S.N.A.P

Protocol Encoder/Decoder DLL

Limitations in this Beta release (DLL version 0.90).

- FEC not implemented.

This is a **beta version** please check our web for updated versions of the **S.N.A.P** Protocol Encoder/Decoder DLL.

1.0 Introduction

This is the preliminary user manual for the Windows 32-bit **S.N.A.P** DLL that can be used with VB, VC, Delphi or any other programming language capable to calling DLL's. The easiest way to start using it if you are a beginner is to study the simple Delphi and Visual Basic examples and use this document as a reference. Before you can start using it copy the SNAP.DLL into your Windows system directory (usually c:\windows\system\).

1.1 What it will do for you.

The **S.N.A.P** DLL performs encoding and decoding of data for the **S.N.A.P** network protocol. What it doesn't do is send or receive this data over serialport or other media. That is left to you to implement. However we will show you how to do this as a very simply matter in many of our Visual Basic examples. To use network terms, your program has to be both the application and the link layer.

1.2 How to use the S.N.A.P DLL

If you want to send a packet you feed the DLL with the data that you want to send, and also give it the destination address. Use the DLL call **SendData** to do that. It will then return an encoded SNAP packet for you. It will calculate and include the error detection mode of your choice. The returned packet is simply an array of bytes that you can send to your microcontrollers on the media that you are using. Use the **GetTXPacket** call to get the encoded packet.

If you on the other hand receive a packet from your serial port from a controller, you just feed the information byte-by-byte to the **RXByteIn** function and the DLL will decode the information for you. It will determine if it's addressed to you, check for CRC errors, and finally return the decoded data to you with the **GetRXData** call.

There is also a smart little thing called the SPYBuffer that you can get data from. When you have several nodes that are communicating with each other, the DLL is "eves-dropping" on this traffic and stores a copy of all their data in the SPY buffer. You can get this data by calling the **GetSPYData** function. For this to work you of course have to feed the **RXByteIn** routine with data from your network Another neat tool for evaluating your **S.N.A.P** network is the built in statistic functions. The DLL keeps track on sent, received and rejected packets/bytes from/to your node as well as the total packets/bytes sent on the network! You get the statistics by the **GetStatistics** call.

Take a look at the figure on the next page to se how the calls operate.

