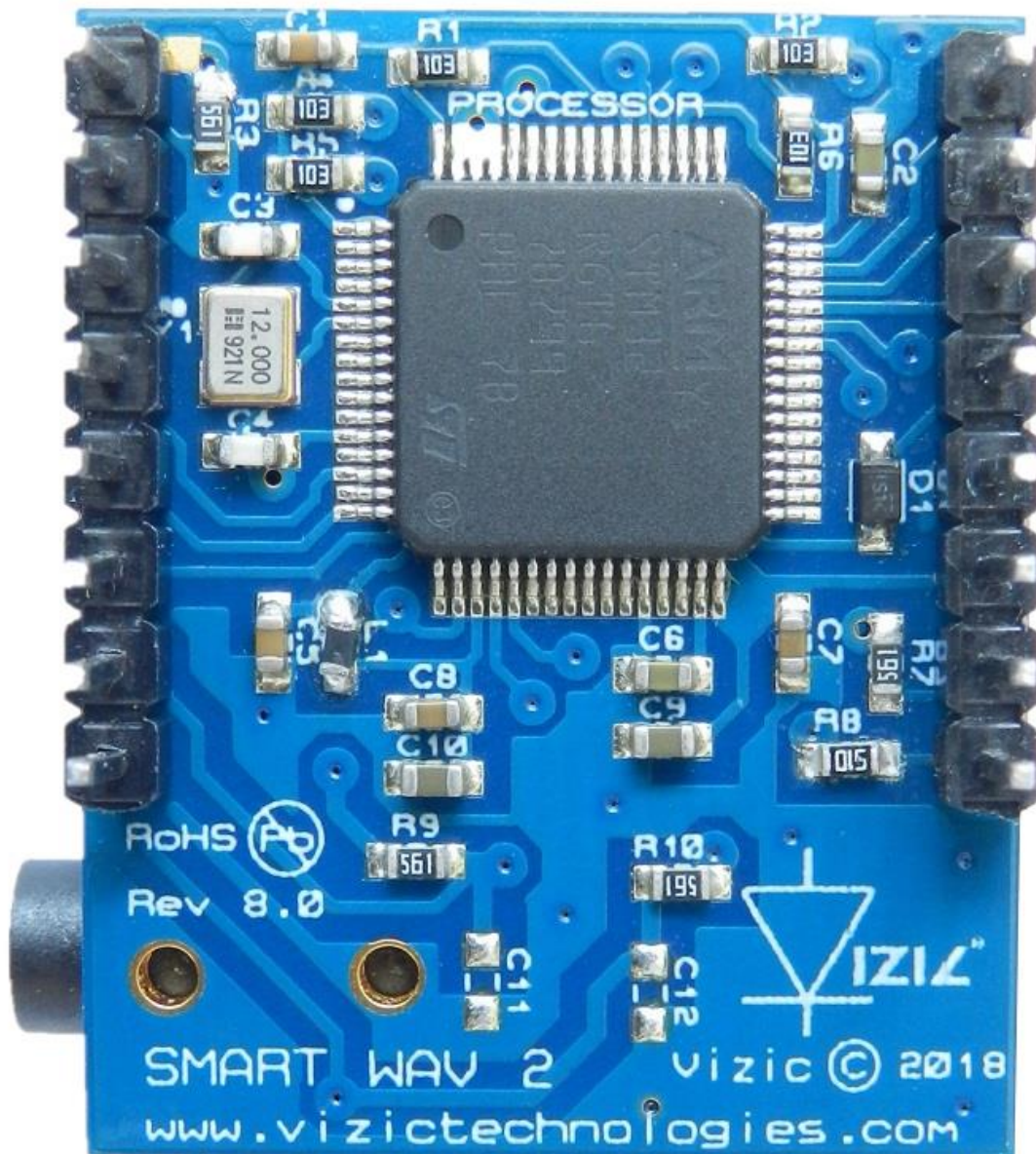


VIZIC
TECHNOLOGIES

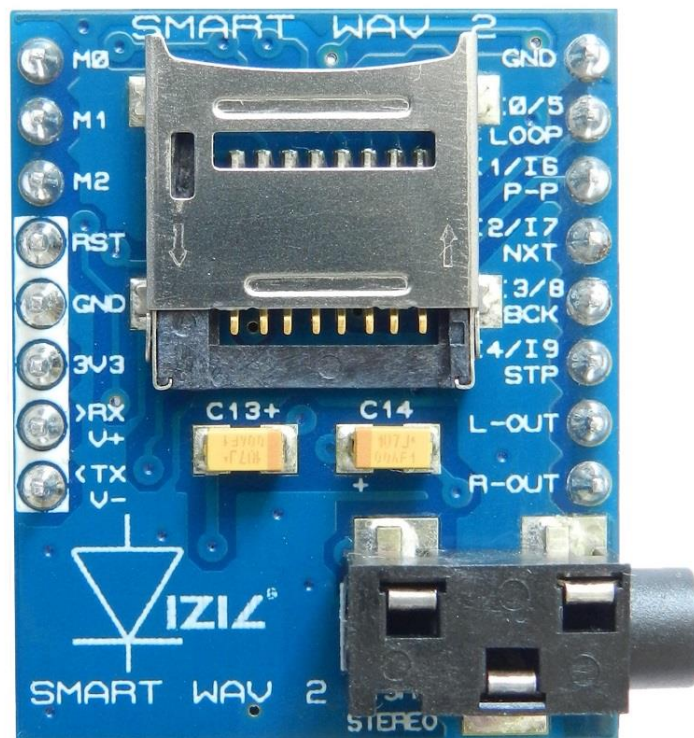
SMARTWAV 2

Serial Mode Command Set----Rev 1.0

SmartWAV 2 – Intelligent Embedded Audio Processor



SmartWAV 2 Top View



SmartWAV 2 Direct phones connection: Pre-Amplified Outputs



Table Of Contents:

1- SmartWAV 2	6
1.1 Introduction	6
1.2 Features	7
1.3 Typical Applications	8
2- SmartWAV 2 System Explained	9
3- Operation Modes	12
3.1 Polyphonic Slave Serial Mode 0 - Standard Serial	12
4- SmartWAV 2 Pinout Configuration.....	13
5- Polyphonic Slave Serial Mode 0 - Standard Serial	14
5.1 Master Host Interface	14
5.2 Command Protocol Flow Control	14
5.3 Power-up, Reset and Serial Setup	15
5.4 Standard Serial Mode 0 Pinout	15
5.5 Standard Serial Mode 0 Typical Connection	17
6- Master Functions - Serial Commands	18
6.1 Initialize System	19
6.2 Reset System	20
6.3 Sleep/Stand-by System	21
6.4 Wake System	23
6.5 Master System Gain	24
6.6 Set Default Master Gain	25
6.7 Master System Sample Rate	26
6.8 Keep Master Sample Rate	27
6.9 System Baud Rate	28
6.10 Set Default Voice Messages	30
6.11 Set Default MIDI Release Time	31
7- Channel Functions - Serial Commands	33
7.1 Mount Track	34
7.2 Play/Pause	36
7.3 Play Immediately	37
7.4 Volume	39

7.5 Panning	40
7.6 Rewind and Play	42
7.7 Advance to Point	43
7.8 Set A and B Points	45
7.9 Loop Points	48
7.10 Stop	49
7.11 Get Mount State	50
7.12 Get Play/Pause	51
7.13 Get Volume	52
7.14 Get Panning	53
7.15 Get Loop	54
7.16 Get Name	55
7.17 Get Current Point	56
7.18 Get A and B Points	57
7.19 Get Remain Time	58
7.20 Get Size (Total Time).....	59
7.21 Get Mono/Stereo	60
7.22 Get Bits Per Sample	61
7.23 Get Sample Rate	62

8- Master FAT/FAT32 Data Management Functions, Data Logger - Serial Commands

8.1 Mount SD Card	65
8.2 List Directories	66
8.3 List Files	67
8.4 Get Dir Item Name	68
8.5 Get File Item Name	70
8.6 Open Directory	72
8.7 Get Current Path	75
8.8 New Directory	76
8.9 New File	78
8.10 Erase Directory	80
8.11 Erase File	81
8.12 Get File Size	82
8.13 Get FAT Attribute	83
8.14 Get Date & Time	84
8.15 Set Date & Time	86
8.16 Dir Rename/Move	88
8.17 File Rename/Move	90
8.18 Get Free & Total Space	92

9- Workspace# FAT/FAT32 Data Management Functions, Data Logger - Serial Commands	93
9.1 Open File	95
9.2 Read File	97
9.3 Write File	100
9.4 Get File Pointer	102
9.5 Set File Pointer	104
9.6 Save File	106
9.7 Truncate File	107
9.8 Test Error In File	108
9.9 Test End of File	109
9.10 Close File	110
10- MicroSD File/Folder Organization - Standard Serial Mode 0	111
11- Formatting MicroSD Card	112
12- Development Software & Hardware Tools	115
13- Proprietary Information	117
14- Disclaimer of Warranties & Limitation of Liability	117

1- SmartWAV 2:

1.1- Introduction:



The SmartWAV 2 is an Intellectual Property smart high-end audio processor running on a state-of-the-art ARM Cortex M4 chip. The processor is mounted on a development board for easy and fast development.

SmartWAV 2 is polyphonic: can play up to 14 channels with high quality stereo sound from a microSD card with universal **FAT/FAT32** format. The processor supports **8/16bit**, **8Khz-48Khz**, **mono/stereo**, WAVE (**.WAV**) files with CD Quality.

The main goal of the SmartWAV 2 it's to bring a very easy way to add polyphonic CD high quality stereo audio to any application or project without the user having experience in handling, mixing or decoding audio, neither managing FAT32 file formatting.

SmartWAV 2 is not a programmable device, no IDEs, programmers, nor debuggers are needed, it is a fully featured end device ready to plug and play in Slave mode, MIDI or Stand-Alone modes.

1.2- Features:

- Pre-amplified dual channel stereo out with 8/16bit, stereo/mono, and up to 48khz sampling rate, CD quality.
- Up to 14 channel polyphonic playback / auto mixing.
- Integrated digital volume control with 1/100 steps.
- Eight modes of operation, including Slave and Stand-Alone modes (M0, M1 and M2 Input pins for mode selection).
- MIDI mode compliant with channel selection.
- On board stereo 3.5mm plug for headphones, or line out.
- Easy 5 pin interface to any host device: **VCC, TX, RX, GND, RESET**. For standard serial mode.
- On-board uSD/uSDHC memory card socket with FAT/FAT32 support up to 32GB for storing thousands of tracks/audio **WAVE** files. No need of special/rare file formatting.
- Special data-logging FAT file create, open, read, write, delete, etc. functions with Long File Names and folder nesting / management support.
- 57600bps standard default baud rate speed for standard serial mode, 8 bits, no parity, 1 stop bit.
- 5V and 3V3 I/O compatible.
- 3V3 power supply, ultra-low current consumption.
- Sleep & Stand-by modes.

1.3- Typical Applications:

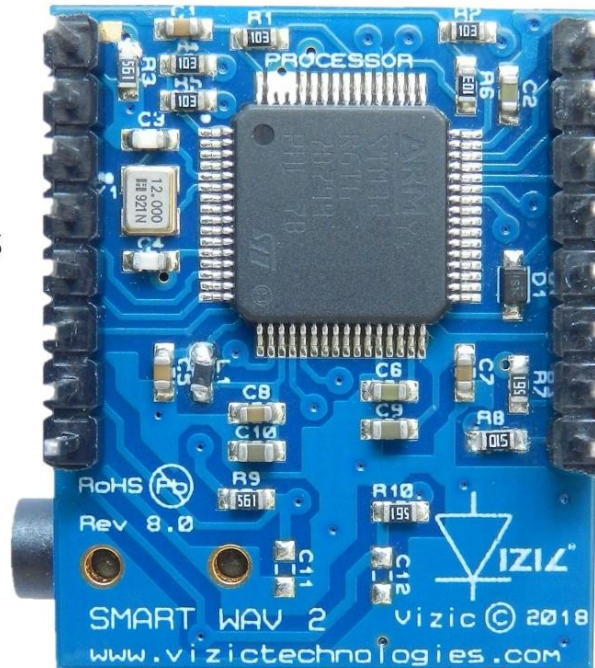
- Embedded polyphonic audio/sound systems.
- All voice annunciator systems.
- MIDI players
- Audio channel mixers.
- Battery powered audio systems.
- Drum machines.
- Automotive, parking, GPS navigation systems.
- Robotics, industrial control.
- Loop stations.
- Traffic facilities: Toll gates, parking lots.
- Sequencers.
- Home automation and domestic appliances.
- Elevator, Security, Access-Control, Warning devices.
- Toys, learning tools, talking books, gaming sound.
- Mp3 like full function simple systems.

2- SmartWAV 2 System Explained:

Main Processor

Stand-Alone
Modes Inputs

Stereo Audio
Outputs



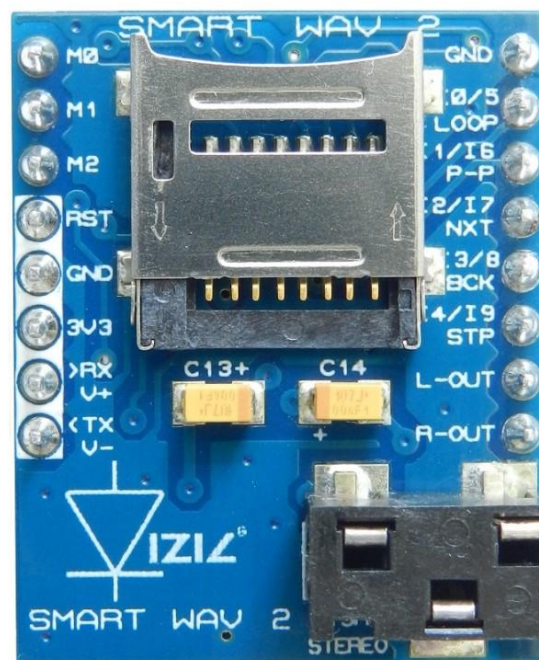
Mode
Selection
Inputs

Serial
Interface
Inputs

MicroSD Card Socket

Mode
Selection
Inputs

Serial
Interface
Inputs



Stand-Alone
Modes Inputs

Stereo Audio
Outputs

3.5mm
Stereo
Output

SmartWAV 2 processor internally is a sophisticated multi-task system, it processes individual .wav tracks mounted on each of the 14 channels, mixing those with a master sample rate and a master gain control, this process is automatically.

