VBAN PROTOCOL

VB-Audio Network Protocol

SPECIFICATIONS

OFFICIAL WEBSITE

www.voicemeeter.com

www.vb-audio.com

Table of Content

INTRODUCTION:

VBAN Protocol has been designed for real-time transport of digital audio stream in IP-based network environments. This has been introduced by VB-Audio Software (Vincent Burel) in June 2015 as a new feature of the Application called Voicemeeter: The Virtual Audio Mixer for Windows (www.voicemeeter.com). It provides easy ways to send / receive audio to / from any computers on a local network.

DATA sub protocols have been implemented in 2017 to provide different solutions to transport serial data or text data.

Why VBAN Protocol?

VBAN protocol has been design as simple and as minimal as possible, to provide to users and developers, the simplest way to use / implement real time audio or data streaming on any local network. VBAN Protocol is public and free, it can be implemented by any developers or companies to make receiver or transmitter applications.

VBAN Capabilities:

VBAN Protocol is using UDP packets to send and receive audio stream(s) over IP-based network. The UDP packet is made by a 28 Bytes Header and various amounts of data behind.

VBAN first implementation (in Voicemeeter) supports native 16bits/24bits PCM audio format in any standard sample rate, for 1 to 8 channels stream.

VBAN Protocol Basis:

VBAN protocol is a true broadcast protocol and does not need any other management than the audio packets processing. The audio stream is defined by an IP Address (to or from) and a Name (16 ASCII CHAR max).

VBAN protocol does not need any special infra–structure and works with any switch or router. QOS is given by the capabilities of the network to route and deliver packet as fast as possible.

VBAN protocol includes no clock and no synchronization process. Like in the radio world, every sender is master and every receiver is slave. The slave is in charge to follow the incoming audio data rate….

VBAN as industrial solution:

VBAN protocol needs a simple UDP / IP stack and can be implemented on very small devices or DSP units easily.

Licensing:

VBAN Protocol base, described in this document, is free to use AS IS. Under this free license, the VBAN protocol base includes:

- The AUDIO protocol with PCM Audio formats.
- The SERIAL protocol (including M.I.D.I.).
- The TEXT protocol.
- The PING protocol for identification service.
- The RT-Packet protocol.

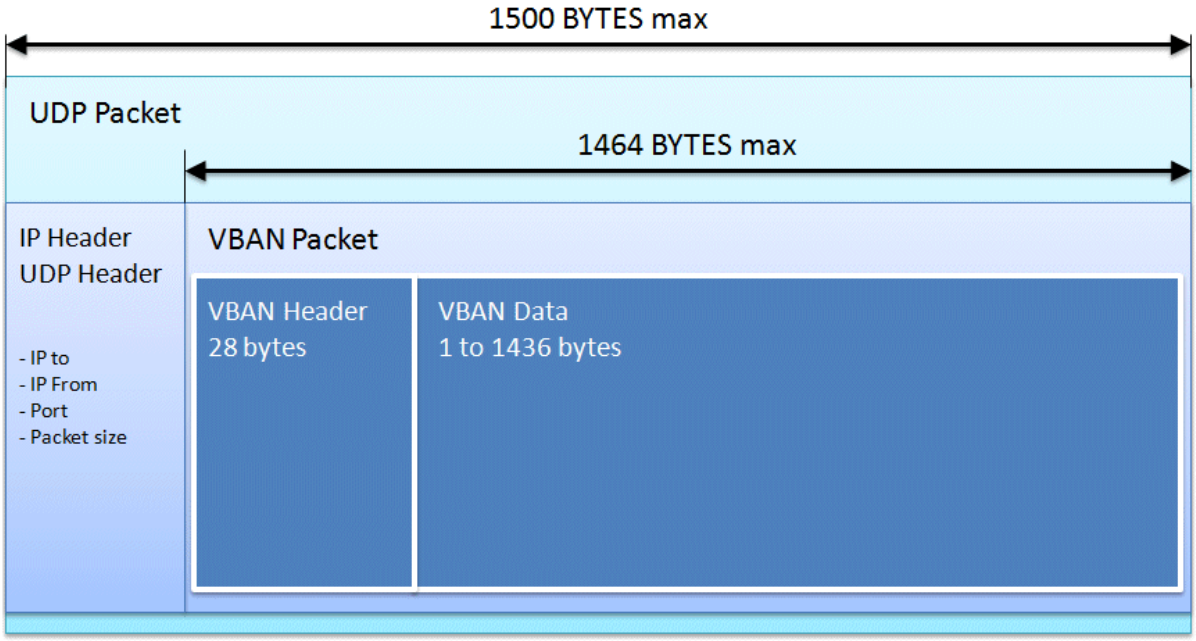
All other protocols or other audio formats (e.g. AOIP / VOIP codec), are not part of this license. All the other rights not expressly stipulated by this License are reserved by Vincent BUREL, for instance the right to modify or enhance the protocol with private or public codec.

Vincent Burel guarantees the compatibility of the current specifications in the time. Voicemeeter can be considered as Test / Validation tool for the current AUDIO PCM protocol implementation.

For implementation / integration questions, you can visit our forum or contact us by e-mail (www.vb-audio.com).

VBAN Packet Structure:

VBAN Packet is a UDP packet composed by a 28 bytes header and a various amount of data after.



VBAN Packet Header:

VBAN Packet header starts by the four letters ‘V’, ‘B’; ‘A’; ‘N’. Then it gives the format of the audio packet: Samplerate, number of Samples, number of multiplexed channels, audio data type, the Stream Name (16x chars) and an optional frame stamp.

