,30,31,32,33,2E,74,78,74,00> Erase File "0123.txt". *Example 2:* <46,45,4F,41,42,43,2E,77,78,6C,00> Erase File "ABC.wxl". *Example 3:* <46,45,4F,72,6F,63,6B,00> Erase the Dir "rock". All data is in hex.

2.9.15 Dir/File Rename/Move –4Dhex – 'M' ascii

Commands (host)	2 bytes + Dir/File OLD name with ".ext" + 1byte(NULL) + Dir/File NEW name with ".ext" + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4D (hex), M (ascii). (Rename/Move Dir/File) 3 up to N (OLD Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii(end of OLD name). N+2 up to M (NEW Dir/File name including extension). M+1.- 0x00 (hex) NULL ascii(end of NEW name).
Responses (device)	1 byte File ACK + 1 byte Command ACK
	1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	This Command can rename or move a Directory or File, both operations can also be achieved in the same command call. If Directory or File to rename/move doesn't exist during a **Dir/File Rename/Move** command, this will fail with NO FILE.
Example (sent commands)	**Rename Examples:** *Example 1:* <46,4D,31,32,33,2E,74,78,74,00,34,35,36,2E,74,78,74,00> Rename a File named "123.txt" to "456.txt", under the current directory path. *Example 2:* <46,4D,30,3A,2F,31,32,33,2E,74,78,74,00,30,3A,2F,34,35,36,2E,74,78,74,00> Rename a File named "123.txt" to "456.txt" under the root "0:/" path. **Move Examples:** *Example 1:* <46,4D,30,3A,2F,46,6F,6C,64,65,72,31,2F,46,6F,6C,64,65,72,32,2F,46,6F,6C,64,65,72,33,00,30,3A,2F,46,6F,6C,64,65,72,33,00> Move a directory from "0:/Folder1/Folder2/Folder3" to "0:/Folder3" path. *Example 2:* <46,4D,30,3A,2F,31,32,33,2E,74,78,74,00,30,3A,2F,46,6F,6C,64,65,72,2F,31,32,33,2E,74,78,74,00> Move a File from "0:/123.txt" to "0:/Folder/123.txt" path.

	Rename + Move Examples: *Example 1:* <46,4D,30,3A,2F,46,6F,6C,64,65,72,31,2F,46,6F,6C,64,65,72,32,00,30,3A,2F,46,6F,6C,64,65,72,58,00> Rename and Move a directory and its contents from "0:/Folder1/Folder2" to "0:/FolderX" path. *Example 2:* <46,4D,30,3A,2F,31,32,33,2E,74,78,74,00,30,3A,2F,46,6F,6C,64,65,72,2F,34,35,36,2E,74,78,74,00> Rename and Move a File from "0:/123.txt" to "0:/Folder/456.txt" path. All data is in hex.

2.9.16 Set/Get Time and Date Dir/File –54hex – 'T' ascii

***This command is divided in 2 sub-commands, each one is explained next:**

Commands (host)	X bytes
	Set Dir/File Time and Date: 1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x54 (hex), T (ascii). (Dir/File Time & Date) 3.- 0x53 (hex), S (ascii). (Set) 4.- Hours 0-23(dec). 5.- Minutes 0-59(dec). 6.- Seconds 0-59(dec). 7.- Day 1-31(dec). 8.- Month 1-12(dec). 9.- Year upper byte 1980-2107(dec). 10.- Year lower byte 11 up to N (Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii. **Get Dir/File Time and Date:** 1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x54 (hex), T (ascii). (Dir/File Time & Date) 3.- 0x47 (hex), G (ascii). (Get) 4 up to N (Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	X bytes
	Set Dir/File Time and Date: 1.- 0xXX (hex) - Refer to "File ACK List". 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **Get Dir/File Time and Date:** 1.- Hours 0-23(dec). 2.- Minutes 0-59(dec). 3.- Seconds 0-59(dec). 4.- Day 1-31(dec). 5.- Month 1-12(dec). 6.- Year upper byte 1980-2107(dec). 7.- Year lower byte 8.- 0xXX (hex) - Refer to "File ACK List". 9.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Set/Get Time Date** command simply Sets or Gets the **last modified** Time and Date timestamp of a Directory or File. **Note that the "New Dir/File" and "Write File" commands update the timestamp of the created/modified Directory or File.**

	FAT commands time and date related functions uses the RTC Real Time Clock data to correctly set timestamps, if this is needed, the RTC must be enabled prior to execute those FAT functions.
Example (sent and received commands)	*Example 1:* <46,54,53,11,1E,14,13,03,07,DD,31,32,33,2E,74,78,74,00> 00,4F - Set Time "17:30:20" and Date "19 March 2013", to the File "123.txt". *Example 2:* <46,54,53,11,1E,14,13,03,07,DD,46,6F,6C,64,65,72,00> 00,4F - Set Time "17:30:20" and Date "19 March 2013", to the Directory "Folder". *Example 3:* <46,54,47,31,32,33,2E,74,78,74,00> 11,1E,14,13,03,07,DD,00,4F – Get Time and Date of the File "123.txt", data obtained is Time "17:30:20" and Date "19 March 2013". All data is in hex.

2.9.17 Get Dir/File Info –49hex – 'I' ascii

***This command is divided in 2 sub-commands, each one is explained next:**

Commands (host)	3 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x49 (hex), I (ascii). (Dir/File Info) 3.- 0x53 (hex), S (ascii). Size or 0x46 (hex), F (ascii). FAT Attribute 4 up to N (Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii.
Responses (device)	X bytes
	Get Dir/File Size: 1.- Size high byte. 2.- Size medium high byte. 3.- Size medium low byte. 4.- Size low byte. 5.- 0xXX (hex) - Refer to "File ACK List". 6.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **The size of a Directory is always Zero 0x00000000.* **Get Dir/File Attribute:** 1.- 0xXX(hex) - **FAT Attribute.** 2.- 0xXX (hex) - Refer to "File ACK List". 3.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **Get Dir/File Info** command simply asks for Information about the received Directory or File, this Information can be the **Size in bytes** or **FAT Attribute.** **The possible FAT Attributes can be one or a logical "OR" combination of the next bytes:** **0x01 – Read Only** **0x02 – Hidden file** **0x04 – System** **0x08 – Volume label** **0x10 – Directory** **0x20 – Archive.**
Example (sent and received commands)	*Example 1:* <46,49,53,31,32,33,2E,74,78,74,00> 00,00,15,DA,00,4F - Get Size of File "123.txt", obtained file size is 5594(dec) bytes ~ 5Kb.

	Example 2: <46,49,53,41,42,43,2E,74,78,74,00> 00,45,87,AB,00,4F - Get Size of File "ABC.txt", obtained file size is 4556715(dec) bytes ~ 4.5Mb. *Example 3:* <46,49,46,31,32,33,2E,74,78,74,00> 21,00,4F - Get FAT Attribute of File "123.txt", obtained Attribute is 0x21(hex), the logical **OR** of 0x20 Archive and 0x01 Read only, so the file is a Read-only Archive. *Example 4:* <46,49,46,31,32,33,2E,74,78,74,00> 23,00,4F - Get FAT Attribute of File "123.txt", obtained Attribute is 0x23(hex), the logical **OR** of 0x20 Archive , 0x01 Read only and 0x02 Hidden File, so the file is a Read-only hidden Archive. All data is in hex.

