S.N.A.P

Scaleable Node Address Protocol

Protocol version	**1.00**
Document revision	**1.04**

Preface.

Thanks for your interest in the **S.N.A.P** network protocol. Our goal was to define a simple and generic network protocol that could be used in many different types of microcontroller applications as well for educational purposes. We are not implying that **S.N.A.P** will solve all network problems in this universe because it won't! There may be other protocols that suit your specific application better. Never the less we think **S.N.A.P** is a great protocol for it's intended use and therefore we decided to share it with the rest of the community. We have used it - and are still using it - for lots of various projects.

Priority has been to keep it as simple as possible and our internal mantra has been KISS (Keep It Simple Stupid). It's harder than one can imagine balancing technical advanced functions with simplicity and we have had many ideas that we scraped due to the reason that it would increase the learning curve and/or make it harder to implement. The persons involved in the development of **S.N.A.P** all have many years background as teachers and knows how important it is to keep things simple for easy understanding. Something to remember is that you many times can build advanced functions even if you are using very simple tools!

One drawback with defining such a flexible network protocol is that it makes it almost impossible to define everything since many parameters (such as timing etc.) depends on what kind of media being used. We could have defined **S.N.A.P** to be used with our (now obsolete) **PLM-24** Power Line Modems only and with strict timing information but then it wouldn't be so versatile as we wanted it.

This document version specifies the **S.N.A.P** packet framing format. Consider **S.N.A.P** as an ongoing project that we will continue to work on and add more functionality to over time. We hope you find our work useful and welcome you to e-mail comments, suggestions and ideas. We are not always able to reply to you personally but we do read all e-mail received.

Note that the purpose of this **S.N.A.P** documentation is not to be a complete book with answers on every question that may arise (we may eventually do something like that in the future). However, we have done our best in given time to provide enough information to get you started and the best way to learn how **S.N.A.P** works in reality is to start experiment with it and study the source code examples that is available on our web-site.

In short, if you find **S.N.A.P** useful then feel free to use it and all we ask is that you give credit where credit is due and if **S.N.A.P** doesn't suit your specific needs then you should consider using another network protocol.

Good luck with your projects!

1.0 Introduction.

Why yet another protocol, aren't there enough already? Because we at HTH needed a protocol for our (now obsolete) **PLM-24** based home automation system. After studying lots of other protocols available we decided to sit down and define a generic protocol that gave us the flexibility we was looking for. We wanted a protocol that could easily be implemented in small microcontrollers with **very** limited computing and memory resources. We also wanted to be able to use the same protocol in larger systems so the solution was to make the protocol scaleable.

S.N.A.P allows for different packet length and different protocol complexity. It can be used as a very simple protocol without any flags or error detection, or the programmer can use up to 24 flags and any of the defined error detection methods, depending on the current need (or personal skill). Since **S.N.A.P** is scaleable, both simple (read as cheap) and sophisticated nodes can communicate with each other in the network.

That is what makes **S.N.A.P** unique!

Features.

- Easy to learn, use and implement.
- Free and open network protocol.
- Free development tools available.
- Scaleable binary protocol with small overhead.
- Requires minimal MCU resources to implement.
- Up to 16.7 million node addresses.
- Up to 24 protocol specific flags.
- Optional ACK/NAK request.
- Optional command mode.
- 8 different error detecting methods (Checksum, CRC, FEC etc.).
- Can be used in master/slave and/or peer-to-peer.
- Supports broadcast messages.
- Media independent (RF, RS-485, PLC, IR, Inter IC, etc)
- Works with simplex, half-, full- duplex links.
- Header is scaleable from 3-12 bytes.
- User specified number of preamble bytes (0-n).
- Works with synchronous and asynchronous communication.

1. 1 What can S.N.A.P be used for?

We developed **S.N.A.P** primarily for use in home automation and control systems but it is a generic protocol and not limited to this. **S.N.A.P** can be used in any type of applications where an easy to learn and flexible network protocol is needed.

1.2 How small MCU's can be used?

S.N.A.P can be implemented in almost any MCU available today. Our first **S.N.A.P** testprogram ran on a BASIC Stamp I from Parallax Inc. It sent 1 Byte data and used 16-bit CRC as error detection method. This tiny microcontroller has only 14 Bytes of available RAM. As another node in the network we used a standard PC and the nodes were communicating over the mains using (the now obsolete) **PLM-24** Power Line Modems.

