P-NET Compact

Joachim Bürmann, *IFTOOLS GmbH*

September 17, 2020

The P-NET protocol is a field-bus protocol with Multi-Master and Multi-Net capabilities and mainly used in applications with an average time requirement of several milliseconds. P-NET uses a ring topology based on RS485 with a maximum number of 125 bus participants, thereof up to 32 masters. The bus access is handled by a token management. The data transmission is asynchron with 76800 bit/sec. A P-NET ring can contains other rings (Multi-Net ability).

Bus access via token management

Only the masters have the right to access the bus by themselves. A slave is only allowed to answer one master request at one time (immediate response). P-NET uses a token based mechanism to handle the bus access for the masters. Every request must be answered by the slave in a specific time given by the token management. Afterwards the token is passed to the next master.

The tokens are managed with the help of two counters implemented in the master devices. One counter increases it's value with the bit rate when the bus is idle (idle-bus-bit-period counter). The master device counters are set to 0 with every start of a new data transmission (bus access).

When the counter reaches 40 another counter - the access counter - is incremented. If there is no master with the need of the access token (maybe there is nothing to do, or no master for the given token exists at all) the idle-bus counter counts further. Without a reset of the idle-bus counter, the access counter is now increased every 10 steps, means: When the idle-bus counter reaches 40, 50, 60 and so on.

The maximum value of the access counter is specified by the number of masters (1...32). Normally it is set to a slightly higher value of the existing bus masters (if less than the maximum) for future bus extensions. As soon as the access counter reaches its maximum value it is reset to 1 and the cycle starts again.

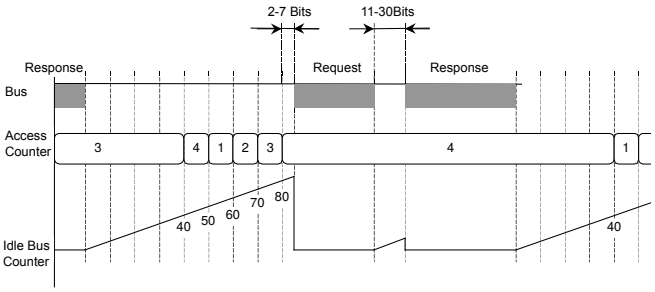


Figure 1: *Virtual Token Passing*

Telegram structure

Telegrams are transmitted as NRZ (Non-Return-To-Zero) coded UART characters with 76800 bit per second (baud rate) and a data frame frame of 9N1 (9 data bits, no parity and one stop bit).

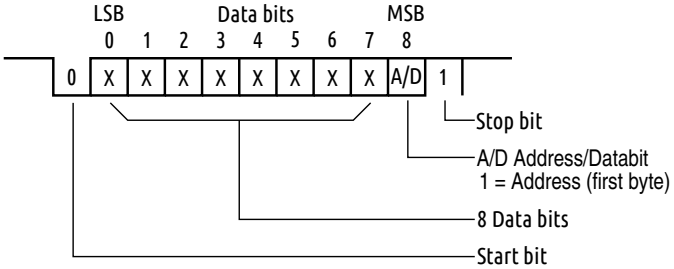


Figure 2: *Data Frame*

The usual parity bit is replaced by an address/data bit (A/D Bit) and always set (=1) in the first byte of a new telegram data sequence. All bytes of a telegram must be transmitted without an interruption, idle times between the bytes are not allowed!
A typical P-NET telegram frame looks like:

Address	Control/Status	Info-Len	Info-Field	Checksum
2..24 Bytes	1 Byte	1 Byte	0..63 Bytes	1 Byte

Table 1: *P-NET telegram structure*

- **Address** Every telegram starts with an address header of 2..24 bytes containing the destination and source/sender address(es) which allows the protocol to route telegrams even to other net segments (Multi-Net ability). The A/D bit in the very first byte of the header is always 1.
- **Control/Status** Next after the address header is the combined control/status field of 1 byte length. In a request it specifies the coded command or service (control), in a response the status of the given command or error information.
- **Info-Len** The lower 6 bits of the info field (0...5) specifies the length of the following info field. Bit 6 and 7 give additional details about the Softwire number and offset in the info field.
- **Info-Field** The info field itself provides the necessary information how to read or write certain data from/to another bus participant.
- **Info-Checksum** The last byte is the two's complement of the sum of all telegram bytes except for the checksum itself.