2.9.18 Get Free and Total Space –46hex – 'F' ascii

Commands (host)	2 bytes
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x46 (hex), F (ascii). (Free and Total Space)
Responses (device)	10 bytes
	Free Space: 1.- Size in Kb high byte. 2.- Size in Kb medium high byte. 3.- Size in Kb medium low byte. 4.- Size in Kb low byte. **Total Space:** 5.- Size in Kb high byte. 6.- Size in Kb medium high byte. 7.- Size in Kb medium low byte. 8.- Size in Kb low byte. **ACKs:** 9.- 0xXX (hex) - Refer to "File ACK List". 10.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **Note that the Size parameters are **Unsigned Long** data types (4 bytes).*
Description	This command simply asks for the available FREE Space and the TOTAL Space of the microSD card Flash Memory, the sizes are returned in Kbytes units. To get the USED Space, the following formula can be applied: **USED Space = TOTAL Space – FREE Space**
Example (sent and received commands)	*Example 1:* <46,46> 00,3A,EE,40,00,3A,F0,00,00,4F Get FREE and TOTAL space of the microSD card flash memory, obtained sizes are: FREE Space: 3862080Kb(dec) and TOTAL Space: 3862528Kb(dec). All data is in hex.

2.10 RTC Real Time Clock Commands

The new Smart GPU 2 includes an embedded RTC Real Time Clock to support full Time-Date calendar functions, also Data Logger applications that require a precise timestamp.

The real-time clock is an independent timer that counts every second and keeps information about date(day, month, year) and time(hours, minutes, seconds). The next RTC Timer Functions enable full possibilities with the Smart GPU 2, as it is now more complete than ever. User can create full real time Graphic User Interfaces and Datalogger applications with clock-calendar functions with the accuracy that only a RTC provides.

The SmartGPU 2 has an internal and ready to run RTC, however to lower production costs, the RTC Crystal hasn't been mounted on the board, a 32.768 Khz Crystal can be soldered to the **OSC32** external pads to enable full SmartGPU 2 RTC support. Also to avoid the RTC lose time and date data each time main VCC power is removed or shut down from the chip, a Standard 3V coin backup battery can be connected to the pads noted as **VBat** and **GND** pin on the board, this way the RTC will continuously running even the SmartGPU 2 chip is power off and RTC time and date data will be conserved.

Please refer to "SmartGPU2LCD320x240datasheet.pdf" file for more detailed information about RTC recommended circuit.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'R'-52hex, as they are RTC commands, followed by the next parameters/bytes.**

- RTC Setup – **53hex 'S'**
- RTC Set/Get Time and Date – **50hex 'P'**

2.10.1 RTC Setup –53hex – 'S' ascii

Commands (host)	2 bytes
	1.- 0x52 (hex), R (ascii). *RTC command. 2.- 0x53 (hex), S (ascii).
Responses (device)	2 bytes
	1.- RTC State: 0x00 Stopped or 0x01 Configured and Running. 2.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **RTC Setup** command setups, Initializes and runs the RTC clock, once the command was sent, the first response byte **RTC State** of this command will tell the new/current RTC state. Each time the SmartGPU chip comes out from a power off/shutdown state and no backup battery is placed on **VBat** pad, this command must be called in order to setup, initialize and run the RTC for the first time. Once the RTC is setup and running, consecutive calls to this command will always return 0x01(running) in the **RTC State** reply parameter, as the RTC is already setup and running. Is recommended to call this command at the beginning of the application program: If the RTC is Stopped, SmartGPU 2 chip will try to initialize and setup the RTC if a crystal is mounted on the board, once some milliseconds have passed, if the RTC wasn't successfully initialized, SmartGPU 2 will reply 0x00(stopped) as **RTC State** parameter. If the RTC is already configured and running: and this command is called, SmartGPU 2 will check RTC internal state and simply ignore the command returning 0x01(running) as **RTC State** reply parameter. **When an application needs time and date, clock and calendar functions, the RTC must be enabled, also the FAT commands time and date related functions uses the RTC data to correctly set timestamps.**

Example (sent and received commands)	*Example 1:* <52,53> 00,4F – Setup RTC, obtained data is 0x00(stopped), the RTC couldn't be initialized due to lack of 32.768Khz crystal. *Example 2:* <52,53> 01,4F – Setup RTC, obtained data is 0x01(running), the RTC is setup and running. *Example 3:* <52,53> 01,4F – Try to Setup an already initialized RTC, obtained data is 0x01(running), the RTC is already running. All data is in hex.

2.10.2 RTC Set/Get Time and Date –50hex – 'P' ascii

***This command is divided in 2 sub-commands, each one is explained next:**

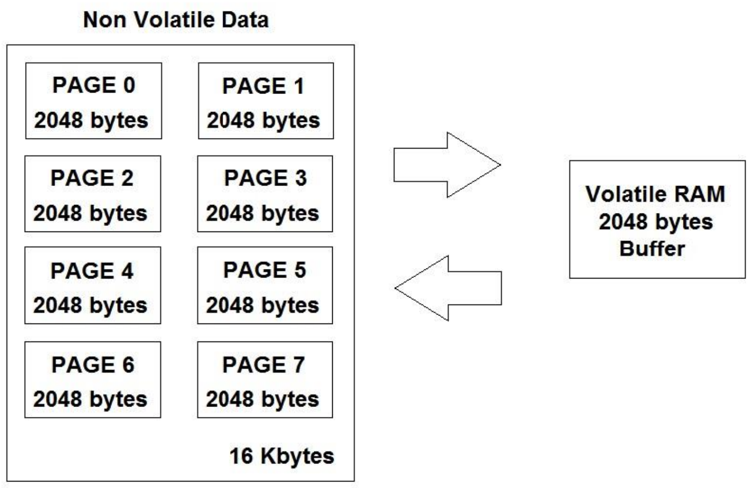
Commands (host)	X bytes
	Set RTC Time and Date: 1.- 0x52 (hex), R (ascii). *RTC command. 2.- 0x50 (hex), P (ascii). (RTC Parameters) 3.- 0x53 (hex), S (ascii). (Set) 4.- Hours 0-23(dec). 5.- Minutes 0-59(dec). 6.- Seconds 0-59(dec). 7.- Day 1-31(dec). 8.- Month 1-12(dec). 9.- Year upper byte 1980-2107(dec). 10.- Year lower byte **Get RTC Time and Date:** 1.- 0x52 (hex), R (ascii). *RTC command. 2.- 0x50 (hex), P (ascii). (RTC Parameters) 3.- 0x47 (hex), G (ascii). (Get)
Responses (device)	X bytes
	Set RTC Time and Date: 1.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **Get RTC Time and Date:** 1.- Hours 0-23(dec). 2.- Minutes 0-59(dec). 3.- Seconds 0-59(dec). 4.- Day 1-31(dec). 5.- Month 1-12(dec). 6.- Year upper byte 980-2107(dec). 7.- Year lower byte 8.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK.
Description	The **RTC Set/Get Time and Date** command simply Sets or Gets the RTC current Time and Date data. Default RTC values are: -Time "05:05:05" -Date "19 March 2013" **Note that if a "RTC Set/Get Time and Date" command is called when the RTC is stopped or hasn't been initialized, retrieved data will be invalid and must be discarded.**

Example (sent and received commands)	*Example 1:* <52,50,53,11,05,0A,13,03,07,DD> 4F - Set Time "17:05:10" and Date "19 March 2013", to the RTC data/current Time and Date. *Example 2:* <52,50,47> 11,05,0A,13,03,07,DD,4F – Get current RTC Time and Date, obtained RTC data is "17:05:10" and Date "19 March 2013". All data is in hex.