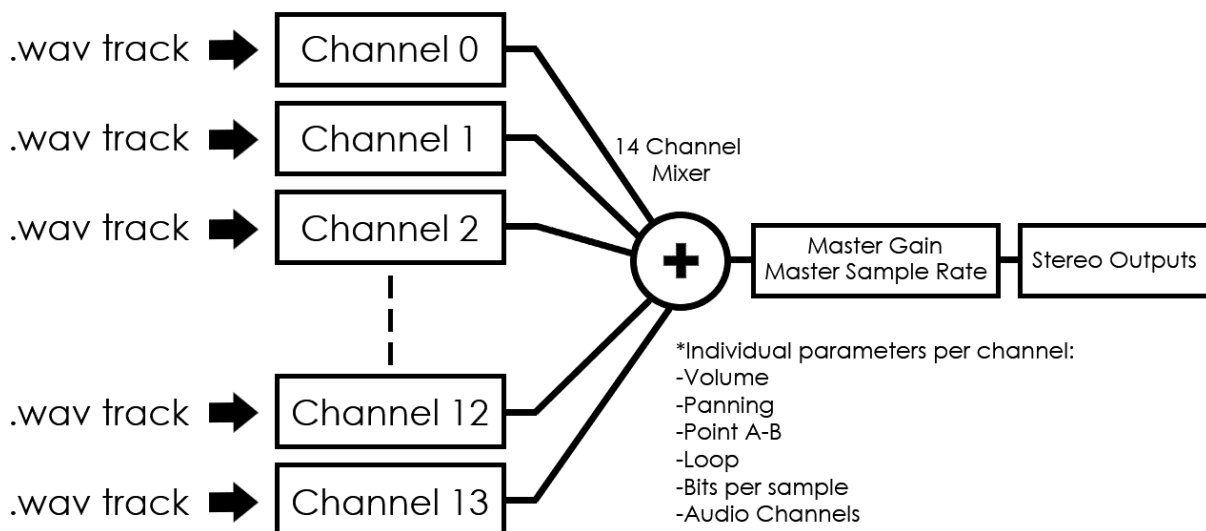
The master sample rate is how the processor outputs audio samples; this parameter is global for all 14 channels.

The master gain control (master mixer volume) is a global parameter that affects all the 14 channels.

For each single channel: volume, panning, point A-B, loop points, bits per sample and audio Channels(mono/stereo) parameters are handled independently:

The next diagram visually explains the system:

SmartWAV 2 Internal Multi-Task System:

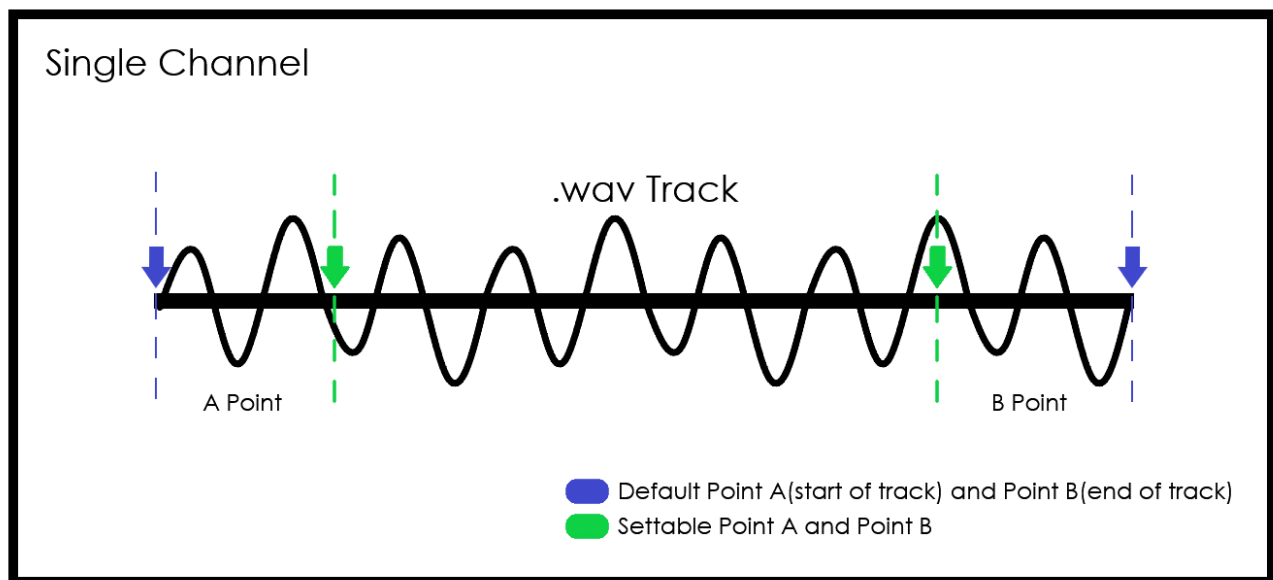


Point A and Point B are markers that the processor handles for each channel for rewind, loop points and other functions. Each time a track is mounted, the points A and B are set at their defaults: A-Start of track and B-End of track (blue color).

User can later set and re-define those A and B points (green color) by the respective command.

The next diagram visually explains the above:

SmartWAV 2 Point A and Point B:



The bits per sample of each channel are internally obtained from the mounted .wav track, those are automatically handled by the system and user does not have to worry on how to handle those. If the user needs this parameters, those can be requested with the respective command.

The audio channels (1 for mono and 2 for stereo) are internally obtained from the mounted .wav track, those are automatically handled by the system and user does not have to worry on how to handle those. If the user needs this parameters, those can be requested with the respective command.

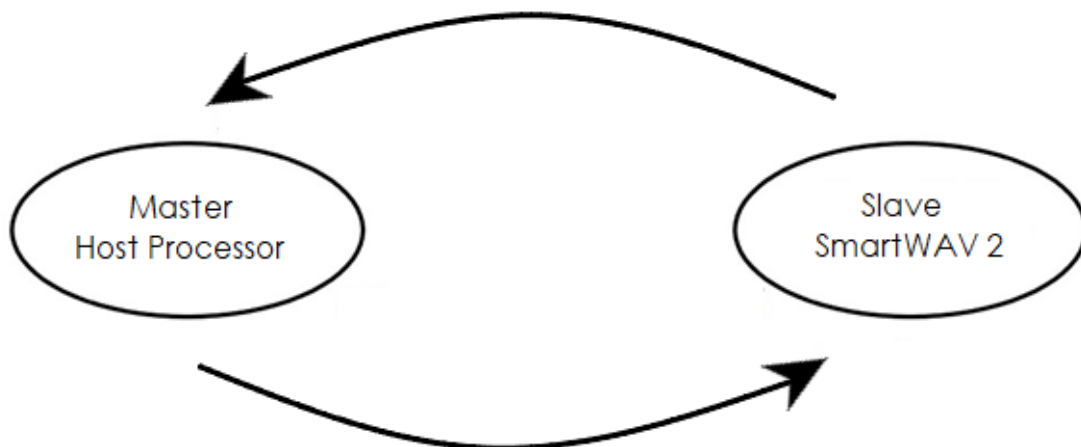
3- Operation Modes:

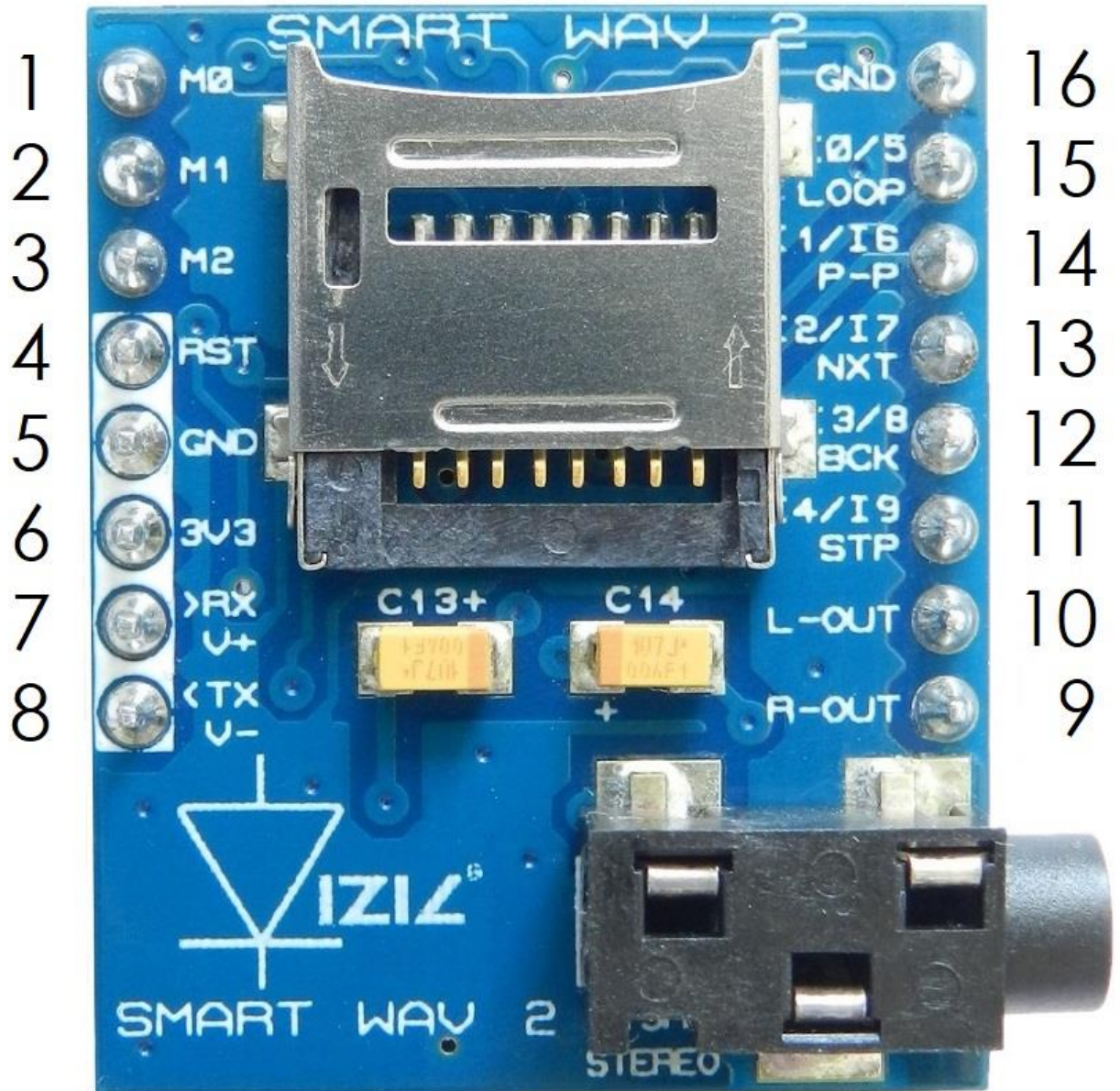
The module offers **8(eight)** modes of operation; selection between modes is by setting the external pins M0, M1 and M2, **this document covers only the Polyphonic Slave Standard Serial Mode 0**, for more operation modes please refer to the document **SmartWAV2-Datasheet.pdf**.

3.1 - Polyphonic Slave Serial Mode 0 - Standard Serial: It is the most complete mode of operation of the system, it offers 14-channel polyphonic audio tracks triggered with a serial interface to any host micro-controller with a serial RX, TX port(UART/USART). Audio related functions are called using simple commands. The SmartWAV 2 allows users to develop their application using their favorite micro-controller or processor and software development tools using the provided libraries.

SmartWAV 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 on the main host controller (8051, PIC, ATMEL, FREESCALE, STMICRO, ARM, CORTEX), or any development platform (ARDUINO, RASPBERRY, FPGA MBED, etc.) or PC (USB-UART SX 2)).

-In Standard Serial mode (master-slave model), the master send commands to the slave and this replies with commands.



4- SmartWAV 2 Pinout:

5- Polyphonic Slave Serial Mode 0 - Standard Serial:

5.1- Master-Slave Host Interface:

SmartWAV 2 in this mode acts as a slave peripheral device, providing a bidirectional serial interface to a master host controller via its UART(Universal Asynchronous Receiver - Transmitter), the required pins of Standard Serial Mode 0 are labeled with a white rectangle.

Any microcontroller or processor (AVR, PIC, mbed, raspberry PI, ARDUINO, beaglebone, 8051, MBED, FPGA, ARM, STM, etc) or PC(by serial interface RS232) as host, can communicate to the device over this serial interface at different bps speeds.

The serial protocol is universal and easy to implement:

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

BaudRate: 57600 bps(default).

Serial data is true and not inverted.

5.2- Command Protocol Flow Control:

SmartWAV 2 processor 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 processor, this executes the command and when the operation is completed it will always return a response*, the processor will also always 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. Processing and execution will take a certain amount of time, depending on the command type and the operation that has to be performed, but once the reply is sent, it means the processor has finished and its ready to receive more commands.*

5.3- Power-up, Reset and Serial Setup:

When the SmartWAV 2 device comes out of a power up or reset, a 50ms delay before sending any command must be met, this time let the processor to configure its internal state machines, do not attempt to communicate with the module before this period. Any command could be sent after this time.

The processor is configured to work at a standard default baud rate of **57600bps**. This baud rate speed can be changed from **9600bps** up to **256000bps** with the corresponding baud rate change command.

5.4- Standard Serial Mode 0 Pinout:

Pin	Symbol	Function	Description
1	M0	INPUT	Digital input power ON/reset mode selection pin, the combination of M0, M1 and M2, determines the working mode of the processor. Internally pulled-down to ground via a 40K resistor.
2	M1	INPUT	Digital input power ON/reset mode selection pin, the combination of M0, M1 and M2, determines the working mode of the processor. Internally pulled-down to ground via a 40K resistor.
3	M2	INPUT	Digital input power ON/reset mode selection pin, the combination of M0, M1 and M2, determines the working mode of the processor. Internally pulled-down to ground via a 40K resistor.
4	RESET	INPUT	Digital input reset pin, an active low pulse greater than 100ns will reset the processor. Internally pulled-up to 3.3V via a 40K resistor. 5V tolerant input.