Addressing

P-NET is a Multi-Master system and allows up to 125 modules (masters or slaves) to be connected via one circular and 2-wired RS485 bus. Termination resistors are not necessary since of the special interface design of the modules.
A special feature of P-Net is it's integrated Multi-Net ability. Single P-NET segments can be combined to complex networks whereas multi-port gateways route the telegrams from one net segment to another. The address header includes all necessary information to send a telegram perhaps through several gateways in case the recipient stays in a further net segment.
Depending on the necessary routing (or addressing) P-NET therefore differentiates between the following address header variants:

- Simple address header (Requests and Responses)
- Complex address header (Requests only)
- Extended address header (Requests only)

The lower bits 0...6 in the address bytes specify an address (therefore max. 125 segment nodes, 0 and 127 are reserved). Bit 7 is used to differentiate between a destination and source address whereas the meaning of bit 7 is reverse for requests and responses. In combination both are used to determine a request or a response.
And: The bit 7 appearance is essential to distinguish between the three address header forms (simple, complex and extended) mentioned before.

Bit 7	Request	Response
0	Destination	Source
1	Source	Destination

Table 2: *Meaning of bit 7 in address bytes*

Simple Address

The simple address type consists of only one destination and source address and is used for addressing a bus participant in the same network segment (no gateway). In a request bit 7 of the second (source) address byte is 1.

Byte	Bit 7	Bit 6..0
1	0	Destination Address
2	1	Source Address

Table 3: *Simple Address Request*

In a response bit 7 has the reverse meaning (as mentioned before). Responses are always of the simple address type!

Byte	Bit 7	Bit 6..0
1	1	Destination Address
2	0	Source Address

Table 4: *Simple Address Response*

It is important to note, that a set bit 7 either in the first or second header byte ALWAYS indicates a simple header!

Simple Address Routing

Figure 3 shows the address header accessing a node in the same network segment.

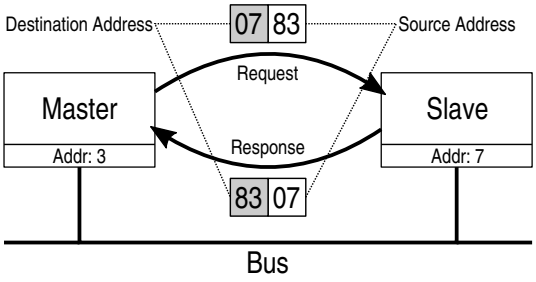


Figure 3: *Simple Address Routing*

The destination fields are grey. The address header is always two bytes long. Note that the very first byte also set the parity or A/D bit marking it as the beginning of the telegram frame which is not shown in the figure. A simple address has always a set bit 7 either in the first (response) or second byte (request). This is essential to differ a simple address from complex or extended addresses!

Complex Address

A soon as a master addresses a bus participant in another net segment (behind one or more gateways), a single source and destination address is not sufficient. Thinking of the address header as a path describing the routing way to it's destination, the header must provide information about the gateway and gateway ports beside the final destination address.
Complex addresses are only used to forward requests through gateways. The passing gateways themselves send only a short acknowledge to the direct sender in the same net segment (indicated as (IR) in figure 4). Thus the acknowledge can be of type simple address.
A complex address header looks like:
The first two address bytes (destination bytes) and also the third length byte are always with a cleared bit 7. Depending on the involved gateways more destination bytes following after the length field. They too have a cleared bit 7. This is essential, otherwise a complex header address is indistinguishable from other header types!
The last gateway stores the origin address of the message source and sends a request with a simple address header to the final recipient. The latter builds the answer message and responses back to the gateway with a simple address header.
To address the original requester, the last gateway then uses the stored address information to create a complex address header (including the final recipient as source) for routing back the answer. In order to avoid new 'immediate responses' by the involved gateways, the address header is 'closed' by a null byte.

Byte	Bit 7	Bit 6..0
1	0	Destination Address Byte 1
2	0	Destination Address Byte 2
3	0	Number of following address bytes
4	0/1	Destination/Source Address Byte 3
⋮		
n-2	0/1	Destination/Source Address Byte n-2
n-1	0/1	Destination/source Address Byte n-1
n	1	Source Address

Table 5: *Complex Address Header*

Complex Address Routing

In the following we will discuss the packet routing when accessing a node or slave in another net segment (Multi-Net Routing), see figure 4.
P-NET connects several net segments with the help of so called Multi-Port masters or gateways. A gateway is a master with at least two ports. Each port has a port number and a unique address for the net segment connected to the port.