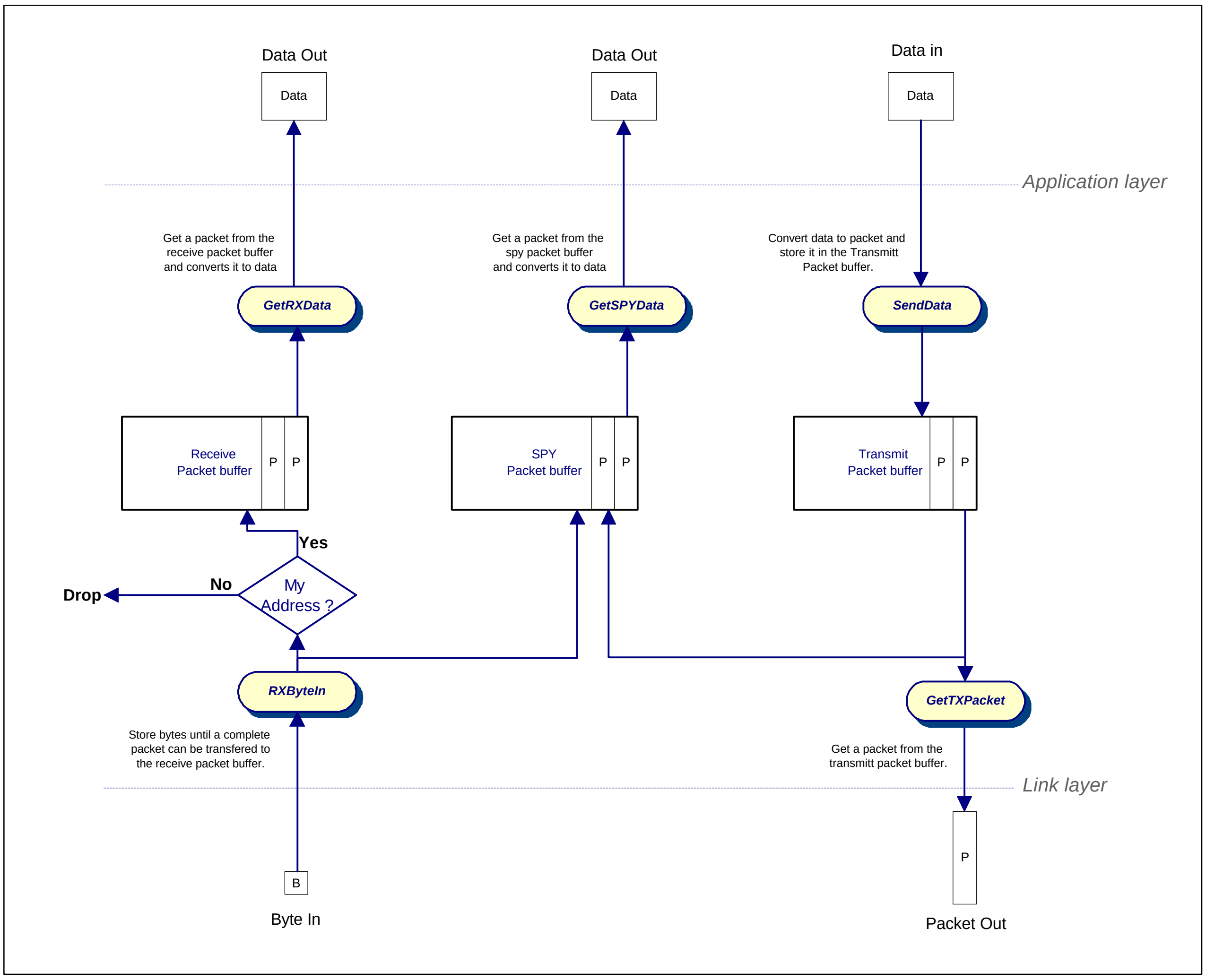
Data Out
Data
Data Out
Data
Data in
Data
Application layer
Get a packet from the receive packet buffer and converts it to data
Get a packet from the spy packet buffer and converts it to data
Convert data to packet and store it in the Transmitt Packet buffer.
GetRXData
GetSPYData
SendData
Receive Packet buffer
P
P
SPY Packet buffer
P
P
Transmit Packet buffer
P
P
Yes
No
Drop
My Address ?
RXByteIn
GetTXPacket
Store bytes until a complete packet can be transfered to the receive packet buffer.
Get a packet from the transmitt packet buffer.
Link layer
B
Byte In
P
Packet Out

Figure 1.0 - DLL functional diagram

2.0 DLL calls overview

Here is a list of the calls that you can make to the DLL.

Type	Call	Name	Function performed
Misc.	Func.	**DLLVersion**	Returns the verison number of the DLL
Misc.	Proc.	**SetMyAddress**	Sets your node address
Send	Func.	**MyAddress**	Returns your node address
Send	Proc.	**SendData**	Compiles a packet with data to specified address
Receive	Proc.	**GetTXPacket**	Returns the compiled packet
Receive	Proc.	**RxByteIn**	Inputs byte to the packet decoding routine
Receive	Proc.	**GetRXData**	Checks if a complete packet that is addressed to us is available
Monitoring	Proc.	**GetStatistics**	Get statistics for sent, received and rejected packets/bytes
Monitoring	Func.	**GetSPYData**	Returns data from the spy buffer

2.1 In-depth explanation off the calls.

Receive Calls.

OK, Let's start with **RXByteIn** call. As seen in the diagram we need to pass a byte to it.

It is declared as: *Function RXByteIn (InByte: Byte): Boolean*$^{(16)}$

The Boolean value returned is false if the receive buffer is full. In that case you have filled up the receive buffer by inputting data with **RXByteIn** in a faster rate than you have emptied it with **GetRXdata** calls. Normally this should not happen if you have coded your program correctly. In the current version of the DLL the receive buffer can store 50 packets before overflow occurs.