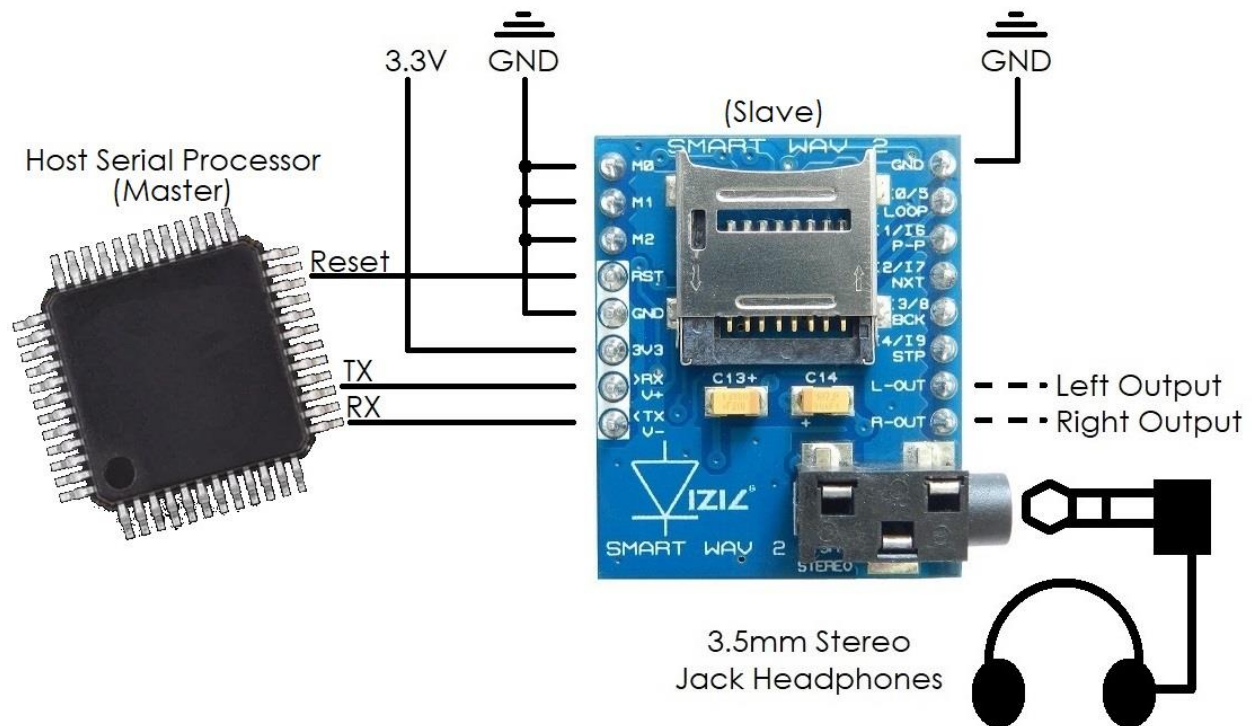
5	GROUND	POWER	Supply ground, be sure to have a well-grounded connection to avoid noise over the audio outputs.
6	3.3V	POWER	Main supply voltage: 2.7v-3.5v. Be sure to use a 10uF and 100nF coupling capacitors to GND as close as possible to avoid noise over the audio outputs.
7	Rx - V+	INPUT	Digital input pin RX (asynchronous serial receiver input pin). Internally pulled-up to 3.3V via a 40K resistor. 5V tolerant input.
8	TX - V-	OUT	Digital output pin TX (asynchronous serial transmitter output pin).
9	R-OUT	OUT	Pre-amplified right channel audio output pin. Connect this pin instead of the 3.5mm plug connector, those are internally connected. (*Do not use both plug and this pin at the same time).
10	L-OUT	OUT	Pre-amplified left channel audio output pin. Connect this pin instead of the 3.5mm plug connector, those are internally connected. (*Do not use both plug and this pin at the same time).
11	I4/I9 - STP	NC	Not used, leave unconnected.
12	I3/I8 - BCK	NC	Not used, leave unconnected.
13	I2/I7 - NXT	NC	Not used, leave unconnected.
14	I1/I6 - P/P	NC	Not used, leave unconnected.
15	I0/I5 - LOOP	NC	Not used, leave unconnected.

16	GND	POWER	Supply ground, be sure to have a well-grounded connection to avoid noise over the audio outputs.
----	-----	-------	--

*NC means no connect.

5.5- Standard Serial Mode 0 Typical Connection:

*The SmartWAV 2 must always be powered with 2.7V-3.5V (Reset and RX pin are 5V tolerant).



*Note: use 3.5mm plug or the L-Out & R-Out output pins, but not both at the same time!

6- Master Functions - Serial Commands:

This section of master functions or master commands are named this way because they affect in general the behavior of the system and all of the 14 channels of the SmartWAV 2 processor, please refer to section **2- SmartWAV 2 System Explained** for more detailed information.

Briefly summary of Functions/Commands in this section:

- | | |
|---------------------------------|-----------------------|
| - Initialize System | - 0x55 hex 'U' ascii |
| - Reset System | - 0x59 hex 'Y' ascii |
| - Sleep/Stand-by System | - 0x5A hex 'Z' ascii |
| - Wake System | - 0x00 hex NULL ascii |
| - Master System Gain | - 0x47 hex 'G' ascii |
| - Set Default Master Gain | - 0x48 hex 'H' ascii |
| - Master System Sample Rate | - 0x54 hex 'T' ascii |
| - Keep Master Sample Rate | - 0x4B hex 'K' ascii |
| - System Baud Rate | - 0x58 hex 'X' ascii |
| - Set Default Voice Messages | - 0x4A hex 'J' ascii |
| - Set Default MIDI Release Time | - 0x44 hex 'D' ascii |

6.1 Initialize System - 55hex - 'U' ascii

Commands (host)	byte(s)
	1.- 0x55 (hex), U (ascii).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Initialize the SmartWAV 2 processor to start receiving serial commands, this command must be called before any other. (please let a 50ms delay before sending this command after a power-on or reset condition).</p> <p>*Note: SmartWAV 2 default power-on or reset baud rate speed is always 57600bps.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -55- Initialize System.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.2 Reset System - 59hex - 'Y' ascii

Commands (host)	byte(s)
	1.- 0x59 (hex), Y (ascii).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	Requests to the SmartWAV 2 processor to perform a software reset, all internal state machines and configurations are reset, this command has the same effect as applying an external hardware reset.
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -59- Reset System.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.3 Sleep/Stand-by System - 5Ahex - 'Z' ascii

Commands (host)	byte(s)
	1.- 0x5A (hex), Z (ascii). 2.- 0x00 (hex) for Sleep mode or 0x01 (hex) for Stand-by mode.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Set the processor to sleep or stand-by.</p> <p>-During sleep mode the processor can be wake up by the "Wake System" command and all internal state machines and configurations are kept.</p> <p>-During the stand-by mode, the processor could only be wake up by performing an external hardware reset, therefore all internal state machines and configurations are reset.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -5A,00- Sleep System.</p> <p>Received data: -4F- OK. (System is slept, now wake system...)</p> <p>Sent data: -00- Wake System.</p> <p>Received data: -4F- OK.</p>

Example 2:

Sent data:

-5A,01- Stand-by System.

Received data:

-4F- OK.

(System is in Stand-by, now wake system by performing an external hardware reset...)

*All data is in hex.

6.4 Wake System - 00hex

Commands (host)	byte(s)
	1.- 0x00 (hex).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Wake the processor from sleep mode, after this command is received, all the internal state machines and configurations are restored.</p> <p>*Note: This command only works for sleep mode, in stand-by mode the processor can only wake up by an external hardware reset.</p>
Examples (sent and received commands)	<p>Example 1: System is in sleep mode</p> <p>Sent data: -00- Wake System.</p> <p>Received data: -4F- OK. (System is running to receive commands)</p> <p>*All data is in hex.</p>

6.5 Master System Gain - 47hex - 'G' ascii

Commands (host)	byte(s)
	1.- 0x47 (hex), G (ascii). 2.- Gain value 0-120 (0x00 to 0x78 hex).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Set the main master gain(volume), this parameter affects the gain/volume of the mix of all the 14 channels. This parameter can be set on the fly at any time.</p> <p>*Even the parameter could be up to 120, it is recommended to set the gain at 100 or below as audio "clipping" can occur.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -47,32- Set Gain at 50.</p> <p>Received data: -4F- OK.</p> <p>Example 2:</p> <p>Sent data: -47,64- Set Gain at 100.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.6 Set Default Master Gain - 48hex - 'H' ascii

Commands (host)	byte(s)
	1.- 0x48 (hex), H (ascii). 2.- Gain value 0-100 (0x00 to 0x64 hex).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Saves to flash and re-writes the default power-on/reset master gain value. This parameter is loaded from flash after a power-on or reset condition.</p> <p>This command is useful if the user needs a fixed power-on/reset master gain volume for the system.</p> <p>*Default factory programmed master gain parameter is 100.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -48,32- Set Default Gain at 50.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.7 Master System Sample Rate - 54hex - 'T' ascii

Commands (host)	byte(s)
	1.- 0x54 (hex), T (ascii). 2.- Sample Rate - High Byte. 3.- Sample Rate - Medium High Byte. 4.- Sample Rate - Medium Low Byte. 5.- Sample Rate - Low Byte.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Sets the main sample rate, this parameter affects all the sampling rate of all of the 14 channels, the parameter could be set from 8kHz up to 48kHz. This parameter can be modified on the fly at any time.</p> <p>*Note: Each time a command "Play Immediately" or "Mount Track" is called, the master sampling rate is updated with the mounted track data.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -54,00,00,36,B0- Set Sample Rate to 14kHz.</p> <p>Received data: -4F- OK.</p> <p>Example 2:</p> <p>Sent data: -54,00,00,AC,44- Set Sample Rate to 44.100kHz.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.8 Keep Master Sample Rate - 4Bhex - 'K' ascii

Commands (host)	byte(s)
	1.- 0x4B (hex), K (ascii). 2.- 0x00 (hex) for Keep mode or 0x01 (hex) for Update mode.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command sets an internal flag to indicate to the processor if must update or keep the master sample rate each time a "Play Immediately" or "Mount Track" command is called.</p> <p>If the value set is 0x00 (hex), the processor will update sample rate based on last mounted track, if the value is set as 0x01 (hex) the processor won't update and will maintain the current sample rate.</p> <p>*Default power-on/reset master sample rate is 44.100kHz.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4B,01- Keep Master Sample Rate.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.9 System Baud Rate - 58hex - 'X' ascii

Commands (host)	byte(s)
	1.- 0x58 (hex), X (ascii). 2.- Baud Rate - High Byte. 3.- Baud Rate - Medium High Byte. 4.- Baud Rate - Medium Low Byte. 5.- Baud Rate -Low Byte.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK at current baud rate. (Delay of 500ms...) 2.- 0x4F (hex), O (ascii) - Success ACK at new baud rate. or 1.- 0x46 (hex), F (ascii) - Fail NAK at current baud rate.
Description	<p>This command sets a new communication baud rate, the parameter could be from 9600bps up to 256000bps. The command sent by the host must be at the actual working baud rate, then 2 things can occur:</p> <p>1-If the command is accepted an ACK will be received at the current baud rate then SmartWAV 2 will change to the new baud rate speed during 500ms and then it will send another ACK at the new baud rate selected.</p> <p>2-If the command is invalid a NAK will be replied at the current baud rate by the SmartWAV 2 and the baud rate speed will not be modified.</p>

	<p>Only when an ACK has been received by the host, the next commands must be sent at the new received baud rate.</p> <p>*Default power-on/reset baud rate is 57600bps.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: At current baud rate -58,00,01,C2,00- Set 115200bps baud rate.</p> <p>Received data: At current baud rate -4F- OK. (delay 500ms)</p> <p>Received data: At new baud rate -4F- OK.</p> <p>*All data is in hex.</p>

6.10 Set Default Voice Messages - 4Ahex - 'J' ascii

Commands (host)	byte(s)
	1.- 0x4A (hex), J (ascii). 2.- 0x00 (hex) for Disable Messages or 0x01 (hex) for Enable Messages.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Saves to flash and re-writes the default enable/disable register that the processor uses to throw audible voice messages “No memory card detected” and “No WAV files”.</p> <p>This command is useful if the user needs to enable or disable the audible output messages feature for the system.</p> <p>*Default factory programmed value is 0x01 (hex) ENABLED.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4A,00- Disable Audible Messages.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

6.11 Set Default MIDI Release Time - 44hex - 'D' ascii

Commands (host)	byte(s)
	1.- 0x44 (hex), D (ascii). 2.- Release Time 0-127 (0x00 to 7F hex).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Saves to flash and re-writes the default MIDI release time parameter, this parameter can be from 0x00 (hex) to 0x7F (hex).</p> <p>NOTE: This parameter is only used by the processor in the Slave Serial MIDI Mode 4, other modes including standard serial mode 0 are not affected by this parameter.</p> <p>The release time is calculated as follows: MIDI Release Time = (parameter x 40) ms.</p> <p>The above equation states that the minimum release time to set is (1 x 40) ms = 40ms and the maximum is (127 x 40) ms = 5,080 ms.</p> <p>This command is useful if the user needs a fixed power-on/reset MIDI release time for the system.</p> <p>*Default factory programmed MIDI release time parameter is 25, about 1 second: (25 x 40) ms = 1000ms.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-44,19- Set Default Release Time 1000ms.

Received data:

-4F- OK.

Example 2:

Sent data:

-44,4B- Set Default Release Time 3000ms.

Received data:

-4F- OK.

*All data is in hex.

7- Channel Functions - Serial Commands:

SmartWAV 2 is a 14 channel polyphonic audio processor, this section of channel functions is named this way because those commands have effect only on one single track / channel behavior of the system, please refer to section **2- SmartWAV 2 System Explained** for more detailed information. Each of the next commands always receive a channel number parameter, this parameter could be from 0 to 13 (0x00 to 0x0D hex), almost all of the commands support single command call to control all channels, for this use 0xFF hex as channel parameter to take command effect on all active channels.

Briefly summary of Functions/Commands in this section:

- | | |
|-----------------------|---------------------------------|
| - Mount Track | - 0x4D hex 'M' ascii |
| - Play/Pause | - 0x50 hex 'P' ascii |
| - Play Immediately | - 0x49 hex 'I' ascii |
| - Volume | - 0x56 hex 'V' ascii |
| - Panning | - 0x41 hex 'A' ascii |
| - Rewind and Play | - 0x52 hex 'R' ascii |
| - Advance to Point | - 0x45 hex 'E' ascii |
| - Set A and B Points | - 0x42 hex 'B' ascii |
| - Loop Points | - 0x4C hex 'L' ascii |
| - Stop | - 0x53 hex 'S' ascii |
| - Get Mount State | - 0x4F, 0x4D hex 'O', 'M' ascii |
| - Get Play/Pause | - 0x4F, 0x50 hex 'O', 'P' ascii |
| - Get Volume | - 0x4F, 0x56 hex 'O', 'V' ascii |
| - Get Panning | - 0x4F, 0x41 hex 'O', 'A' ascii |
| - Get Loop | - 0x4F, 0x4C hex 'O', 'L' ascii |
| - Get Name | - 0x4F, 0x4E hex 'O', 'N' ascii |
| - Get Current Point | - 0x4F, 0x4F hex 'O', 'O' ascii |
| - Get A and B Points | - 0x4F, 42 hex 'O', 'B' ascii |
| - Get Remain Time | - 0x4F, 0x52 hex 'O', 'R' ascii |
| - Get Size Time | - 0x4F, 0x53 hex 'O', 'S' ascii |
| - Get Mono/Stereo | - 0x4F, 0x43 hex 'O', 'C' ascii |
| - Get Bits Per Sample | - 0x4F, 0x49 hex 'O', 'I' ascii |
| - Get Sample Rate | - 0x4F, 0x54 hex 'O', 'T' ascii |

7.1 Mount Track - 4Dhex - 'M' ascii

Commands (host)	byte(s)
	1.- 0x4D (hex), M (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3 to n .- File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Mounts the received audio(.wav) file name on the received channel to use (0 to 13), this is almost the first command that must be called before any other command related to single channel playback.</p> <p>Once this command is acknowledged, the track is mounted in the received channel and set in pause state, now the channel is ready to be controlled with more commands.</p> <p>The file name must always be ended with the 0x00 (hex) NULL character that ends the command.</p> <p>*The file name could be received with or without including the ".wav" extension, both ways are accepted by the processor, also the received name must not be larger than 128 characters including the .wav extension.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-4D,02,73,6F,6E,67,00,- Mount track called "song.wav" in channel 2.