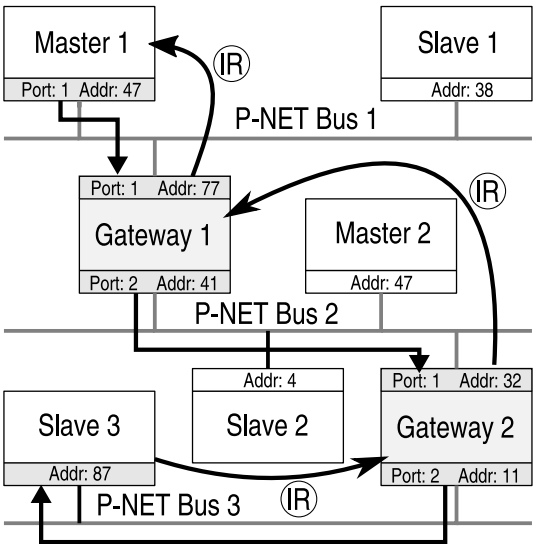


Figure 4: *Multi-Net Application*

To fulfill the immediate response (required by the token principle), every gateway confirms the received

telegram by sending the acknowledge 'answer comes later' using simple address routing (the sender is in the same segment). The destination node itself sends it's response also via simple address to the according gateway. The responses are shown as curved arrows (IR = Immediate Response).

Now let's see how the address routing takes place when Master 1 (in net segment 1) tries to contact Slave 3 (in net segment 3).
Figure 5 shows a simplified schematic of the data routing. Port numbers are displayed with a leading 'P' for better readability. In reality they are 7 bit values like the addresses. The device addresses are abbreviated as A.

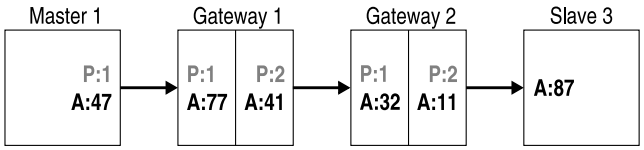


Figure 5: *Simplified Multi-Net Routing*

Please note! Sometimes a master works as a kind of gateway too - for instance if the master operates as a bridge between the P-NET and an external UDP network as described in [4]. In that case the master uses an additional address byte similar to the gateway ports to route telegrams from and to its internal forwarding port (e.g. an UDP port with port number 20).
Table 6 shows the five steps, necessary to route the telegram to it's destination. To simplify matters bit 7 (differing between source and destination) is not shown in the table.
The last two bytes in the complex header contain the master address and (as an example) an optional UDP port (20) in the master device. The optional master port does not affect the bit 7 sequence in the header bytes during the request. But we will see, that the optional master port byte makes a difference when the response is routed back to the master.

Step	Header with length field (col 3)							
Step 1	77	P2	5	32	P2	87	47	P20
Step 2	P2	32	5	P2	87	P1	47	P20
Step 3	32	P2	5	87	41	P1	47	P20
Step 4	P2	87	5	P1	41	P1	47	P20
Step 5	87	11	5	P1	41	P1	47	P20

The routing path is the address 77, port 2 (gateway 1), address 32, port 2 (gateway 2) and address 87 of the final node. The source address is 47 (Master 1

7-bit Destination Address Byte (decimal value), Bit 7 = 0
7-bit Source Address Byte (decimal value), Bit 7 = 1

Table 6: *Complex Header Generation - Request*

address). This is step 1 in the table.
Step 2 happens in the gateway. The next destination is port 2 in the gateway itself. The destination gateway address (77) is removed and the port 1 (pointing to the former segment) is add to the source addresses.
Step 3 routes the packet from gateway 1 to gateway 2 (address 32, port 2). The gateway 1 address 41 is added as the now previous source.
Step 4 is again a routing in the gateway 2 to port 2, then address 87.
At last step 5. It addresses the final recipient (address 87) and the following source bytes describe the route back to Master 1. According to [2] the last gateway stores the information for the return path and uses a simple address header for the final contact. Therefore the relevant header consists of only the first two byte of step 5.

87	11

Letting only masters (and multi-port masters like gateways) dealing with complex addresses reduces the expense for simple slaves (or servers in a client server view) when processing address headers, thus simplify their implementation.
Note again! Every gateway transmits a short acknowledge to the requester in the same net segment (IR) in figure 4 to fulfill P-NET's immediate response rule.

The way back

We start with the optional master port (UDP port 20) as described in [4]. Table 7 shows the individual steps.

