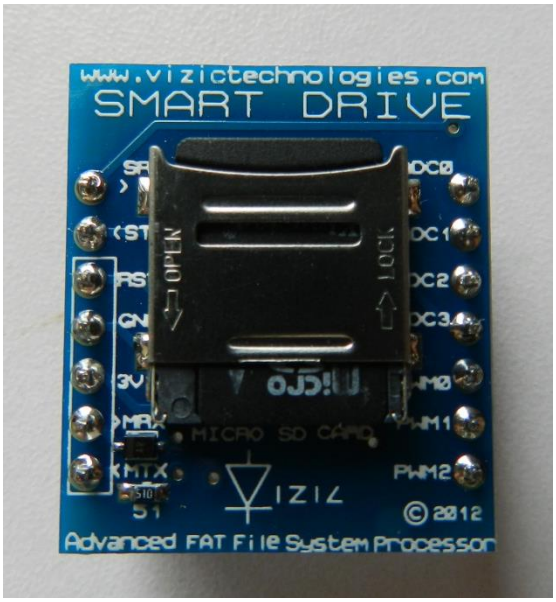
VIZIC TECHNOLOGIES

SMART DRIVE

COMMAND SET----Rev 1.0

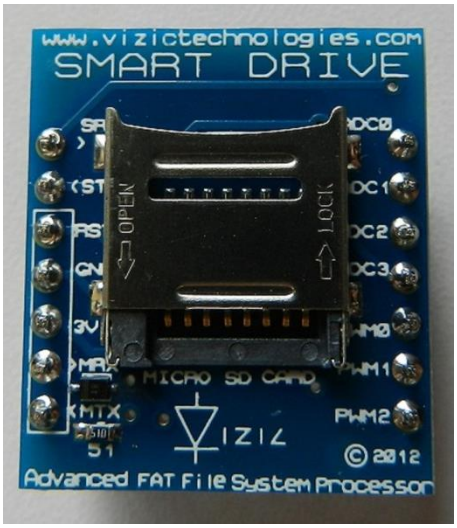
SMART DRIVE – Advanced FAT File System Processor Unit



Smart DRIVE Bottom View

Table of Contents:

Introduction:



The Smart DRIVE is a powerful, high quality, easy to use professional FAT File System management processor with an easy to use serial UART interface. It's the perfect embedded tool to any data-logger application as it's the most complete and advanced FAT processor on the market + the unique that supports LFN(Long File Names). From a microSD card, the Smart DRIVE can Read, Write, Create, Rename, Move, Copy, set Date, set Time, Modify, list files and much more from Files and Directories/Folders.

The module offers a simple, yet effective serial UART interface to any host micro-controller(8051, PIC, ATMEL, FREESCALE, STMICRO, ARM, CORTEX, ARDUINO, FPGA, MBED, or PC(USB-UART SX)). All Data Management related functions can be called using simple commands via the serial interface.

The Smart DRIVE 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 on your favorite main processor. The module has an on board microSDHC memory card socket with up to 32GB of data storage capacity with the integrated FAT/FAT12/FAT16 and FAT32 windows PC universal format.

In addition, the Smart DRIVE has an extra Serial Port(UART), 4 ADC(Analog to Digital Converters) channels to easy log analog values, 3 PWM channels to directly drive servos, control motors, fade LEDs or any other application that requires PWM.

The main goal of the Smart DRIVE it's to bring a very easy way to add high storage capabilities and universal FAT system to any application or project like "data loggers", without the user having experience in handling FAT file system management.

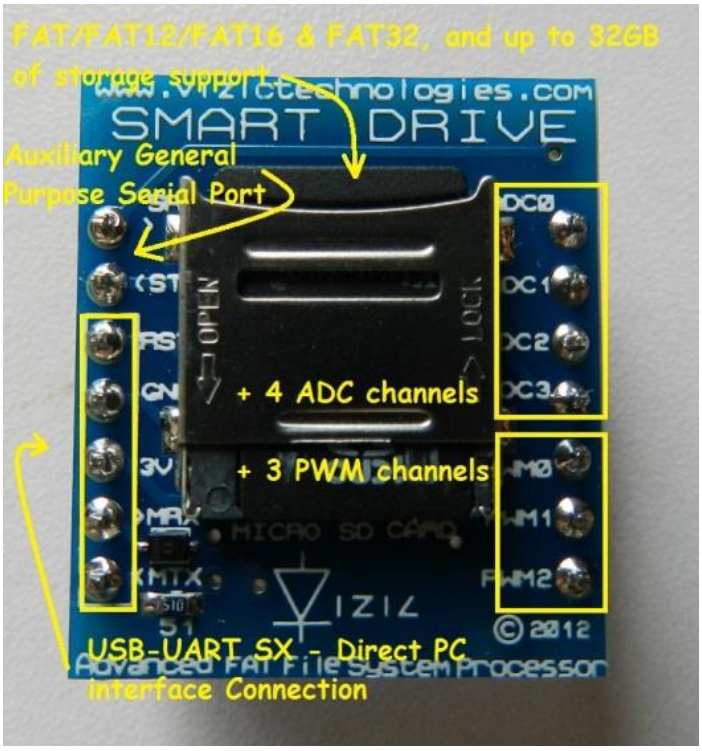
Features:

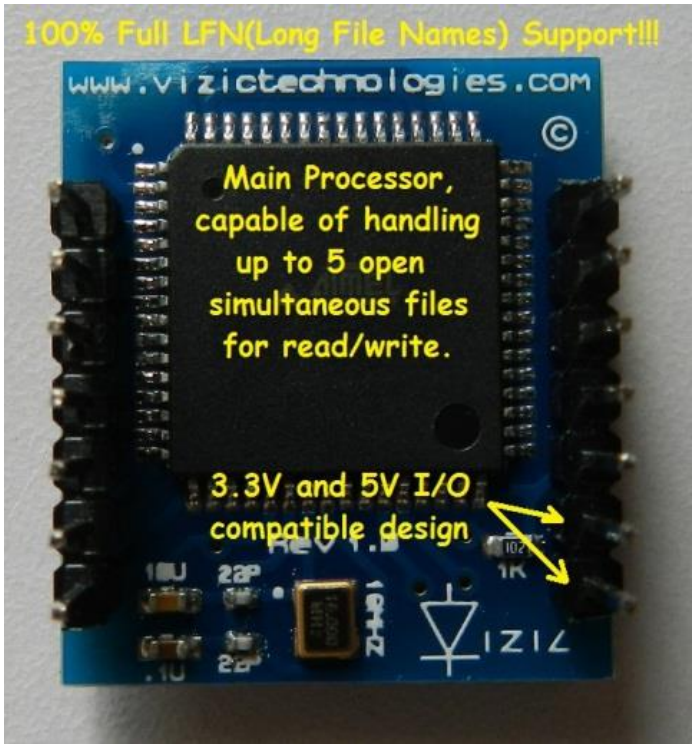
- Read, Write, Modify, Create, Erase, Copy, set Time, set Date, and many other functions on Files and Directories/Folders.
- Long File Names (LFN) up to 255 characters support.
- Up to 5 simultaneously open files for read/write.
- Easy 5 pin interface to any host device: **VCC, TX, RX, GND, RESET.**
- 9600 up to 2000000 Baud Rate speeds, 8 bits, no parity, 1 stop bit + an extra/auxiliary serial port (UART).
- On-board uSD/uSDHC memory card socket with FAT (windows PC), Support up to 32GB for storing thousands of files and directories. No need of special/rare file format.
- Full Directories and folder management support.
- 4 **ADC**(10 bit) Analog to Digital Converter channels for easy log analog data.
- 3 **PWM**(pulse width modulation) channels for direct servo, motor, LEDs or any other PWM control needs.
- Sleep mode.
- 5V and 3V3 I/O compatible, 3V3 power supply, ultra-low current consumption.

Applications:

- Embedded Data Logger systems.
- Point of sale/terminals.
- Temperature logging, green houses.
- Battery powered audio systems, with FAT management.
- Automotive, parking, GPS navigation systems.
- Robotics, industrial control.
- Traffic facilities: Toll gates, parking lots.
- Home automation and domestic appliances.
- Elevator, Security, Access-Control, Warning devices.
- Toys, learning tools, electronic books, gaming.
- GPS type data storage.

SMART DRIVE-EXPLAINED:





1.-Host Interface

The SMART DRIVE must be used as a slave peripheral device, providing a bidirectional serial interface to a host controller via its UART(Universal Asynchronous Receiver - Transmitter) by its main MTX and MRX port.

Any microcontroller or processor (AVR, PIC, BASICstamp, ARDUINO, 8051, MBED, FPGA, ARM, STmicro, etc) or PC(by serial interface RS232) as host, can communicate to the device over this serial interface.

The SMART DRIVE 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 UART.

The serial protocol is universal and very easy to implement.
Serial Data Format: 8 Bits, No Parity, 1 Stop Bit.
BaudRate: 9600 bps (can be selected, up to 2000000bps).
Serial data is true and not inverted.

1.1 Command Protocol: Flow Control

The SMART DRIVE Advanced FAT File System 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 Power-up, Reset and Serial Set-up

When the SMART DRIVE 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. Any command could be sent after this point.

The SMART DRIVE is configured to always initialize at a standard **baud rate of 9600 bps**. Any other baudrate speed can be set after this initialization.

2. SMART DRIVE Command Set - Software Interface Specification

As mentioned before the command interface between the SMART DRIVE and the host is via the serial interface UART.

A list of very easy to learn commands provide complete access to all the available functions. Commands and responses are a byte packages. All commands always return a pair or more of responses/ACKs.

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

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

Main Processor:
-main application processing.
-math processing
-I/O processing

VS

SMART DRIVE Processor:
-FAT System Management
-File/Folder sync
-Low level SD communication
-PWM output processing
-ADC processing
-FAT System Tasks
-And more...

File ACK/NAK List:

Smart Drive always respond 2 ACKs, first ACK is for File Operation, and second ACK is for command success ‘O’ or fail ‘F’.

The next list of bytes, are the possible ACKs that could be obtained from 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)	ALREADY EXIST	Access denied to prohibited access.
0x09 (hex)	INVALID OBJECT	File object is invalid.
0x0A (hex)	WRITE PROTECTED	Physical drive is write protected.
0x0C (hex)	NOT ENABLED	The volume has no work area.
0x0D (hex)	NO FILESYSTEM	There’s no valid FAT volume.
0x10 (hex)	LOCKED	There’s no access granted.
0x11 (hex)	NOT ENOUGH CORE	LFN Working Buffer couldn’t be allocated.
0x12 (hex)	TOO MANY OPEN FILES	Acceptable open files Number is exceeded.
0x13 (hex)	INVALID PARAMETER	Given parameters are invalid.

2.1 General Commands

Briefly Summary of Commands in this section:

- Initialize –Mount/Unmounts SmartDRIVE – **55hex 'U'**
- Get bytes from Secondary USART – **42hex 'B'**
- Transmit bytes to Secondary USART – **4Bhex 'K'**
- ADC or PWM action – **41hex 'A'**
- Sleep Mode – **5Ahex 'Z'**
- Set Master/Secondary USART BaudRate – **58hex 'X'**

**** Note that commands on all of this section always respond 2 ACKs or bytes, the first byte is one of the above File ACK/NAK List that informs about command execution status, the second byte correspond to 'O'-Ok or 'F'-Fail. Please refer to "File ACK/NAK List".***

2.1.1 Initialize Mount/Unmounts SmartDRIVE - 55hex - U ascii

Commands (host)	1 byte
	1.- 0x55 (hex), U (ascii).
Responses (device)	2 bytes(ACK)
	1.- ACK or NAK(Refer to File ACK/NAK List). 2.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	This command Mounts and Unmounts the microSD card and initialized the SmartDRIVE to start receiving commands, remember to wait at least 200ms after any power up or reset before sending this command. Please note that no command is available or could be called while the SmartDRIVE is unmounted, the only available/permitted command to be called while the SmartDRIVE is unmounted is “Sleep in/out Mode”.
Example (sent commands)	**If SmartDRIVE isn’t Initialized or microSD card is Unmounted:** <55> 00,4F Initializes SMART DRIVE/Mount microSD card. **If SmartDRIVE is Initialized and microSD card is Mounted:** <55> 00,4F Unmounts microSD card. All data is in hex.