2.11 EEPROM-FLASH Commands

The Smart GPU 2 includes an embedded EEPROM like-FLASH **16Kb** storage, this is mapped as **8 Pages of 2048 bytes**, this EEPROM-FLASH is useful to store non-volatile data when an application require to maintain data even if power is removed from the system.

The next image shows the EEPROM-FLASH memory map and the Internal 2048 bytes RAM Buffer for Read-Write Operations:



The internal 2048 RAM buffer is used to perform **single byte** read-write operations, once data is ready to be saved, contents of all EEPROM RAM Buffer must be saved on the EEPROM-FLASH Page 0 – 7. A typical EEPROM-FLASH usage procedure example is described next:

**Consider that Page0 contains data that it needs to be copied to Page2:*
1.-Init/Clear all EEPROM RAM Buffer(2048b) with 0xFF.
2.-Fill all EEPROM RAM Buffer(2048b) with Contents of Page 0(2048b).
3.-Data is ready to be saved from EEPROM RAM Buffer to EEPROM Page2.
4.-Erase EEPROM Page2(2048b).
5.-Save data from EEPROM RAM Buffer(2048b) to EEPROM Page2(2048b).
6.-Compare EEPROM RAM Buffer contents(2048b) to EEPROM Page2 contents(2048b).
7.-Contents must be the same of EEPROM Page0 and EEPROM Page2.

User must take in account that all the "Fill RAM Buffer with EEPROM PageX" , "Save RAM Buffer to EEPROM PageX", "compare Buffer VS PageX", "Erase PageX" are 2048bytes operations, this means that all PageX is copied to all the Buffer, all Buffer contents are Saved to the PageX, all Buffer contents are compared VS PageX and all PageX contents are erased depending on the called command.

Briefly Summary of Commands in this section:

***All of Those next commands always begin with the byte 'E'-45hex, as they are EEPROM-FLASH commands, followed by the next parameters/bytes.**

• Init/Clear EEPROM Buffer	– **49hex 'I'**
• Read Bytes from EEPROM Buffer	– **52hex 'R'**
• Write Bytes to EEPROM Buffer	– **57hex 'W'**
• Fill Buffer with EEPROM Page#	– **46hex 'F'**
• Save Buffer to EEPROM Page#	– **53hex 'S'**
• Erase EEPROM Page#	– **45hex 'E'**
• Compare Buffer to EEPROM Page#	– **43hex 'C'**

****EEPROM-FLASH Erase-Write Endurance is 10 000 Cycles.***
****EEPROM-FLASH Minimum Data Retention is >15 years.***

2.11.1 Init/Clear EEPROM Buffer –49hex – 'I' ascii

Commands (host)	2 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x49 (hex), I (ascii).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This command Initializes to 0xFFs all the EEPROM RAM Buffer 2048bytes. As user may know, an erased EEPROM Page is filled with 0xFFs, not 0x00s, so by Initializing the EEPROM RAM Buffer to 0xFFs and then comparing this EEPROM RAM buffer with an EEPROM Page, user can determine if the Page is clean/erased or not. *During Read, Write, Erase, Compare, etc. EEPROM Functions/Commands, user may not call any other command rather than EEPROM commands, as the EEPROM RAM Buffer data is only valid during calls to EEPROM only functions, if any other function/command different than EEPROM related is called, EEPROM RAM Buffer will be discarded and data in it will be lost.*
Example (sent and received commands)	*Example 1:* <45,49> 4F – Initialize RAM Buffer, after this command succeeds, all contents of EEPROM RAM Buffer are 0xFFs (2048 bytes). All data is in hex.

2.11.2 Read Bytes From EEPROM Buffer –52hex – 'R' ascii

Commands (host)	6 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x52 (hex), R (ascii). 3.- EEPROM RAM Buffer Address High Byte. 4.- EEPROM RAM Buffer Address Low Byte. 5.- Bytes To Read High Byte. 6.- Bytes To Read Low Byte.
Responses (device)	DRB(N bytes) + SRB(2 bytes) + ACK/NAK(1 byte).
	1-N.- DRB(Data Read Bytes). N+1.-SRB(Successfully Read Bytes) High byte N+2.-SRB(Successfully Read Bytes) Low byte N+3.-0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Reads Data from the EEPROM RAM Buffer starting from the **EEPROM RAM Buffer Address**(0-2047) and reading the **Bytes To Read** parameter, this could be from 1-2048 bytes. After the command succeeded, **Successfully Read bytes** should be checked to detect the valid data bytes. In case of **Successfully Read bytes < Bytes to Read**, it means the read pointer reached end of the EEPROM RAM Buffer during read operation. This command always will try to read the **Bytes to Read** parameter, however note that even **Successfully Read bytes < Bytes to Read,** this command will always return the requested **Byte to Read** bytes, that is: if **Successfully Read bytes < Bytes to Read** then the SmartGPU 2 will complete/fill requested data bytes with 0x00(hex). **Successfully Read bytes** parameter will be the number of returned valid read bytes, the rest will be just 0x00(hex). *During Read, Write, Erase, Compare, etc. EEPROM Functions/Commands, user may not call any other command rather than EEPROM commands, as the EEPROM RAM Buffer data is only valid during calls to EEPROM only functions, if any other function/command different than EEPROM related is called, EEPROM RAM Buffer will be discarded and data in it will be lost.*

Example (sent and received commands)	*Example 1:* <45,52,00,00,00,0A> (10 Bytes Data),00,0A,4F – Read 10(dec) Bytes of EEPROM RAM Buffer Data starting from Address 0(dec). *Example 2:* <45,52,00,0A,00,32> (50 Bytes Data),00,32,4F – Read 50(dec) Bytes of EEPROM RAM Buffer Data starting from Address 10(dec). *Example 3:* <45,52,07,F8,00,0A> (10 Bytes Data),00,08,4F – Read 10(dec) Bytes of EEPROM RAM Buffer Data starting from Address 2040(dec), Successfully Read Bytes Is minor than Bytes To Read, this means that the only valid data are the first 8(dec) bytes, the other 2 bytes are just 0x00s, this is because starting address is 2040(dec) and the maximum address is 2047(dec). All data is in hex.