1.3 Is S.N.A.P easy to learn?

As mentioned before it can be scaled down and used as a very simple protocol and is therefore easy to learn. This also means it's great for educational purposes and for electronic hobbyists. And many of the professionals appreciate it, since it's easy to implement and a very flexible protocol.

1.4 Using S.N.A.P in commercial applications.

S.N.A.P is free for private and commercial (requires a vendor ID#) use. All that we ask in return is that you give credit where credit is due and that you enclose the latest version of this original PDF-file (or a link to our web-site) with your applications/products.

If you intend to use **S.N.A.P** in any commercial application you **<u>must</u>** request a vendor ID#. This will not cost anything. For details on how to request your own vendor ID# see our web-site at...

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

1.5 Support.

S.N.A.P is released "AS IS" and we are not able to provide any free support.

1.6 Future development of S.N.A.P.

We reserve us the right to change, modify or enhance the **S.N.A.P** network protocol without any prior notice. Our intention is to enhance **S.N.A.P** with more features in the future. To stay up-to-date with the development of **S.N.A.P** feel free to join our mailing list, details at the end of this document. We welcome any feedback and suggestions from users. We are aware about some missing information such as collision detection an recommended time-out values. This will be eventually be included in a future revision of this document.

2.0 Protocol description.

Below is the structure of **S.N.A.P** described. Before we begin a few words of explanation. All communication between network nodes, is in the form of packets. These packets can be of different length. The total packet length will depend on how many address bytes (0-6 Bytes) you decide to use, how many flags bytes (0-3 Bytes), how much data you want to send (0-512 Bytes) and what error detection method you use. All of this is defined in the header definition bytes (**HDB2** and **HDB1**).

Each packet could be preceded with optional preamble bytes (0-n). The packet starts with a unique byte (01010100_2). This byte is called the synchronization byte. Any type of preamble characters can be used as long as they are not the same as the synchronization byte.

In the example below you see a small **S.N.A.P** packet with the following structure.

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB1	Destination Address Byte
SAB1	Source Address Byte
DB1	Data Byte 1
CRC2	High byte of CRC-16
CRC1	Low byte of CRC-16

n bytes preamble	**SYNC**	**HDB2**	**HDB1**	**DAB1**	**SAB1**	**DB1**	**CRC2**	**CRC1**

The total length of this packet would be 8 Bytes (excluding the optional preamble bytes). All bytes within a group are positioned with the least significant byte to the right.

2.1 Synchronization byte - SYNC.

This byte is pre-defined to 01010100_2 and indicates the start of each packet.

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

2.2 Overview of header definition bytes (HDB2 and HDB1).

The first two bytes after the synchronization byte are called header definition bytes, they are used to define the structure of the complete packet. The name on the fields are chosen to be easy to remember.

HDB2

Bit	7	6	5	4	3	2	1	0
	D	D	S	S	P	P	A	A
	DAB		**SAB**		**PFB**		**ACK**	

HDB1

Bit	7	6	5	4	3	2	1	0
	C	E	E	E	N	N	N	N
	CMD	**EDM**			**NDB**			

DAB = Number of **D**estination **A**ddress **B**ytes
SAB = Number of **S**ource **A**ddress **B**ytes
PFB = Number of **P**rotocol specific **F**lag **B**ytes
ACK = **ACK**/NAK bits
CMD = **C**o**M**man**D** mode bit
EDM = **E**rror **D**etection **M**ethod
NDB = **N**umber of **D**ata **B**ytes

2.3 Header Definition Byte 2 - HDB2.

Bit	7	6	5	4	3	2	1	0
	D	D	S	S	P	P	A	A

Bit 7 and 6 - Destination Address Bytes (DAB)

These two bits defines number of destination address bytes in the packet. With the maximum size of 3 Bytes it gives a total of 16 777 215 different destination node addresses.

Bit 7	6	
0	0	0 Byte destination address
0	1	1 Byte destination address
1	0	2 Bytes destination address
1	1	3 Bytes destination address

Bit 5 and 4 - Source Address Bytes (SAB)

These two bits defines the number of source address bytes in the packet. With the maximum size of 3 Bytes, it gives a total of 16 777 215 different source node addresses.

