Vilistus Communication Protocol
User Summary

This documentation is supplied by Durham Systems Management Limited for use alongside The Vilistus Digital Sampling Unit and the Vilistus Professional software

Comments and suggestions relating to the products should be addressed to:

Durham Systems Management Limited

Fernlea House
Newby
Penrith CA10 3EX
United Kingdom

t: +44 (0) 1 931 714053

e: support@vilistus.com
www.vilistus.com

Version 1.0 – August 2011
Version 2.0 – February 2013
Version 3.0 – November 2013

Vilistus V4.3.4 – Communications Summary

The Vilistus v3.3.4 Digital Sampling Unit (DSU) can communicate with a host PC by Bluetooth, WiFi or tethered USB Cable. The FTDI USB driver is always installed as part of the Vilistus PC software installation procedure. Note that we don't recommend that you use a Bluetooth radio that uses the Blue Soleil software.

Bluetooth and USB

The serial port parameters are as follows:

Baud Rate:	115200
Data Bits:	8
Stop Bits:	1
Parity:	No Parity
Flow Control:	None

WiFi

Vilistus can operate in either Ad-hoc or Infrastructure mode. As standard, the Vilistus is set up for Ad-hoc mode and uses

IP address:	169.254.1.1
Port:	2000

We recommend that Ad-hoc mode is used except when using multiple DSU's.

Communications Protocol (extract)

The Vilistus DSU uses the open source P3 data format.

The DSU is turned on using a long press on the on/off button. At this point the Bluetooth and battery light will briefly flash and the white Power light will illuminate.

For units that do not support ambulatory recording, pressing and holding the on/off button for more than two seconds will turn DSU transmission on (and the Blue transmission LED will illuminate); pressing again for two seconds will turn the DSU transmission off. This is not the normal mode of operation.

There are two commands to start and stop data transfer:

1. To begin communications : 0x0A**RING**0x0A
2. To terminate communications: 0x0A**NO C**0x0A

Note the space in NO CARRIER. This can be tested with products such as Hyperterminal

where the 0x0A character can be simulated by holding the CTRL key and pressing “J”.

Sample Rates

As a default, the Vilistus unit will internally sample at 256 samples per second although this can be changed tif required.

It is important to note, however, that in the case of a 8-Channel DSU that uses the Bluetooth transmission method, due to bandwidth issues, 256 is the highest sample rate.

The DSU will, as a default, transmit 8 channels of data even if, as with a Vilistus-4, there are only 4 data channels exposed. The size of the P3 packet can be tailored to 2, 4 or 8 channels.

It is possible to increase the internal sample rate by reducing the number of active sensors. For example, an internal sample rate of 1KHz can be achieved by using 2 sensors.

These changes require access to the restricted Vilistus protocol commands but will be supplied by Technical Support if required.

P3 Protocol

Vilistus uses the open source P3 protocol which uses the high bit on the last byte to signal the end of packet.

We have noticed that a number of open source products expect a 6-channel packet and assume a 12 byte packet size without checking bit 7 on the last byte. As standard, Vilistus using an 8-channel packet with bit 7 set on the 14th byte. The P3 data format, along with sample code for expanding the data can be supplied if required.

The following code snippet shows how to decode a 14 byte packet:

```
P3Buffer->PacketCounter[PacketID] = *decode_ptr>>1;
P3Buffer->AuxByte[PacketID]  = *(decode_ptr+1);

P3Buffer->Channel0[PacketID] = ( *(decode_ptr+4) >>4<<7 ) | *(decode_ptr+2);
P3Buffer->Channel1[PacketID] = ( (*(decode_ptr+4) & 0x0F) <<7 ) | *(decode_ptr+3);
P3Buffer->Channel2[PacketID] = ( *(decode_ptr+7) >>4<<7 ) | *(decode_ptr+5);
P3Buffer->Channel3[PacketID] = ( (*(decode_ptr+7) & 0x0F) << 7 )  | *(decode_ptr+6);
P3Buffer->Channel4[PacketID] = ( *(decode_ptr+10) >>4<<7 ) | *(decode_ptr+8);
P3Buffer->Channel5[PacketID] = ( (*(decode_ptr+10) & 0x0F) << 7 )  | *(decode_ptr+9);
P3Buffer->Channel6[PacketID] = ( (*(decode_ptr+13) & 0x7F) >>4<<7 ) | *(decode_ptr+11);
P3Buffer->Channel7[PacketID++] = ( (*(decode_ptr+13) & 0x0F) << 7 ) | *(decode_ptr+12);
```