Received data:

-4F- OK.

Example 2:

Sent data:

-4D,00,73,6F,6E,67,2E,77,61,76,00,- Mount track called "song.wav" in channel 0.

Received data:

-4F- OK.

*All data is in hex.

7.2 Play/Pause - 50hex - 'P' ascii

Commands (host)	byte(s)
	1.- 0x50 (hex), P (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- 0x00 (hex) for Pause or 0x01 (hex) for Play.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Pause or Play the received channel (0 to 13).</p> <p>A track must be already mounted in the selected channel for this command to work.</p> <p>*For each single channel, once the playing track reaches the point B, it is automatically rewinded to the point A and set in pause state (if loop points is active the playing continues).</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -50,02,01,- Play track in channel 2.</p> <p>Received data: -4F- OK.</p> <p>Example 2:</p> <p>Sent data: -50,FF,00,- Pause all channels.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.3 Play Immediately - 49hex - 'I' ascii

Commands (host)	byte(s)
	1.- 0x49 (hex), I (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3 to n .- File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command is a fast shortcut for the “Mount Track” + “Play/Pause” commands; the command immediately performs the mount of the received audio(.wav) file name on the received channel to use (0 to 13) and set the track in play state.</p> <p>The file name must always be ended with the 0x00 (hex) NULL character that ends the command.</p> <p>*The file name could be received with or without including the “.wav” extension, both ways are accepted by the processor, also the received name must not be larger than 128 characters including the .wav extension.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -49,01,73,6F,6E,67,00,- Play Immediately track called “song.wav” in channel 1.</p> <p>Received data: -4F- OK.</p>

Example 2:

Sent data:

-49,00,73,6F,6E,67,2E,77,61,76,00,- Mount
track called "song.wav" in channel 0.

Received data:

-4F- OK.

*All data is in hex.

7.4 Volume - 56hex - 'V' ascii

Commands (host)	byte(s)
	1.- 0x56 (hex), V (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- Volume 0-100 (0x00 to 0x64 hex).
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Set the independent audio volume in percentage (0% to 100%) for the received channel (0 to 13).</p> <p>*This parameter only affects the received single channel, the processor internally mixes the volume of the 14 channels and then the mix is finally controlled by the master gain volume.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -56,03,64,- Set volume to 100% of channel 3.</p> <p>Received data: -4F- OK.</p> <p>Example 2:</p> <p>Sent data: -56,FF,32,- Set volume to 50% of all channels.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.5 Panning - 41hex - 'A' ascii

Commands (host)	byte(s)
	1.- 0x41 (hex), A (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- Panning -100 to 100 (0x9C to 0x64 hex). <i>*Panning parameter is received as a signed char value.</i>
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Set the independent audio panning in percentage: -100%(left) to 100%(right) for the received channel (0 to 13).</p> <p><i>*This parameter only affects the received single channel.</i></p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -41,03,64,- Set panning to 100%(Right) of channel 3.</p> <p>Received data: -4F- OK.</p> <p>Example 2:</p> <p>Sent data: -41,03,9C,- Set panning to -100%(Left) of channel 3.</p> <p>Received data: -4F- OK.</p>

Example 3:

Sent data:

-41,03,00,- Set panning to 0%(Center) of channel 3.

Received data:

-4F- OK.

Example 4:

Sent data:

-41,FF,00,- Set panning to 0%(Center) of all channels.

Received data:

-4F- OK.

*All data is in hex.

7.6 Rewind and Play - 52hex - 'R' ascii

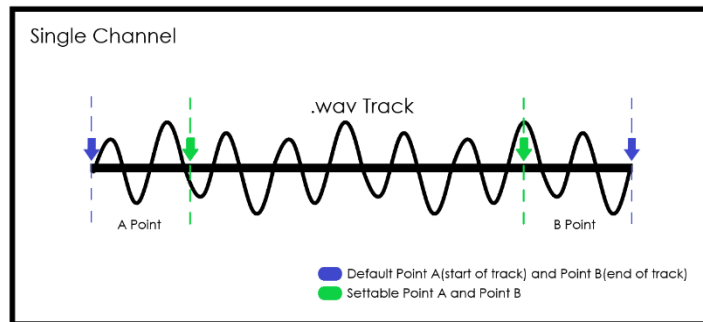
Commands (host)	byte(s)
	1.- 0x52 (hex), R (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Rewinds to the point A and set the received channel (0 to 13) to play state. This command is a fast shortcut of the "Advance to Point (point A)" + "Play/Pause" commands.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1: Sent data: -52,02,- Rewind and play track in channel 2. Received data: -4F- OK.</p> <p>Example 2: Sent data: -52,FF,- Rewind and play all active channels. Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.7 Advance to Point (In Milliseconds) - 45hex - 'E' ascii

Commands (host)	byte(s)
	1.- 0x45 (hex), E (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- Milliseconds - High Byte. 4.- Milliseconds - Medium High Byte. 5.- Milliseconds - Medium Low Byte. 6.- Milliseconds - Low Byte.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Advance to point in time (Milliseconds) the mounted track on the received channel (0 to 13), the time is always calculated from the beginning of the track (not depends on Point A). This command is useful to fast forward or fast backward a track.</p> <p>The current audio playing position/point of each channel track is handled by the processor in Milliseconds; In order to convert the next formula can be applied: Milliseconds = (seconds x 1000) or seconds = (Milliseconds / 1000)</p> <p>A track must be already mounted in the selected channel for this command to work.</p> <p>*The advance to point command is independent of point A and point B, could be set inside or outside the A and B markers. If the received point parameter is farther</p>

	away in time than the current set point B , the track will continue playing until the end of the track , then it will be rewinded to point A .
Examples (sent and received commands)	<p>Example 1: Sent data: -45,05,00,00,75,30- Advance channel 5 to time point 00:00:30 (30000 milliseconds). Received data: -4F- OK.</p> <p>Example 2: Sent data: -45,01,00,00,00,00- Advance channel 1 to time point 00:00:00 seconds (rewind). Received data: -4F- OK.</p> <p>Example 3: Sent data: -45,03,00,00,EA,60- Advance channel 3 to time point 00:01:00 (60000 milliseconds). Received data: -4F- OK.</p> <p>Example 4: Sent data: -45,FF,00,00,EA,60- Advance all active channels to time point 00:01:00 (60000 milliseconds). Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.8 Set A and B Points (In Milliseconds) - 42hex - 'B' ascii

Commands (host)	byte(s)
	1.- 0x42 (hex), B (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- Milliseconds A - High Byte. 4.- Milliseconds A - Medium High Byte. 5.- Milliseconds A - Medium Low Byte. 6.- Milliseconds A - Low Byte. 7.- Milliseconds B - High Byte. 8.- Milliseconds B - Medium High Byte. 9.- Milliseconds B - Medium Low Byte. 10.- Milliseconds B - Low Byte.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command sets the A and B points in time (Milliseconds) for the received channel (0 to 13), the time is always calculated from the beginning of the track. This command is useful to set an A and B time markers for the track, or to set loop regions.</p>  <p>The diagram, titled 'Single Channel', shows a waveform labeled '.wav Track'. Two vertical dashed green lines mark 'A Point' and 'B Point' on the waveform. Blue arrows at the start and end of the track indicate the default points. A legend at the bottom states: 'Default Point A(start of track) and Point B(end of track)' (blue) and 'Settable Point A and Point B' (green).</p> <p>Each time a command “Play Immediately” or “Mount Track” is called, the point A and B of the received channel are reset to the</p>

defaults: **beginning of the track** 00:00:00 and **end of the track** XX:XX:XX in time.

If the received **point A** and **point B** parameters are equal, the system also resets those points to their defaults: **beginning of the track** 00:00:00 and **end of the track** XX:XX:XX in time, this case can be used as a default/reset point A and B markers.

The current audio playing position/point of the channel is not modified by this command.

A track must be already mounted in the selected channel for this command to work.

*Note: For each single channel, once the playing track reaches the **point B**, it is automatically rewinded to the **point A** and set in pause state (if **loop points** is active the playing continues).

If the "Advance to Point" command is called with a time parameter farther away in time than the current set **point B**, the track will continue playing until the **end of the track**, then it will be rewinded to **point A**.

Examples (sent and received commands)

Example 1:

Sent data:

-42,05,00,00,75,30,00,00,EA,60- Set channel 5 point A to time 00:00:30 (30000 milliseconds) and point B to time 00:01:00 (60000 milliseconds).

Received data:

-4F- OK.

Example 2:

Sent data:

-42,01,00,00,00,00,00,00,00- Reset channel 1 point A and point B parameters are equal), this will set points to their defaults: beginning of the track 00:00:00 and XX:XX:XX end of the track.

Received data:

-4F- OK.

Example 3:

Sent data:

-42,03,00,01,D4,C4,00,04,93,E0- Set channel 3 point A to time 00:02:00 (120000 milliseconds) and point B to time 00:05:00 (300000 milliseconds).

Received data:

-4F- OK.

Example 4:

Sent data:

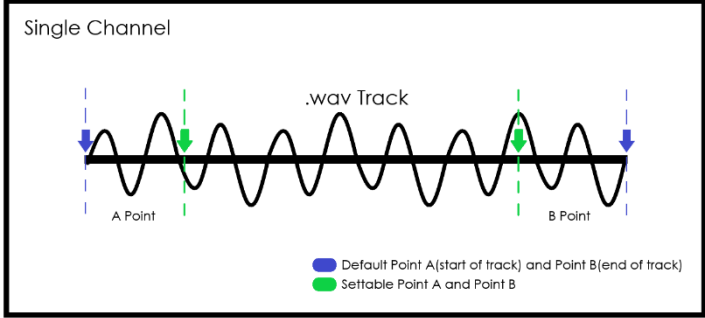
-42,FF,00,01,D4,C4,00,04,93,E0- Set all active channels point A to time 00:02:00 (120000 milliseconds) and point B to time 00:05:00 (300000 milliseconds).

Received data:

-4F- OK.

*All data is in hex.

7.9 Loop Points - 4Chex - 'L' ascii

Commands (host)	byte(s)
	1.- 0x4C (hex), L (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All. 3.- 0x00 (hex) for Disable or 0x01 (hex) for Enable.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command Enables or Disables the loop point A to point B feature. This command is useful to set A - B loop regions.</p>  <p>For each single channel, once the playing track reaches the point B, it is automatically rewinded to the point A and if loop points is enabled the playing continues.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4C,05,01- Enable channel 5 loop A-B feature.</p> <p>Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.10 Stop - 53hex - 'S' ascii

Commands (host)	byte(s)
	1.- 0x53 (hex), S (ascii). 2.- Channel 0x00 to 0x0D or 0xFF (hex) for All.
Responses (device)	byte(s)
	1.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>Stop playing and unmounts the track on the received channel (0 to 13).</p> <p>Once this command is acknowledged, the channel is cleared and reset, however the current volume, panning and loop point parameters are preserved.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1: Sent data: -53,02,- Stop channel 2. Received data: -4F- OK.</p> <p>Example 2: Sent data: -53,FF,- Stop all active channels. Received data: -4F- OK.</p> <p>*All data is in hex.</p>

7.11 Get Mount State - 0x4F, 0x4Dhex - 'O', 'M' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x4D (hex), M (ascii).
Responses (device)	byte(s)
	1.- 0x00 (hex) for No Mounted Track or 0x01 (hex) for Mounted Track. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current mounted or not mounted track state in the received channel (0-13).</p> <p>If a track is mounted on the channel the processor sends 0x01 as reply.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,4D- Ask for track mount state of channel 2.</p> <p>Received data: -01,4F- Track is mounted, OK.</p> <p>Example 1:</p> <p>Sent data: -4F,00,4D- Ask for track mount state of channel 0.</p> <p>Received data: -00,4F- Track is not mounted, OK.</p> <p>*All data is in hex.</p>

7.12 Get Play/Pause - 0x4F, 0x50hex - 'O', 'P' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x50 (hex), P (ascii).
Responses (device)	byte(s)
	1.- 0x00 (hex) for Paused State or 0x01 (hex) for Playing State. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current play/pause state of the received channel (0-13).</p> <p>If a track is playing on the channel the processor sends 0x01 as reply.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,50- Ask for play/pause state of channel 2.</p> <p>Received data: -00,4F- Channel is paused, OK.</p> <p>Example 2:</p> <p>Sent data: -4F,05,50- Ask for play/pause state of channel 5.</p> <p>Received data: -01,4F- Channel is playing, OK.</p> <p>*All data is in hex.</p>

7.13 Get Volume - 0x4F, 0x56hex - 'O', 'V' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x56 (hex), V (ascii).
Responses (device)	byte(s)
	1.- Volume 0-100 (0x00 to 0x64 hex). 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current audio volume parameter of the received channel (0-13).</p> <p>Volume is returned from 0% to 100%.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,56- Ask for audio volume of channel 2.</p> <p>Received data: -32,4F- Channel volume is 50%, OK.</p> <p>Example 2:</p> <p>Sent data: -4F,05,56- Ask for audio volume of channel 5.</p> <p>Received data: -64,4F- Channel volume is 100%, OK.</p> <p>*All data is in hex.</p>

7.14 Get Panning - 0x4F, 0x41hex - 'O', 'A' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x41 (hex), A (ascii).
Responses (device)	byte(s)
	1.- Panning -100 to 100 (0x9C to 0x64 hex). <i>*Panning parameter is sent as a signed char value.</i> 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	This command asks for the current panning parameter of the received channel (0-13). Panning is returned from -100% to 100%.
Examples (sent and received commands)	Example 1: Sent data: -4F,02,41- Ask for panning of channel 2. Received data: -64,4F- Channel panning is 100%(Right), OK. Example 2: Sent data: -4F,05,41- Ask for panning of channel 5. Received data: -9C,4F- Channel panning is -100%(Left), OK. *All data is in hex.