Bit 5	4	
0	0	0 Byte source address
0	1	1 Byte source address
1	0	2 Bytes source address
1	1	3 Bytes source address

Bit 3 and 2 - Protocol specific Flag Bytes (PFB)

These two bits defines how many protocol specific flag bytes the packet includes, from 0-3 Bytes which give a total of 24 flags.

Bit 3	2	
0	0	0 Byte flags
0	1	1 Byte flags
1	0	2 Bytes flags
1	1	3 Bytes flags

Bit 1 and 0 - ACK/NACK Bits (ACK)

These two bits defines if the sending node requests an ACK/NAK packet in return. These bits also acts as the actual ACK/NAK response sent from the receiving node.

Bit 1	0	
0	0	No ACK request (Tx)
0	1	ACK request (Tx)
1	0	ACK response (Rx)
1	1	NAK response (Rx)

2.4 Header Definition Byte 1 - HDB1.

Bit	7	6	5	4	3	2	1	0
	C	E	E	E	N	N	N	N

Bit 7 - Command mode bit

This bit indicates what's called command mode. This is an optional feature and if a node is not implementing it this bit should always be set to zero (CMD=0).

A node implementing this feature will be able to respond on queries from other nodes as well as send responses when for example the receiving node can't handle the packet structure in a received packet. It can be used to scan large networks for nodes and have them respond with their capabilities or for two nodes negotiating the right packet structure, among other things.

If this bit is set (CMD=1) it indicates that the data in DB1 contains a command (query or a response). This results in total 256 different commands.

The range is divided in two half's, commands between 1-127 are queries and commands between 128-255 are responses. The commands specified to date are the following. Note this is the value in DB1, not the actual CMD bit.

CMD	Query	CMD	Response
0	Command mode supported?	128	Command mode supported
1	Preferred packet structure?	129	Preferred packet structure
2	Reserved but not yet defined	130	Reserved but not yet defined
...	...	...	...
127	Reserved but not yet defined	255	Reserved but not yet defined

There are some things to think about for this to work properly. The sending node can not use an higher address range than the receiving node. This is not a problem if the receiving nodes that are implementing this feature are capable to handle all the address range (i.e. 1-16 777 215). Another solution is to assign all masters in the network (in a master/slave network) to the low address range (i.e. between 1-255).

Bit 6 to 4 - Error Detection Method (EDM)

These three bits defines what kind error detecting method is being used to validate the packet. A node does not need to support any error detection method at all (i.e. EDM = 0) or you can choose to implement a subset or all of them. For more information about this topic see section 2.6.

Bit 6	5	4	
0	0	0	No error detection
0	0	1	3 times re-transmission
0	1	0	8-bit checksum
0	1	1	8-bit CRC
1	0	0	16-bit CRC
1	0	1	32-bit CRC
1	1	0	FEC (specific FEC standard TBD)
1	1	1	User specified

TBD = To Be Determined

Bit 3 to 0 - Number of Data Bytes (NDB)

These four bits defines how many bytes data there is in the packet (0-512 Bytes).

Bit 3	2	1	0	
0	0	0	0	0 Byte
0	0	0	1	1 Byte
0	0	1	0	2 Bytes
0	0	1	1	3 Bytes
0	1	0	0	4 Bytes
0	1	0	1	5 Bytes
0	1	1	0	6 Bytes
0	1	1	1	7 Bytes
1	0	0	0	8 Bytes
1	0	0	1	16 Bytes
1	0	1	0	32 Bytes
1	0	1	1	64 Bytes
1	1	0	0	128 Bytes
1	1	0	1	256 Bytes
1	1	1	0	512 Bytes
1	1	1	1	User specified

2.5 Protocol specific Flag Bytes (PFB3-PFB1).

These three flag bytes are **reserved** and intended for future enhancements of **S.N.A.P**. They may be defined in a future protocol version and should **not be used**!

Below are some ideas for their use and we welcome any further comments and suggestions.