Composition of the VBAN 28 Bytes header for audio packet:

FOURC::	V	B	A	N	4 bytes header
SR / sub protocol:		5 bits for SR index, 3 bits for sub protocol selection.			
nb Sample per frame:		8 bits unsigned integer (0 to 255 fits 1 to 256 samples).			
nb Channels:		8 bits unsigned integer (0 to 255 fits 1 to 256 channels).			
Data format / codec		3 bits for data format, 1 bit reserved, 4 bits for Codec selector.			
Stream Name::	String				16 char name (ASCII)
Frame Counter					32 bits unsigned integer growing counter

‘C’ Structure to manage VBAN header.

```
#ifdef _WINDOWS

#pragma pack(1)

struct tagVBAN_HEADER
{
   VBT_DWORD vban;         // contains 'V' 'B', 'A', 'N'
   VBT_BYTE format_SR;     // SR index (see SRList above)
   VBT_BYTE format_nbs;    // nb sample per frame (1 to 256)
   VBT_BYTE format_nbc;    // nb channel (1 to 256)
   VBT_BYTE format_bit;    // mask = 0x07 (see DATATYPE table below)
   char streamname[16];    // stream name
   VBT_DWORD nuFrame;      // growing frame number
};

#pragma pack()

#endif

#if defined(__APPLE__) || defined(__ANDROID__)

    struct tagVBAN_HEADER
    {
        uint32_t    vban;          /* contains 'V' 'B', 'A', 'N' */
        uint8_t     format_SR;     /* SR index (see SRList above) */
        uint8_t     format_nbs;    /* nb sample per frame (1 to 256) */
        uint8_t     format_nbc;    /* nb channel (1 to 256) */
        uint8_t     format_bit;    /* mask = 0x07 (nb Byte integer from 1 to 4) */
        char        streamname[16];/* stream name */
        uint32_t    nuFrame;       /* growing frame number. */
    } __attribute__((packed));

#endif

typedef struct tagVBAN_HEADER T_VBAN_HEADER;
typedef struct tagVBAN_HEADER *PT_VBAN_HEADER;
typedef struct tagVBAN_HEADER *LPT_VBAN_HEADER;

#define expectedsize_T_VBAN_HEADER (4 + 4 + 16 +4)
```

REM: VBAN protocol has been developed on x86 architecture. Data storage is consequently following little-endian rules (least significant bytes are stored on first memory addresses).

IMPORTANT CONSTRAINT:

The VBAN Header is followed by the audio data in the format defined in this header. For Ethernet V2, The entire UDP packet might be smaller than 1472 Bytes (1500 bytes MTU - 20 bytes for IP header - 8 bytes for UDP header) to avoid possible packet fragmentation by the network infrastructure. Voicemeeter VBAN Implementation is using 1500- 64 = 1436 Bytes Max (as effective data size limitation). It lets at least 2x 4 bytes free for VLAN tags or other tunneling process.

VBAN PACKET MAX SIZE = 1436 +28 = 1464 BYTES

VBAN DATA MAX SIZE = 1436 BYTES

To respect this limitation, it's up to the sender process to adjust the number of samples/data it can send in a single packet.

VBAN Packet DATA:

VBAN Packet data are following the header: data size = Packet size – VBAN header size.

PCM AUDIO DATA STRUCTURE:
Samples are stored multiplexed according the number of channels (exactly like in WAVE File format). Consequently the sample size is pending on data type (16 bits, 24 bits…) and number of channels. The sample size for 8 channel 16 bits PCM audio is 8x 2 = 16 Bytes.

If we consider the 1436 BYTES limit for VBAN Data size, we can store only 89 samples 16 bits PCM 8 channels (7.1) while we could store 359 of 16 bits Stereo samples. However the VBAN protocol is limited to 256 samples per packet.

HEADER fields:

Packet identification: the four first byte are forming the name “VBAN”

```
header.vban='NABV';
```

Header: Sample Rate

```
header.format_SR = SR_index;
```

Sample Rate is given by an index (0 to 31) given in the 5 first bits. Bits 6 to 7 must be ZERO.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	0					

```
#define VBAN_SR_MAXNUMBER 21

static long VBAN_SRList[VBAN_SR_MAXNUMBER]=
{6000, 12000, 24000, 48000, 96000, 192000, 384000,
8000, 16000, 32000, 64000, 128000, 256000, 512000,
11025, 22050, 44100, 88200, 176400, 352800, 705600};
```

index	SR	index	SR	index	SR
0	6000 Hz	7	8000 Hz	14	11025 Hz
1	12000 Hz	8	16000 Hz	15	22050 Hz
2	24000 Hz	9	32000 Hz	16	44100 Hz
3	48000 Hz	10	64000 Hz	17	88200 Hz
4	96000 Hz	11	128000 Hz	18	176400 Hz
5	192000 Hz	12	256000 Hz	19	352800 Hz
6	384000 Hz	13	512000 Hz	20	705600 Hz

index	SR	index	SR	index	SR
21	Undefined	25	Undefined	29	Undefined
22	Undefined	26	Undefined	30	Undefined
23	Undefined	27	Undefined	31	Undefined
24	Undefined	28	Undefined		

Header: Sub Protocol:

Bits 5 to 7 of **format_SR** field are used as sub protocol selector (0 to 7 scaled to 0x00 to 0xE0) should be used to be tested (ZERO = AUDIO PROTOCOL).

```
#define VBAN_PROTOCOL_AUDIO                 0x00
#define VBAN_PROTOCOL_SERIAL                0x20
#define VBAN_PROTOCOL_TXT                   0x40
#define VBAN_PROTOCOL_SERVICE               0x60
#define VBAN_PROTOCOL_UNDEFINED_1           0x80
#define VBAN_PROTOCOL_UNDEFINED_2           0xA0
#define VBAN_PROTOCOL_UNDEFINED_3           0xC0
#define VBAN_PROTOCOL_USER                  0xE0
```

Header: Number of samples

```
header.format_nbs = nbSample-1;
```

Number of sample is given by a 8 bits unsigned integer (0 – 255) where 0 means 1 sample and 255 means 256 samples (all bits are used).