2.1.2 Get Bytes from Secondary USART- 42hex - B ascii

Commands (host)	1 byte + BTG(2 bytes)
	1.- 0x42 (hex), B (ascii). 2.- High byte BTG. 3.- Low byte BTG. *BTG stands for Bytes To Get (MAX 255 per call).
Responses (device)	BTG(data) + SRB(2 bytes) + 2 bytes(ACK)
	1 up to BTG.- (Data of the USART Buffer). BTG + 1.- (SRB High Byte). BTG + 2.- (SRB Low Byte). BTG + 3.- ACK or NAK(Refer to File ACK/NAK List). BTG + 4.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii). *SRB stands for Successfully Read Bytes.
Description	This command asks for the **BTG** number of bytes and outputs the data through the main USART. The Secondary USART is Internal Interrupt based, so it can receive data in any moment(except during sleep mode), each time the **SRX** pin receive a byte, it is stored on a **255 bytes buffer** and the automatic **buffer byte pointer** advances by the number of received bytes. Each time user retrieves data or bytes, the **buffer byte pointer** is decreased by the number of **SRB**. If receive buffer overflows(**buffer byte pointer>255**), all data is discarded and **buffer byte pointer** takes value of 0(zero). The user can ask for more than the previously received/buffered bytes in a "Get Bytes from Secondary USART" command, however, if the **BTG** parameter is bigger than the actual **buffer byte pointer** number, those **buffer byte pointer** valid bytes will be sent, and the remaining bytes to complete **BTG** will be sent as 0x00. Always check the **SRB** received parameter to know how many obtained data bytes are valid, and how many received data bytes are just 0x00 filling bytes. In this case the returned ACK will be **INVALID PARAMETER** as only a % of the retrieved data will be valid and the rest(**BTG** - **buffer byte pointer**) will be only filling bytes. It's **highly** recommended that the MAIN USART and SECONDARY USART use the same baud rate speed, however this isn't absolutely necessary.

Example (sent commands)	*Example 1: Try to get 5 bytes from Secondary USART, buffer has received/stored 6 bytes(10, 20, 30, 40, 50, 60).* *Before command:* ***buffer byte pointer*** *= 6.* **Sent data:** <42,00,05> (Receive/Read 5 bytes). **Received data:** -10, 20, 30, 40, 50- (5 buffered bytes). -Zero bytes of 0x00.(Filling 0x00 bytes). -0x00,0x05- (Number of valid sent Data bytes). -0x00- (OK – ACK list Successful command). -0x4F- OK. *After command:* ***buffer byte pointer*** *= 6-5 equal 1.* *Example 2: Try to get 10 bytes from Secondary USART, but it hasn't received/buffered any byte.* *Before command:* ***buffer byte pointer*** *= 0.* **Sent data:** <42,00,0A> (Receive/Read 10 bytes). **Received data:** -Ten bytes of 0x00.(Filling 0x00 bytes). -0x00,0x00- (Number of valid sent Data bytes). -0x13- (INVALID PARAMETER – ACK list). -0x46- FAIL. *After command:* ***buffer byte pointer*** *= 0-0 equal 0.* *Example 3: Try to get 10 bytes from Secondary USART, buffer has received/stored 4 bytes(10, 20, 30, 40).* *Before command:* ***buffer byte pointer*** *= 4.* **Sent data:** <42,00,0A> (Receive/Read 10 bytes). **Received data:** -10, 20, 30, 40- (4 buffered bytes). -Six bytes of 0x00.(Filling 0x00 bytes). -0x00,0x04- (Number of valid sent Data bytes). -0x13- (INVALID PARAMETER – ACK list). -0x46-FAIL. *After command:* ***buffer byte pointer*** *= 10-4 equal 0.* All data is hex.

2.1.3 Transmit Bytes to Secondary USART – 4Bhex - K ascii

Commands (host)	1 byte + BTT(2 bytes) + BTT(data)
	1.- 0x4B (hex), K (ascii). 2.- High byte BTT. 3.- Low byte BTT. 4 up to BTT value.- Bytes to be transmitted/transfer. *BTT stands for Bytes To Transfer (MAX 255).
Responses (device)	2 bytes(ACK)
	1.- ACK or NAK(Refer to File ACK/NAK List). 2.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	This command receives and re-transmits bytes from the MAIN USART to the SECONDARY USART. A maximum of 255 can be received and transmitted in the same burst/command. Only the BTT(data) are transmitted to the SECONDARY USART, the other parameters are only to control the MAIN USART. If User tries to transmit more than 255 bytes (BTT>255), data will be discarded and no data will be transfer, receiving the ACK "INVALID PARAMETER". It's **highly** recommended that the MAIN USART and SECONDARY USART use the same baud rate speed, however this isn't absolutely necessary.
Example (sent commands)	*Example 1: Try to transmit 5 bytes from Main USART to Secondary USART (10, 20, 30, 40, 50).* **Sent data:** <4B,00,05,10,20,30,40,50>(Transmit 5 bytes). **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Secondary USART successfully transmitted 5 bytes .* *Example 2: Try to transmit 513 bytes from Main USART to Secondary USART (10, 20, 30, 40, 50……).* **Sent data:** <4B,02,01,10,20,30,40,50,……XX>(Transmit 256 bytes). **Received data:** -0x13- (INVALID PARAMETER – No data was transferred). -0x46- FAIL. All data is hex.

2.1.4 ADC or PWM Action – 41hex - A ascii

This Command is divided in two cases: case 1 ADC:

Commands (host)	3 bytes
	1.- 0x41 (hex), A (ascii). 2.- 0x41(hex) 'A' -**ADC action** 3.- Channel Number to Read (0-3).
Responses (device)	2 bytes(ADC) + 2 bytes(ACK)
	1.- ADC lecture High byte. 2.- ADC lecture Low byte. 3.- ACK or NAK(Refer to File ACK/NAK List). 4.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	This command returns the ADC lecture of the desired channel. Please note that the available ADC channels on the SmartDRIVE are only 0x00, 0x01, 0x02 and 0x03. The ADC conversion will be obtained in hex data going from GND-0x0000(hex) up to VCC-0x0400(hex) that is 10 bit (0 – 1024) possible values.
Example (sent commands)	*Example 1: Get lecture of ADC channel 0.* **Sent data:** <41,41,00> **Received data:** -0xXX- ADC conversion High byte. -0xXX- ADC conversion Low byte. -0x00- (OK - Successful command). -0x4F- OK. *Example 2: Get lecture of ADC channel 2.* **Sent data:** <41,41,02> **Received data:** -0xXX- ADC conversion High byte. -0xXX- ADC conversion Low byte. -0x00- (OK - Successful command). -0x4F- OK. All data is in hex.