- Remote reset
- Remote re-programming
- Remote configuration
- Media traverse flag
- Data encryption flag
- Extended command mode
- Routing information flag
- Repeater flag
- Packet counter
- Time sync flag
- Packet numbering
- Packet priority levels
- Data streaming

Bit 7 to 0 - PFB3

Bit	7	6	5	4	3	2	1	0
	R	E	S	E	R	V	E	D

Eight protocol specific flags. Specific functions TBD.

Bit 7 to 0 - PFB2

Bit	7	6	5	4	3	2	1	0
	R	E	S	E	R	V	E	D

Eight protocol specific flags. Specific functions TBD.

Bit 7 to 0 - PFB1

Bit	7	6	5	4	3	2	1	0
	R	E	S	E	R	V	E	D

Eight protocol specific flags. Specific functions TBD.

2.6 General information about different EDM.

Below is a brief overview of the different error detection methods supported by **S.N.A.P**. Some of you may ask why there are more than one error detection mode available?

The reason to why **S.N.A.P** supports different error detection methods comes from the idea of making a light weighted generic network protocol that could be used over any media available, such as cables, RF, IR, power line (PLC), inter IC etc. Also by offering none or simple error detecting methods very simple **S.N.A.P** nodes (for example using just logic gates) could coexist with more sophisticated nodes on the same network.

A simple way to view it is that some medias are more reliable than others. Using a 32-bit CRC checksum on data sent just a few centimeters between two MCU's directly connected to each other on the same PCB (i.e Inter IC) would probably be overkill. In this case it would be wiser to use no error detection at all since it's very rare packets get corrupted in a design like this.

But if the same data should be sent over RS-485, RF or the power lines where noise and other disturbances are a fact you would probably want to use a higher degree of error detection to make sure that the data sent and received are correct.

It's not only the type of media that you have to take into consideration when selecting a error detection method. Factors like datapacket size, preferred overhead and if the communication link is bi-directional all affects the decision.

S.N.A.P supports the following error detection methods...

No error detection

This one is obvious. Packets will be sent without any error detection information at all.

3 times re-transmission

This is a very simple way of error detection that is very easy to implement. The sending node send the exact same packet three times.The receiving node just compare the received packets and if 3 equal packets are received then the data is assumed to be OK.

8-bit checksum

This method adds one byte (8-bits) containing the checksum at the end of each packet. The calculation of the checksum is simple and all that’s done is that all bytes (except the SYNC byte) is summed together and the 8-bit result is the checksum byte. This is sometimes referred as Longitudinal Redundancy Check (LRC) in some documentation.

8-bit CRC

This method also adds one byte (8-bits) containing the checksum at the end of each packet. The calculation of the CRC checksum is a bit more sophisticated than 8-bit checksum and therefor this is a bit more reliable.

16-bit CRC

This method adds two bytes (16-bits) containing the checksum at the end of each packet. The calculation of the CRC checksum is similar to 8-bit CRC but a bit more reliable due to the increased size of the checksum (16-bits).

32-bit CRC

This method adds four bytes (32-bits) containing the checksum at the end of each packet. The calculation of the CRC checksum is similar to 8-bit and 16-bit CRC but even more reliable.

FEC

Specific FEC standard to be determined. FEC (Forward Error Correction) is often used in simplex RF links since it offers the possibility to not only detect but also to correct corrupt data. There are many different "standards" available.

User specified

If none of the above error detection methods suits your application you can use your own.

A last note on error detection methods supported in **S.N.A.P**. There are many different error detection algorithms available. We have chosen some of the most common ones. If you are interested to learn more details about CRC (Cyclic Redundant Check) calculations we recommend you to read "A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS" written by Ross N. Williams. This document can be obtained from the following URL...

http://www.hth.com/filelibrary/snap/crc_v3.txt

2.6 CRC polynoms.

Below are the polynoms used for the different CRC error detection modes supported in **S.N.A.P**.

8-bit CRC (a.k.a DOW CRC)

X^8+X^5+X^4+1

Polynominal = h18
Initial reminder = h00

16-bit CRC (a.k.a CRC-CCITT)

X^16+X^12+X^5+1

Polynominal = h1021
Initial reminder = h0000

32-bit CRC (a.k.a Ethernet standard)

X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X^1+1

Polynominal = h04C11DB7
Initial reminder = hFFFFFFFF

Note that the result is inverted.