Bit number	7	6	5	4	3	2	1	0
Bit Value								

Header: Number of channels:

```
header.format_nbc = nbChannel-1;
```

Number of channel is given by a 8 bits unsigned integer (0 – 255) where 0 means 1 channel and 255 means 256 channels (all bits are used).

Bit number	7	6	5	4	3	2	1	0
Bit Value								

Header: Bit Resolution:

```
header.format_bit = index;
```

Data type used to store audio sample in the packet. The index is stored on 3 first bits. Bit 3 must be ZERO. Bits 4 to 7 are used for additional CODEC (see next page).

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	0	0	0			

index	Data type	Remark
0	8 bits unsigned integer	0 to 255 (128 = 0)
1	16 bits Integer	-32768 to 32767
2	24 bits Integer	-8388608 to 8388607
3	32 bits Integer	-2147483648 to 2147483647
4	32 bits float	-1.0 to +1.0 (normal signal)
5	64 bits float	-1.0 to +1.0 (normal signal)
6	12 bits integer	-2048 to +2047
7	10 bits integer	-512 to +511

```
#define VBAN_DATATYPE_BYTE8     0x00
#define VBAN_DATATYPE_INT16     0x01
#define VBAN_DATATYPE_INT24     0x02
#define VBAN_DATATYPE_INT32     0x03
#define VBAN_DATATYPE_FLOAT32   0x04
#define VBAN_DATATYPE_FLOAT64   0x05
#define VBAN_DATATYPE_12BITS    0x06
#define VBAN_DATATYPE_10BITS    0x07
```

Header: CODEC:

Bit 4 to 7 of **format_bit** field are used to define additional CODEC.
(ZERO = no codec = Native PCM audio packet).

```
#define VBAN_CODEC_PCM                0x00
#define VBAN_CODEC_VBCA               0x10   //VB-AUDIO AOIP CODEC
#define VBAN_CODEC_VBCV               0x20   //VB-AUDIO VOIP CODEC
#define VBAN_CODEC_UNDEFINED_1        0x30
#define VBAN_CODEC_UNDEFINED_2        0x40
#define VBAN_CODEC_UNDEFINED_3        0x50
#define VBAN_CODEC_UNDEFINED_4        0x60
#define VBAN_CODEC_UNDEFINED_5        0x70
#define VBAN_CODEC_UNDEFINED_6        0x80
#define VBAN_CODEC_UNDEFINED_7        0x90
#define VBAN_CODEC_UNDEFINED_8        0xA0
#define VBAN_CODEC_UNDEFINED_9        0xB0
#define VBAN_CODEC_UNDEFINED_10       0xC0
#define VBAN_CODEC_UNDEFINED_11       0xD0
#define VBAN_CODEC_UNDEFINED_12       0xE0
#define VBAN_CODEC_USER               0xF0
```

Header: Stream Name:

```
header.streamname = szName;
```

The stream name, containing up to 16 ASCII char, is made to identify the stream. This name can be possibly defined by user.

An incoming VBAN stream must be identified by its name and its IP from.

Header: Frame Counter

```
header.nuFrame = i++;
```

This unsigned 32bits value is there to give a simple frame stamp. For each packet sent in the same stream, just increase this number. This is made for the client application (receiver) to detect possible problem: missing frame, double frame, bad frame order etc…

In our implementation in Voicemeeter, this counter is used to detect error(s) and count them. Since version 1.0.5.3 / 2.0.3.3, it is used to correct the audio stream (discontinuity problem).

Implementation Details

Even if the protocol is simple, VBAN application may have to process a huge amount of packet per second. It is important to implement an optimal process.

While UDP socket is created (bind to a given port, 6980 per default) the listen process will have to check every incoming UDP packet according the following minimal algorithm:

```
PROCESS_UDP_PACKET(void * packet, long packetsize, SOCKADDR_IN from);
{
   LPT_VBAN_HEADER lpHeader = (LPT_VBAN_HEADER)packet;
   //Detect a VBAN packet in a single 32 bit instruction
   if (lpHeader->vban == 'NABV')
   {
      //check the IP-From and Stream Name (different stream can come in)
      fOk=CheckIfWeProcessThisStream(lpHeader, from);
      if (fOk)
      {
         //get datasize (ater header) and protocol/codec index
         Datasize = packetsize - sizeof(T_VBAN_HEADER);
         protocol = lpHeader->format_SR & VBAN_PROTOCOL_MASK;
         codec = lpHeader->format_bit & VBAN_CODEC_MASK;

         //protocol selector
         if (protocol == VBAN_PROTOCOL_AUDIO)
         {
            //codec selector
            switch (codec)
            {
               case VBAN_CODEC_PCM:
                  //get audio stream information
                  SR =VBAN_SRList[lpHeader->format_SR & VBAN_SR_MASK];
                  nbChannel =((long)lpHeader->format_nbc)+1;
                  fReserved =lpHeader->format_bit & 0x08;
                  BitResolution = lpHeader->format_bit & VBAN_DATATYPE_MASK;
                  sampleSize = VBAN_SampleSize[BitResolution];
                  nbSample =((long)lpHeader->format_nbs)+1;
                  nbByte = sampleSize * nbChannel * nbSample;

                  //Push Data in Audio Stack (to be sent to audio device output for
                  //example) NOTE that waiting cycles are not allowed in this thread.

                  if (fReserved == 0)
                  {
                     if (nbByte == datasize)
                     {
                        MyStream_PushSignal(pStreamObj, lpHeader+1, nbByte,
                                            SR, nbChannel, BitResolution);
                     }
                     else nError_corrupt++
                  }
               }
            }
         }
      }
   }
}
```

Network Quality and Stack size

By “Network Quality” we are considering the capability of the network (including Ethernet Stack managed by Operating System) to send/receive audio packet as fast as possible and support different buffering sizes.