2.11.3 Write Bytes to EEPROM Buffer –57hex – 'W' ascii

Commands (host)	6 bytes + N bytes: Bytes To Write
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x57 (hex), W (ascii). 3.- EEPROM RAM Buffer Address High Byte. 4.- EEPROM RAM Buffer Address Low Byte. 5.- Bytes To Write High Byte. 6.- Bytes To Write Low Byte. 7-N – Data Bytes.
Responses (device)	SWB(2 bytes) + ACK/NAK(1 byte).
	1.-SWB(Successfully Written Bytes) High byte. 2.-SWB(Successfully Written Bytes) Low byte. 3.-0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Writes Data to the EEPROM RAM Buffer starting from the **EEPROM RAM Buffer Address**(0-2047) and writing the **Bytes To Write** parameter, this could be from 1-2048 bytes. After the command succeeded, **Successfully Written bytes** should be checked to detect the amount of valid written data bytes. In case of **Successfully Written bytes < Bytes to Write**, it means the EEPROM RAM Buffer address pointer overflowed position 2047. Note that **EEPROM RAM Buffer Address + Bytes to Write** parameters must always be less than or equal to **2048 bytes**, if the sum of those parameters is more than 2048 bytes, command will fail. *During Read, Write, Erase, Compare, etc. EEPROM Functions/Commands, user may not call any other command rather than EEPROM commands, as the EEPROM RAM Buffer data is only valid during calls to EEPROM only functions, if any other function/command different than EEPROM related is called, EEPROM RAM Buffer will be discarded and data in it will be lost.*
Example (sent commands)	*Example 1:* <45,57,00,0A,00,32,(data)> Write 100(dec) Bytes to the EEPROM RAM Buffer starting from Address 10(dec). All data is in hex.

2.11.4 Fill Buffer with EEPROM Page# –46hex – 'F' ascii

Commands (host)	3 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x46 (hex), F (ascii). 3.- EEPROM Page #(number).
Responses (device)	1 byte
	1.-0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Fills all the EEPROM RAM Buffer 2048 bytes with the contents of a received EEPROM Page Number(2048 bytes). This Command could be seen also as "Copy EEPROM Page to EEPROM RAM Buffer". Page parameter can be 0-7. *During Read, Write, Erase, Compare, etc. EEPROM Functions/Commands, user may not call any other command rather than EEPROM commands, as the EEPROM RAM Buffer data is only valid during calls to EEPROM only functions, if any other function/command different than EEPROM related is called, EEPROM RAM Buffer will be discarded and data in it will be lost.*
Example (sent commands)	*Example 1:* <45,56,02> Fill EEPROM RAM Buffer with contents of EEPROM Page 2. All data is in hex.

2.11.5 Save Buffer to EEPROM Page# –53hex – 'S' ascii

Commands (host)	3 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x53 (hex), S (ascii). 3.- EEPROM Page #(number).
Responses (device)	1 byte
	1.-0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Saves all the current contents of EEPROM RAM Buffer 2048 bytes to the received EEPROM Page Number(2048 bytes). This Command could be seen also as "Store in non-volatile flash memory/EEPROM Page, all the current contents of EEPROM RAM Buffer". Page parameter can be 0-7. *During Read, Write, Erase, Compare, etc. EEPROM Functions/Commands, user may not call any other command rather than EEPROM commands, as the EEPROM RAM Buffer data is only valid during calls to EEPROM only functions, if any other function/command different than EEPROM related is called, EEPROM RAM Buffer will be discarded and data in it will be lost.*
Example (sent commands)	*Example 1:* <45,53,04> Save EEPROM RAM Buffer contents to EEPROM Page 4. All data is in hex.

2.11.6 Erase EEPROM Page# –45hex – 'E' ascii

Commands (host)	3 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x45 (hex), E (ascii). 3.- EEPROM Page #(number).
Responses (device)	1 byte
	1.-0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Simply Erases all contents of the received EEPROM Page Number(2048 bytes), Note that Erased contents of an EEPROM Page are 0xFFs.
Example (sent commands)	*Example 1:* <45,45,05> Erase EEPROM Page 5 (0xFFs). All data is in hex.

2.11.7 Compare Buffer to EEPROM Page# –43hex – 'C' ascii

Commands (host)	3 bytes
	1.- 0x45 (hex), E (ascii). *EEPROM command. 2.- 0x43((hex), C (ascii). 3.- EEPROM Page #(number).
Responses (device)	1 byte Result + 1 byte ACK/NAK
	1.- 0x00-Contents Differ or 0x01-Contents are Equal. 2.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command Simply Compares all the current EEPROM RAM Buffer contents to the contents of the received EEPROM Page Number(2048 bytes). If contents differ, the SmartGPU 2 will reply 0x00, if contents are equal, it will reply 0x01.
Example (sent and received commands)	*Example 1:* <45,43,06> -01,4F- Compare EEPROM Page 6 with EEPROM RAM Buffer contents, received 0x01; contents are equal. *Example 2:* <45,43,01> -00,4F- Compare EEPROM Page 1 with EEPROM RAM Buffer contents, received 0x00; contents differ. All data is in hex.

2.12 OBJECTS Commands

The Smart GPU 2 integrates a series of hardware drawing assisted objects, those objects help user to create easy GUI controls like Scroll Bars, Progress Bars, Buttons, etc.

The next objects are currently supported:

-Checkbox.
-Button with or without text.
-Binary Switch ON/OFF.
-Progress Bar with text indicator.
-Scroll Bar with steps.
-Sliders Vertical and Horizontal.
-Windows with X(close) button and text.

Without those hardware objects, simple working panels or settings pages would be not so easy to construct, however by simple calling commands, those objects auto-adjust and center text/steps, letting the user take care in main application instead of expending hours by designing those objects.

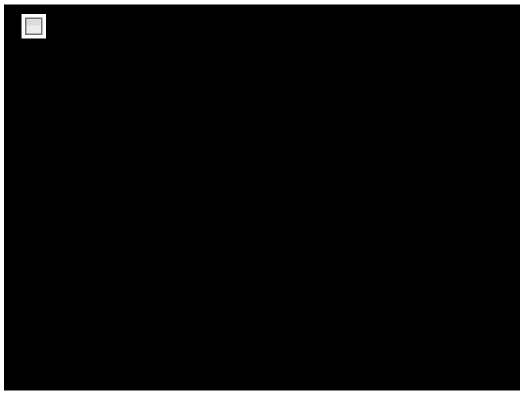
Briefly Summary of Commands in this section:

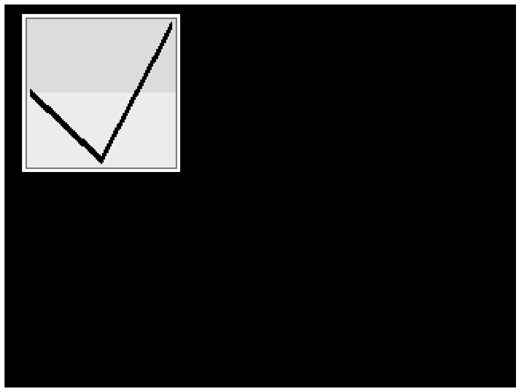
***All of Those next commands always begin with the byte 'O'-4Fhex, as they are OBJECTS commands, followed by the next parameters/bytes.**