2.7 EDM checkvalues.

Below is a table with pre calculated checksums for all different EDM modes in **S.N.A.P**. This is usefull for verifying your own calculation routines. The checksums listed is the result you should get when performing calculating on the string “SNAP” or “snap”.

EDM Mode	SNAP	snap
000 - No error detection	-	-
001 - 3 times re-transmission	-	-
010 - 8-bit checksum	h32	hB2
011 - 8-bit CRC	h11	h17
100 - 16-bit CRC	h8C43	h1F4F
101 - 32-bit CRC	h00F1F02A	h36641D9E
110 - FEC (specific FEC standard TBD)	TBD	TBD
111 - User specified	?	?

2.8 Miscellaneous information.

- Address 0 is reserved as a broadcast address and should not be used for anything else.

- All packets must start with a synchronization byte (01010100_2).

- The synchronization byte is not included in the error detection calculation.

- If a node is capable to use 2 or 3 Bytes DAB it should also be capable to decode 1 or 2 Bytes DAB for compatibility.

- If a node is capable to use 2 (or 3) Bytes SAB addresses lengths but configured to use an low address that “fits” whitin 1 Byte (i.e 1-255) it should use 1 Byte SAB length for all it’s outgoing packets for compability with nodes that only can handle 1 Byte addresses.

- If you are sending packets over media's like power line, RF or IR keep the packets small (less than 40 Bytes) to improve performance.

3.0 Examples.

Below are some additional examples on how **S.N.A.P** packets may look like. In the examples the optional preamble bytes isn't shown. More examples on bit level can be found in appendix A.

Example 1

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB1	Destination Address Byte
SAB1	Source Address Byte
DB1	Data Byte 1
CHK	8-bit checksum

SYNC	HDB2	HDB1	DAB1	SAB1	DB1	CHK

Example 2

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB2	Destination Address Byte 2
DAB1	Destination Address Byte 1
SAB2	Source Address Byte 2
SAB1	Source Address Byte 1
DB2	Data Byte 2
DB1	Data Byte 1
CRC2	High byte of CRC-16
CRC1	Low byte of CRC-16

SYNC	HDB2	HDB1	DAB2	DAB1	SAB2	SAB1	DB2	DB1	CRC2	CRC1

Example 3

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB1	Destination Address Byte
SAB1	Source Address Byte
DB1	Data Byte 1
CRC4	High byte of most significant word in CRC-32
CRC3	Low byte of most significant word in CRC-32
CRC2	High byte of least significant word in CRC-32
CRC1	Low byte of least significant word in CRC-32

SYNC	HDB2	HDB1	DAB1	SAB1	DB2	DB1	CRC4	CRC3	CRC2	CRC1

Example 4

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB1	Destination Address Byte
SAB1	Source Address Byte
DB8	Data Byte 8
DB7	Data Byte 7
DB6	Data Byte 6
DB5	Data Byte 5
DB4	Data Byte 4
DB3	Data Byte 3
DB2	Data Byte 2
DB1	Data Byte 1
CRC	CRC-8

SYNC	HDB2	HDB1	DAB1	SAB1	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	CRC

Example 5

SYNC	Synchronization byte
HDB2	Header Definition Byte 2
HDB1	Header Definition Byte 1
DAB3	Destination Address Byte 3
DAB2	Destination Address Byte 2
DAB1	Destination Address Byte 1
SAB3	Source Address Byte 3
SAB2	Source Address Byte 2
SAB1	Source Address Byte 1
PFB1	Protocol specific Flag Byte
DB1	Data Byte 1
CRC2	High byte of CRC-16
CRC1	Low byte of CRC-16

SYNC	HDB2	HDB1	DAB3	DAB2	DAB1	SAB3	SAB2	SAB1	PFB1	DB1	CRC2	CRC1

4.0 FAQ - Frequently Asked Questions.

Q. Scaleable? Can't you spell?
A. Since English isn't our native language we sometimes have spelling problems but "scaleable" is spelled right. Both scalable and scaleable could be used according to the language experts.

Q. What is/was the **PLM-24** Power Line Modem and where can I get it?
A. It was a power line modem that sent serial data over the mains at 2400 bps. The **PLM-24** product line are discontinued and not available anymore due to component obsolescence.