Step	Header with length field (col 3)								
Step 1	11	87							
Step 2	41	P1	6	47	P20	32	P2	87	00
Step 3	47	P20	6	77	P2	32	P2	87	00

Table 7: *Multi-Net Response to Multi-Master*

The final recipient (slave 3 with address 87) builds the response telegram regarding to the requested action and answers back to gateway 2 with a simple address header (step 1 in table 7).
Gateway 2 still knows the complete return path and complete the address header for the way back to the

original sender (master 47) in P-Net Bus 1. Due to the fact, that the original sender stays in another net segment, the answer of Slave 3 is packed in a request with a complex header. Remember: Complex header addresses are limited to requests and must not used for responses!
Gateway 2 also attaches a NULL byte to the header so to avoid an otherwise immediate response by the passing gateways, see [2]. The length field therefore contains 6 and not 5, see Step 2 in table 7.
The final transmission takes place in step 3. Gateway 1 removes it's own address (41) and port (P1) from the destination and adds the route to P-Net Bus 2 (address 77, port P2) in the source fields.
Even if the telegram now addresses a participant in the same net segment, the complex part must be add to specify the source of the response!

As mentioned before: The optional master port (in our example the UDP port 20) leads to a different address header sequence when the answer of the request is routed back. Table 8 below shows the address headers without the optional master port.

Step	Header with length field (col 3)							
Step 1	11	87						
Step 2	41	P1	5	47	32	P2	87	00
Step 3	47	77	5	P2	32	P2	87	00

Table 8: *Multi-Net Response to Master*

The real difference appears in step 3 when the master receives the final response from gateway 1.
Instead of 3 destination bytes (Step 3 in table 7) the header now starts with the destination 47 (the master address) followed by the source 77 (gateway 1 address). Regarding to table 3 the address header thereby looks like a simple request.

DISCUSSION - Ambiguous address headers?

Even if the header sequence indicates a simple request the header is still of a complex address type!
But without knowing the real address type the further parsing of the telegram content becomes completely different! In case of a simple address the third byte is the control/status byte followed by the Info-Len and Info-Field.
If we handle it as a complex address, the third byte indicates the length of the following address fields including the NULL byte.
How can the final recipient - Master 1 - knows?

The author assumes that the master still waits for the requested answer. (It only got an immediate response 'answer comes later' from gateway 2 when it first sends its request to slave 3). Therefore it handles the header correct just by knowing the expected data.
Unfortunately the author of this articles didn't find any explanation to prove this assumption.
Figure 2.7 in [2] shows this final telegram with a complex header which seems ambiguous and not distinguishable from a simple address header - with all following consequences. But it does not explain the further handling.
Other books (e.g. [5]), pages and web sites don't worry about the response through gateways at all or limit their description to telegrams within a single net segment only.
The tutorial part 2 found at [4] introduces an additional number (internal master port, task, control id) which solves the problem. But it does not explain why other sources don't mention this number at all or how the responses to normal masters looks like.

Without a valid proof - for instance a record with an IFTOOLS analyzer showing the response of a former request via complex addressing - the question still remains open.
The author would be happy about any hints, example records and further information to bring more light into this question.
You can reach him under jbuermann@iftools.com.

Extended Address

The extended address type is special case of the complex type when the number of following address bytes (byte 3 in table 5) is 0. As like with complex addressing, one gateway is involved *(Is this really true? Or is an extended address only used to address a variable in a gateway? What says the P-NET specification?)*

Byte	Bit 7	Bit 6..0
1	0	Destination Address
2	0	Destination Address (Port?)
3	1	Source Address
4	1	Source Address

Table 9: *Extended Address*

Address header evaluation - conclusion

At this point it should be clear that for the right processing of a P-NET telegram it is essential to determine the correct header type. We learned that there are simple headers for requests and responses, and complex and extended headers for requests only. When parsing a P-NET transmission the beginning of each telegram is clearly visible by the set bit 8 giving a value higher than FFh.

Bit 7 in the first 1..24 address bytes plays the decisive factor to differentiate between the various headers. The following table 10 shows the 7th bit in the first 4 bytes of a P-NET telegram.

Byte 1	Byte 2	Byte 3	Byte 4	Header
0	1	x	x	Simple Request
1	0	x	x	Simple Response
0	0	1	1	Extended Request
0	0	0	x	Complex Request

Table 10: *Header differentiation*

A set bit 7 in the first byte indicates a simple response (line 2). In all other cases the header type is easily distinguishable by counting the bytes with a NOT set bit 7. Complex header addresses always beginning with at least 3 destination bytes (bit 7 = 0). Only two destination bytes means an extended address header and only one destination byte indicates a simple address.