GetRXdata will take complete packets from the receive buffer.

It is declared as: *Function GetRXData (RXData:Type_RXData):Boolean*$^{(16)}$

The call returns true if there was a packet in the receive packet buffer that it could convert to data. If the buffer was empty the call returns false. The returned data is a record called Type_RXData.

It is defined as :

```
Type_RXData
     DestAdd        :LongInteger(32)
     SourceAdd      :LongInteger(32)
     Data           :Type_DataBytes
     NumOfDataBytes :Integer(16)
     EDMByte        :Byte
     ACKBits        :Byte
     CMDBit         :Byte
     RXTime         :Date
```

DestAdd is the destination address that the packet was sent to. This either the address that you set with **SetMyAddress** call or 0 if it was transmitted as a broadcast. Usually you don't bother with this variable, but if your application wants to sort out the information that was explicit sent to you and the broadcasted data it may use this variable to do that. It can also have the special value of -1. -1 indicates that no destination address was included in the packet. Normally packets that has no destination address is dropped by the receive decoding routines. However, if you **SetMyAddress** to -1 you indicate that you want to have all packets that has no destination address.

SourceAdd is the address of the sending node that transmitted the data. If no source address was sent it will be given the value of -1.

Data is the transmitted data. It may be 0 to 512 bytes. Data is an array of byte from 1 to 512

It is defined as:

```
Type_DataBytes= Array [1..512] of Byte
```

NumOfDataBytes indicates how many bytes were passed in the data array.

EDMByte returns what Error detection mode the sending node used to transmit the data to you. Usually this is not a very interesting thing for you program but it may be useful to know if you want to send an acknowledge packet back to the sending node. In that case you should send the ACK back with the same error detection mode.

ACKBits passes the ACK bits from the packet header. You may use this to implement your own ACK/NAK scheme. 0 means that the sending node don't want an ACK in return. 1 means that the sending node do want an ACK in response. You should then send a packet back to the sending node using the **SendData** call. Use the same Error detection mode and set ACKBits to 2 in that case. 2 means that you have received an ACK from the node. This can only happen if you have sent a packet to the node previously and requested that it ACK your packet. 3 means that you have received a NAK (Negative acknowledge) from a node. This is a rare case and can only happen if you have sent data to a node and requested an ACK from it and it has for some reason decided that your data was not valid or correctly received.

CMDBit passes the CMD bit from the packet header. 0 means that command mode is disabled and 1 means command mode is enabled. For more information about the optional command mode see **S.N.A.P** protocol description.

RXTime is a timestamp of the packet when it was received by the RXByteIn routine. It is a standard OLE Date. I.e. the date is calculated with reference to midnight of December 1899. The integral part is the date and the fractional part is the time.

GetSPYdata will take complete packets from the spy buffer. It is identical to GetRXData in is call and parameters. However the spy buffer contains all transmitted and received packets regardless of the destination address. You may use this routine to "spy" on your network traffic and evaluate the communication from or to other nodes in your network.

Transmit Calls.

The **SendData** call will take your data and encode it to a packet and place this packet in the transmit buffer.

It is declared as: *Function SendData (TXData:Type_TXData) :Boolean*$^{(16)}$

The boolean value will return false if the transmit buffer was full. In that case you have filled up the transmit buffer by inputting data in a faster rate than you have emptied it with **GetTXPacket** calls. Normally this should not happen if you have coded your program correctly. In the current version of the DLL the transmit buffer can store 50 packets before overflow occurs.

The TXData is a record defined as:

```
Type_TXData
     DestAdd        :LongInteger(32)
     Data           :Type_DataBytes
     NumOfDataBytes :Integer(16)
     EDMByte        :Byte
     ACKBits        :Byte
     CMDBit         :Byte
```

DestAdd is the destination address that the data shall be sent to.