Q. Examples uses a **PLM-24**, must I have one to use/test **S.N.A.P**?
A. No. **S.N.A.P** is a generic network protocol and you could use it over any media you like. Easiest way to get started is to hook up a PC to a microcontroller (or another PC by using a null modem cable) via the serial port or use a simple RS-485 network.

Q. When do you want to use zero bytes DAB and SAB?
A. This is useful if you for example connect two microcontrollers directly to each other in a point-to-point communication. Since there only is two nodes on the "network" there is no need for any addresses and thus saving protocol overhead.

Q. Must a node support all EDM modes?
A. No, you can choose to just implement one or a subset of them.

Q. What are the best EDM?
A. Selecting correct EDM depends on several factors but we recommend 16-bit CRC as a good generic EDM. It offers a pretty high degree of security without adding to much overhead.

Q. Should the SYNC byte be included in the error detection calculation?
A. No.

Q. Whats the purpose of the pre-amble bytes?
A. The optional pre-amble bytes is used to calibrate the so called data-slicer in some hardware such as RF receivers etc.

Q. Must 01010101_2 be used as pre-amble character?
A. No, you can use whatever pre-amble character you want except for 01010100_2.

Q. If a node is capable to use 2 or 3 bytes addresses it should also be able to decode 1 or 2 bytes addresses . Why?
A. What we mean is that both simpler one address byte nodes, and more sophisticated two and three address byte nodes could co-exist on the same network. For exampel a two address byte node could communicate with a "one byter" if the "two byter" has an address below 256 and it sends that address in one byte. So the "two byter" becomes a "one byter" when it's address by such a node. The same reasoning applies to three versus two and one address byte nodes.

Q. How do I do if I want to send 9-bytes data?
A. Since there isn't any support for 9-bytes data you need to select the nearest higher number of data bytes supported. In this case 16-bytes.

Q. Is **S.N.A.P** free to use?
A. Yes, it's totally free to use for anyone but if you intend to use it in any commercial application you must register a Vendor ID#.

Q. Why do I need to register a Vendor ID for commercial products?
A. The main reason for the required Vendor ID is to set us free from any potential legal liabilities (see Disclaimer of Liability at the end of this document) but it also gives us an indication on how many are using the **S.N.A.P** protocol.

Q. Does it cost anything to register a Vendor ID# for our company?
A. No, just fill in the form on our web-site.

Q. You say **S.N.A.P** is free, but there is always a copyright notice in the docs. If we implement the protocol into our system, we must supply the origin docs to the user. So can we distribute your origin docs with our system?
A. Yes, as long as they are not modified in any way.

Q. Are there any **S.N.A.P** tools available to speed up the learning process?
A. Yes, on our web-site there are simple program examples that shows how to implement **S.N.A.P** in a small microcontroller as well as a free Windows 32-bit DLL that takes care of the encoding and decoding of **S.N.A.P** packets.

Q. How do I use the **S.N.A.P** protocol encoder/decoder DLL?
A. Included in the ZIP-archive is a DLL-documentation that describes how to interface to the DLL as well as source code examples for Visual Basic 4, Visual Basic 6 and Delphi.

Q. Are there any **S.N.A.P** tools available for Linux?
A. Yes, a beta version of our **S.N.A.P** library is available on our web-site. We are working on update shared library for Linux and hope to release it Q4 2014. It will be available for the x86, x64 and ARM architectures. Let us know if you are interested to test it.

Q. Are there any **S.N.A.P** tools available for other operating systems?
A. There are some old test programs for PalmPilot but they are probably not relevant 2014!

Q. Is the source code for the Windows 32-bit DLL and the Linux library available?
A. No. If you would like to see support for other platforms let us know and we will see what we can do (no guarantees).

Q. Can I get any support from you?
A. Sorry, we are not able to provide any free support.

Q. What if I find a bug in some of the programs or **S.N.A.P** tools?
A. We would appreciate if you took the time to fill in the bug report on our web-site. Please include as much details as possible since we are only able to fix bugs that we are able to reproduce.

5.0 S.N.A.P network protocol history.

1998-09-09 - **S.N.A.P** Version 0.91 [Preliminary Draft].