Control/Status field

This one byte field is equal in both, request and response telegrams.

Control field

In case of a request it contains the command or service to be executed by the recipient, see table 11.

Bit 2..0	Service	Function
001	STORE	Write variable(s)
010	LOAD	Read variable(s)
011	AND	bit-wise AND operation of variable with telegram data
100	OR	bit-wise OR operation of variable with telegram data
101	TEST-AND-SET	Check, allocate and free resources
Continued on next page		

Continued from previous page		
Bit 2..0	**Service**	**Function**
110	LONG LOAD	Read data block of more than 56 bytes (max 64 kBytes)
111	LONG STORE	Write data block of more than 56 bytes (max 64 kBytes)

Bit 3	Function
0	Softwire-List addressing (default)
1	absolute addressing (service only)

Table 11: *Control field*

P-NET uses a reduced instruction set to increase the efficient and processing speed. The instructions in detail:

Store

A master transfers certain data to a slave via the STORE command. The data follows immediately after the info field which contains the according software wire - or with other words - where the data has to be stored by the recipient. The slave confirms the command with an empty info field (info-len is null) whereas the operation result is set in the status byte.

Load

With the LOAD command the master fetches data from a certain slave. The requested data is specified by a softwire number (2/4 bytes), an optional offset (2/4 bytes) and the data type (kind of variable like byte, word, ...). The size of the softwire number and of the optional offset is set in the info field, see table 14.

And

As the name indicates, this command executes an AND operation of the transmitted data with the destination data addressed like in the STORE instruction.

Or

The same as the AND instruction but now executing an OR operation of the transmitted data with the destination data addressed like in the STORE instruction.

Test and set

This instruction is used to handle resources between several masters, similar to the atomic operation in multitasking systems. With Test-And-Set a given service or resource can be checked for availability and allocated or freed again (by the owner) by one (atomic) instruction.

Long-Store

Stores a data block of more than 56 Bytes (limited by the normal total length of the Info-Field of 63 when using STORE). The maximum data block size with LONG-STORE is 64 kBytes and will be automatically segmented.

Long-Load

Reads a data block of more than 56 Bytes (limited by the normal total length of the Info-Field of 63 when using LOAD). The maximum data block size with LONG-LOAD is 64 kBytes and will be automatically segmented.

Status field

In a response this field stores the confirmation state (table 12) or an error code (table 13), differentiated by the value of the lowest bits 0..2.
The confirmation state (application status) is indicated by Bit 2..0 not 0.

Bit 6..4	Bit 2..0	Status
000	$<> 000$	ok
001	$<> 000$	busy
010	$<> 000$	data error
011	$<> 000$	not defined
100	$<> 000$	wait
101	$<> 000$	answer comes later
110	$<> 000$	not defined
111	$<> 000$	not defined

Bit 7	Function
0	No historical data error
1	Historical data error

Table 12: *Application status*

Whereas an error status is indicated by all bits 2..0 are cleared.

Bit 7..3	Bit 2..0	Status
00000	$= 000$	no repsonse
00001	$= 000$	time out
00010	$= 000$	too busy
00011	$= 000$	wait too long
00100	$= 000$	buffer full/empty
00101	$= 000$	data format error
00110	$= 000$	SWNo error
00111	$= 000$	node address error
01000	$= 000$	write protection
Continued on next page		

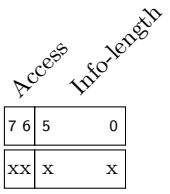
Continued from previous page

Bit 7..3	Bit 2..0	Status
01001	$= 000$	info length error
01010	$= 000$	instruction error
10000	$= 000$	error detect failure
10001	$= 000$	overrun/framing error
10010	$= 000$	net short circuit
10011	$= 000$	port not master
10100	$= 000$	out of sync
10101	$= 000$	RS-232 handshake error

Table 13: *Bus status*

Info-Len

The format of the info length field is the same for requests and responses. It specifies the length (number of bytes) of the following info (data) field and how to address the data (Software-Wire number and offset).



