GRUNDFOS

GENIbus Protocol Specification

Version 01.00.05

July 01 2016

Contents

Version history

Date	Initials	Description	Version
19-10-2012	HAM	Baseline for version history	V01.00.00

1. Introduction

GENIbus, the **G**rundfos **E**lectronics **N**etwork **I**ntercommunications **bus** is a fieldbus developed by Grundfos to meet the need for data transfer in all typical Grundfos motor/pump applications. In the field of *Building Management*, control of *Water Purifying Plants*, *Water Works* and *Industry applications* etc. Grundfos devices with GENIbus can be wired together in networks and integrated in automation systems. The major employment's are:

1) Set point control
2) Close loop control of slow systems (sampling rate < 10Hz)
3) Monitoring and data logging
4) Configuration
5) Faultfinding

GENIbus is based on the RS485 hardware standard and operates at a baud rate of 9600 bits/s. This relatively slow communication speed makes it possible to communicate up to 1200 m without the use of termination resistors. On the other hand, the slow speed makes GENIbus unsuitable for applications that requires fast control loops e.g. servo applications.

The GENIbus protocol is based on master/slave communication and can handle multi-master networks (not described in this document) if needed. However, a standard GENIbus device from Grundfos, like an E-pump or a CU-control unit, acts as a slave. It will not interfere with the bus control and it will only send a reply when it receives a request from a master device. This means that a GENIbus network will normally have only one master which could be the central management system (SCADA), a local controller like a PLC, or a gateway to another type of network. A total of 32 devices can be connected.

Like most other fieldbusses, GENIbus supports the mechanisms for single-casting (single-addressing), multicasting (group addressing) and broadcasting (global addressing). A unique feature of GENIbus is the *Connection Request*, which makes it possible to recognize all connected units on a network without having to poll through all possible addresses.

A summary of GENIbus technical data is given in chapter 2. This gives an overview of the functionality, the performance and the limitations.

Chapter 3 provides a detailed specification of the GENIbus telegram format. It describes how data is organized and how operations on data take place. The mechanism in cyclic redundancy checking (CRC) is explained and straight forward guidance in the CRC implementation is given. The chapter ends with some illustrative telegram examples.

Chapter 4 explains how GENIbus values, which represent physical units are scaled and shows some examples.

Finally chapter 5 gives an overview of available GENIbus functional profiles. A GENIbus functional profile document is associated with each GENIbus device. The functional profile specifies all data that can be exchanged and how to use it. It is indispensable when making software to operate GENIbus devices.

2. Technical Data Summary

Physical Layer (hardware)	
Topology	Bus
Transmitter	EIA RS485, half duplex
Coding	NRZ (non return to zero)
Data format	Start bit (=0), 8 data bits with least significant bit first, stop bit (=1)
Baud rate	9600 bits/s
Distance	Daisy chain: 1200m Multidrop: 500m Twisted pair cable with shield is recommended
No. of bus units	Max. 32
Data Link Layer (timing, verification)	
Inter Byte Delay	<=1.2ms
Inter Telegram Delay	>=3ms
Reply Delay	[3ms; 50ms]
Cyclic redundancy checking	16 bit CCITT, polynomial is 0x1021, Start Delimiter excluded. Initialised to 0xFFFF, CRC value bit inverted after calculation. High order byte transmitted first.
Medium access	Master/Slave
Physical address range	Master address range: [0; 231] Slave address range: [32; 231] *) Connection request address: 254 Broadcast address: 255
Presentation Layer (data processing)	
Data orientation	byte

*) The physical address range [32; 95] corresponds to the infra red remote controller R100 number range [1; 64]