First public release.

1999-01-20 - **S.N.A.P** Version 1.00 [Initial release].

Changes/additions to previous protocol version:

ACK/NAK structure changed.
Removed user specific flag bytes.
Added user specified number of databytes.
Command mode implemented.

5.1 Document revision history.

1998-09-09 - Document revision 0.91 [Preliminary Draft].

1999-01-20 - Document revision 1.00 [Initial release].

1999-12-30 - Document revision 1.01.

2000-02-13 - Document revision 1.02.

2002-01-04 - Document revision 1.03.

2014-08-23 - Document revision 1.04.

6.0 S.N.A.P mailing list.

The old PLM-News mailing list is history but a new announcement mailing list is available. To subscribe got to URL beloow and fill in your e-mail address.

http://www.freelists.org/list/snap

If there are any interest to have a **S.N.A.P** user discussion list or maybe a forum we will set one up. Let us know what you would prefer via an e-mail.

6.1 More information.

More information about **S.N.A.P** can be found at the URL's given below.

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

6.2 User supplied S.N.A.P examples.

If you have any source code that you would like to share with others feel free to e-mail them to us and we will add it to the list on our web-page.

We had some requests for Arduino examples lately so if you have a example for Arduino to share it would be appreciated by the Arduino community.

6.3 Feedback and suggestions.

If you have any comments or suggestions or have any program examples that are using **S.N.A.P** and want to share them with others please feel free to contact us by e-mail. We are looking forward to hear from different people using **S.N.A.P** and if you send us a short story about your use and we may publish it on our web (with your consent of course!).

snap@hth.com

6.3 Special thanks.

Special thanks to the following people for their ideas and comments.

Claus Kühnel
Mats Ekberg

Disclaimer of Liability.

Overview of header definition bytes (HDB2 and HDB1)

HDB2

Bit	7	6	5	4	3	2	1	0
	D	D	S	S	P	P	A	A

HDB1

Bit	7	6	5	4	3	2	1	0
	C	E	E	E	N	N	N	N

See section 2.0 in the manual for a detailed description of each bit position in HDB2 and HDB1. The following examples shows several packets on a bit level, to make it easier for the **S.N.A.P** beginner.

Example 1 - Packet size 8 Bytes

In the example below the transmitting node has address 00000001_2 and is sending data 11111111_2 to node 00000010_2. Since no acknowledge is required the transmitting node will not expect any ACK or NAK packet in return.

Packet structure.

DD=01	- 1 Byte destination address
SS=01	- 1 Byte source address
PP=00	- No protocol specific flags
AA=00	- No ACK request
C=0	- Command mode not supported
EEE=100	- 16-bit CRC
NNNN=0001	- 1 Byte data

Packet sent from node 00000001_2

SYNC	HDB2	HDB1	DAB1	SAB1	DB1	CRC2	CRC1
0 1 0 1 0 1 0 0	0 1 0 1 0 0 0 0	0 1 0 0 0 0 0 1	0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1	1 1 1 1 1 1 1 1	0 1 0 0 1 1 1 0	1 0 1 1 1 0 1 1

Example 2 - Packet size 8 Bytes

In the example below the transmitting node has address 00000001_2 and is sending data 11110000_2 to node 00000011_2. This time acknowledge is requested and the transmitting node expect to receive an ACK or NAK packet, else it should time-out and take proper action.

Packet structure.

DD=01	- 1 Byte destination address
SS=01	- 1 Byte source address
PP=00	- No protocol specific flags
AA=01- ACK request	
C=0	- Command mode not supported
EEE=100	- 16-bit CRC
NNNN=0001	- 1 Byte data

Packet sent from node 00000001_2

SYNC	HDB2	HDB1	DAB1	SAB1	DB1	CRC2	CRC1
01010100	01010001	01000001	00000011	00000001	11110000	00100010	00110101

ACK packet returned from node 00000011_2

SYNC	HDB2	HDB1	DAB1	SAB1	DB1	CRC2	CRC1
01010100	01010010	01000001	00000001	00000011	00000000	00101011	11111010

NAK packet returned from node 00000011_2

SYNC	HDB2	HDB1	DAB1	SAB1	DB1	CRC2	CRC1
01010100	01010011	01000001	00000001	00000011	00000000	10000001	10101011

Note: Red color indicates changed bits/bytes.