Bit 7	Bit 6	Function
0	0	2 Bytes SoftWire-No, no offset
0	1	2 Bytes SoftWire-No, 2 Byte offset
1	0	4 Bytes Code/SoftWire-No, no offset
1	1	4 Bytes Code/SoftWire-No, 2/4 Byte offset

Bit 5..0	Function
xxxxxx	Info length

Table 14: *Info Len Field*

Info-Field

The info field presents the data or information depending on the control/status byte. This can be a Softwire number with or without offset, the number of involved data, raw data or a LONG code (for the LONG LOAD-/STORE functions).

Checksum

The checksum is calculated by add up all telegram bytes (except for the last checksum byte) and return the sum as two's compliment. Here coded in Lua:

```lua
function chksum( data )
  local sum = 0
  for i=1,#data-1 do
    sum = sum + data:byte(i)
  end
  return bit.band( 0-sum, 0xFF );
end
```

Data types

P-NET specifies a number of predefined data types which are used for the data exchange. The number in parentheses shows the according type number in hex. Data types consisting of more than one bytes are transmitted with the highest byte (MSB) first.
And: Data types with a size of more than one byte must always start at an even address, requiring a fill byte if necessary! This has to be considered when calculating the offset, see table 14.

- Boolean (00h)
- Byte (01h)
- Word (02h)
- Integer (03h)
- Long Integer (04h)
- Real (05h)
- Long Real (06h)
- Complex (80h)

Boolean

A boolean variable is represented by one byte whereas only the LSB (bit 0) is taken into account.

Byte

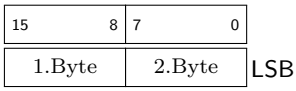
A byte is transferred as a byte and contains a value between 0 and 255.

Char

Like a byte, but it's content is interpreted as an ASCII character (ISO 8-bit code).

Word

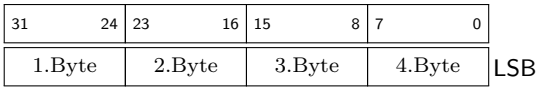
A 16 bit value formed by two bytes and with a range of 0 to 65535.



Integer

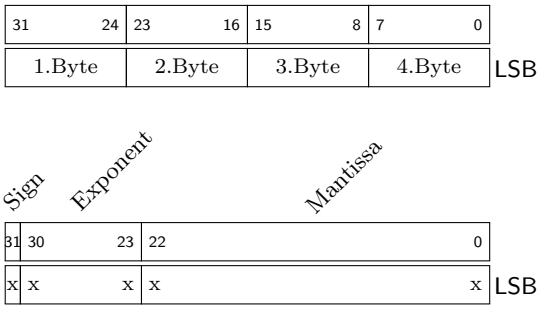
The integer type is a signed word type with a range of -32768 to 32767. It is transmitted like a word with the highest byte first.

Long Integer



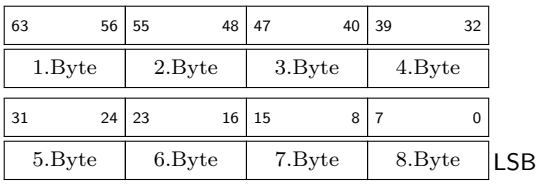
Real

The real data type is equal describes a 32-bit single precision float value according to the standard IEEE 754. It covers a range from $\approx 1 \cdot 10^{-38}$ bis $\approx 3 \cdot 10^{38}$ with a precision of 7...8 digits.



Long Real

The long real data type describes a 64-bit double precision float value according to the standard IEEE 754. It covers a range from $\approx 2 \cdot 10^{-308}$ bis $\approx 3 \cdot 2^{308}$ with a precision of 15...16 digits.



The partitioning of the long real type is similar to the real type. The sign is the highest bit, followed by a now 11 bit exponent and a 52 bit mantissa.

Complex

The complex data type covers all data without a hard coded length. Something like strings, arrays, records or general buffers of a certain length.
The optional offset in the Info Len Field (table 14) is used to access a certain data member in a complex data structure.

Channels

One of the characteristics of the P-NET protocol is the so called channel concept which is fundamental for the transport of any data. A channel covers all necessary information to handle process signals, inputs and more. For example:
A single P-NET node (or slave) reads the temperature via an analog input and temperature sensor. Without further information it's up to the application layer how to interpret this data. E.g.: How is the value stored (number format), what are the scaling factors, the physical unit, possible errors, configuration and so on. In this case all information to handle the process signal properly is separated from the module and must be adapted every time a module is exchanged or replaced by another one.
With the idea of the P-NET channel every module itself provides the application with all necessary information how to process the data. Back to our example of an analog input these should are the number format of the input value, calibration offsets, minimum and maximum values, scaling factors, precision and more.
It's clear that such a range of information is only useful if organized. Therefore P-NET specifies the structure of a channel as a basic part of the protocol, providing a common valid interface for accessing even different process signals.