Table 1: *Short form technical specification for GENIbus.*

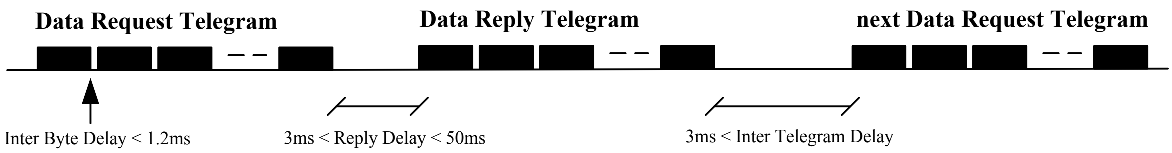


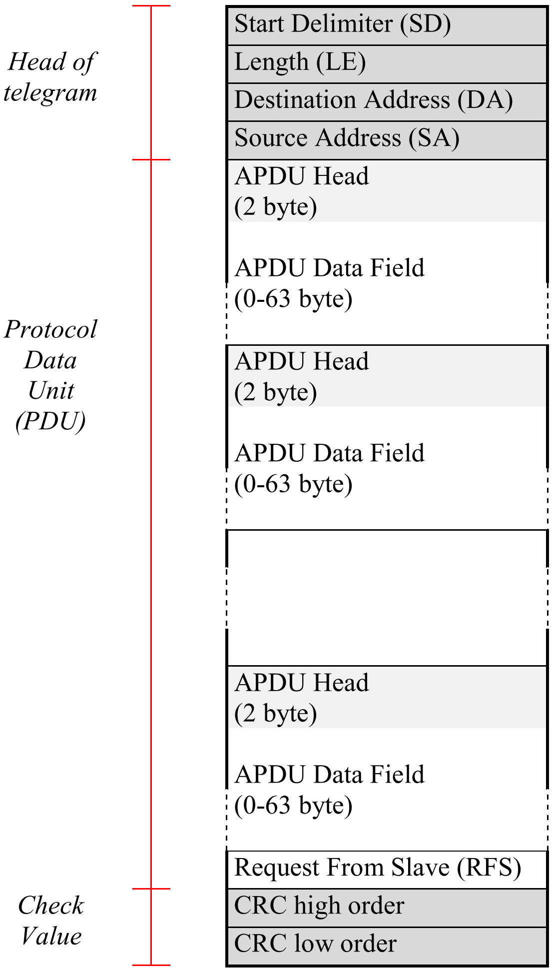
Figure 1: *Illustration of the timing requirements for a Communication Session. Each black rectangle shows a byte (10 bits character) on the RS485 line. These requirements imply the following:*

- *The bytes in a telegram must be sent consecutively with an Inter Byte Delay less than 1.2 ms.*
- *The Data Reply from a GENIbus unit will always be delayed at least 3 ms and maximum 50 ms. A Reply Timeout of approximately 60 ms in a master is suitable.*
- *A master must leave the bus idle for at least 3 ms after the reception of a reply telegram before the next request is transmitted. This triggers the idle detection circuit in all GENIbus units.*

When using a Data Message telegram (which is not very common) the bus must be left idle by the master for a time period corresponding to the maximum Reply Delay (= 50 ms) before the next Communication Session can be initiated.

3. Telegram Specification

3.1 Telegram Format



SD=27H: Data Request
SD=26H: Data Message (no reply)
SD=24H: Data Reply

Length: Number of following bytes excluding the Check Value

Destination Address: Address of the receiving unit (destination unit)

Source Address: Address of the transmitting unit (source unit)

PDU: Protocol Data Unit, the data field of the telegram

APDU: Application Program Data Unit. This reflects the organization of application data in Classes and the data access in Operations.

RFS: Request From Slave. This byte is optional. It is used for transfer of bus control in multi-master networks

CRC: A 16 bit Cyclic Redundancy Check Value for validation of a correct data transfer.

***Figure 2**: Format of GENIbus telegram. Each horizontal field is a byte unless otherwise stated. A PDU can consist of zero, one or many APDU's*

3.2 APDU Specification

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Class			
OS/ACK		Length of APDU Data Field					
APDU Data Field (0-63 byte)							

***Figure 3**: Format of Application Program Data Unit*

Class: Specifies which Data Class the APDU belongs to. The APDU Data Field will be interpreted accordingly. Table 2 shows a survey of the Data Classes together with the possible Operations that can be performed on them.

OS/ACK: OS: *Operation Specifier* (for Data Request and Data Message)
00: GET, to read the value of Data Items

10: SET, to write the value of Data Items
11: INFO, to read the scaling information of Data Items, an Info Data structure (fig. 4) will be returned.