Vilistus DLL

We supply a DLL which completely wraps the communications with the Vilistus DSU and decodes the P3 data stream, exposing an array of the next data values to be read. The code sample below shows the flow of control using a simple Visual Basic program:

```
Private sub cmdReadDSU()

     dim RC as long
     dim P3Array() as long
     dim PacketCounter as long
     dim AuxByte as long
     dim i as integer
     dim cFileName as string

     'Setup communications
     rc = SetComPorts(1, 100, 10)

     if rc < 0 then
        msgbox "Initalisation Failure Code is " & rc
        exit sub
     end if

     'Set COM Port scan wait time (usually 1.5 seconds)
     rc = SetScanWaitTime(1, 1500)

     if rc < 0 then
        msgbox "Invalid wait time"
        exit sub
     end if

     ' Initialise the DSU and run variables
     rc = InitialiseDSU(1)
     redim P3Array(7) as long
     bProcessing = True

     Do While bProcessing = True

        rc = GetP3Data(P3Array(0), 8, 1, PacketCounter, AuxByte)

        if rc = 0 and PacketCounter > 0 then
           lblPacketCounter = PacketCounter
           lbllbufferlen = Bufferlength(1)
           lblBytesRead = TotalBytesRead(1)
           lblPacketsRemainingInBuffer = PacketsInBuffer(1)

           for i = 0 to 7
               txtValue(i) = P3Array(i)
           next I
        end if

        Do Events

     Loop

     ' Terminate after writing data to file
     cFilename = "Data.SES"
     rc = TerminateDSU(1, 1, cFilename)
```

The key processing functions are described below. Note that in all cases the DSUId should be set to the same positive integer (we suggest 1).

InitialiseDSU

```
int InitialiseDSU(int DSUID);
```

The **InitialiseDSU** function sets up the internal environment for the run and spawns the sub-task for processing the DSU data stream.

GetP3Data

```
int GetP3Data(int array,int ArraySize,int DSUID,int PacketCounter,int AuxByte);
```

The **GetP3Data** function collects, if data exists, the next packet of data as decoded by the Vilistus DLL. The current PacketCounter (which is a number between 0 and 63) and the AuxByte (which is normally 0) are returned in the specified variables.

Return Codes:

```
 0    -          Data Returned in array
-1    -          At end of data buffer (*)
-2    -          No data written to output buffer (and PacketCounter set to -1)
-3    -          System not set (**)
```

(*) Not necessarily an error. It could be that the processing program has exhausted the read buffer before the Vilistus DLL has had a chance to add further rows
(**) This is usually a transmission error. Check that the Vilistus is switch on and connected.

SetComPorts

```
int SetComPorts(int  DSUID, int FROMPORT, int TOPORT, int HINTPORT);
```

The **SetComPorts** function scans all active COM Ports (from FROMPORT to TOPORT) looking for a Vilistus DSU. The HINTPORT should be set to the COM Port that was assigned when the unit was paired with the PC .

SetScanWaitTime

```
int SetScanWaitTime(int DSUID, int SLEEPWAIT);
```

The **SetScanWaitTime** function sets the time period opening/reading of COM Ports. We suggest 200mS to 1500mS

TerminateDSU

```
int TerminateDSU(int DSUID, int WriteIndicator, char Filename)
```

The **TerminateDSU** function closes communications with the Vilistus DSU.

WriteIndicator Value	**Meaning**
0	Do not write the session to disk
1	Write the session to disk using the supplied file name

Vilistus Consumer Application Programming Interface

The Vilistus Consumer API has multiple levels of support for 3rd party products to interoperate with the Vilistus hardware and software. These can be categorised as:

Level 1 – Receive threshold, amplitude and status information via DLL
Level 2 – Send and receive game control information via TCP/IP
Level 3 – Initiate Vilistus command functions through the use of callbacks to the Vilistus code

Level 2 and 3 access to the Vilistus API are restricted and a request to Durham Systems Management should be made if these are appropriate to your application.

Level 1 API Access

Architecture

The Vilistus DLL, as well as providing low level access to functions including serial ports and digital filters, provides a simple one-direction applications programming interface allowing client programs to check the status of the Vilistus application and to receive key data values.

The Vilistus DLL is available to be used by multiple client applications simultaneously.