This Command is divided in two cases: case 2 PWM:

Commands (host)	5-6 bytes
	1.- 0x41 (hex), A (ascii). 2.- 0x50(hex) 'P' -**PWM action** 3.- 0x46(hex) 'F' –Frequency or 0x44(hex) 'D' -Duty Cycle 4.- Freq/Duty Parameter High byte. 5.- Freq/Duty Parameter Low byte. *6.- Channel Number(0-2) – only on Duty Cycle.*
Responses (device)	2 bytes(ACK)
	1.- ACK or NAK(Refer to File ACK/NAK List). 2.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	This command can set the PWM Frequency(5 bytes command) of all Channels or the Individual PWM Duty Cycle(6 bytes command) of a Single Channel. When the 0x46(hex) 'F' byte is sent(5 bytes command), then the next 2 bytes(Freq/Duty Parameter) will set the frequency of all the PWM channels. The next simple formula can be used to set a desired value in Hz to the PWM frequency: ***FreqParameterValue = (2000000/FreqHz)*** The **FreqHz** is the desired frequency to set in Hz, and the **FreqParameterValue** is the parameter that must be given to obtain this desired frequency. *Please note that the 3 PWM channels share the same frequency, frequencies that can be obtained go from 30.5180Hz (0xFFFF) up to 2000000Hz (0x0001). Default value is 0x9C40.* When the 0x44(hex) 'D' byte is sent(6 bytes command), then the next 2 bytes(Freq/Duty Parameter) will set the Duty Cycle of Single PWM channel, the 6th byte is the Channel Number. Please note that the available PWM channels on the SmartDRIVE are only 0x00, 0x01, and 0x02. The next simple formula can be used to set a desired value in % to the PWM Duty Cycle: ***DutyParameterValue* =(FreqParameterValue*Duty)/100**

	The **FreqParameterValue** is the last/previously set frequency parameter, the **Duty** is the value from 0-100 that represents the % of the desired Duty Cycle, final the **DutyParameterValue** is the parameter that must be given to obtain this desired Duty Cycle. *Please note that the default value of Duty Cycle is 100%(full HIGH state), the value is 0x9C40.*
Example (sent commands)	*Example 1: Set PWMs Frequency at 150Hz.* *(2000000/150Hz)= 13333 - 0x3415(hex)* **Sent data:** <41,50,46,34,15> Set 150Hz frequency. **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Example 2: Set PWMs Frequency at 100Hz.* *(2000000/100Hz)= 20000 - 0x4E20(hex)* **Sent data:** <41,50,46,4E,20> Set 100Hz frequency. **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Example 3: Set PWM channel 0 Duty Cycle at 50%. (Current PWM Freq Param is 50Hz 0x9C40)* *(0x9C40 x 50%)/100 = 20000 - 0x4E20(hex)* **Sent data:** <41,50,44,4E,20,00> Set 50% Duty Cycle. **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Example 4: Set PWM channel 2 Duty Cycle at 80%. (Current PWM Freq Param is 100Hz 0x4E20)* *(0x4E20 x 80%)/100 = 16000 - 0x3E80(hex)* **Sent data:** <41,50,44,3E,80,02> Set 80% Duty Cycle. **Received data:** -0x00- (OK - Successful command). -0x4F- OK. All data is in hex.

2.1.5 Sleep Mode – 5Ahex - Z ascii

Commands (host)	1 byte
	1.- 0x5A (hex), Z (ascii). To enter Sleep Mode. 0x00 (hex), To exit Sleep Mode.
Responses (device)	2 bytes(ACK)
	1.- ACK or NAK(Refer to File ACK/NAK List). 2.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	This command sets the Sleep Mode. It takes 150ms to get in/out sleep mode after command is accepted. During sleep mode the Drive and PWMs turn Off, but the memory is conserved. During Sleep Mode any LOW pulse in the MAIN RX line will awake the SmartDRIVE, It' recommended to always awake the SmartDRIVE with a single 0x00(hex) byte. *Please note that this is the ONLY command that can be called even if the SmartDRIVE is mounted or unmounted.*
Example (sent commands)	*Example 1: Set Sleep Mode* **Sent data:** <5A> **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Now the SmartDRIVE is in Sleep Mode.* *Example 2: Awake/Get Out from Sleep Mode* **Sent data:** <00> **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Now the SmartDRIVE is in Active Mode.* All data is in hex.

2.1.6 Set Master/Secondary USART Baud Rate – 58hex - X ascii

Commands (host)	3 bytes
	1.- 0x58 (hex), X (ascii). 2.- 0x4D(hex), M (ascii) Master USART 0x53(hex), S (ascii) Secondary USART 3.- 9600 bps : 00(hex) or 19200 bps : 01(hex) or 57600 bps : 02(hex) or 115200 bps : 03(hex) or 256000 bps : 04(hex) or 500000 bps : 05(hex) or 1000000 bps : 06(hex) or 2000000 bps : 07(hex).
Responses (device)	2-3 bytes
	Set Master USART Baud Rate: Accepted 1.- 0x4F(hex),O(ascii) –Success byte, changing Master Baud Rate… -Delay 500ms- Success ACKs at new Baud Rate: 2.- ACK(Refer to File ACK/NAK List). 3.- 0x4F(hex) O (ascii). **Set Master USART Baud Rate: Denied** Fail NAK at actual(same) baudrate: 1.- NAK(Refer to File ACK/NAK List). 2.- 0x46(hex) F (ascii). **Set Secondary USART Baud Rate: Accepted** 1.- ACK(Refer to File ACK/NAK List). 2.- 0x4F(hex) O (ascii). **Set Secondary USART Baud Rate: Denied** 1.- NAK(Refer to File ACK/NAK List). 2.- 0x46(hex) F (ascii).
Description	Commands needed to set different baud rates to the Master or Secondary USART. When changing the Master USART Baud Rate, the command must be sent by the host at the actual baud rate speed that is being used; If the command is accepted, an 'O' - 0x4F(hex) will be received at the actual baud rate then SmartDRIVE will change to the new baud rate during 500ms and then it will send the ACK(2 bytes) at the new baud rate selected.