• Object Checkbox	– **43hex 'C'**
• Object Button	– **42hex 'B'**
• Object Switch	– **54hex 'T'**
• Object Progress Bar	– **50hex 'P'**
• Object Scroll Bar	– **53hex 'S'**
• Object Slider	– **4Chex 'L'**
• Object Window	– **57hex 'W'**

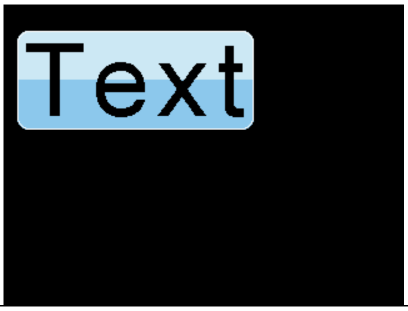
***some objects have size limitations or minimum accepted sizes, those are explained in each object command description.**

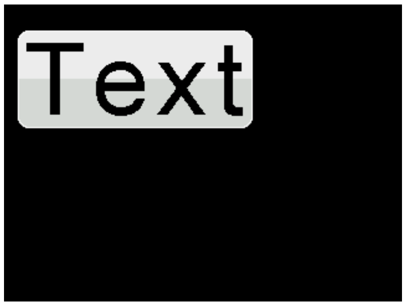
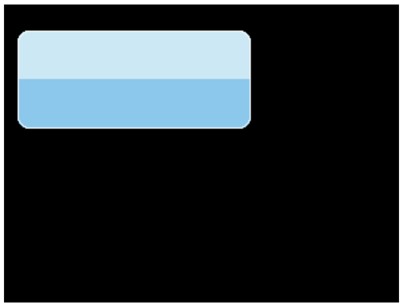
2.12.1 Object Checkbox –43hex – ‘C’ ascii

Commands (host)	9 bytes
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x43((hex), C (ascii). 3.- X coord high byte. 4.- X coord low byte. 5.- Y coord high byte. 6.- Y coord low byte. 7.- Checkbox Size high byte. 8.- Checkbox Size low byte. 9.- Active State (check/un-check).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws a Checkbox object with the received top left X(16bit) and Y(16bit) coordinates, the parameter Checkbox size(16bit), will determine size in pixels of the object in X and Y axis. This is a simple but useful object to create GUIs that need checkboxes. *Minimum “Checkbox size” accepted parameter is 15(dec), which means that the smallest Checkbox that can be created is 15x15 pixels.
Example (sent commands)	*Example 1:* <4F,43,00,0A,00,05,00,10,00> Create a checkbox (un-checked) of size 16(dec)x16(dec), starting on the top left corner X:10(dec), Y:5(dec). 

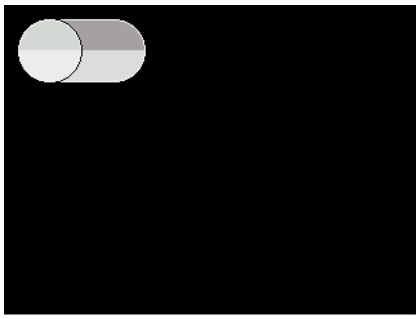
	Example 2: <4F,43,00,0A,00,05,00,64,01> Create a checkbox (checked) of size 100(dec)x100(dec), starting on the top left corner X:10(dec), Y:5(dec).  All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

2.12.2 Object Button –42hex – 'B' ascii

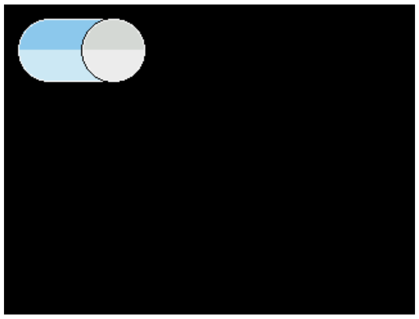
Commands (host)	11 bytes + Text on Button + 1byte(NULL)
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x42((hex), B (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Active State (selected/un-selected). 12 – N.- Button Text N+1.- 0x00(hex) NULL character.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws a Selected/Unselected button object with the received X1(16bit), Y1(16bit), X2(16bit), Y2(16bit) coordinates, the Button text will be automatically centered and font is adjusted in size to fit the Button, if the Button is too Small to hold a large text inside it, this will be discarded. Buttons without text can also be created by passing only the 0x00(hex) NULL Character as "Text on Button" parameter. *Minimum Button Size accepted parameters are 20(dec)x20(dec), that means that the smallest Button that can be created is 20x20 pixels. (X2-X1) >= 20 and (Y2-Y1) >= 20.
Example (sent commands)	*Example 1:* <4F,42,00,0A,00,14,00,C8,00,64,01,54,65,78,74,00> Draw a Selected(0x01) button with the coordinates: X1:10(dec), Y1:20(dec), X2:200(dec), Y2:100(dec), and the word "Text" inside. 

	Example 2: <4F,42,00,0A,00,14,00,C8,00,64,00,54,65,78,74,00> Draw a Unselected(0x00) button with the coordinates: X1:10(dec), Y1:20(dec), X2:200(dec), Y2:100(dec), and the word "Text" inside.  *Example 3:* <4F,42,00,0A,00,14,00,C8,00,64,01,00> Draw a Selected(0x01) button with the coordinates: X1:10(dec), Y1:20(dec), X2:200(dec), Y2:100(dec), without any word or text inside.  All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

2.12.3 Object Switch –54hex – 'T' ascii

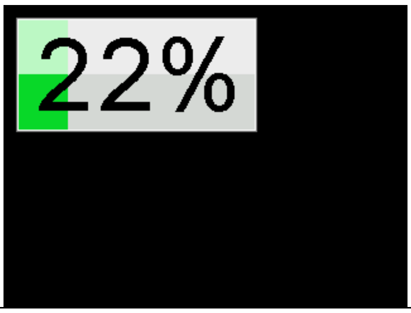
Commands (host)	9 bytes
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x54((hex), T (ascii). 3.- X coord high byte. 4.- X coord low byte. 5.- Y coord high byte. 6.- Y coord low byte. 7.- Switch Size high byte. 8.- Switch Size low byte. 9.- Active State (On/Off).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws an On/Off Switch object with the received top left X(16bit) and Y(16bit) coordinates and received Switch Size(16bit), the resulting Switch will measure "Switch Size" pixels in X axis and "Switch Size"/2 pixels in Y axis. This is a simple but useful object to create GUIs that need switches. *Minimum "Switch size" accepted parameter is 40(dec), which means that the smallest switch that can be created is 40x20 pixels.
Example (sent commands)	*Example 1:* <4F,54,00,0A,00,0A,00,64,00> Draw an Off(0x00) state Switch object starting at the top left corner: X:10(dec), Y:10(dec). 

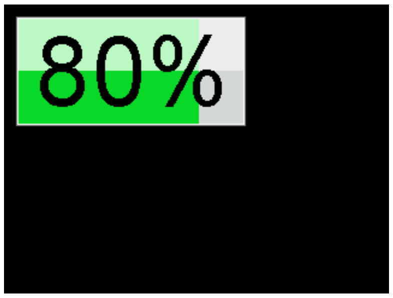
Example 2:
<4F,54,00,0A,00,0A,00,64,01> Draw an On(0x01) state Switch object starting at the top left corner: X:10(dec), Y:10(dec).