7.15 Get Loop - 0x4F, 0x4Chex - 'O', 'L' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x4C (hex), L (ascii).
Responses (device)	byte(s)
	1.- 0x00 (hex) for Loop Disabled or 0x01 (hex) for Loop Enabled. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current loop points register state of the received channel (0-13).</p> <p>If loop points bit is enabled on the channel the processor sends 0x01 as reply.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,4C- Ask for loop state of channel 2.</p> <p>Received data: -01,4F- Channel loop points is enabled, OK.</p> <p>*All data is in hex.</p>

7.16 Get Name - 0x4F, 0x4Ehex - 'O', 'N' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x4E (hex), N (ascii).
Responses (device)	byte(s)
	1 to n .- File Name. n+1 .- 0x00 (hex), NULL ascii. n+2 .- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the name of the current mounted track on the received channel (0-13).</p> <p>The track name is sent byte by byte followed with a NULL 0x00hex character to indicate end of name, the .wav extension of the file name is omitted and not sent.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,4E- Ask for name of track in channel 2.</p> <p>Received data: -73,6F,6E,67,00,4F- Track name is "song.wav" (" .wav" not sent), OK.</p> <p>*All data is in hex.</p>

7.17 Get Current Point - 0x4F, 0x4Fhex - 'O', 'O' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x4F (hex), O (ascii).
Responses (device)	byte(s)
	1.- Milliseconds - High Byte. 2.- Milliseconds - Medium High Byte. 3.- Milliseconds - Medium Low Byte. 4.- Milliseconds - Low Byte. 5.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the currently playing point, file audio pointer or elapsed time (In Milliseconds) for the received channel (0-13), the time is always calculated from the beginning of the track.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,4F- Ask for current playing point of channel 2.</p> <p>Received data: -00,01,D4,C0,4F- Current file playing point or pointer is 00:02:00 (120000 milliseconds), OK.</p> <p>*All data is in hex.</p>

7.18 Get A and B Points - 0x4F, 0x42hex - 'O', 'B' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x42 (hex), B (ascii).
Responses (device)	byte(s)
	1.- Milliseconds A - High Byte. 2.- Milliseconds A - Medium High Byte. 3.- Milliseconds A - Medium Low Byte. 4.- Milliseconds A - Low Byte. 5.- Milliseconds B - High Byte. 6.- Milliseconds B - Medium High Byte. 7.- Milliseconds B - Medium Low Byte. 8.- Milliseconds B - Low Byte. 9.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current set point A and point B time markers (In milliseconds) for the received channel (0-13).</p> <p>*Each time a command "Play Immediately" or "Mount Track" is called, the point A and B of the received channel are reset to the defaults: beginning of the track 00:00:00 and end of the track XX:XX:XX in time.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,42- Ask for point A and B of channel 2.</p> <p>Received data: -00,01,D4,C0,00,04,93,30,E0- Point A is 00:02:00 (120000 milliseconds) and point B 00:05:00 (300000 milliseconds), OK.</p> <p>*All data is in hex.</p>

7.19 Get Remain Time - 0x4F, 0x52hex - 'O', 'R' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x52 (hex), R (ascii).
Responses (device)	byte(s)
	1.- Milliseconds - High Byte. 2.- Milliseconds - Medium High Byte. 3.- Milliseconds - Medium Low Byte. 4.- Milliseconds - Low Byte. 5.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the current remaining time (In Milliseconds) for the channel track to reach point B for the received channel (0-13), the processor basically subtracts the point B time minus current playing point.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,04,52- Ask for remaining time to reach point B for track in channel 4.</p> <p>Received data: -00,01,5F,90,4F- Remaining time is 00:01:30 (90000 milliseconds), OK.</p> <p>*All data is in hex.</p>

7.20 Get Size (Total Time) - 0x4F, 0x53hex - 'O', 'S' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x53 (hex), S (ascii).
Responses (device)	byte(s)
	1.- Milliseconds - High Byte. 2.- Milliseconds - Medium High Byte. 3.- Milliseconds - Medium Low Byte. 4.- Milliseconds - Low Byte. 5.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the total audio size in time (Milliseconds) for the track mounted on received channel (0-13).</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,05,53- Ask for total audio size time of track in channel 5.</p> <p>Received data: -00,09,27,C0,4F- Size in time is 00:10:00 (600000 milliseconds), OK.</p> <p>*All data is in hex.</p>

7.21 Get Mono/Stereo - 0x4F, 0x43hex - 'O', 'C' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x43 (hex), C (ascii).
Responses (device)	byte(s)
	1.- 0x01 (hex) for 1 (Mono) or 0x02 (hex) for 2 (Stereo). 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the audio channels: 1 for mono or 2 for stereo of the mounted track in the received channel (0-13).</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,43- Ask for mono/stereo of track in channel 2.</p> <p>Received data: -01,4F- One channel, track is mono, OK.</p> <p>Example 1:</p> <p>Sent data: -4F,05,43- Ask for mono/stereo of track in channel 5.</p> <p>Received data: -02,4F- Two channels, track is stereo, OK.</p> <p>*All data is in hex.</p>

7.22 Get Bits per Sample - 0x4F, 0x49hex - 'O', 'I' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x49 (hex), I (ascii).
Responses (device)	byte(s)
	1.- 0x08 (hex) for 8 bit or 0x10 (hex) for 16 bit. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the audio bits per sample of the track mounted track in the received channel (0-13).</p> <p>.wav files could only be 8 or 16 bits per sample.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,02,49- Ask for the bits per sample of track in channel 2.</p> <p>Received data: -08,4F- Track is 8 bps, OK.</p> <p>Example 1:</p> <p>Sent data: -4F,05,49- Ask for the bits per sample of track in channel 5.</p> <p>Received data: -10,4F- Track is 16 bps, OK.</p> <p>*All data is in hex.</p>

7.23 Get Sample Rate - 0x4F, 0x54hex - 'O', 'T' ascii

Commands (host)	byte(s)
	1.- 0x4F (hex), O (ascii). 2.- Channel 0x00 to 0x0D (hex), 0-13 (ascii). 3.- 0x54 (hex), T (ascii).
Responses (device)	byte(s)
	1.- Sample Rate - High Byte. 2.- Sample Rate - Medium High Byte. 3.- Sample Rate - Medium Low Byte. 4.- Sample Rate - Low Byte. 5.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command asks for the audio sample rate data of the track mounted in received channel (0-13). This sample rate data is not the same as the actual master system sample rate, the parameters may be equal or different.</p> <p>A track must be already mounted in the selected channel for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -4F,05,54- Ask for audio sample rate data of track in channel 5.</p> <p>Received data: -00,00,AC,44,4F- Sample rate track data is 44.100kHz, OK.</p> <p>*All data is in hex.</p>

8- Master FAT/FAT32 Data Management Functions, Data Logger -Serial Commands:

SmartWAV 2 includes full FAT/FAT32 data management data logger functions, this enables the processor to be able to create, read, modify, write, save, etc. files and folders inside the microSD card, those functions enable full possibilities for advanced embedded audio systems.

This section is for **Master FAT Data Management Functions**.

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

*Never remove micro SD card during any FAT operation as data could be corrupted and damaged.

Briefly summary of Functions/Commands in this section:

- Mount SD Card - 0x46, 0x58 hex 'F', 'X' ascii
- List Directories - 0x46, 0x4C hex 'F', 'L' ascii
- List Files - 0x46, 0x4C hex 'F', 'L' ascii
- Get Dir Item Name - 0x46, 0x47 hex 'F', 'G' ascii
- Get File Item Name - 0x46, 0x47 hex 'F', 'G' ascii
- Open Directory - 0x46, 0x44 hex 'F', 'D' ascii
- Get Current Path - 0x46, 0x48 hex 'F', 'H' ascii
- New Directory - 0x46, 0x4E hex 'F', 'N' ascii
- New File - 0x46, 0x4E hex 'F', 'N' ascii
- Erase Directory - 0x46, 0x45 hex 'F', 'E' ascii
- Erase File - 0x46, 0x45 hex 'F', 'E' ascii
- Get File Size - 0x46, 0x49 hex 'F', 'I' ascii
- Get FAT Attribute - 0x46, 0x49 hex 'F', 'I' ascii
- Get Date & Time - 0x46, 0x54 hex 'F', 'T' ascii
- Set Date & Time - 0x46, 0x54 hex 'F', 'T' ascii
- Dir Rename/Move - 0x46, 0x4D hex 'F', 'M' ascii
- File Rename/Move - 0x46, 0x4D hex 'F', 'M' ascii
- Get Free & Total Space - 0x46, 0x46 hex 'F', 'F' ascii

FAT ACK/NAK List:

The next list of bytes, are the possible FAT access execution ACKs/NAKs 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 / not inserted.
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	No more files can be opened.
0x12 (hex)	TOO MANY OPEN FILES	No more files can be opened.
0x13 (hex)	INVALID PARAMETER	Given parameters are invalid.

8.1 Mount SD Card - 0x46, 0x58hex - 'F', 'X' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x58 (hex), X (ascii). *Mount SD Card. 3.- 0x00 (hex) for Unmount or 0x01 (hex) for Mount.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command mounts/unmounts a FAT /FAT32 file access system on the inserted microSD card. This command is useful If the user needs to safely remove and insert a different microSD card in the running application.</p> <p>*This command is automatically performed inside SmartWAV 2 with a parameter "Mount" just after the "Initialize System" command is ACKnowledged, if a microSD card is inserted, then FAT file system will be ready to use.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,58,00- Unmount SDcard FAT file system.</p> <p>Received data: -00,4F- OK, OK. (microSD card can be safely removed).</p> <p>*All data is in hex.</p>

8.2 List Directories - 0x46, 0x4Chex - 'F', 'L' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4C (hex), L (ascii). *List/Enumerate. 3.- 0x44 (hex), D (ascii). *Directories.
Responses (device)	byte(s)
	1.- Number of Directories - High Byte. 2.- Number of Directories - Low Byte. 3.- 0xFF - FAT ACK/NAK List Byte Reply 4.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The List Directories command enumerates the directories inside the current selected/working directory path.</p> <p>This command is useful to be called each time that the current directory path is modified ("Open Directory" command), in order to know the available directories inside the new path.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,4C,44- Retrieve count of directories inside the current directory path.</p> <p>Received data: -00,0A,00,4F- Found 10 directories in the current directory path, OK, OK.</p> <p>*All data is in hex.</p>

8.3 List Files - 0x46, 0x4C hex - 'F', 'L' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4C (hex), L (ascii). *List/Enumerate. 3.- 0x46 (hex), F (ascii). *Files.
Responses (device)	byte(s)
	1.- Number of Files - High Byte. 2.- Number of Files - Low Byte. 3.- 0xFF - FAT ACK/NAK List Byte Reply 4.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The List Files command enumerates the files inside the current selected/working directory path.</p> <p>This command is useful to be called each time that the current directory path is modified ("Open Directory" command), in order to know the available files inside the new path.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,4C,46- Retrieve count of files inside the current directory path.</p> <p>Received data: -00,7E,00,4F- Found 126 files in the current directory path, OK, OK.</p> <p>*All data is in hex.</p>

8.4 Get Dir Item Name - 0x46, 0x47hex - 'F', 'G' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x47 (hex), G (ascii). *Get Name. 3.- 0x44 (hex), D (ascii). *Directory. 4.- Item Number - High Byte. 5.- Item Number - Low Byte.
Responses (device)	byte(s)
	1 to n .- Directory Item Number Name. n +1 .- 0x00 (hex), NULL ascii. n +2 .- 0xFF - FAT ACK/NAK List Byte Reply n +3 .- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command retrieves the Directory Name of the received Item Number inside the current directory path. This command is designed to work in conjunction with the "List Directories" command.</p> <p>By using the "List Directories" command combined with this "Get Dir Item Name" command, the user can have a detailed list with names of all the directories inside the current selected/working directory path.</p> <p>*The Item Number parameter in this command must always be less than the previously found number of directories with the "List Directories" command. As an example: when the "List Directories" command is called, and it returns 5 directories, then the "Get Dir Item Name" could only ask for the directory name of the items 0 to 4.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,47,44,00,02- Get dir name of item# 2.

Received data:

-61,62,63,00,00,4F- Dir name of item# 2 is "abc", OK, OK.

*All data is in hex.

8.5 Get File Item Name - 0x46, 0x47hex - 'F', 'G' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x47 (hex), G (ascii). *Get Name. 3.- 0x46 (hex), F (ascii). *File. 4.- Item Number - High Byte. 5.- Item Number - Low Byte.
Responses (device)	byte(s)
	1 to n .- File Item Number Name. n +1 .- 0x00 (hex), NULL ascii. n +2 .- 0xFF - FAT ACK/NAK List Byte Reply n +3 .- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command retrieves the File Name of the received Item Number inside the current directory path. This command is designed to work in conjunction with the "List Files" command.</p> <p>By using the "List Files" command combined with this "Get File Item Name" command, the user can have a detailed list with names of all the directories inside the current selected/working directory path.</p> <p>*The Item Number parameter in this command must always be less than the previously found number of files with the "List Files" command. As an example: when the "List Files" command is called, and it returns 10 files, then the "Get File Item Name" could only ask for the file name of the items 0 to 9.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,47,46,00,05- Get file name of item# 5.

Received data:

-78,79,7A,2E,74,78,74,00,00,4F- File name of item# 5 is "xyz.txt", OK, OK.

*All data is in hex.

8.6 Open Directory - 0x46, 0x44hex - 'F', 'D' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x44 (hex), D (ascii). *Open Directory. 3 to n .- Directory Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command Opens a Directory/Folder that is inside the current directory path.</p> <p>After "Open Directory" command is executed and succeeds (ACK), the current directory path name will add a slash '/' plus the new requested to open directory name.</p> <p>To go inside a directory, call this command with the Directory Name as parameter.</p> <p>To go outside a directory or to parent directory, call this command with the ".." as Directory Name parameter.</p> <p>*Once a directory is opened or closed, it is recommended to call the command "Get Current Path" to get/know the new current directory path.</p>

Examples (sent and received commands)

Example 1:

**Current directory path is "/" (root path).*

Sent data:

-46,44,30,31,32,33,00- Open a directory called "0123".

Received data:

-00,4F- OK, OK.

**Current directory path is now "/0123".*

Example 2:

**Current directory path is "/ABC".*

Sent data:

-46,44,30,31,32,33,00- Open a directory called "0123".

Received data:

-00,4F- OK, OK.

**Current directory path is now "/ABC/0123".*

Example 3:

**Current directory path is "/ABC".*

Sent data:

-46,44,2E,2E,00- Go to parent directory: ".." goes up one level.

Received data:

-00,4F- OK, OK.

**Current directory path is now "/" (root path).*

Example 4:

**Current directory path is "/ABC/0123".*

Sent data:

-46,44,2E,2E,00- Go to parent directory: ".." goes up one level.

Received data:

-00,4F- OK, OK.

**Current directory path is now "/ABC".*

Example 5:

**Current directory path is "/ABC/0123".*

Sent data:

-46,44,2F,00- Go fast to root directory "/".

Received data:

-00,4F- OK, OK.

**Current directory path is now "/" (root path).*

Example 6:

**Current directory path is "/" (root path).*

Sent data:

-46,44,2F,41,42,43,2F,30,31,32,33,00- Open
fast nested directory "/ABC/0123".

Received data:

-00,4F- OK, OK.