	Otherwise, if the command is invalid only NAKs(2 bytes) will be responded by the SmartDRIVE and the baud rate will not be modified. *Only when a Success ACK has been received by the host, the next commands must be sent at the new baud rate defined.* --- When changing the Secondary USART Baud Rate, user must only wait for an ACK to successfully change the baud rate speed, if a NAK is received, baud rate speed will not be modified. Default reset/power on baud rate for both USARTs is 9600bps.
Example (sent commands)	*Example 1: Set Master USART's baud rate - 57600bps* **Sent data:** <58,4D,02> Set 57600bps. **Received data:** -0x4F- At current baud rate speed. -delay 500ms- -0x00- (OK - Successful command). At new baud rate. -0x4F- OK. At new baud rate speed. *Now the SmartDRIVE Master USART BaudRate Speed is 57600bps.* *Example 2: Set Master USART's baud rate - 2000000bps* **Sent data:** <58,4D,07> Set 2000000bps. **Received data:** -0x4F- At current baud rate speed. -delay 500ms- -0x00- (OK - Successful command). At new baud rate. -0x4F- OK. At new baud rate speed. *Now the SmartDRIVE Master USART BaudRate Speed is 2000000bps.* *Example 3: Set Secondary USART's baud rate - 19200bps* **Sent data:** <58,53,01> Set 19200bps. **Received data:** -0x00- (OK - Successful command). -0x4F- OK. *Now the SmartDRIVE Secondary USART BaudRate Speed is 19200bps.* All data is in hex.

2.2 FAT Management – General Functions

Briefly Summary of Commands in this section:

- Get Drive Used/Free Space – **46hex 'F'**
- List/Count of Files and Folders – **4Chex 'L'**
- Get Dir/File Name # – **47hex 'G'**
- Get Current Directory Path – **48hex 'H'**

**** Note that commands on all of this section always respond 2 ACKs or bytes, the first byte is one of the above File ACK/NAK List that informs about command execution status, the second byte correspond to 'O'-Ok or 'F'-Fail. Please refer to "File ACK/NAK List".***

2.2.1 Get DRIVE Free/Total Space – 46hex - F ascii

Commands (host)	1 byte
	1.- 0x46 (hex), F (ascii).
Responses (device)	**Free Space(4 bytes) + Total Space(4 bytes) + 2 bytes(ACK)**
	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 returns the Free Space(first 4 bytes), and the Total Space(next 4 bytes). Finally it returns the 2 ACK bytes. Used Space could be obtained by the formula: **UsedSpace = TotalSpace - FreeSpace**
Example (sent commands)	*Example: Get Free and Total Space of a 2GB micro SD card with ~50% Free Space.* **Sent data:** <46> **Received data:** -00, 0F, 01, 30- Free Space (hex) -00, 1E, 02, 60- Total Space (hex) -0x00- (OK - Successful command). -0x4F- OK. All data is hex.

2.2.2 List/Count of Files and Folders – 4Chex – 'L' ascii

Commands (host)	1 byte
	1.- 0x4C (hex), L (ascii).
Responses (device)	Dirs(2 bytes) + Files(2 bytes) + 2 bytes(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.- ACK or NAK(Refer to File ACK/NAK List). 6.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii).
Description	The "List/Count Files and Folders" command reads and count all the number of Directories and Files under the current microSD Directory Path. This number of Dirs and Files will serve in the next command as reference of available Item Numbers. This is one of the simplest yet effective commands available on the Smart DRIVE FAT Data Management functions.
Example (sent commands)	*Example 1: Get List success* **Sent data:** <4C> **Received data:** -XX, XX- Number of Dirs. -XX, XX- Number of Files. -0x00- (OK - Successful command). -0x4F- OK. *Example 2: Get List fail* **Sent data:** <4C> **Received data:** -00, 00- Number of Dirs. -00, 00- Number of Files. -0xXX- (Any ACK/NAK List – Reason of Error). -0x46- FAIL. All data is hex.

2.2.3 Get Dir/File Name # – 47hex – 'G' ascii

Commands (host)	1 bytes + Dir/File(1 byte) + Item #(2 bytes)
	1.- 0x47 (hex), G (ascii). 2.- 0x46 (hex), F (ascii) –**File Name** or 0x44 (hex), D (ascii) – **Dir Name**. 3.- Item Number of Dir or File(high byte). 4.- Item Number of Dir of File(low byte). ***Item Number** must go from 0 up to "Number of Dirs/Files" - 1, obtained with the previous command.
Responses (device)	Dir/File Name(N bytes) + NULL(1 byte) + 2 bytes(ACK)
	1 up to N.- Dir/File Name. N+1.- 0x00(hex) NULL character. N+2.- ACK or NAK(Refer to File ACK/NAK List). N+3.- 0x4F(hex) O (ascii) or 0x46(hex) F (ascii). *N - Means the length of the Dir or File name.*
Description	The "Get Dir/File Name #" command gets the name of a Directory or File Name(0x46 or 0x44) in the current directory path with the sent **Item Number** parameter. **Item Number** must be less than a previously obtained "Number of Dirs and files" value with the "List/Count Files and Folders" command on the current directory path. This command return the name of the Item followed by a NULL(0x00) character and ending with the 2 standard ACKs.
Example (sent commands)	*Example 1: Get Directory Name of item # 3.* **Sent data:** <47, 44, 00, 03> **Received data:** -XX, XX, XX…- Dir Name. -0x00- NULL character (0x00). -0x00- (OK - Successful command). -0x4F- OK. *Example 2: Get File Name of item # 300.* **Sent data:** <47, 46, 01, 2C> **Received data:** -XX, XX, XX…- File Name. -0x00- NULL character (0x00). -0x00- (OK - Successful command). -0x4F- OK. All data is hex.