For 1 to 8 channels streaming (as it is implemented in Voicemeeter application), packet can contain up to 256 samples and then applies a minimal stack of 3 x 256 samples.

```
#define VBAN_PROTOCOL_MAXNBS 256
```

Voicemeeter proposes 5 Network Quality Level conditionning the stack size for incoming packet starting by 6x 256 samples (Audio Pro Stack, managing 32, 64, 128 or more channel could be smaller). This is subject to change in the time but the current stack size (nnn) in Voicemeeter is following the algorithm below.

```
nnn= current audio buffer size

nmin = 6 * VBAN_PROTOCOL_MAXNBS;
switch(pStream->format.networkquality)
{
   case 0://Optimal
      if (nnn < 512) nnn=512;
      break;
   case 1://Fast
      if (nnn < 1024) nnn=1024;
      break;
   case 2://Medium
      if (nnn < 2048) nnn=2048;
      break;
   case 3://Slow
      if (nnn < 4096) nnn=4096;
      break;
   case 4://Very Slow
      if (nnn < 8192) nnn=8192;
      break;
}
nnn=nnn*3;
if (nnn < nmin) nnn=nmin;
```

For general audio stream management, we are obliged to use big stack to fit the different possible buffering size used by audio applications. Default audio buffering on windows is around 512 to 1024 samples. So we might consider receiving burst of 2, 3 or 4 VBAN packets at a time, simply because audio buffering is 2, 3, or 4 time 256 samples (max size for a VBAN packet).

VBAN SUB PROTOCOL
SERIAL

VBAN sub protocol: SERIAL

VBAN Packet header can be used to send serial data by using the sub protocol field:

#define VBAN_PROTOCOL_SERIAL : 0x20

Some members of the header are consequently used in different way to describe the SERIAL stream.

Composition of the VBAN 28 Bytes header for serial packet:

FOURC::	V	B	A	N	4 bytes header
SR / sub protocol:		5 bits for **bps** index, 3 bits for sub protocol selection (001).			
bitmode:		8 bits field to define the serial settings (to fit serial hardware).			
Channels Ident:		8 bits unsigned integer (0 to 255 fits 1 to 256 channels).			
Data format		3 bits for data format, 1 bit reserved & 4 bits for serial type.			
Stream Name::	String				16 char name (ASCII)
Frame Counter					32 bits unsigned integer growing counter

Serial Header: SR gives BPS

```
header.format_SR = bps_index;
```

Bit rate is given in bps for information only. But it can be useful if serial data come from or go to a particular COM port. Set to ZERO if there is no particular bit rate.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	1					

```
#define VBAN_BPS_MAXNUMBER 25

static long VBAN_BPSList[VBAN_BPS_MAXNUMBER]=
{0, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 14400,
19200, 31250, 38400, 57600, 115200, 128000, 230400, 250000, 256000, 460800,
921600, 1000000, 1500000, 2000000, 3000000};
```

index	SR	Index	SR	index	SR
0	0 bps	7	4800 bps	14	115200 bps
1	110 bps	8	9600 bps	15	128000 bps
2	150 bps	9	14400 bps	16	230400 bps
3	300 bps	10	19200 bps	17	250000 bps
4	600 bps	11	31250 bps	18	256000 bps
5	1200 bps	12	38400 bps	19	460800 bps
6	2400 bps	13	57600 bps	20	921600 bps

index	SR	Index	SR	index	SR
21	1000000 bps	25	Undefined	29	Undefined
22	1500000 bps	26	Undefined	30	Undefined
23	2000000 bps	27	Undefined	31	Undefined
24	3000000 bps	28	Undefined		

Serial Header: channel ident:

```
header.format_nbc = channel ident;
```

Can be used to defined a sub channel (sub serial link) and then manage up to 256 different serial virtual pipes (ZERO by default).

Serial Header: COM Port configuration

```
header.format_nbs = bitmode;
```

This field is used to give possible information on COM port and serial transmission mode related to a Hardware COM port. This is made to possibly emulate COM to COM port connections and let the receiver configure the physical COM port in the right mode.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	0	0				

Bit 0-1 : STOP BIT
- 0: 1 stop bit
- 1: 1,5 stop bit.
- 2: 2 stop bit.
- 3: unused.

Bit 2 : START BIT
- 0: no start bit
- 1: start bit.

Bit 3: PARITY CHECKING:
- 0: no parity checking
- 1: parity checking.

Bit 7: MULTIPART DATA BLOCK (optional)
- Set to 1 if the serial data block requires several VBAN Packets (1436 BYTE max) to be completed.

Serial Header: format bit:

```
header.format_bit = index;
```

Data type used to store data in the packet (ZERO per default). The index is stored on 3 first bits. Bit 3 must be ZERO. Bits 4 to 7 gives additional mode.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	0	0	0			

index	Data type	Remark
0	8 bits data	0 to 255
1	Undefined	
2	Undefined	
3	Undefined	
4	Undefined	
5	Undefined	
6	Undefined	
7	Undefined	

#define VBAN_DATATYPE_BYTE8 0x00

Bit 4-7: SERIAL stream type:

Bit 4 to 7 of **format_bit** field are used to define additional type of serial.
(ZERO = generic).