All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

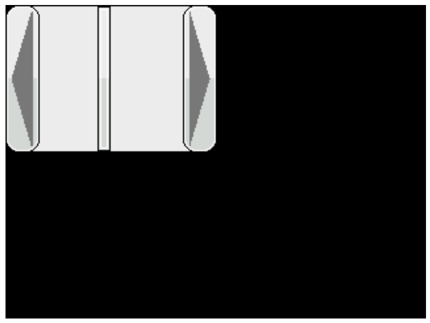
2.12.4 Object Progress Bar –50hex – 'P' ascii

Commands (host)	11 bytes
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x50((hex), P (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Progress/Percentage 0x00(hex) – 0x64(hex).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws a Progress Bar object with the received X1(16bit), Y1(16bit), X2(16bit), Y2(16bit) coordinates, the progress bar will be automatically filled with GREEN colour up to the received "Progress/Percentage" parameter, the text will be automatically centered and the font is adjusted in size to fit the Progress Bar, if the Progress Bar is too Small to hold text inside it, this will be discarded. *Minimum Progress Bar accepted parameters are 25(dec)x25(dec), that means that the smallest Progress Bar that can be created is 25x25 pixels. (X2-X1) >= 25 and (Y2-Y1) >= 25.
Example (sent commands)	*Example 1:* <4F,50,00,0A,00,0A,00,C8,00,64,16> Draw a Progress Bar object with the coordinates: X1:10(dec), Y1:10(dec), X2:200(dec), Y2:100(dec), and 22(dec) as Progress/Percentage. 

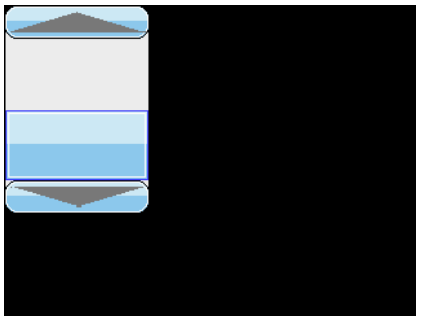
	Example 2: <4F,50,00,0A,00,0A,00,C8,00,64,50> Draw a Progress Bar object with the coordinates: X1:10(dec), Y1:10(dec), X2:200(dec), Y2:100(dec), and 80(dec) as Progress/Percentage.  All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

2.12.5 Object Scroll Bar –53hex – 'S' ascii

Commands (host)	14 bytes
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x53((hex), S (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Bar Position (must be < Divisions). 12.- Divisions. 13.- Orientation: 0x00-Horizontal or 0x01-Vertical. 14.- Active State(selected/un-selected).
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws a Scroll Bar object with the received X1(16bit), Y1(16bit), X2(16bit), Y2(16bit) coordinates, the "Divisions" parameter will determine actual Bar Size, the "Bar Position" determines where to draw the Bar, the Scroll Bar can be Vertical or Horizontal and selected or un-selected. This is a simple but useful object to create GUIs that need Scroll Bars. *Minimum Scroll Bar accepted parameters are 25(dec)x25(dec), that means that the smallest Scroll Bar that can be created is 25x25 pixels. (X2-X1) >= 25 and (Y2-Y1) >= 25, Bar Position must be always less than Divisions, Bar Position < Divisions, and it's also recommended to set small numbers to "Divisions" (<= 10).
Example (sent commands)	*Example 1:* <4F,53,00,00,00,00,00,A0,00,70,04,0A,00,00> Draw a Horizontal(0x00), Un-selected(0x00) Scroll Bar object with the coordinates: X1:0(dec), Y1:0(dec), X2:160(dec), Y2:112(dec), Bar Position:4(dec), Divisions:10(dec).



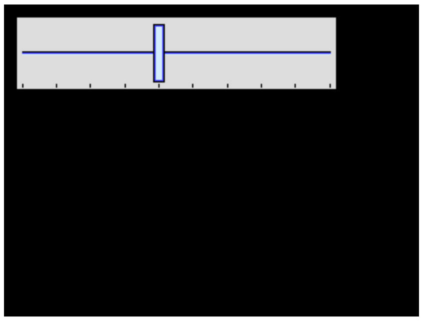
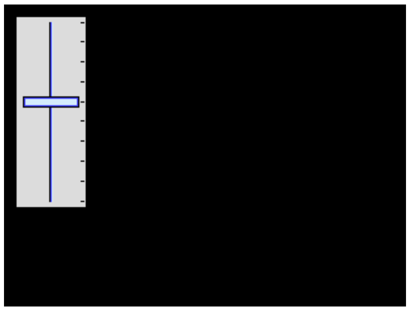
Example 2:
<4F,53,00,00,00,00,00,70,00,A0,01,02,01,01>
Draw a Vertical(0x01), Selected(0x01) Scroll Bar object with the coordinates: X1:0(dec), Y1:0(dec), X2:112(dec), Y2:160(dec), Bar Position:1(dec), Divisions:2(dec).



All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

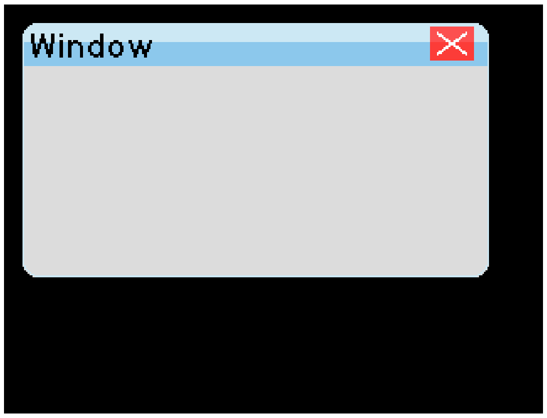
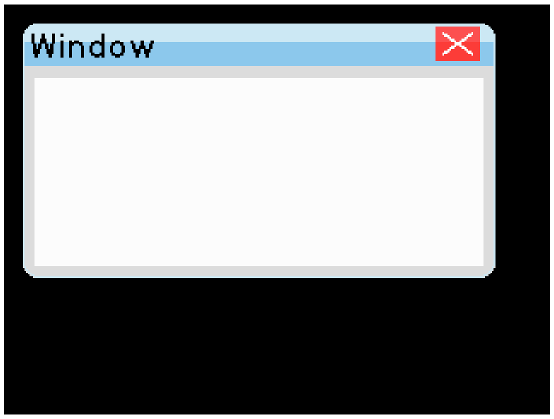
2.12.6 Object Slider –4Chex – 'L' ascii

Commands (host)	13 bytes
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x4C((hex), L (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Slider Position (must be < Divisions). 12.- Divisions. 13.- Orientation: 0x00-Horizontal or 0x01-Vertical.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws a Slider object with the received X1(16bit), Y1(16bit), X2(16bit), Y2(16bit) coordinates, the "Divisions" will be automatically draw in the Slider area, if the Slider area is too small to fit all the Divisions, those will be discarded and not drawn, the "Slider Position" determines where to draw the Slider, finally the Slider Bar can be Vertical or Horizontal. This is a simple but useful object to create GUIs that need Sliders. *Minimum Slider Size accepted parameters are 25(dec)x25(dec), that means that the smallest Slider Object that can be created is 25x25 pixels. (X2-X1) >= 25 and (Y2-Y1) >= 25, Slider Position must be always less than Divisions, Slider Position < Divisions.
Example (sent commands)	*Example 1:* <4F,4C,00,0A,00,0A,00,A0,00,40,04,0A,00> Draw a Horizontal(0x00), Slider object with the coordinates: X1:10(dec), Y1:10(dec), X2:160(dec), Y2:64(dec), Slider Position:4(dec), Divisions:10(dec).