Shared Data

The data provided within the Vilistus level-1 API consists of three shared variables defined (and initialised) as:

```
double vsAPIAmplitudeValue[256] = {0};
double vsAPIThresholdValue[256] = {0};
int vsAPISystemStatus = 0;
```

The Amplitude and threshold values are populated by the Vilistus application using appropriate API Functions (see below). The vsAPISystemStatus flag is set as following:

VSAPI_STATUSNOTINITIALISED = 0
is the status before Vilistus initialises the data fields
VSAPI_TRANSMITTING = 1
is set when data is being sent after processing from a live feed or stored session
VSAPI_PAUSED = 2
is set when the data processing is paused
VSAPI_STOPPED = 3
is set when either recording or playback has ended
VSAPI_ENDED = 4
is set just before the Vilistus program terminates.
VSAPI_VILISTUS_LOADED = 5
is just when the Vilistus program finishes initialising

Functions

The Vilistus DLL provides six functions to access and update the status and the data:

Setting functions:

These functions should only used within the Vilistus program but are documented here for completeness.

```
int vsAPISetSystemStatus( int nStatus );
```

Sets the current status. Always returns a "0" return code

```
int vsAPISetThresholdValues( double *nThresholdValueValues );
```

Sets the current thresholds. This value is updated every time the threshold processor is accessed even when the threshold is fixed. Always returns a 0 return code. The address points to an array of 256 threshold values.

```
int vsAPISetAmplitudeValues( double *nAmplitudeValues );
```

Sets the current amplitude from the array of amplitude values generated within the Vilistus program. Always returns a 0 return code. The address points to an array of 256 amplitude values.

IMPORTANT: The Amplitude and Threshold values are in the order of the instruments created on the Vilistus desktop

Retrieval functions:

Three functions are provided to access the Amplitude, threshold and status information.

```
int vsAPIGetSystemStatus();
```

returns the current system status (values documented above)

```
double nAmplitudeValues[256];
int vsAPIGetAmplitudeValues( double *nAmplitudeValues );
```

Returns the current amplitude values from the internal array into a user defined array. The function returns a 0 return code.

```
Double nThresholdvalues[256];
Int vsAPIGetThresholdValues( double *nThresholdValues );
```

Returns the current threshold values from the internal array into a user defined array. The function returns a 0 return code.

Testing your connection to Vilistus

You may need to link your code with our DLL. Instructions on how to do that are beyond the scope of this document as the linking mechanism is dependent on your development environment.

If you have having issues connecting to our software then contact our support term at support@vilistus.com

The easiest way to test the connectivity between Vilistus and 3rd party code is to use a stored EEG session. We provide a number of demo sessions with the software and we recommend that you use these as, in Vilistus, there is no difference in internal processing once the data has been acquired from either the serial port or stored on file.

We recommend that you follow these instructions to get the Vilistus program to update the values in the Vilistus API.

1. Start the Vilistus program either from the Start Menu or the desktop icon.
2. Load the an appropriate Screenset by going to FILE / LOAD SCREEN SET, select and press Open.
3. Load an existing session by selecting FILE / OPEN EXISTING SESSION and then selecting the first Session “2 channels EEG” . Press “LOAD SESSION”.
4. The “PLAY” button at the bottom of the screen will become active which, when you press, will start the data transfer.
5. The data in the DLL will now be being updated.
6. To Stop, press the “STOP” button, to Pause the data, press “PAUSE” and then “RESUME” to continue.

Vilistus DLL – Miscellaneous functions

RR Interval Identification

The RRInterval function takes a series of raw data points and returns either the length of the interbeat gap or the current pulse rate. This function can be used with either BVP or ECG. The function is used in two modes: to initialise internal data areas and to produce the RR data.

The function is defined in Visual Basic as:

```
Public Declare Function RRInterval Lib "vilistus.dll" _
 (ByVal nRawValue As Single, ByVal nSPS As Long, ByVal nLowerLimit As Long, _
  ByVal bUpperlimit As Long, ByVal nAGC As Single, ByVal nOrder As Long, _
  ByVal nDuplicate As Long, ByVal nUseFilter As Long, ByVal nBPMFlag As Long) As Single
```

with the fields having the following meaning:

Data Name	Description	Comments[1]
nRawValue	The current data value extracted from the input array. This will be number between0 and 1023.	Value between 0 and 1023. **A value of -1 in this field re-initialises the internal data fields**
nSPS	the current number of samples per second	usually 256 but can be 512 (*the latter disables the IIR filter)*
nLowerLimit	used to sanity check the RR interval, the lowest pulse rate that qualifies	Normally 60
nUpperLimit	used to sanity check the RR interval, the highest pulse rate that qualifies	Normally 100
nAGC	the AGC factor	Normally 0.001
nOrder	The Filter Order to be used	Normally 9
nDuplicate	Whether invalid data should be replaced by the previous good value	**1 = use previous value** 0 = Return Code of -2
nUseFilter	A flag to indicate whether the IIR filter should be used to pre-condition the data	**1 = Use the Filter** 0 = Use the raw data value
nBMPFlag	A flag to indicate whether the BPM or Interbeat gap should be returned	1 = Return Pulse Rate **0 = Return Interbeat gap**

The function returns either the data value computed from the input array or -1 to indicate that no RR interval has been computed in this call to the function.

1 Highlighted values are the default