Channel structure

P-NET organizes the internal channel information (process signal, scale, ...) as a sequence of 16 register variables. It is important not to mix up a channel register with a - for example - CPU register. Channel registers exists in any of the former data types. A register can consists of only one byte but also as a record of different data types.
According to the P-NET standard every channel register has to serve a well defined purpose. Table 15 below shows the mandatory register entries. The memory column displays the used kind of memory (EEPROM, RAM, ROM) and the accessibility shortened as: ro: Read only, rpw: Read Protected Write, rw: read and write.

Rg	Name	Memory	Format	Description
00h	Primary Value	...	...	...
09h	ChConfig	EEPROM rpw	record	Configuration
0Ah	Maintenance	EEPROM rpw	long int	Maintenance info (date, category)
Continued on next page				

Continued from previous page				
Rg	**Name**	**Memory**	**Format**	**Description**
0Eh	ChType	ROM ro	record	Type & register info
0Fh	ChError	RAM ro	record	Error status/code

Table 15: *Channel registers*

In a P-NET bus a channel is identified by it's unique channel number. This number is assigned and administered by the P-NET user organisation.
Every bus participant can contain several channels (e.g. a Multi-IO module) whereas channel 0 has a special meaning. Channel 0 is also named as the Service Channel and must be provided by every bus module. Table 17 below shows the service channel, the mandatory entries are gray.

Rg	Name	Memory	Format	Description
00h	NumberOfSwNo	ROM ro	integer	highest SWNo in module
01h	DeviceID	ROM ro	record	device number, program version, producer info
03h	Reset	RAM rw	byte	optional module reset
04h	PnetSerialNo	function	record	node address, serial number
06h	TimeDate	function	record	date and time value
07h	FreeRunTimer	RAM ro	word	module time counter
08h	WDTimer	RAM rw	real	watchdog counter
09h	ModuleConfig	EEPROM rpw	record	module configuration
0Ah	WDPreset	EEPROM rpw	real	watchdog presettings
0Bh	MailFilter	RAM rw Init EEPROM	string	message filter
0Ch	MailBox	RAM rw	buffer	message buffer
0Dh	WriteEnable	RAM rw	boolean	EEPROM write release
0Eh	ChType	ROM ro	record	Type register info
0Fh	ChError	RAM rw	record	Error status/code

Table 16: *Service channel*

Analog input channel

The following channel definition serves as an example, picking up our former temperature analogy.

Rg	Name	Memory	Format	Description
00h	AnalogIn	RAM rw	real	input value
07h	HighLevel	RAM ro Init EEPROM	real	high alarm value
Continued on next page				

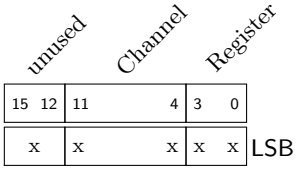
Continued from previous page				
Rg	**Name**	**Memory**	**Format**	**Description**
08h	LowLevel	RAM ro Init EEPROM	real	low alarm value
09h	ChConfig	EEPROM rpw	record	module configuration
0Bh	FullScale	EEPROM rpw	real	full scale value
0Ch	ZeroPoint	EEPROM rpw	real	zero point value
0Dh	Maintenance	EEPROM rpw	record	maintenance info (date, category)
0Eh	ChType	ROM ro	record	Type register info (used register flags)
0Fh	ChError	RAM rw	record	Error status/-code (high, low alarm,...)

Table 17: *Service channel*

By accessing the service channel 0, the application software is able to collect all necessary information to handle the process signal (the provided analog input) properly and without 'hard-coding' some module properties in the software itself!

Softwire Numbers

We know so far that every P-NET module is organized by a series of channels. Channel 0 is the service channel and describes the basic module properties. There are also more channels, depending on the module capacities. Each channel has a unique number authorized by the P-NET user organisation.
But how do we access a certain register in a channel? The data represented by a register value can not only varying in type (byte, word, real, record, ...) and access permission but also in the kind of memory where they were stored in the module (EEPROM, RAM, ROM). The tables indicated this already in the memory column.
Where may also exists some dependencies between a EEPROM and RAM variable. For example: P-NET allows the initialisation of a RAM variable by reading it's counterpart in the EEPROM.
The result is a huge number of addresses, scattered widely between the three storage types - and the different channels!
To make things easier, P-NET hides the final address of a register variable by using a mapping mechanism called **Softwire Numbers**.
In doing so every register address is stored in a list and every list entry is assigned with an unique number - the softwire number. The list itself is completely transparent for the user (or application). And: The internal structure of a module can be changed without any impact of the application, because the assigned number (or index) remains the same!
To access a certain register in a channel is therefore achieved by using a Softwire number which is a combination of the channel and the register index.