2.2.4 Get Current Directory Path –48hex – 'H' ascii

Commands (host)	1 bytes
	1.- 0x48 (hex), H (ascii).
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:* <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.

2.3 FAT Management – File Functions

Briefly Summary of Commands in this section:

- Open File – **4Fhex 'O'**
- New File – **4Ehex 'N'**
- Close File – **43hex 'C'**
- Read Data from File – **52hex 'R'**
- Write Data to File – **57hex 'W'**
- Sync File – **53hex 'S'**
- Set/Get File Pointer Pos – **50hex 'P'**
- Truncate File Size – **56hex 'V'**
- Get File Attribute/Size – **49hex 'I'**
- Erase/Delete File – **45hex 'E'**
- Set/Get File Time-Date – **54hex 'T'**
- Re-Name/Move File – **4Dhex 'M'**
- Copy File – **59hex 'Y'**
- Test End Of File/Error – **51hex 'Q'**

**** Note that commands on all of this section always respond 2 ACKs or bytes, the first byte is one of the above File ACK/NAK List that informs about command execution status, the second byte correspond to 'O'-Ok or 'F'-Fail. Please refer to "File ACK/NAK List".***

2.3.1 Open File –4Fhex – 'O' ascii

Commands (host)	3 bytes + file name with ".ext" + 1byte(NULL).
	1.- 0x4F (hex), O (ascii). (Open File) 2.- **Workspace Block#** 0x00(hex) – 0x03(hex). 3.- Open Mode: 0x01 (hex)- Read Only 0x02 (hex)- Write Only 0x03 (hex)- Read +Write 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 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:* <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:* <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:* <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.

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

Commands (host)	2 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x4E (hex), N (ascii). (New Dir/File) 2.- Dir/File Item: 0x44 (hex), D (ascii) Dir Item or 0x46 (hex), F (ascii) File Item. 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 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 SmartDRIVE 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:* <4E,46,30,31,32,33,2E,74,78,74,00> Creates new file named "0123.txt", that doesn't exist. *Example 2:* <4E,44,41,42,43,00> Create a new directory named "ABC", that doesn't exist. All data is in hex.

2.3.3 Close File –43hex – 'C' ascii

Commands (host)	2 bytes
	1.- 0x43 (hex), C (ascii). (Close File) 2.- **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:* <43,**02**> Save and Close the file contained under the Workspace block 0x02 file object. All data is in hex.

2.3.4 Read File –52hex – 'R' ascii

Commands (host)	4 bytes
	1.- 0x52 (hex), R (ascii). (Read File) 2.- **Workspace Block#** 0x00(hex) – 0x03(hex) 3.- **Bytes to Read** (High byte). 4.- **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 SmartDRIVE 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:* <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:* <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:* <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.

2.3.5 Write File –57hex – 'W' ascii

Commands (host)	4 bytes + N Data bytes
	1.- 0x57 (hex), W (ascii). (Write File) 2.- **Workspace Block#** 0x00(hex) – 0x03(hex). 3.- **Bytes to Write** (High byte). 4.- **Bytes to Write** (Low byte). 5 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:* <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:* <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.

2.3.6 Sync File –53hex – 'S' ascii

Commands (host)	2 bytes
	1.- 0x53 (hex), S (ascii). (Sync File) 2.- **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:* <53,**01**> Sync/Save changes to Workspace block 0x01 file object. All data is in hex.

2.3.7 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.- 0x50 (hex), P (ascii). (File Pointer) 2.- **Workspace Block#** 0x00(hex) – 0x03(hex) 3.- 0x53 (hex), S (ascii). (Set) 4.- Position high byte. 5.- Position medium high byte. 6.- Position medium low byte. 7.- Position low byte. **Get File Pointer position:** 1.- 0x50 (hex), P (ascii). (File Pointer) 2.- **Workspace Block** 0x00(hex) – 0x03(hex). 3.- 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:* <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:* <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:* <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.

2.3.8 Truncate File –56hex – 'V' ascii

Commands (host)	2 bytes
	1.- 0x56 (hex), V (ascii). (Truncate File) 2.- **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:* <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.

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

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

Commands (host)	2 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x49 (hex), I (ascii). (Dir/File Info) 2.- 0x53 (hex), S (ascii). Size or 0x46 (hex), F (ascii). FAT Attribute 3 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:* <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: <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:* <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:* <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.

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

Commands (host)	2 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x45 (hex), E (ascii). (Erase Dir/File) 2.- 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:* <45,4F,30,31,32,33,2E,74,78,74,00> Erase File "0123.txt". *Example 2:* <45,4F,41,42,43,2E,77,78,6C,00> Erase File "ABC.wxl". *Example 3:* <45,4F,72,6F,63,6B,00> Erase the Dir "rock". All data is in hex.

2.3.11 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.- 0x54 (hex), T (ascii). (Dir/File Time & Date) 2.- 0x53 (hex), S (ascii). (Set) 3.- Hours 0-23(dec). 4.- Minutes 0-59(dec). 5.- Seconds 0-59(dec). 6.- Day 1-31(dec). 7.- Month 1-12(dec). 8.- Year upper byte 1980-2107(dec). 9.- Year lower byte 10 up to N (Dir/File name including extension). N+1.- 0x00 (hex) NULL ascii. **Get Dir/File Time and Date:** 1.- 0x54 (hex), T (ascii). (Dir/File Time & Date) 2.- 0x47 (hex), G (ascii). (Get) 3 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 to DEFAULT values.**

Example (sent and received commands)	*Example 1:* <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:* <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:* <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.

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

Commands (host)	1 bytes + Dir/File OLD name with ".ext" + 1byte(NULL) + Dir/File NEW name with ".ext" + 1byte(NULL).
	1.- 0x4D (hex), M (ascii). (Rename/Move Dir/File) 2 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:* <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:* <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:* <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:* <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:* <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:* <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.