#define VBAN_SERIAL_GENERIC 0x00
#define VBAN_SERIAL_MIDI 0x10
#define VBAN_SERIAL_UNDEFINED_2 0x20
#define VBAN_SERIAL_UNDEFINED_3 0x30
#define VBAN_SERIAL_UNDEFINED_4 0x40
#define VBAN_SERIAL_UNDEFINED_5 0x50
#define VBAN_SERIAL_UNDEFINED_6 0x60
#define VBAN_SERIAL_UNDEFINED_7 0x70
#define VBAN_SERIAL_UNDEFINED_8 0x80
#define VBAN_SERIAL_UNDEFINED_9 0x90
#define VBAN_SERIAL_UNDEFINED_10 0xA0
#define VBAN_SERIAL_UNDEFINED_11 0xB0
#define VBAN_SERIAL_UNDEFINED_12 0xC0
#define VBAN_SERIAL_UNDEFINED_13 0xD0
#define VBAN_SERIAL_UNDEFINED_14 0xE0
#define VBAN_SERIAL_USER 0xF0

Header: Stream Name:

```
header.streamname = szName;
```

The stream name, containing up to 16 ASCII char, is made to identify the stream. This name can be possibly defined by user.

An incoming VBAN stream must be identified by its name (and possibly by its IP from). For SERIAL Stream, it's up to the receiver to identify the stream strictly with IP and name or just by name… It can be interesting to be able to manage VBAN SERIAL stream by its name only, because allowing multi remotes system (the ability for the receiver to be controlled by multiple devices, including MIDI devices…).

Header: Frame Counter

```
header.nuFrame = i++;
```

This unsigned 32bits value is there to give a simple frame stamp. For each packet sent in the same stream, just increase this number. This is made for the client application (receiver) to detect possible problem: missing frame, double frame, bad frame order etc…

SERIAL stream in MIDI mode:

If transmitting MIDI data, configure **format_bit** stream type field with VBAN_SERIAL_MIDI value.

Optionally you may set **format_SR** field with the MIDI bit rate (31250 bps) and set **format_nbs** field in the regular MIDI port mode: 1 start bit, 1 stop bit, no parity.

A VBAN packet must contain one or several entire MIDI messages (up to 1436 bytes max). a VBAN packet can then contains a list of several Note-On, Note-Off, Pitch Bend, Controller or other MIDI message.

If the MIDI message cannot enter in a single packet (for SYSEX for example), set the bit 7 of **format_nbs** field. Then the receiver will be able to rebuild the complete packet before processing it.

Implementation Details

Voicemeeter or MacroButtons application use 115200 kbps serial to manage MIDI stream (without multi packet management): a single MIDI Message must be less than 1436 BYTES.

HEADER: 'V', 'B', 'A', 'N', 0x2E, 0x00, 0x00, 0x00, "MIDI1", …

VBAN SUB PROTOCOL
TEXT

VBAN sub protocol: TEXT

VBAN Packet header can be used to send text data by using the sub protocol field:

#define VBAN_PROTOCOL_TXT: 0x40

Similar to SERIAL protocol, TEXT protocol allows transporting text command in different format (ASCII, UTF8…).

Composition of the VBAN 28 Bytes header for serial packet:

FOURC::	V	B	A	N	4 bytes header
SR / sub protocol:		5 bits for **bps** index, 3 bits for sub protocol selection (010).			
bitmode:		Unused (must be zero).			
Channels Ident:		8 bits unsigned integer (0 to 255 fits 1 to 256 channels).			
Data format		3 bits for data format, 1 bit reserved & 4 bits for serial type.			
Stream Name::	String				16 char name (ASCII)
Frame Counter					32 bits unsigned integer growing counter

Serial Header: SR gives BPS

```
header.format_SR = bps_index;
```

Bit rate is given in bps for information only. But it can be used internally to limit the bandwidth of the stream and for example gives more priority to audio stream or RT MIDI stream. It can be set to ZERO if there is no particular bit rate.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	1	0					

```
#define VBAN_BPS_MAXNUMBER 25

static long VBAN_BPSList[VBAN_BPS_MAXNUMBER]=
{0, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 14400,
19200, 31250, 38400, 57600, 115200, 128000, 230400, 250000, 256000, 460800,
921600, 1000000, 1500000, 2000000, 3000000};
```

index	SR	Index	SR	index	SR
0	0 bps	7	4800 bps	14	115200 bps
1	110 bps	8	9600 bps	15	128000 bps
2	150 bps	9	14400 bps	16	230400 bps
3	300 bps	10	19200 bps	17	250000 bps
4	600 bps	11	31250 bps	18	256000 bps
5	1200 bps	12	38400 bps	19	460800 bps
6	2400 bps	13	57600 bps	20	921600 bps

index	SR	Index	SR	index	SR
21	1000000 bps	25	Undefined	29	Undefined
22	1500000 bps	26	Undefined	30	Undefined
23	2000000 bps	27	Undefined	31	Undefined
24	3000000 bps	28	Undefined		

Serial Header: channel ident:

```
header.format_nbc = channel ident;
```

Can be used to defined a sub channel (sub text channel) and then manage up to 256 different virtual pipes (ZERO by default).

Serial Header: nbs is not used

```
header.format_nbs = 0;
```

This field is not used for TEXT protocol.

Serial Header: format bit:

```
header.format_bit = index;
```

Data type used to store data in the packet (ZERO per default). The index is stored on 3 first bits. Bit 3 must be ZERO. Bits 4 to 7 gives additional mode.