The lowest half byte indicated the register (a number between 00h and 0Fh). The Bits 4...11 (8 bits) contain the channel number.

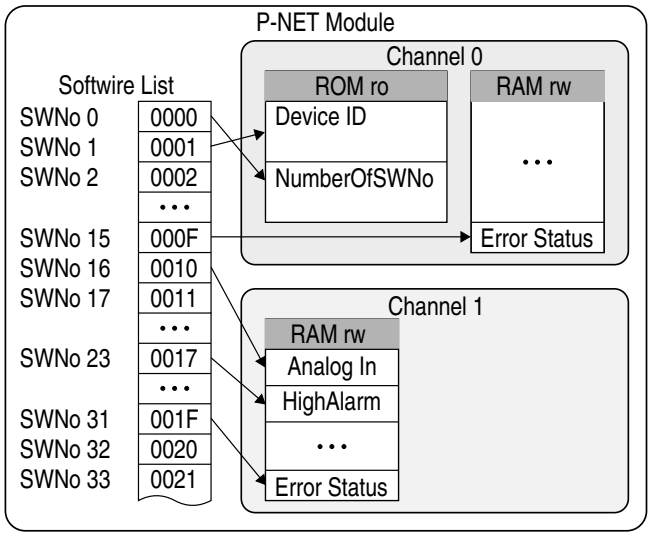


Figure 6: *Softwire List*

Note that the node address is coded in the header. When the addressed module or node receives the telegram it knows by its internal softwire list which memory address the given softwire number means.
The softwire number is transmitted in the Info-Field which also hold the data type of the accessing register in the byte following the softwire number. With these information the node not only knows where it has to look for the data but also which kind of data it accesses, see figure 6.

Examples

The following telegram sequence reads register 10 in channel 1 (a single byte) of the module or node with address 57 (first byte). The sender or source has address 9 (second byte with set 7th bit since it is a request).

39	89	02	03	00	1A	01	1E

DES	SRC	CONTROL	INFO-TYPE	INFO-LEN	Chn	Reg	TYPE	CKS
57	9	LOAD/SW-List	2 Byte SWNo	3	1	10	Byte	1E

STORE command sequence

In the following the master with address 1 wants to clear the error status (register 0Fh in the service channel) of module 64.

40	81	01	03	00	0F	00	2C

DES	SRC	CONTROL	INFO-TYPE	INFO-LEN	Chn	Reg	TYPE	CKS
64	1	STORE/SW-List	2 Byte SWNo	3	0	15	Boolean	2C

The module responses with an error, indicating that the given register is write protected.

81	40	40	00	FF

DES	SRC	STATUS	DATA-LEN	CKS
1	64	Write Protection	0	FF

Further links

https://www.p-net.org/
https://en.wikipedia.org/wiki/IEEE_754
https://de.wikipedia.org/wiki/P-NET
https://classic.proces-data.com/63/002J1EFSED/8KQR9-01/Description_ENG.htm
https://classic.proces-data.com/6D/002J1EFSED/THPUS-01/Description_ENG.htm
https://classic.proces-data.com/63/002J1EFSED/9HS9H-01/Description_ENG.htm
https://classic.proces-data.com/63/002J1EFSED/RF7DD-01/Description_ENG.htm

References

[1] Franz J. Gira, Dipl.-Ing. Mag. Martin Manninger. Skriptum zum Hochschulkurs Feldbus Systeme, P-NET, 24 April, 1994

[2] Otto-von-Guericke-Universität Magdeburg, Laborpraktikum Nachrichtentechnik, Versuch P-NET, NT 20

[3] Markus Haag, Diplomarbeit, P-NET-Slave auf einem 88C166 Mikrocontroller, Implementierung eines P-NET Knotens, Institut für Computertechnik, Universität Wien, Juni 1999

[4] P-Net tutorials on https://www.p-net.org/p-net-tutorials/

[5] Hans-Jürgen Gevatter, Ulrich Grünhaupt, Handbuch der Mess- und Automatisierungstechnik in der Produktion, Springer Verlag, 2006