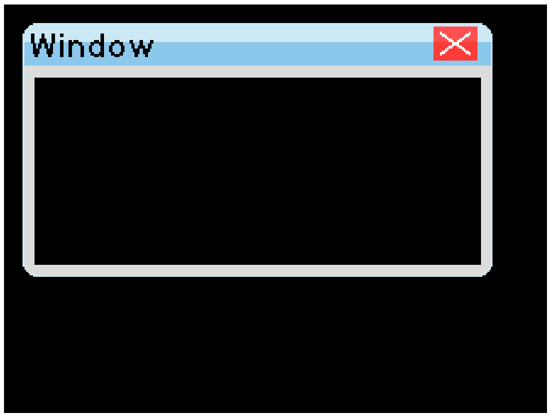
	 *Example 2:* <4F,4C,00,0A,00,0A,00,40,00,A0,04,0A,01> Draw a Vertical(0x01), Slider object with the coordinates: X1:10(dec), Y1:10(dec), X2:64(dec), Y2:160(dec), Slider Position:4(dec), Divisions:10(dec).  All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

2.12.7 Object Window –57hex – 'W' ascii

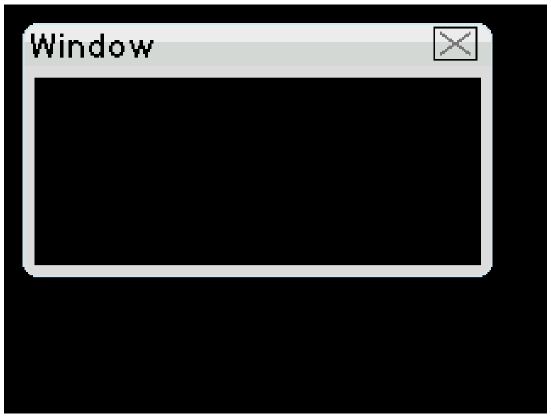
Commands (host)	12 bytes + Window Text + 1byte(NULL)
	1.- 0x4F (hex), O (ascii). *OBJECT command. 2.- 0x57((hex), W (ascii). 3.- X1 coord high byte. 4.- X1 coord low byte. 5.- Y1 coord high byte. 6.- Y1 coord low byte. 7.- X2 coord high byte. 8.- X2 coord low byte. 9.- Y2 coord high byte. 10.- Y2 coord low byte. 11.- Window Font Size 12.- Active State: 0x00(hex) Un-selected Trans, 0x01(hex) Selected Trans, 0x02(hex) Selected Gray, 0x03(hex) Selected White. 13 – N.- Window Text N+1.- 0x00(hex) NULL character.
Responses (device)	1 byte
	1.- 0x4F (hex), O (ascii) – success ACK or 0x46 (hex), F (ascii) – fail NAK.
Description	This Command generates-draws an: -Un-selected header with Transparent center -Selected header with Transparent center -Selected header with Gray center -Selected header with White center Window object with the received X1(16bit), Y1(16bit), X2(16bit), Y2(16bit) coordinates, the Window header will be automatically fitted with the received "Window Font Size". The parameters selected/un-selected and transparent/colour windows let user draw window objects that require backgroundcolours(gray/white) or when the transparent center is needed, this allows the user to switch states of the window from selected to unselected without overwriting the contents of the entire window. *Minimum Window Size accepted parameters are 50(dec)x50(dec), that means that the smallest Window object that can be created is 50x50 pixels. (X2-X1) >= 50 and (Y2-Y1) >= 50.

Example (sent commands)	*Example 1:* <4F,57,00,0A,00,0A,01,20,00,A0,02,**02**,57,69,6E,64,6F,77,00> Draw a Selected header with Gray center(0x02) Window object with the coordinates: X1:10(dec), Y1:10(dec), X2:288(dec), Y2:160(dec), and the word "Window" with font 2(dec) as header text.  *Example 2:* <4F,57,00,0A,00,0A,01,20,00,A0,02,**03**,57,69,6E,64,6F,77,00> Draw a Selected header with White center(0x03) Window object with the coordinates: X1:10(dec), Y1:10(dec), X2:288(dec), Y2:160(dec), and the word "Window" with font 2(dec) as header text. 

Example 3:
<4F,57,00,0A,00,0A,01,20,00,A0,02,**01**,57,69,6E,64,6F,77,00> Draw a Selected header with Transparent center(0x01) Window object with the coordinates: X1:10(dec), Y1:10(dec), X2:288(dec), Y2:160(dec), and the word "Window" with font 2(dec) as header text.*(this kind of transparent window is used when window center data needs to be kept and not overwritten).*



Example 4:
<4F,57,00,0A,00,0A,01,20,00,A0,02,**00**,57,69,6E,64,6F,77,00> Draw a Unselected header with Transparent center(0x00) Window object with the coordinates: X1:10(dec), Y1:10(dec), X2:288(dec), Y2:160(dec), and the word "Window" with font 2(dec) as header text.*(this kind of transparent window is used when window center data needs to be kept and not overwritten).*