Bit number	7	6	5	4	3	2	1	0
Bit Value	0	0	0	0	0			

index	Data type	Remark
0	8 bits data	0 to 255
1	Undefined	
2	Undefined	
3	Undefined	
4	Undefined	
5	Undefined	
6	Undefined	
7	Undefined	

#define VBAN_DATATYPE_BYTE8 0x00

Bit 4-7: SERIAL stream type:

Bit 4 to 7 of **format_bit** field are used to define TEXT format.

#define VBAN_TXT_ASCII 0x00
#define VBAN_TXT_UTF8 0x10
#define VBAN_TXT_WCHAR 0x20
#define VBAN_TXT_UNDEFINED_3 0x30
#define VBAN_TXT_UNDEFINED_4 0x40
#define VBAN_TXT_UNDEFINED_5 0x50
#define VBAN_TXT_UNDEFINED_6 0x60
#define VBAN_TXT_UNDEFINED_7 0x70

```
#define VBAN_TXT_UNDEFINED_8        0x80
#define VBAN_TXT_UNDEFINED_9        0x90
#define VBAN_TXT_UNDEFINED_10       0xA0
#define VBAN_TXT_UNDEFINED_11       0xB0
#define VBAN_TXT_UNDEFINED_12       0xC0
#define VBAN_TXT_UNDEFINED_13       0xD0
#define VBAN_TXT_UNDEFINED_14       0xE0
#define VBAN_TXT_USER               0xF0
```

Header: Stream Name:

```
header.streamname = szName;
```

The stream name, containing up to 16 ASCII char, is made to identify the stream. This name can be possibly defined by user.

An incoming VBAN stream must be identified by its name (and possibly by its IP from). For TEXT Stream, it's up to the receiver to identify the stream strictly with IP and name or just by name… It can be interesting to be able to manage VBAN TEXT stream by its name only, because allowing multi remotes system (the ability for the receiver to be controlled by multiple devices, sending different textual commands).

Header: Frame Counter

```
header.nuFrame = i++;
```

This unsigned 32bits value is there to give a simple frame stamp. For each packet sent in the same stream, just increase this number. This is made for the client application (receiver) to detect possible problem: missing frame, double frame, bad frame order etc…

Implementation Details

Voicemeeter or MacroButtons application use 256000 kbps UTF8 TEXT to manage Command stream (without multi packet management): a single complete TEXT Message must be less than 1436 BYTES.

HEADER: 'V', 'B', 'A', 'N', 0x52, 0x00, 0x00, 0x10, "Command1", …

VBAN SUB PROTOCOL
SERVICE

This section is preliminary specification
partially implemented

VBAN sub protocol: SERVICE

VBAN Packet header can be used to send service request. For example a specific PING packet to identify VBAN device on the network:

#define VBAN_PROTOCOL_SERVICE 0x60

Some members of the header are consequently used in different way to describe the SERIAL stream.

Composition of the VBAN 28 Bytes header for Service packet:

FOURC::	V	B	A	N	4 bytes header
sub protocol:		5 bits unused, 3 bits for sub protocol selection (011).			
function:		8 bits field to define the service type.			
Service:		8 bits field to define the service function.			
Additional info		Must be zero.			
Stream Name::	String				16 char name (ASCII)
Frame Counter					32 bits unsigned integer growing counter

Service Header:

```
header.format_SR = x60;
```

This field gives only the sub protocol SERVICE (0x60) on 3 last bit (5 first bits must be ZERO).

Bit number	7	6	5	4	3	2	1	0
Bit Value	**0**	**1**	**1**	0	0	0	0	0

Service Header: service type:

```
header.format_nbc = service type;
```

This field is used to define up to 256 different services. The first one is the IDENTIFICATION service = 0.

#define VBAN_SERVICE_IDENTIFICATION	0
#define VBAN_SERVICE_CHATUTF8	1
#define VBAN_SERVICE_RTPACKETREGISTER	32
#define VBAN_SERVICE_RTPACKET	33

Service Header: service function

```
header.format_nbs = function;
```

This field is used to define 128 different functions per service and bit 7 is used to indicate that the request is a reply.

Bit number	7	6	5	4	3	2	1	0

Bit Value	0							

#define VBAN_SERVICE_FNCT_PING0 0

#define VBAN_SERVICE_FNCT_REPLY 0x80

Service Header: format bit:

```
header.format_bit = 0;
```

Can be used according service type (otherwise must be ZERO)

Service Header: Stream Name:

```
header.streamname = szName;
```

The stream name, is not used and can optionally be used to label the service function. e.g: VBAN SPOT PING.

Service Header: Frame Counter / request ident.

```
header.nuFrame = i++;
```

This unsigned 32bits value must be a growing number to give a unique ident to the service request. This frame number will be used for the reply to identify the transaction (REQUEST - REPLY). In other word, the reply will have to use the same frame number of the request to let the server associates the received request to the right sent request.

For some service type, this counter can be used for extra information.