**Current directory path is now "/ABC/0123".*

**All data is in hex.*

8.7 Get Current Path - 0x46, 0x48hex - 'F', 'H' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x48 (hex), H (ascii). *Get Current Path.
Responses (device)	byte(s)
	1 to n .- Directory Path Name. n +1 .- 0x00 (hex), NULL ascii. n +2 .- 0xFF - FAT ACK/NAK List Byte Reply n +3 .- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command returns the current directory path name.</p> <p>The path name is returned in the following convention: starting with "/" and ending with the last character of the current directory/folder name plus the 0x00(hex) NULL character:</p> <p>"/(NULL)" -for root path "/dir1/dir2/dir3(NULL)" -for nested path</p> <p>It is recommended to call this command after each "Open Directory" command, to verify/obtain the new current directory path.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,48- Get current directory path name.</p> <p>Received data: -2F,41,42,43,00,00,4F- Current directory path name is "/ABC", OK, OK.</p> <p>*All data is in hex.</p>

8.8 New Directory - 0x46, 0x4Ehex - 'F', 'N' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4E (hex), N (ascii). *New. 3.- 0x44 (hex), D (ascii). *Directory. 4 to n .- New Directory Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command creates a New Directory inside the current directory path based on the received directory name parameter.</p> <p>After a new directory is created, its contents are empty, to add files, first open the directory with the "Open Directory" command and then create files with the "New File" command.</p> <p>The directory name can be up to 500 characters, only numbers and letters are recommended to name the directory, some special characters are not allowed and may not work.</p> <p>This Command fails with EXIST(0x08) if the directory already exists.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,4E,44,41,42,43,00- Create new directory "ABC" inside current directory path.

Received data:

-00,4F- OK, OK.

Example 2:

Sent data:

-46,4E,44,31,32,33,00- Create new directory "123" inside current directory path.

Received data:

-00,4F- OK, OK.

*All data is in hex.

8.9 New File - 0x46, 0x4Ehex - 'F', 'N' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4E (hex), N (ascii). *New. 3.- 0x46 (hex), F (ascii). *File. 4 to n .- New File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command creates a New File inside the current directory path based on the received file name parameter.</p> <p>After a new file is created, its contents are empty (0 bytes in size), to append data to the file, first open the file with the "Open File" command and then write data with the "Write File" command.</p> <p>The file name can be up to 500 characters including the extension(".xxx"), only numbers and letters are recommended to name the file, some special characters are not allowed and may not work.</p> <p>This Command fails with EXIST(0x08) if the file already exists.</p> <p>*The ".xxx" extension may be sent within the file name; if not sent, a valid file but without extension will be created.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,4E,46,31,32,33,2E,74,78,74,00- Create new file "123.txt" inside current directory path.

Received data:

-00,4F- OK, OK.

Example 2:

Sent data:

-46,4E,46,41,42,43,2E,64,6F,63,00- Create new file "ABC.doc" inside current directory path.

Received data:

-00,4F- OK, OK.

*All data is in hex.

8.10 Erase Directory - 0x46, 0x45hex - 'F', 'E' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x45 (hex), E (ascii). *Erase Directory. 3.- 0x4F (hex), O (ascii). *Un-Lock Byte. 4 to n .- Directory Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command Erases an existing directory inside the current directory path. The security Un-Lock byte is a simple double security byte that avoids unwanted delete operations.</p> <p>When deleting a directory, it must be empty or this command will fail.</p> <p>If the directory to erase doesn't exist, the command will fail with INVALID NAME(0x06).</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,45,4F,72,6F,63,6B,00- Erase the empty directory named "rock" inside current directory path.</p> <p>Received data: -00,4F- OK, OK.</p> <p>*All data is in hex.</p>

8.11 Erase File - 0x46, 0x45hex - 'F', 'E' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x45 (hex), E (ascii). *Erase File. 3.- 0x4F (hex), O (ascii). *Un-Lock Byte. 4 to n .- File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command Erases an existing file inside the current directory path. The received file name must always include the .xxx extension of the file. The security Un-Lock byte is a simple double security byte that avoids unwanted delete operations.</p> <p>When deleting a File, it must be closed and not allocated / mounted under any workspace# to avoid data corruption.</p> <p>If the file to erase doesn't exist, the command will fail with INVALID NAME(0x06).</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,45,4F,31,32,33,2E,74,78,74,00- Erase the file named "123.txt" inside current directory path.</p> <p>Received data: -00,4F- OK, OK.</p> <p>*All data is in hex.</p>

8.12 Get File Size - 0x46, 0x49hex - 'F', 'I' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x49 (hex), I (ascii). *Get Info. 3.- 0x53 (hex), S (ascii). *File Size. 4 to n .- File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- File Size - High Byte. 2.- File Size - Medium High Byte. 3.- File Size - Medium Low Byte. 4.- File Size - Low Byte. 5.- 0xFF - FAT ACK/NAK List Byte Reply 6.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	The Get File Size command simply asks for the file size in bytes of the received file name inside the current directory path .
Examples (sent and received commands)	Example 1: Sent data: -46,49,53,31,32,33,2E,74,78,74,00- Get file size of "123.txt". Received data: -00,00,15,DA,00,4F- Size is 5594 bytes, OK, OK. *All data is in hex.

8.13 Get FAT Attribute - 0x46, 0x49hex - 'F', 'I' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x49 (hex), I (ascii). *Get Info. 3.- 0x46 (hex), F (ascii). *FAT Attribute. 4 to n .- Directory/File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - FAT Attribute (see description). 2.- 0xXX - FAT ACK/NAK List Byte Reply 3.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Get FAT Attribute Info command request the FAT information about the received directory or file name inside the current directory path.</p> <p>The possible FAT Attributes can be one or a logical "OR" combination of the next bytes:</p> <ul style="list-style-type: none"> 0x01 (hex)- Read Only 0x02 (hex)- Hidden 0x04 (hex)- System 0x08 (hex)- Volume Label 0x10 (hex)- Directory 0x20 (hex)- Archive
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data:</p> <p>-46,49,46,31,32,33,2E,74,78,74,00- Get FAT attribute of file named "123.txt".</p> <p>Received data:</p> <p>-23,00,4F- Attribute is 0x23(hex): logical OR of 0x20 (Archive), 0x01 (Read only) and 0x02 (Hidden File), so the file is a Read-only Hidden Archive, OK, OK. *All data is in hex.</p>

8.14 Get Date & Time - 0x46, 0x54hex - 'F', 'T' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x54 (hex), T (ascii). *Date & Time. 3.- 0x47 (hex), G(ascii). *Get. 4 to n .- Directory/File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - Hours. 2.- 0xXX - Minutes. 3.- 0xXX - Seconds. 4.- 0xXX - Day. 5.- 0xXX - Month. 6.- 0xXX - Year - High Byte. 7.- 0xXX - Year - Low Byte. 8.- 0xXX - FAT ACK/NAK List Byte Reply. 9.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Get Date & Time command request the last modified Date and Time timestamp of the received directory or file name inside the current directory path.</p> <p>*Note that the "New Directory", "New File" and "Write File" commands update the timestamp of the created / modified directory or file. As the SmartWAV 2 does not integrates a RTC clock, the updated timestamp is generic when using those commands.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,54,47,31,32,33,2E,74,78,74,00- Get date and time of file named "123.txt".

Received data:

-11,1E,14,13,03,07,DD,00,4F- Last modified date is "19 March 2013" and time "17:30:20", OK, OK.

*All data is in hex.

8.15 Set Date & Time - 0x46, 0x54hex - 'F', 'T' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x54 (hex), T (ascii). *Date & Time. 3.- 0x53 (hex), S(ascii). *Set. 4.- 0xFF - Hours. 5.- 0xFF - Minutes. 6.- 0xFF - Seconds. 7.- 0xFF - Day. 8.- 0xFF - Month. 9.- 0xFF - Year - High Byte. 10.- 0xFF - Year - Low Byte. 11 to n .- Directory/File Name. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Set Date & Time command modifies the last modified Date and Time timestamp of the received directory or file name inside the current directory path.</p> <p>*Note that the "New Directory", "New File" and "Write File" commands update the timestamp of the created / modified directory or file. As the SmartWAV 2 does not integrates a RTC clock, the updated timestamp is generic when using those commands.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,54,53,11,1E,14,13,03,07,DD,31,32,33,2E,74,78,74,00- Set last modified date "19 March 2013" and time "17:30:20" to file named "123.txt".

Received data:

-00,4F- OK, OK.

Example 2:

Sent data:

-46,54,53,11,1E,14,13,03,07,DD,46,6F,6C,64,65,72,00- Set last modified date "19 March 2013" and time "17:30:20" to directory named "Folder".

Received data:

-00,4F- OK, OK.

*All data is in hex.

8.16 Dir Rename/Move - 0x46, 0x4Dhex - 'F, 'M' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4D (hex), M (ascii). *Rename/Move. 3 to n .- Old Directory Name/Path. n +1 .- 0x00 (hex), NULL ascii. n +2 to m .- New Directory Name/Path. m +1.- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command can rename or move a Directory with the received names, rename and move operations can be achieved at the same command call.</p> <p>If the directory to rename/move doesn't exist, the command will fail with NO FILE(0x04).</p>
Examples (sent and received commands)	<p>Example 1: Rename Only Directory.</p> <p>Sent data: -46,4D,53,6F,6E,67,73,00,4D,75,73,69,63,00- Rename the named "Songs" directory to "Music" directory name.</p> <p>Received data: -00,4F- OK, OK.</p>

Example 2: Move Only Directory.

Sent data:

-46,4D,2F,44,69,72,31,2F,44,69,72,32,2F,44,69
72,33,00,2F,44,69,72,33,00- Move directory
named "Dir3" from "/Dir1/Dir2/Dir3" path to
"/Dir3" root path.

Received data:

-00,4F- OK, OK.

Example 3: Move and Rename Directory.

Sent data:

-46,4D,2F,44,69,72,31,2F,44,69,72,32,2F,44,69
72,33,00,2F,46,6F,6C,00- Move and rename
directory named "Dir3" from
"/Dir1/Dir2/Dir3" path to "/Fol" root path with
new name "Fol".

Received data:

-00,4F- OK, OK.

*All data is in hex.

8.17 File Rename/Move - 0x46, 0x4Dhex - 'F', 'M' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4D (hex), M (ascii). *Rename/Move. 3 to n .- Old File Name/Path. n +1 .- 0x00 (hex), NULL ascii. n +2 to m .- New File Name/Path. m +1.- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xXX - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command can rename or move a File with the received names, rename and move operations can be achieved at the same command call.</p> <p>If the file to rename/move doesn't exist, the command will fail with NO FILE(0x04).</p>
Examples (sent and received commands)	<p>Example 1: Rename Only File.</p> <p>Sent data: -46,4D,61,62,63,2E,74,78,74,00,31,32,33,2E,64 6F,63,00- Rename the named "abc.txt" file to "123.doc" file name.</p> <p>Received data: -00,4F- OK, OK.</p>

Example 2: Move Only File.

Sent data:

-46,4D,2F,44,69,72,31,2F,44,69,72,32,2F,61,62
63,2E,74,78,74,00,2F,44,69,72,31,2F,61,62,63
2E,74,78,74,00- Move file named "abc.txt"
from "/Dir1/Dir2/abc.txt" path to
"/Dir1/abc.txt" directory path.

Received data:

-00,4F- OK, OK.

Example 3: Move and Rename File.

Sent data:

-46,4D,2F,44,69,72,31,2F,44,69,72,32,2F,61,62
63,2E,74,78,74,00,2F,44,69,72,31,2F,31,32,33
2E,64,6F,63,00- Move and rename file
named "abc.txt" from "/Dir1/Dir2/abc.txt"
path to "/Dir1/123.doc" directory path with
new name "123.doc".

Received data:

-00,4F- OK, OK.

*All data is in hex.

8.18 Get Free & Total Space - 0x46, 0x46hex - 'F', 'F' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x46 (hex), F (ascii). *Get Free & Total.
Responses (device)	byte(s)
	1.- Free Space - High Byte. 2.- Free Space - Medium High Byte. 3.- Free Space - Medium Low Byte. 4.- Free Space - Low Byte. 5.- Total Space - High Byte. 6.- Total Space - Medium High Byte. 7.- Total Space - Medium Low Byte. 8.- Total Space - Low Byte. 9.- 0xFF - FAT ACK/NAK List Byte Reply. 10.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command requests the available Free Space and Total Space of the mounted microSD card flash memory, the sizes are returned in Kbytes units.</p> <p>To calculate the Used Space, the following formula can be applied: Used Space = Total Space – Free Space</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,46- Get free and total space.</p> <p>Received data: -00,17,47,90,00,3A,F0,00,00,4F- Current free space is 1,525,648(dec) and TOTAL Space: 3,862,528Kb(dec), OK, OK.</p> <p>*All data is in hex.</p>

9- Workspace# FAT/FAT32 Data Management Functions, Data Logger -Serial Commands:

SmartWAV 2 uses the 14 audio channels also as workspaces# for those next workspace# FAT commands, that means that up to 14 workspaces (0 to 13) could be used at the same time for file management (the physical 14 spaces inside SmartWAV 2 could be used as workspace or audio channels but not both at the same time).

As an example: the command "Open File" is used to assign a file to a workspace#, only one file/object can be allocated at the same time in the same workspace, the command "Close File" frees the workspace.

This section is for **Workspace# FAT Data Management Functions**, all of this section commands always require a file allocated/mounted on a workspace# to work.

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

*Never remove micro SD card during any FAT operation as data could be corrupted and damaged.

Briefly summary of Functions/Commands in this section:

- | | |
|----------------------|---------------------------------|
| - Open File | - 0x46, 0x4F hex 'F', 'O' ascii |
| - Read File | - 0x46, 0x52 hex 'F', 'R' ascii |
| - Write File | - 0x46, 0x57 hex 'F', 'W' ascii |
| - Get File Pointer | - 0x46, 0x50 hex 'F', 'P' ascii |
| - Set File Pointer | - 0x46, 0x50 hex 'F', 'P' ascii |
| - Save File | - 0x46, 0x53 hex 'F', 'S' ascii |
| - Truncate File | - 0x46, 0x56 hex 'F', 'V' ascii |
| - Test Error in File | - 0x46, 0x51 hex 'F', 'Q' ascii |
| - Test End of File | - 0x46, 0x51 hex 'F', 'Q' ascii |
| - Close File | - 0x46, 0x43 hex 'F', 'C' ascii |

FAT ACK/NAK List:

The next list of bytes, are the possible FAT access execution ACKs/NAKs 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 / not inserted.
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	No more files can be opened.
0x12 (hex)	TOO MANY OPEN FILES	No more files can be opened.
0x13 (hex)	INVALID PARAMETER	Given parameters are invalid.

9.1 Open File - 0x46, 0x4Fhex - 'F', 'O' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x4F (hex), O (ascii). *Open File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- Open Mode: 0x01 (hex)- Read Only 0x02 (hex)- Write Only 0x03 (hex)- Read +Write 5 to n .- File Name Including Extension. n+1 .- 0x00 (hex), NULL ascii.
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command opens / allocates a file that is inside the current directory path on the received workspace#, the file access depends on the "Open Mode" parameter:</p> <ul style="list-style-type: none"> - 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 and write access to the object, file data can be write or read.