Data is the data that you want to send. It may be 0 to 512 bytes.

NumOfDataBytes indicates how many bytes you passed in the Data array.

EDMByte indicates what error detection mode you want to use.

ACKBits is the ACK bits that you want to include in the packet header. You may use this to implement your own ACK/NAK scheme. 0 means that the receiving node doesn't have to send an ACK in response of your packet. 1 means that the receiving must send an ACK in response to your packet. 2 means that you have previously received a packet from a node and you are now sending an ACK response back to it. 3 means that you have received a packet previously from a node and have decided that you need to have it re-transmitted to you.

CMDBit is the CMD bit in the packet header. 0 means that command mode is disabled and 1 means command mode is enabled. For more information about the optional command mode see **S.N.A.P** protocol description.

GetTXPacket gets a packet from the transmit buffer.

It is declared as: *Function GetTXPacket (Packet:Type_Packet): Boolean*$^{(16)}$

The Boolean value is false if the buffer is empty.

Packet is a record defined as:

```
Type_Packet
     NumOfBytes    :Integer
     PacketBytes   :Type_PacketBytes
     Time          :Date
```

NumOfDataBytes indicates how many bytes that were passed in the **PacketBytes** array.

PacketBytes is really the packet itself. It's an array of byte from 1 to 1042.

It's declared as: Type_DataBytes=[1..512] of Byte.

Time is a timestamp of the packet when it was encoded by the **SendData** routine. It is a standard OLE Date. You have limited use of this data. Future versions of the DLL or software may make use of this field.

Miscellaneous Calls.

SetMyAddress set your node number.

It is declared as : *Procedure SetMyAddress (Address : LongInteger*$^{(32)}$*)*

Address is the node address that you want to assign to your computer. You should do this first in your program. The receive calls will use this address to sort out which packets that are addressed to you and the transmit routines will include this address as the source address in its packets.

MyAddress returns your address.

It is declared as : *Function MyAddress :LongInteger*$^{(32)}$*)*

DLLVersion returns the version number of the DLL.

It is declared as: *Function DLLVersion :Type_Version*

Type_Version is a record defined as:

```
Type_Version
     Major  :Byte
     Minor  :Byte
```

Major is the major version.

Minor is the minor version .

Combine these two numbers in the format major.minor to get the version number.

For more information See Delphi, VB4 and VB6 examples for more information how to declare the calls and to get information on the data structures that is passed to or from the calls.

That's all folks!

3.0 Bug reports.

If you find a bug that you are able to repeat than I am interested to know about it. Please report the bug by using our webform on the following URL.

http://www.hth.com/snap/dllbugs.htm

3.1 S.N.A.P Protocol Encoder/Decoder DLL history.

1999-02-01 **S.N.A.P** Version 0.83 (Beta).

First public beta release.

1999-04-01 **S.N.A.P** Version 0.84 (Beta).

Second public beta release.

1999-09-27 **S.N.A.P** Version 0.90 (Beta).

Third public beta release.

4.0 PLM-News mailing list.

If you want to stay up-to-date with **S.N.A.P** you can subscribe to our PLM-News mailing list. We will announce updates, new products, tips & tricks when available.

To subscribe to the PLM-News mailing list send an e-mail to...

listserv@hth.com

with the following in the body...

subscribe plm-news <e-mail address>

Substitute "<e-mail address>" with your own e-mail address.

After you subscribed you will receive a confirmation via e-mail and there after receive every issue of PLM-News.

4.1 More information.

More information about **S.N.A.P** and the **PLM-24** Power Line Modem can be found at the URL's given below. If you are interested to implement **S.N.A.P** in any commercial application please send e-mail for free licensee registration.

info@hth.com

Information about **S.N.A.P** and **PLM-24** Power Line Modem can be found at...

http://www.hth.com/snap/
http://www.hth.com/plm-24/

Disclaimer of Liability.

High Tech Horizon
Asbogatan 29 C
S-262 51 Angelholm
SWEDEN

E-mail: info@hth.com
WWW: http://www.hth.com