Example 3 - Packet size 9 Bytes

In the example below the transmitting node has address 00000001_2 and is sending data 11110000_2 to node 00000011_2. This time to, acknowledge is requested and the transmitting node expect to receive an ACK or NAK packet, else it should time-out and take proper action. Further more one byte of protocol specific flags are used, these flags are set to 00000011_2 in PFB1. Also note that the ACK/NAK packet returned contains 0 Byte data.

Packet structure.

DD=01	- 1 Byte destination address
SS=01	- 1 Byte source address
PP=01	- 1 Byte protocol specific flags
AA=01- Acknowledge is required	
C=0	- Command mode not supported
EEE=100	- 16-bit CRC
NNNN=0001	- 1 Byte data

Packet sent from node 00000001_2

SYNC	HDB2	HDB1	DAB1	SAB1	PFB1	DB1	CRC2	CRC1
01010100	01010101	01000001	00000011	00000001	00000011	11110000	10011110	00001100

ACK packet returned from node 00000011_2

SYNC	HDB2	HDB1	DAB1	SAB1	PFB1	CRC2	CRC1
01010100	010101**10**	0100000**0**	**00000001**	**00000011**	00000011	**11100100**	**00101011**

NAK packet returned from node 00000011_2

SYNC	HDB2	HDB1	DAB1	SAB1	PFB1	CRC2	CRC1
01010100	010101**11**	0100000**0**	**00000001**	**00000011**	00000011	**01001110**	**01111010**

Note: Red color indicates changed bits/bytes.

S.N.A.P examples.

Below is a list of examples currently available for download. The **S.N.A.P** examples includes schematics and well documented source code so they should give you an easy start. In some examples there are room for much improvement and possibility to add more functionality, we kept it as simple as possible for easy understanding.

WIP = Work In Progress

Name	MCU	Ver.	Description
SNAP-001	BS1-IC	1.02	Turn a LED on and off
SNAP-002	BS1-IC	1.02	Lampdimmer node for PLM-24
SNAP-003	BS1-IC	1.02	Domestic AC current meter with PLM-24
SNAP-004	BS1-IC	1.02	Simple temperature node for PLM-24
SNAP-005	BS1-IC	1.02	Simple light measuring node for PLM-24
SNAP-006	BS1-IC	1.02	Air quality node for PLM-24
SNAP-007	BS1-IC	1.02	Simple humidity node for PLM-24
SNAP-008	BS1-IC	1.02	Simple 4-bit input node for PLM-24
SNAP-009	BS1-IC	1.02	WakeUp alarm node for PLM-24
SNAP-010	BS1-IC	1.02	Four channel plant moisture sensor I
SNAP-011	BS2-IC	1.02	Turn a LED on and off
SNAP-012	BS2-IC	1.02	Shows how to implement background tasks
SNAP-013	BS2-IC	1.02	PLM-24 to X-10 Gateway
SNAP-014	BS2-IC	1.02	8-bit parallel input node for PLM-24
SNAP-015	BS2-IC	WIP	Programmable light monitor node for PLM-24
SNAP-016	89C2051	1.02	Turn a LED on and off
SNAP-017	BS2-IC	1.02	IR detector alarm node for PLM-24
SNAP-018	BS2-IC	1.02	Four channel relay node with local control
SNAP-019	BS2-IC	1.02	1-8 zones security system node for PLM-24
SNAP-020	BS2-IC	WIP	DCF-77 atomic clock node for PLM-24
SNAP-021	BS2-IC	1.02	Fire alarm node for PLM-24
SNAP-022	BS1-IC	1.02	1-channel 8-bit A/D converter node
SNAP-023	89C2051	1.02	Simple 16 x 1 LCD terminal node for PLM-24
SNAP-024	89C2051	1.02	Simple 16 x 1 LCD info node for PLM-24
SNAP-025	89C2051	1.02	S.N.A.P packet spy node for PLM-24
SNAP-026	BS2-IC	1.00	Testprogram for S.N.A.P Serial for PalmPilot
-	-	-	More to come...

Distributor:

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

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