VBAN PING0 DATA STRUCTURE

```
#ifdef _WINDOWS

   #pragma pack(1)

struct tagVBAN_GPS_STRUCT
{
       unsigned char latitude_degre; //0 to 90
       unsigned char latitude_minute; //0 to 59
       unsigned char latitude_seconde; //0 to 59
       unsigned char latitude_scent; //0 to 99 (100ieme seconde) + sign mask 0x80 (0 = N / 1 =
Sud)
       unsigned char longitude_degre; //0 to 180
       unsigned char longitude_minute; //0 to 59
       unsigned char longitude_seconde; //0 to 59
       unsigned char longitude_scent; //0 to 99 (100ieme seconde) + sign mask 0x80 (0 = E / 1 =
West)
} T_VBAN_GPS_STRUCT, *PT_VBAN_GPS_STRUCT, *LPT_VBAN_GPS_STRUCT;


   struct tagVBAN_PING0
   {
        VBT_DWORD    bitType;               /* VBAN device type*/
        VBT_DWORD    bitfeature;            /* VBAN bit feature */
        VBT_DWORD    bitfeatureEx;          /* VBAN extra bit feature */
        VBT_DWORD    PreferedRate;          /* VBAN Preferred sample rate */
        VBT_DWORD    MinRate;               /* VBAN Min samplerate supported */
        VBT_DWORD    MaxRate;               /* VBAN Max Samplerate supported */
        VBT_DWORD    color_rgb;             /* user color */
        VBT_BYTE     nVersion[4];           /* App version 4 bytes number */

        char      GPS_Position[8];          /* Device position */
        char      USER_Position[8];         /* Device position defined by a user process */
        char      LangCode_ascii[8];        /* main language used by user FR, EN, etc..*/
        char      reserved_ascii[8];        /* unused : must be ZERO*/

        char       reservedEx[64];          /* unused : must be ZERO*/
        char       DistantIP_ascii[32];     /* Distant IP*/
        VBT_WORD   DistantPort;             /* Distant port*/
        VBT_WORD   DistantReserved;         /* Reserved*/

        char        DeviceName_ascii[64];       /* Device Name (physical device) */
        char        ManufacturerName_ascii[64]; /* Manufacturer Name */
        char        ApplicationName_ascii[64];  /* Application Name */
        char        HostName_ascii[64];         /* dns host name  */
        char        UserName_utf8[128];         /* User Name */
        char        UserComment_utf8[128];      /* User Comment/ Mood/ Remark/ message */
   };

   #pragma pack()

#endif

#if defined(__APPLE__) || defined(__ANDROID__)

    struct tagVBAN_PING0
    {
        uint32_t     bitType;           /* VBAN device type*/
        uint32_t     bitfeature;        /* VBAN bit feature */
        uint32_t     bitfeatureEx;      /* VBAN extra bit feature */
        uint32_t     PreferedRate;      /* VBAN Preferred sample rate */
        uint32_t     MinRate;           /* VBAN Min samplerate supported */
        uint32_t     MaxRate;           /* VBAN Max Samplerate supported */
        uint32_t     color_rgb;         /* user color */
        uint8_t      nVersion[4];       /* App version 4 bytes number */

        char      GPS_Position[8];      /* Device position */
        char      USER_Position[8];     /* Device position defined by a user process */
        char      LangCode_ascii[8];    /* main language used by user FR, EN, etc..*/
        char      reserved_ascii[8];    /* unused : must be ZERO*/

        char       reservedEx[64];      /* unused : must be ZERO*/
        char       DistantIP_ascii[32]; /* Distant IP*/
        uint16_t   DistantPort;         /* Distant port*/
```

```
        uint16_t   DistantReserved;      /* Reserved*/

        char        DeviceName_ascii[64];           /* Device Name (physical device) */
        char        ManufacturerName_ascii[64];     /* Manufacturer Name */
        char        ApplicationName_ascii[64];      /* Application Name */
        char        HostName_ascii[64];             /* dns host name  */
        char        UserName_utf8[128];             /* User Name */
        char        UserComment_utf8[128];          /* User Comment/ Mood/ Remark/ message */

    } __attribute__((packed));

#endif
```

Application type

bitType gives information on the VBAN device/application capabilities / main function:

```
#define VBANPING_TYPE_RECEPTOR          0x00000001  // Simple receptor
#define VBANPING_TYPE_TRANSMITTER       0x00000002  // Simple Transmitter
#define VBANPING_TYPE_RECEPTORSPOT      0x00000004  // SPOT receptor (able to receive several
streams)
#define VBANPING_TYPE_TRANSMITTERSPOT   0x00000008  // SPOT transmitter (able to send several
streams)

#define VBANPING_TYPE_VIRTUALDEVICE     0x00000010  // Virtual Device
#define VBANPING_TYPE_VIRTUALMIXER      0x00000020  // Virtual Mixer
#define VBANPING_TYPE_MATRIX            0x00000040  // MATRIX
#define VBANPING_TYPE_DAW               0x00000080  // Workstation

#define VBANPING_TYPE_SERVER            0x01000000  // VBAN SERVER
```

Features (supported sub protocol)

bitfeature gives information on supported VBAN protocol:

```
#define VBANPING_FEATURE_AUDIO          0x00000001
#define VBANPING_FEATURE_AOIP           0x00000002
#define VBANPING_FEATURE_VOIP           0x00000004

#define VBANPING_FEATURE_SERIAL         0x00000100
#define VBANPING_FEATURE_MIDI           0x00000300

#define VBANPING_FEATURE_FRAME          0x00001000

#define VBANPING_FEATURE_TXT            0x00010000
```

bitfeatureEx is not defined for the moment (set it to ZERO).

DistantIP_ascii and DistantPort are planned to make peer to peer relationship. Then the PING packet can be used to talk with a server and get information for P2P link with another computer on internet.

Undefined or unexplained field must be ZERO.

RT-Packet Service

VBAN RT-Packet can be implemented in some server applications (like Voicemeeter) to broadcast periodically current parameters or states and any real time value (e.g. peak meter value). Client application just has to register to the service, in order to receive RT-Packet for a given period (time out). Client Application will have to register periodically according the wanted time out (for example every 10 seconds if the time out is set to 11 seconds).