2.3.13 Copy File –59hex – 'Y' ascii

Commands (host)	1 bytes + File name to Copy with ".ext" + 1byte(NULL) + New File name with ".ext" + 1byte(NULL).
	1.- 0x59 (hex), Y (ascii). (Copy File) 2 up to N (File name to copy including extension). N+1.- 0x00 (hex) NULL ascii(end of File name). N+2 up to M (New File name including extension). M+1.- 0x00 (hex) NULL ascii(end of New File name).
Responses (device)	4 bytes + 1 byte File ACK + 1 byte Comm ACK
	1.- SCB high byte. 2.- SCB medium high byte. 3.- SCB medium low byte. 4.- SCB low byte. 5.- 0xXX (hex) - Refer to "File ACK List". 6.- 0x4F (hex), O (ascii) – success ACK. 0x46 (hex), F (ascii) – fail NAK. **SCB stands for Successfully Copied Bytes, this parameter is an **unsigned long** type.*
Description	This Command copies the File contents of the "File name to Copy" file, and creates a new "New File Name" file with the contents of the previous file. If the File to copy doesn't exist during a **Copy File** command, this will fail with NO FILE.
Example (sent commands)	*Example 1:* <59,31,32,33,2E,74,78,74,00,31,32,33,63,6F,70,79 2E,74,78,74,00> Copy contents from a File named "123.txt" and create a new file named to "123copy.txt" with the contents of file "123.txt" file. *Example 2:* <59,31,32,33,2E,74,78,74,00,41,42,43,2E,74,78,74,00> Copy contents from a File named "123.txt" and create a new file named to "ABC.txt" with the contents of file "123.txt" file. All data is in hex.

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

Commands (host)	3 bytes
	1.- 0x51 (hex), Q (ascii). (Test File) 2.- **Workspace Block#** 0x00(hex) – 0x03(hex) 3.- 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:* <51,**01**,52> Test for an Error on the Workspace block 0x01 file object. *Example 2:* <51,**01**,45> Test for an End of File on the Workspace block 0x01 file object. All data is in hex.

2.4 FAT Management – Folder/Dir Functions

Briefly Summary of Commands in this section:

- Open/Enter Folder/Dir – **44hex 'D'**
- New Folder/Dir – **4Ehex 'N'**
- Erase/Delete Folder/Dir – **45hex 'E'**
- Re-Name Folder/Dir – **4Dhex 'M'**

**** Note that commands on all of this section always respond 2 ACKs or bytes, the first byte is one of the above File ACK/NAK List that informs about command execution status, the second byte correspond to 'O'-Ok or 'F'-Fail. Please refer to "File ACK/NAK List".***

2.4.1 Open Folder/Dir –44hex – 'D' ascii

Commands (host)	1 bytes + Dir name + 1byte(NULL).
	1.- 0x44 (hex), D (ascii). (Open Dir) 2 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. SmartDRIVE 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.* <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".* <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".* <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".* <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".* <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.* <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.

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

Commands (host)	2 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x4E (hex), N (ascii). (New Dir/File) 2.- Dir/File Item: 0x44 (hex), D (ascii) Dir Item or 0x46 (hex), F (ascii) File Item. 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 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 SmartDRIVE 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:* <4E,46,30,31,32,33,2E,74,78,74,00> Creates new file named "0123.txt", that doesn't exist. *Example 2:* <4E,44,41,42,43,00> Create a new directory named "ABC", that doesn't exist. All data is in hex.

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

Commands (host)	2 bytes + Dir or File name with ".ext" + 1byte(NULL).
	1.- 0x45 (hex), E (ascii). (Erase Dir/File) 2.- 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:* <45,4F,30,31,32,33,2E,74,78,74,00> Erase File "0123.txt". *Example 2:* <45,4F,41,42,43,2E,77,78,6C,00> Erase File "ABC.wxl". *Example 3:* <45,4F,72,6F,63,6B,00> Erase the Dir "rock". All data is in hex.

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

Commands (host)	1 bytes + Dir/File OLD name with ".ext" + 1byte(NULL) + Dir/File NEW name with ".ext" + 1byte(NULL).
	1.- 0x4D (hex), M (ascii). (Rename/Move Dir/File) 2 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:* <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:* <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:* <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:* <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:* <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:* <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.