	<p>After open file command is executed and succeeds(ACK), the file object workspace is then valid, now the workspace can be used for subsequent read/write operations.</p> <p>Only one file can be open at the same time in the same workspace#, always be sure close a workspace file object before opening a new one on it.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,4F,02,03,74,65,73,74,2E,74,78,74,00- Open file "test.txt" in workspace# 2 with read+write access(0x03).</p> <p>Received data: -00,4F- OK, OK. File correctly open and allocated in workspace# 2 for read+write access.</p> <p>Example 2:</p> <p>Sent data: -46,4F,0A,01,74,65,73,74,2E,74,78,74,00- Open file "test.txt" in workspace# 10 with read only access(0x01).</p> <p>Received data: -00,4F- OK, OK. File correctly open and allocated in workspace# 10 for read only access.</p> <p>*All data is in hex.</p>

9.2 Read File - 0x46, 0x52hex - 'F', 'R' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x52 (hex), R (ascii). *Read File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- Number of Bytes to Read - High Byte. 5.- Number of Bytes to Read - Low Byte. <i>*Number of bytes to read parameter could be any value from 1 up to 512 bytes per command call.</i>
Responses (device)	byte(s)
	1 to Bytes to Read .- File Requested to Read Data Bytes. Bytes to Read + 1.- Successfully Read Bytes - High Byte. Bytes to Read + 2.- Successfully Read Bytes - Low Byte. Bytes to Read + 3.- 0xXX - FAT ACK/NAK List Byte Reply. Bytes to Read + 4.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Read File command reads the requested number of Bytes to Read data from the file open / allocated in the received workspace#, the file must be open with "Read Only" or "Read+Write" access mode for this command to succeed.</p> <p>The returned parameter Successfully Read Bytes must always be considered; the processor always replies (In bytes) the requested number of Bytes to Read, however 2 cases could happen:</p>

1: All of the requested **Bytes to Read** have been successfully read from the file data, then the next parameters are equal:

Bytes to Read == Successfully Read Bytes

**All of the received data bytes are valid file data.*

2: Not all of the requested **Bytes to Read** could be successfully read from the file data, the cause may be an error or the end of file data has been reached, in this case the processor fills/complete with 0x00 (hex) NULL bytes until the **Bytes to Read** parameter is completed. In this scenario the next parameters are not equal:

Bytes to Read > Successfully Read Bytes

From the received data bytes, the **Successfully Read Bytes parameter indicates the valid data, the complementary received data are dumb filled 0x00 (hex) NULL bytes.*

The returned **Successfully Read bytes** parameter should always be checked to ensure that the complete / part of the received data bytes are valid.

The current **File Pointer** value will be increased by the number of **Successfully Read bytes** from the last pointer position.

A file must be already opened/allocated in the selected workspace for this command to work.

Examples (sent and received commands)

Example 1:

Sent data:

-46,52,08,00,64- Request to read 100 bytes from the file in workspace# 8.

Received data:

-XX...XX,00,64,00,4F- Received 100 bytes, then Successfully Read Bytes(100), OK, OK.

This is a case:1

Bytes to Read == Successfully Read Bytes

**All of the received data bytes are valid file data.*

Example 2:

Sent data:

-46,52,04,02,00- Request to read 512 bytes from the file in workspace# 4.

Received data:

-XX...XX,00,64,00,4F- Received 512 bytes, then Successfully Read Bytes(100), OK, OK.

This is a case 2:

Bytes to Read > Successfully Read Bytes

From the received data bytes, the **Successfully Read Bytes parameter (100 bytes) indicates the valid data, and the complementary received data (412 bytes) are dumb filled 0x00 (hex) NULL bytes.*

**All data is in hex.*

9.3 Write File - 0x46, 0x57hex - 'F', 'W' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x57 (hex), W (ascii). *Write File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- Number of Bytes to Write - High Byte. 5.- Number of Bytes to Write - Low Byte. 6 to Bytes to Write .- Data Bytes to Write. <i>*Number of bytes to write parameter could be any value from 1 up to 512 bytes per command call.</i>
Responses (device)	byte(s)
	1.- Successfully Written Bytes - High Byte. 2.- Successfully Written Bytes - Low Byte. 3.- 0xFF - FAT ACK/NAK List Byte Reply. 4.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Write File command writes the requested number of Bytes to Write data to the file open / allocated in the received workspace#, the file must be open with "Write Only" or "Read+Write" access mode for this command to succeed.</p> <p>The returned parameter Successfully Written Bytes must always be considered as those are actually the correctly written bytes to the file data.</p> <p>The current File Pointer value will be increased by the number of Successfully Written bytes from the last pointer position.</p>

	<p>A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,57,08,00,64,XX...XX- Request to write 100 sent bytes, to the file in workspace# 8.</p> <p>Received data: -00,64,00,4F- Successfully Written Bytes(100), OK, OK.</p> <p>Example 2:</p> <p>Sent data: -46,57,04,02,00,XX...XX- Request to write 512 sent bytes, to the file in workspace# 4.</p> <p>Received data: -02,00,00,4F- Successfully Written Bytes(512), OK, OK.</p> <p>*All data is in hex.</p>

9.4 Get File Pointer - 0x46, 0x50hex - 'F', 'P' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x50 (hex), P (ascii). *File Pointer. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- 0x47 (hex), G (ascii) Get Pointer.
Responses (device)	byte(s)
	1.- Pointer Position - High Byte. 2.- Pointer Position - Medium High Byte. 3.- Pointer Position - Medium Low Byte. 4.- Pointer Position - Low Byte. 5.- 0xFF - FAT ACK/NAK List Byte Reply. 6.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>When the Get Pointer command is called, it retrieves the current file pointer position in bytes of the file allocated in the received workspace#.</p> <p>The current file pointer position parameter is always computed from the file origin.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p> <p>*Note that the "Read File" or "Write File" commands calls automatically advance the current file pointer position depending on the successfully read or written bytes.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,50,08,47- Get current file pointer position of the file in workspace# 8.

Received data:

-00,00,00,64,00,4F- Current file pointer position of the file in workspace# 8 is byte:100, OK, OK.

Example 2:

Sent data:

-46,50,02,47- Get current file pointer position of the file in workspace# 2.

Received data:

-00,01,E2,40,00,4F- Current file pointer position of the file in workspace# 2 is byte: 123456, OK, OK.

*All data is in hex.

9.5 Set File Pointer - 0x46, 0x50hex - 'F', 'P' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x50 (hex), P (ascii). *File Pointer. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- 0x53 (hex), S (ascii) Set Pointer. 5.- Pointer Position - High Byte. 6.- Pointer Position - Medium High Byte. 7.- Pointer Position - Medium Low Byte. 8.- Pointer Position - Low Byte.
Responses (device)	byte(s)
	1.- 0xXX - FAT ACK/NAK List Byte Reply. 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>When the Set Pointer command is called, it moves the current file pointer position in bytes of the received workspace#. This command can also be used to increase the file size when writing data (cluster pre-allocation; data contained in the expanded area is undefined).</p> <p>The current file pointer position parameter is always computed from the file origin.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p> <p>*Note that the "Read File" or "Write File" commands calls automatically advance the current file pointer position depending on the successfully read or written bytes.</p>

Examples (sent and received commands)

Example 1:

Sent data:

-46,50,02,53,00,00,02,00- Set current file pointer position to byte:512 of the file in workspace# 2.

Received data:

-00,4F- Current file pointer position of the file in workspace# 2 is now byte:512, OK, OK.

Example 2:

Sent data:

-46,50,05,53,00,01,E2,40- Set current file pointer position to byte:123456 of the file in workspace# 5.

Received data:

-00,4F- Current file pointer position of the file in workspace# 5 is now byte: 123456, OK, OK.

*All data is in hex.

9.6 Save File - 0x46, 0x53hex - 'F', 'S' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x53 (hex), S (ascii). *Save File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii).
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>This command sync/saves any changes of the file in the received workspace#. The changes to save are normally written or truncated data bytes to the file, this command is suitable for applications that require to have opened files for a long time in "Write Only" or "Read+Write" modes, such as long-time data loggers.</p> <p>Performing sync / save file periodically or immediately after a "Write File" command can minimize the risk of data loss due to a sudden blackout or an unintentional disk removal. A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,53,02- Save changes to file in workspace# 2.</p> <p>Received data: -00,4F- OK, OK. Changes made to file in workspace# 2 are saved.</p> <p>*All data is in hex.</p>

9.7 Truncate File - 0x46, 0x56hex - 'F', 'V' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x56 (hex), V (ascii). *Truncate File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii).
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Truncate File command cuts all data bytes up from the current File Pointer Position of the file in the received workspace#.</p> <p>Once this command is acknowledged, the current File Pointer Position will now be the last data byte of the file, therefore this command reduces the total file size.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,56,02- Truncate file data bytes / contents up from the current file pointer position of workspace# 2.</p> <p>Received data: -00,4F- OK, OK. File in workspace# 2 size was reduced to the current filer pointer position bytes.</p> <p>*All data is in hex.</p>

9.8 Test Error in File - 0x46, 0x51hex - 'F', 'Q' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x51 (hex), Q (ascii). *Test File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- 0x52 (hex), R (ascii). *Error in File.
Responses (device)	byte(s)
	1.- 0x00 (hex) for FALSE condition or 0x01 (hex) for TRUE condition. 2.- 0xFF - FAT ACK/NAK List Byte Reply 3.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Test Error File command checks for any error in a file allocated in the given workspace#. An error could be bad / corrupted data, write errors, etc. This command is helpful to check file sanity.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,51,02,52- Test file in workspace# 2 for an Error condition.</p> <p>Received data: -00,00,4F- OK, OK. Error in file is FALSE, the written bytes or any other procedure was correctly performed; file sanity is correct.</p> <p>*All data is in hex.</p>

9.9 Test End of File - 0x46, 0x51hex - 'F', 'Q' ascii

Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x51 (hex), Q (ascii). *Test File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii). 4.- 0x45 (hex), E (ascii). *End of File.
Responses (device)	byte(s)
	1.- 0x00 (hex) for FALSE condition or 0x01 (hex) for TRUE condition. 2.- 0xFF - FAT ACK/NAK List Byte Reply 3.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The test End of File command, checks for an end of file of the allocated file in the received workspace#, End of File means that the Current File Pointer is pointing at the very last data byte of the file. This command is helpful when reading to know if the file pointer has reached the last byte of the file and no more data can be read.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,51,02,45- Test file in workspace# 2 for a EOF condition.</p> <p>Received data: -01,00,4F- OK, OK. End of File is TRUE, then the current file pointer is pointing to the very last data byte of file.</p> <p>*All data is in hex.</p>

9.10 Close File - 0x46, 0x43hex - 'F', 'C' ascii

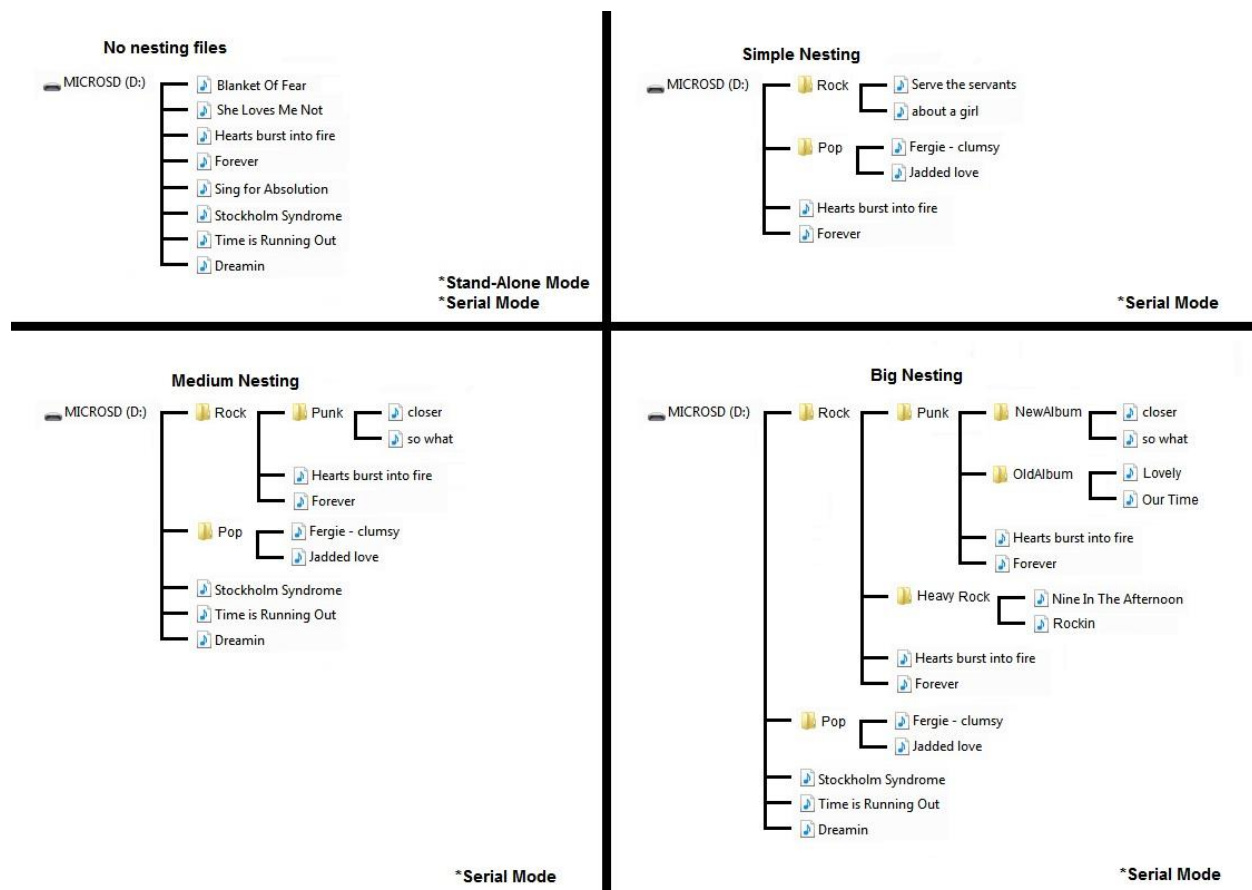
Commands (host)	byte(s)
	1.- 0x46 (hex), F (ascii). *FAT command. 2.- 0x43 (hex), C (ascii). *Close File. 3.- Workspace 0x00 to 0x0D (hex) 0-13 (ascii).
Responses (device)	byte(s)
	1.- 0xFF - FAT ACK/NAK List Byte Reply 2.- 0x4F (hex), O (ascii) - Success ACK or 0x46 (hex), F (ascii) - Fail NAK.
Description	<p>The Close File command, sync/save changes first, and then closes the file allocated on the received workspace#. If a modified file is not closed, the file data can be collapsed.</p> <p>Once this command is acknowledged, the file is unmounted/deallocated from the workspace#, this become free and no more workspace# FAT related commands can be performed.</p> <p>A file must be already opened/allocated in the selected workspace for this command to work.</p>
Examples (sent and received commands)	<p>Example 1:</p> <p>Sent data: -46,43,02- Save and close file allocated in workspace# 2.</p> <p>Received data: -00,4F- OK, OK. File in workspace# 2 is now closed and the workspace is free.</p> <p>*All data is in hex.</p>

10-MicroSD File/Folder Organization - Standard Serial Mode0:

The SmartWAV 2 is capable of managing and folders in this mode, so a complete library organized by artist/ album/ genre/ year/ etc. could be done inside the micro SD card. Also the processor could access nested folders for example: "D:/rock/punk/oldies/song.wav".

