NSRT_mk4_Dev

Com Protocol

February 6 2025

Bruno Paillard

1 Introduction

The *NSRT_mk4_Dev* is a variant of the *NSRT_mk4* series that introduces an open virtual Com port communication protocol. That means that the instrument can be used on any platform that has a generic driver to support the CDC (Communication) USB class. Nowadays most platforms support that class, including Windows, Mac and Linux.

That instrument is targeted at developers. Because its communication protocol is open, developers can design their own application supporting the instrument.

In addition to the Com port, the *NSRT_mk4_Dev* can have an optional USB Audio interface to stream the actual Audio signal captured by the microphone.

2 COM Port Enumeration

When the instrument is enumerated by the host PC, one of the interfaces that it presents is a virtual COM port (a CDC-Class USB device). On Windows 10 and up the generic Windows COM port driver is automatically instantiated and bound to that interface. On Windows 7 and 8, even though Microsoft provides a generic driver, the user must manually load the driver when the device is connected to the PC for the first time. After the driver is loaded, a new COM port is shown in the list of devices connected to the PC.

3 COM Port Configuration

The COM port can be configured (bit rate, number of stop bits… etc.), either using the controls in Windows *Device Manager*, or in an application by using the appropriate API functions. However such settings have **no effect** on the actual communication. They are only exposed for compatibility. At the hardware level there is no physical serial line present, and the ultimate communication speed is only determined by the throughput of the USB link. That throughput is typically around 3 Mbps when there are no other devices on the USB bus.

4 Communication Structure

Exchanges between the host PC and the instrument always follow a Master-Slave model. The host initiates an exchange using a *Command Packet*. The host may also send data following that *Command Packet*. The instrument responds either with data, or with an Ack byte if no data is to be transmitted back to the host.

In all cases after sending a command, the host PC must not send another command before the instrument sends a response back. That response may be data or may be an Ack if no data is requested by the command.

5 Endianness

Unless otherwise noted, the endianness is Little-Endian.

6 Basic Types

The following basic types may be used in this protocol:

Type Name	Description	Endianness
U8	Single byte unsigned	N/A
U16	16-bit word unsigned	Little-Endian
U32	32-bit word unsigned	Little-Endian
U64	64-bit word unsigned	Little-Endian
I8	Single byte signed	N/A
I16	16-bit word signed	Little-Endian
I32	32-bit word signed	Little-Endian
I64	64-bit word signed	Little-Endian
Sgl	32-bit word in IEEE 754 floating point format	Little-Endian
Dbl	64-bit word in IEEE 754 floating point format	Little-Endian
String	Strings are concatenations of 8-bit ASCII characters, terminated by an end-of-text (0x00) byte.	N/A

Table 1

7 Packet Structure

7.1 Command Packet

The *Command Packet* is structured as follows:

Field	Size (bytes)	Function
Command	4	The command indicates the data transmitted or operation performed. The indicated direction of transmission is host-centric. Bit 31 of the command word indicates the direction of transfer: • 0: OUT (Host to Device) • 1: IN (Device to Host)
Address	4	The function of the address field varies with the command
Count	4	This field indicates the number of bytes to be transferred in the following data packet (either an IN or an OUT). How the bytes are interpreted is defined by the command. This number DOES NOT INCLUDE the command packet.

Table 2

7.2 Data Packet

Data Packets are simply a concatenation of bytes. The way the bytes are interpreted is a function of the command that precedes the *Data packet*.

7.3 Acknowledge

The *Ack* is a single byte with value 0x06. The *Ack* byte is only sent back to the host if the command is a *Write*, and therefore does not require a data response from the device. When the command is a *Read*, the actual data sent back to the host serves that purpose.

7.4 Commands

Command	Description	Address	Count	Data/Ack
0x80000010	*Read_Level* This command retrieves the current running level in dB. That is an exponentially averaged level, using the time constant and weighting function set for the instrument. That is not an LEQ.	The address field is not relevant for this command	4	Data: 32-bit IEEE-754 Float representing the Level in dB Ack: No
0x80000011	*Read_LEQ* This command retrieves the current running LEQ and starts the integration of a new LEQ. This way the next *Read_LEQ* command returns the LEQ calculated between the present time and the retrieval of the previous LEQ.	The address field is not relevant for this command	4	Data: 32-bit IEEE-754 Float representing the LEQ in dB Ack: No
0x80000012	*Read_Temperature* This command retrieves the temperature.	The address field is not relevant for this command	4	Data: 32-bit IEEE-754 Float representing the temperature in degC Ack: No
0x80000020	*Read_Weighting* This command returns the weighting curve that is currently selected	The address field is not relevant for this command	1	Data: 1 byte representing the weighting curve: 0: dB-C 1: dB-A 2: dB-Z Ack: No