RT Packet frequency and Data Structure are pending on the server Application and packet identifier. Meaning a server application can provide several RT-Packet having different frequency period (basically from 100Hz to 0.1Hz).

Request to register to RT packet service:

FOURC::	V	B	A	N	4 bytes header
sub protocol:		0x60 (SERVICE).			
Function:		0 = register to RT packet service ID (0-127).			
Service:		32 = VBAN_SERVICE_RTPACKETREGISTER.			
Additional info		0 – 255 = Time out in second (to stop RT packet broadcast)			
Stream Name::	String				unused (ASCII) user define
Frame Counter					32 bits unsigned integer growing counter

Packet ID List	0	1	…	…	list of requested RT packets (0 to 127).

With this RT-Packet Service, a server application can provide up to 128 different packets. A client application can register to receive only the wanted packet, for example packet ID 0 and packet ID 5, while another application could request for the packet ID 2. If the Packet ID List is not specified, all RT-Packets will be sent to the client.

Reply will give the code error in the Additional info field:

```
header.format_bit = reply;
```

0 : no RT packet service (could mean the packet ID is not existing).
1 : RT packet service registered
2 : RT packet service busy (no more slot).

RT packets:

FOURC::	V	B	A	N	4 bytes header
sub protocol:		0x60 (SERVICE).			
Function:		0 = RT packet service ID (0-127).			
Service:		33 = VBAN_SERVICE_RTPACKET.			
Additional info		0			
Stream Name::	String				Can contain a packet name
Frame Counter					32 bits unsigned integer growing counter.

RT Packet Data::	1 to 1436 Bytes	Pending on server application.

The RT Packet Data Structure is pending on the server application specification and packet ID List description.

Implementation Details

For example the VM-Streamer View Application is sending the RTPACKETREGISTER request every 10 seconds to subscribe the RT-Packet service for 15 seconds. Packet ID List is optional since it requires a single packet (the packet 0).

HEADER: ‘V’, ‘B’, ‘A’, ‘N’, 0x60, 0x00, 0x20, 0x0F, “Register RTP”, 0

After Registration to the RT-Packet service, you can expect to receive periodically the RT-Packet starting by this header:

HEADER: ‘V’, ‘B’, ‘A’, ‘N’, 0x60, 0x00, 0x21, 0x00, “Voicemeeter RTP”, …

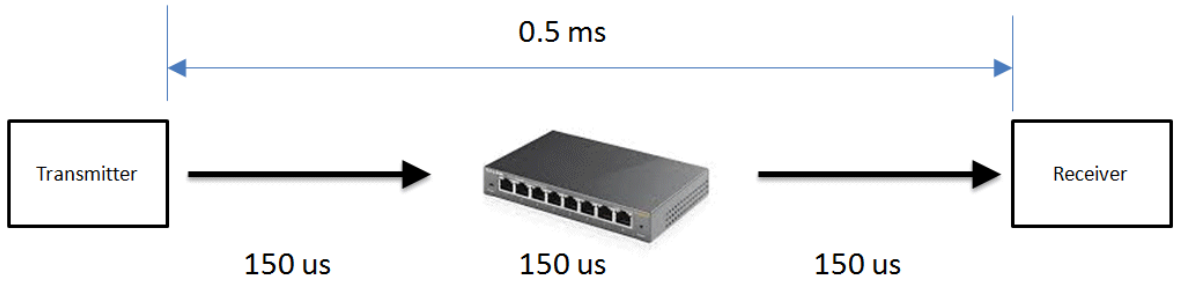
The data structure behind is pending on server application and RT-Packet Identifier of course. For Voicemeeter, see RT-Packet Structure in VoicemeeterRemote.h

AOIP
AUDIO OVER IP

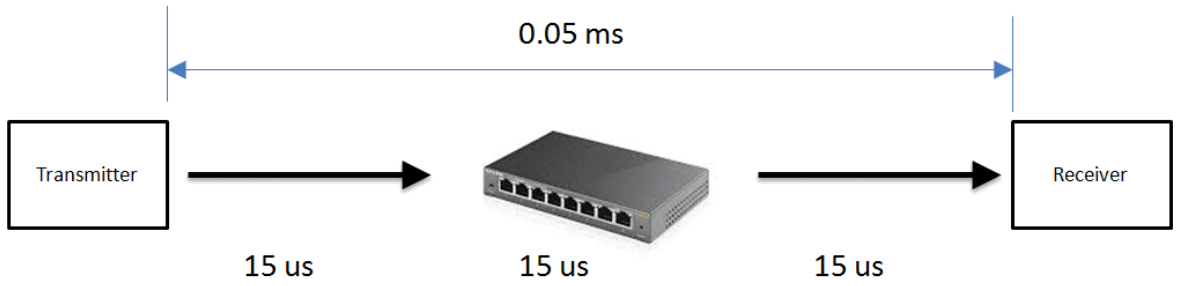
AUDIO NETWORK CAPABILITIES

With T100 or T1000 network infra structure, it is possible to transport audio with a very small latency, basically less than one millisecond on T100 and less than 0.1 ms on T1000.

T100 : Max time to transport 1500 Bytes UDP Packet



T1000 : Max time to transport 1500 Bytes UDP Packet



Consequently, real time audio transport with the best latency or multi point synchronization is mainly pending on transmitter or receiver capabilities (for a computer device, the network stack managed by the operating system). The transmitter points should be able to feed the network with a minimal latency and a perfect time regularity. The receiver point should be able to process the audio packet as fast as it is received, with a constant processing time.

WARNING: network timing is also pending on wire length. Due to the 300.000 km second maximum speed of the light (electricity), additional latency must be added for long distance: 1ms for 300 km, 1us for 300 meters (not significant for usual local network of course).