In Stand-Alone Modes, the SmartWAV 2 only recognize .wav files stored in the root path of the microSD card, all .wav files must be placed in this path.

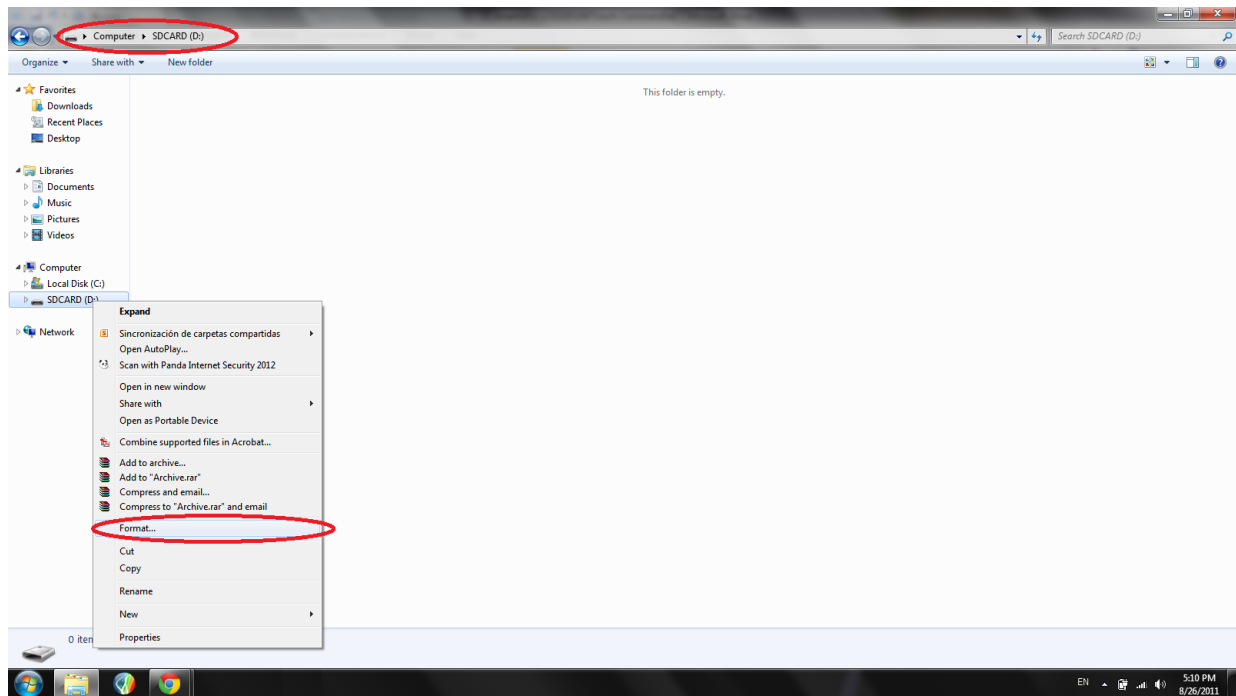
The next image gives some examples of files/folders organization/nesting that can be achieved and accessed, folder nesting is allowed:



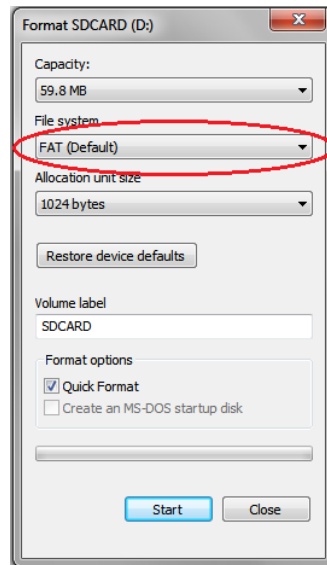
11- Formatting MicroSD Card:

It is recommended but not strictly necessarily to format the micro SD card for first use, in this section a format to new micro SD card to FAT format is explained.

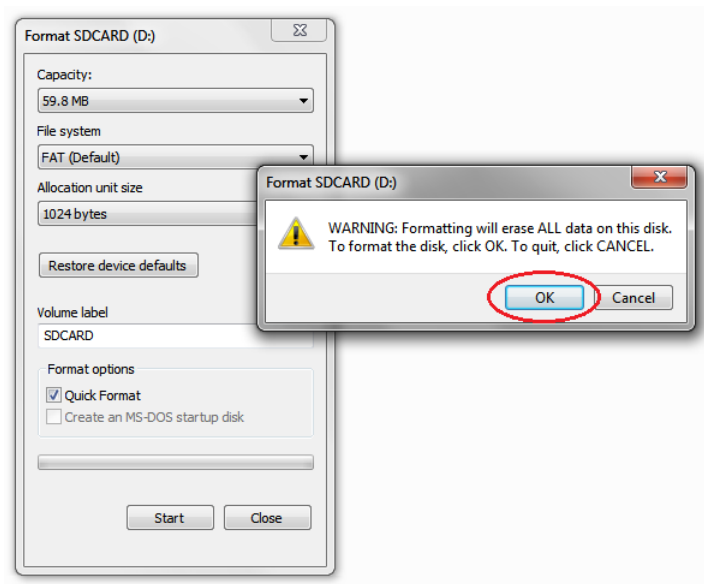
A.- Open a new windows explorer right click on the microSD card and a menu appears, select the “FORMAT...” and click on it. *(Note that formatting a micro SD card will erase all the contents of it).*



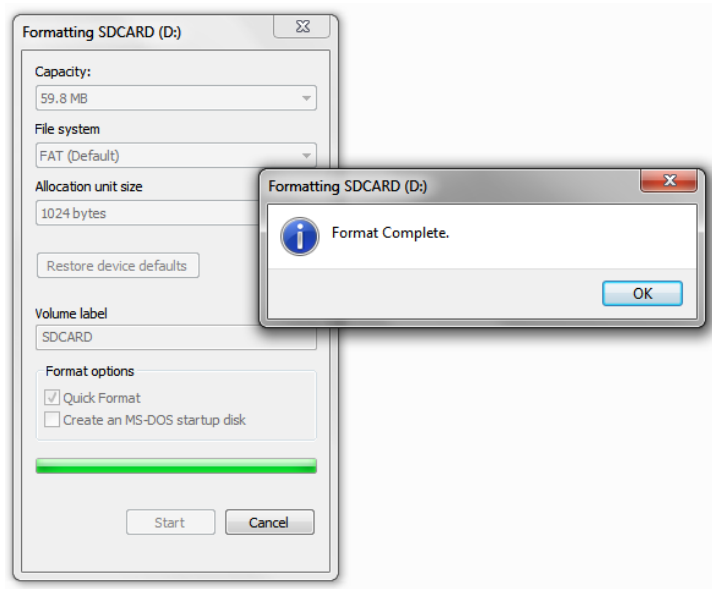
B.- A new window will pop-up, chose FAT/FAT32(default) on the File System menu, and click start.



C.- Click OK on the new window and wait to the PC to perform the format.



D.- Now the microSD card is ready to load tracks and songs!

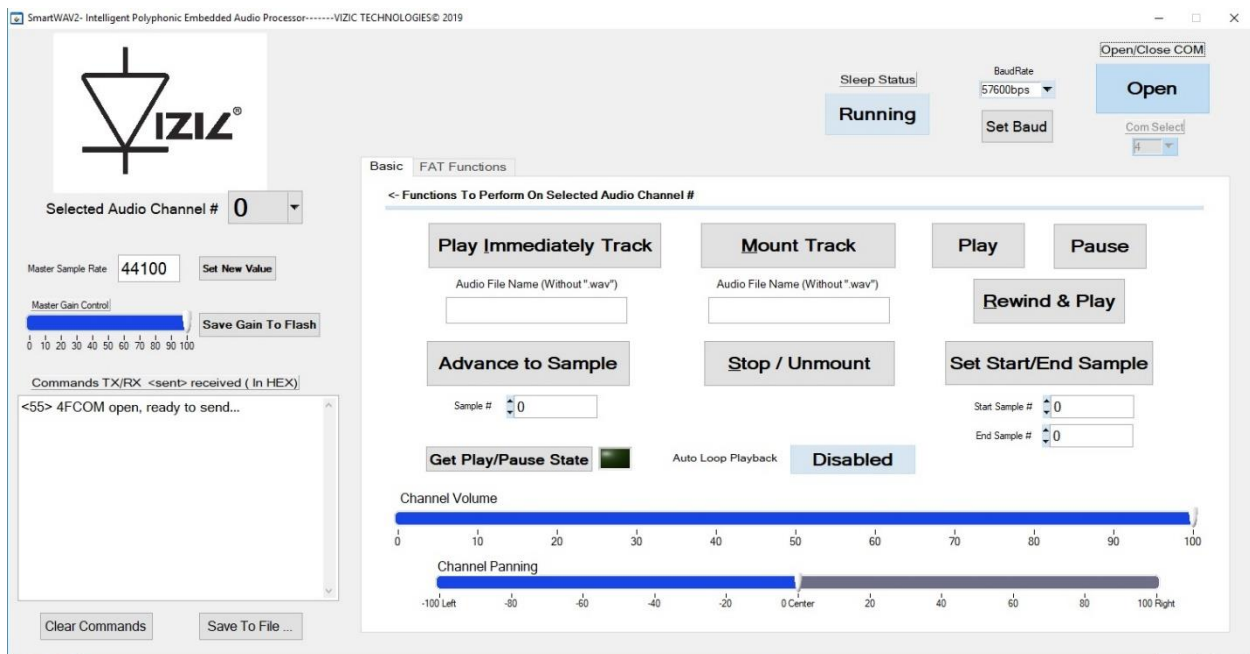


12- Development Software & Hardware Tools

In order to make easier the learning about how to communicate with the SmartWAV 2, free software could be downloaded and used in any PC. This software simulates almost all of the functions of the processor in Standard Serial Mode 0, this is achieved by connecting the hardware tool USB-UART SX Bridge to the SmartWAV 2 enabling real time audio processing, the required pins of Standard Serial Mode 0 are labeled with a white rectangle.

This software greatly reduces the time of learning the commands, and helps the user to understand how commands are created.

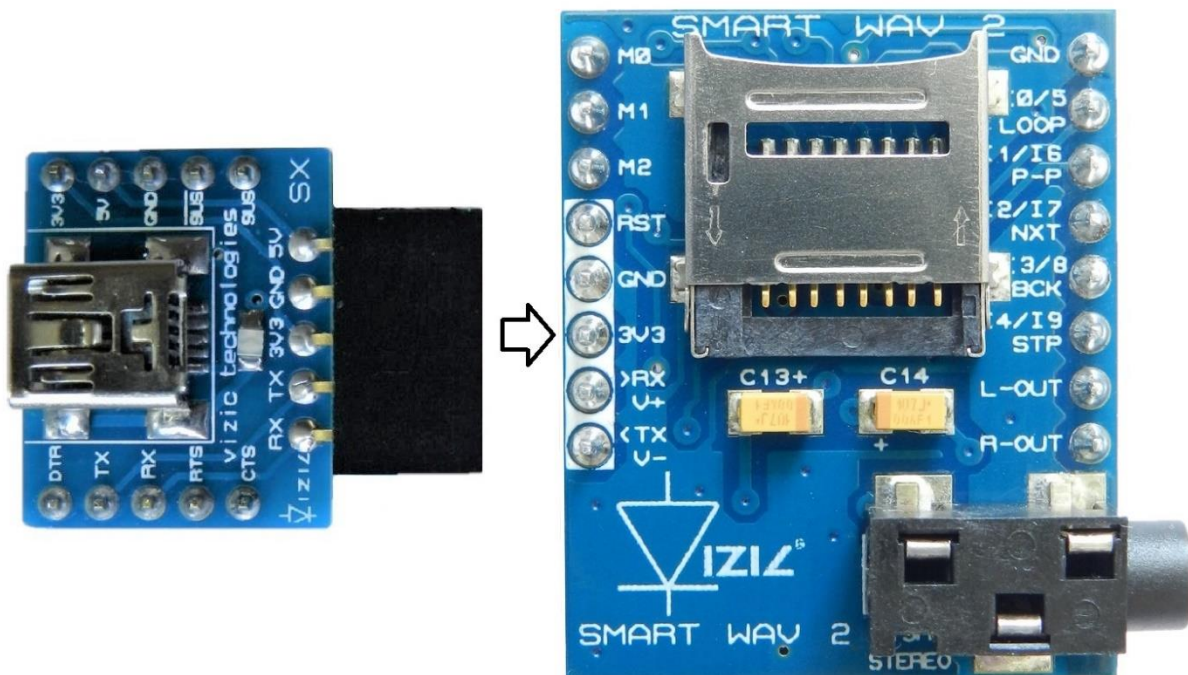
SmartWAV 2 PC Interface:



The USB-UART SX:



SmartWAV 2 connected to the USB-UART SX, the required pins of Standard Serial Mode 0 are labeled with a white rectangle:



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

13- 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.

14- 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.

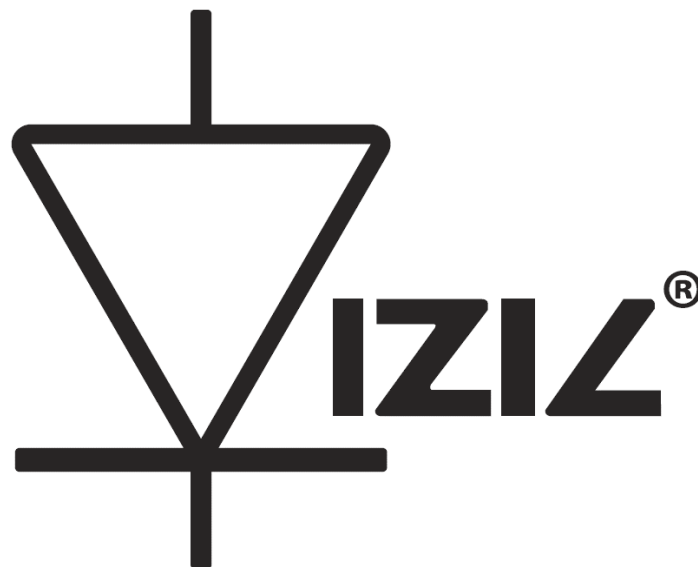
In no event shall Vizic be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by Vizic, 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.

THE DATASHEETS AND SOFTWARE ARE PROVIDED "AS IS." VIZIC EXPRESSLY DISCLAIM ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT.

IN NO EVENT SHALL VIZIC BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENCE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

VIZIC TECHNOLOGIES. COPYRIGHT 2020.



www.VIZICTECHNOLOGIES.COM