0x00000020	*Write_Weighting* This command selects the weighting curve	The address field is not relevant for this command	1	Data: 1 byte representing the weighting curve: 0: dB-C 1: dB-A 2: dB-Z Ack: Yes
0x80000021	*Read_FS* This command reads the current sampling frequency	The address field is not relevant for this command	2	Data: U16 representing the sampling frequency in Hz: 32000: 32 kHz 48000: 48 kHz Ack: No
0x00000021	*Write_FS* This command sets the sampling frequency	The address field is not relevant for this command	2	Data: U16 representing the sampling frequency in Hz. There are only two choices: 32000: 32 kHz 48000: 48 kHz Ack: Yes
0x80000022	*Read_Tau* This command reads the current time constant	The address field is not relevant for this command	4	Data: 32-bit IEEE-754 Float representing the time constant in s. Ack: No
0x00000022	*Write_Tau* This command sets the time constant	The address field is not relevant for this command	4	Data: 32-bit IEEE-754 Float representing the time constant in s. Ack: Yes
0x80000031	*Read_Model* This command reads the instrument model	The address field is not relevant for this command	0-32	Data: ASCII string representing the Model.

				Size: Up to 32 bytes, including the termination byte Ack: No
0x80000032	*Read_SN* This reads the serial number of the instrument	The address field is not relevant for this command	0-32	Data: ASCII string representing the serial number of the instrument. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000033	*Read_FW_Rev* This command reads the firmware revision number.	The address field is not relevant for this command	0-32	Data: ASCII string representing the Firmware revision. Size: Up to 32 bytes, including the termination byte Ack: No
0x80000034	*Read_DOC* This command reads the date of last calibration.	The address field is not relevant for this command	8	Data: U64 number representing the UTC (Universal Time Code) of the date/time of last calibration. The UTC represents the number of seconds elapsed since Jan 1 1904. Ack: No
0x80000035	*Read_DOB* This command reads the date of birth of the instrument.	The address field is not relevant for this command	8	Data: U64 number representing the UTC (Universal Time Code) of the date/time of birth. The UTC represents the number of seconds elapsed since Jan 1 1904.

				Ack: No
0x80000036	*Read_User_ID* This command reads the User_ID field. That field can be written in persistent memory using the *Write_User_ID* command.	The address field is not relevant for this command	0-32	Data: ASCII string representing the User-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte Ack: No
0x00000036	*Write_User_ID* This command writes the User_ID field in persistent memory.	The address field is not relevant for this command	0-32	Data: ASCII string representing the user-ID, as defined by the user. Size: Up to 32 bytes, including the termination byte Ack: Yes
0x00000036	*Write AudioDebug Mode* This command sets or resets the Audio debug mode. When in debug mode the instrument outputs a perfect 1 kHz sine wave at an amplitude of 94 dB on its USB Audio interface. Note: This mode only affects the USB Audio interface. The rest of the instrument keeps outputting the levels measured by the microphone Note: This command is only supported in firmware V1.4 and up.	The address field is not relevant for this command	1	Data: 1 byte representing the Audio Debug mode: 0: Normal mode 1: Debug mode Ack: Yes

Table 3

8 Data Persistence

The following parameters are stored in Flash memory and are persistent:

- **User_ID:** The user-modifiable identifier for the instrument

- **Tau:** The time constant of the instrument. That time constant applies to the instantaneous level, but NOT to LEQs. LEQs are calculated using a rectangular averaging between two reads of the value.

- **FS:** The sampling frequency

- **Weighting:** The weighting function (A, C or Z)

The Flash memory that is used to contain these values can sustain approximately 10 000 write cycles over the lifetime of the instrument. Even though that is a large number, the instrument is not designed to sustain constantly changing the values in rapid succession. For instance, switching the weighting function back and forth in an attempt to read both A and C levels all the time will quickly exhaust the number of cycles guaranteed for that Flash memory.

Whenever *Tau*, *Fs* or *Weighting* are modified, the instrument's correction filters are reset and that creates a transient spike in the indicated levels. To read valid levels after changing one of these parameters, a delay of at least the largest of 1 second, or 10 times the value of *Tau* should be observed.