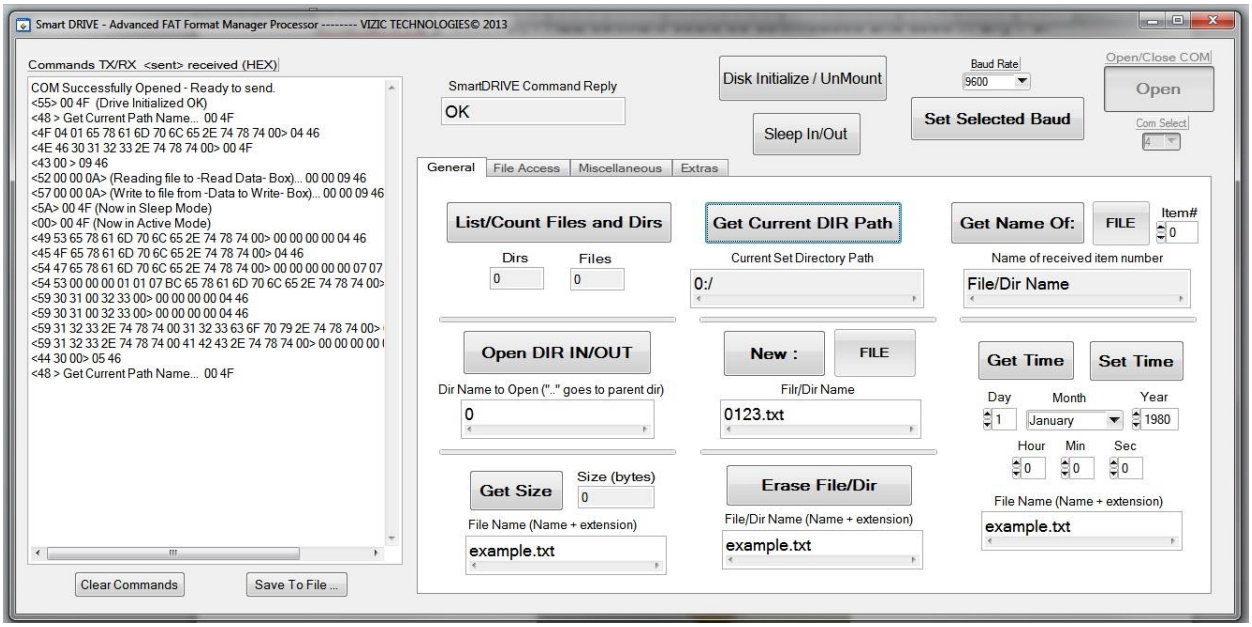
3 Development software tools

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

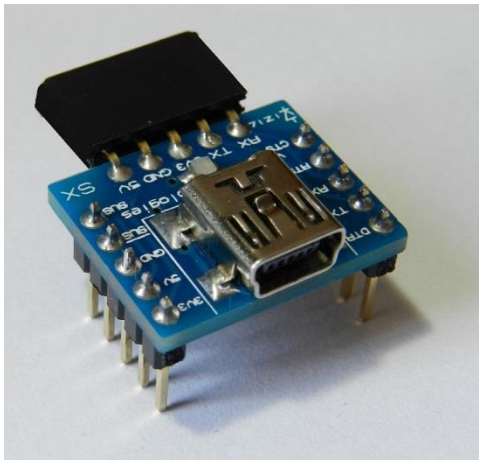
This software simulates most of the functions of the SmartDRIVE chip by connecting it to the PC through the USB-UART SX Bridge, this connection enables real live graphics processing.

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<-> SmartDRIVE Chip.

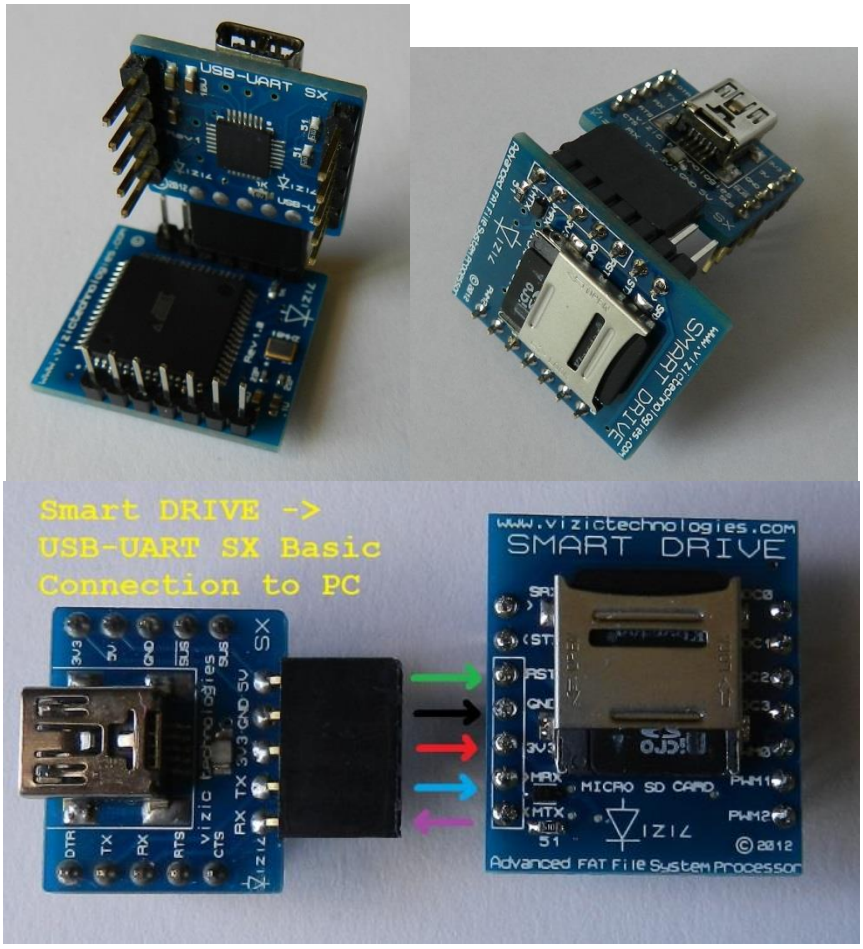
SmartDRIVE PC Interface:



The USB-UART SX:



Smart DRIVE connected to the USB-UART SX



For detailed information about this software and how to use it, please refer to the "SmartDRIVE-PCsimulation.pdf" sheet that could be downloaded in the web site.

For detailed information about the USB-UART SX Bridge, 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.

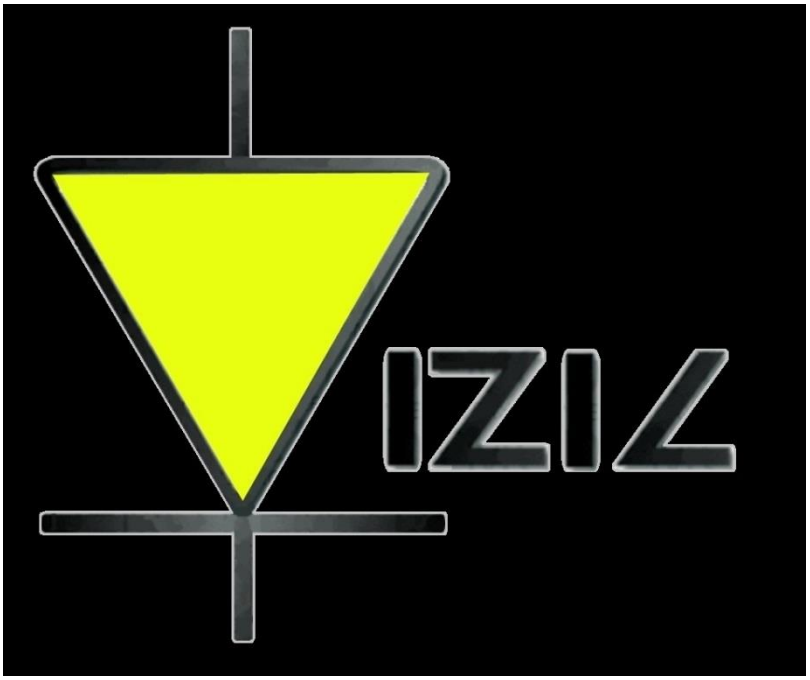
Disclaimer of Warranties & Limitation of Liability:

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.

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

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



www.VIZICTECHNOLOGIES.COM