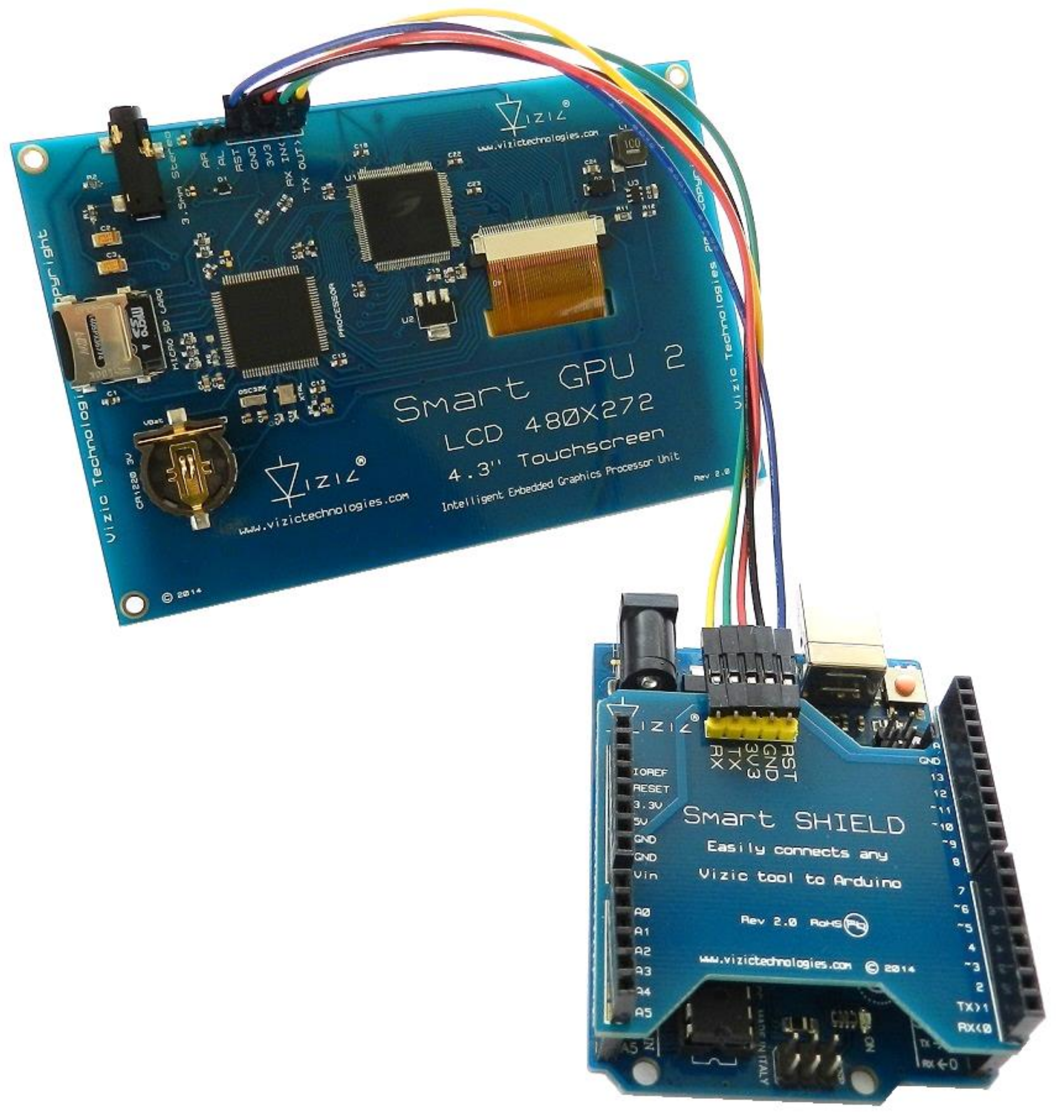


All data is in hex. **Note:** Maximum X or Y acceptable size values depend on display orientation.

3 Development Hardware tools

SmartGPU2 is compatible with any system that contains a USART/Serial port, a common device with this feature is the Arduino board, SmartGPU2 is fully compatible with all Arduino and Arduino-like boards, however as most of those boards supply only ~150mA in their 3.3V power output, then Vizic Technologies designed the SmartSHIELD, this board takes 5V of the boards and regulates the output to 3.3V via its internal circuitry sourcing up to 1000mA of current, with this any Vizic tool can be powered without current issues.

Also the smartSHIELD simplifies the connection of any Vizic tool to Arduino-like boards, this is because a single bus is required to connect all the tools instead of individual jumper wires.

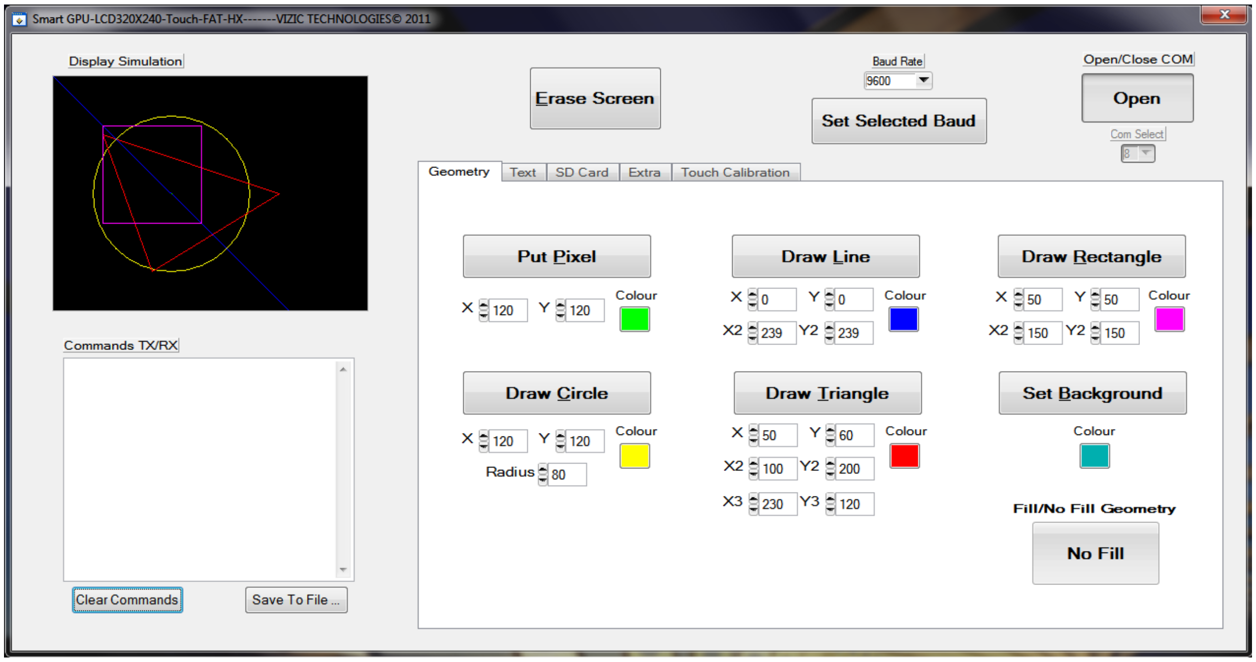


4 Development software tools

In order to make even easier the learning about how to communicate with the SmartGPU 2 processor, FREE software could be downloaded and used in any PC.

This software simulates most of the functions of the SmartGPU 2 chip by connecting it to the PC through the USB-UART SX Bridge, this connection enables real live graphics processing.

This software greatly reduces the time of learning the commands, and helps the user to understand how commands are created as it shows the sent and received commands by the PC<->SmartGPU 2 Chip.



Baud Rate: 9600
Com Select: 8

Put Pixel: X 120, Y 120, Colour 🟩
Draw Line: X 0, Y 0, X2 239, Y2 239, Colour 🟦
Draw Rectangle: X 50, Y 50, X2 150, Y2 150, Colour 🟪
Draw Circle: X 120, Y 120, Radius 80, Colour 🟨
Draw Triangle: X 50, Y 60, X2 100, Y2 200, X3 230, Y3 120, Colour 🟥
Set Background: Colour 🟦

For detailed information about this please download it from the website under the smartGPU2 LCD480x320 page.

For detailed information about the USB-UART SX Bridge, please visit our web site.

For detailed information about the SmartSHIELD, please visit our web site.

Proprietary Information:

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

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

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

Disclaimer of Warranties & Limitation of Liability:

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

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

www.VIZICTECHNOLOGIES.COM