ACK: *Acknowledge Code* (for Data Reply)
00: Everything is OK
01: Data Class unknown, reply APDU data field will be empty
10: Data Item ID unknown, reply APDU data field contains first unknown ID
11: Operation illegal or Data Class write buffer is full, APDU data field will be empty

	Operation		
Data Class	**GET**	**SET**	**INFO**
1. -			
2. Measured Data	X		X
3. Commands		X	X
4. Configuration Parameters	X	X	X
5. Reference Values	X	X	X
6. -			
7. ASCII-strings	X		

Table 2: *The Data Classes and possible Operations*

<table>
<tr><th>Bit 7</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0</th></tr>
<tr><td>1</td><td>0</td><td>VI</td><td>BO</td><td>*</td><td>*</td><td colspan="2">SIF=2</td></tr>
<tr><td>SZ</td><td colspan="7">UNIT index</td></tr>
<tr><td colspan="8">ZERO scale factor</td></tr>
<tr><td colspan="8">RANGE scale factor</td></tr>
</table>

<table>
<tr><th>Bit 7</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0</th><th></th></tr>
<tr><td>1</td><td>0</td><td>VI</td><td>BO</td><td>*</td><td>*</td><td colspan="2">SIF=3</td><td>INFO Head</td></tr>
<tr><td>SZ</td><td colspan="7">UNIT index</td><td rowspan="3">INFO Data Field</td></tr>
<tr><td colspan="8">ZERO scale factor high order</td></tr>
<tr><td colspan="8">ZERO scale factor low order</td></tr>
</table>

Figure 4: *Info Data structure, the reply to the INFO Operation. Requesting INFO from data items which are not scaled will only return the INFO Head (1 byte). Scaled data items result in a reply with an INFO Data Field as well. 8/16 bit data items use the left Data Field format (standard). The format to the right is for 8/16/24/32 bit data items (extended precision). See details in chapter 4.*

VI: *Value Interpretation:*
0: Only values from 0-254 are legal. 255 means "data not available"
1: All values 0-255 are legal values

BO: *Byte Order:*
0: High order byte, this is default for all values that are only 8 bit
1: Low order byte to a 16 bit, 24 bit or 32 bit value

SIF: *Scale Information Format:*
00: Scale information not available (no UNIT, ZERO or RANGE in reply)
01: Bit wise interpreted value (no UNIT, ZERO or RANGE in reply)
10: Scaled 8/16 bit value (UNIT, ZERO and RANGE in reply)
11: Extended precision, scaled 8/16/24/32 bit value (UNIT and ZERO hi/lo in reply)

SZ: *Sign of ZERO:*
0: Positive
1: Negative

UNIT index: A 7 bit index to the GENIbus Unit Table (Chapter 4)

ZERO/RANGE scale factors: For conversion between the physical representation and the computer representation of a value. See details in chapter 4.

	GET Operation		SET Operation		INFO Operation	
	Request APDU	*Reply APDU*	*Request APDU*	*Reply APDU*	*Request APDU*	*Reply APDU*
Class 2 Measured Data	0 0 0 0 Class=2 0 0 Length ID Code ID Code :	0 0 0 0 Class=2 0 0 Length Value Value :	***Illegal***		0 0 0 0 Class=2 1 1 Length ID Code ID Code :	0 0 0 0 Class=2 0 0 Length 1 or 4 byte INFO Data 1 or 4 byte INFO Data :
Class 3 Commands	***Illegal***		0 0 0 0 Class=3 1 0 Length ID Code ID Code :	0 0 0 0 Class=3 0 0 Length	0 0 0 0 Class=3 1 1 Length ID Code ID Code :	0 0 0 0 Class=3 0 0 Length INFO Head INFO Head :
Class 4 Configuration Parameters	0 0 0 0 Class=4 0 0 Length ID Code ID Code :	0 0 0 0 Class=4 0 0 Length Value Value :	0 0 0 0 Class=4 1 0 Length ID Code Value ID Code Value :	0 0 0 0 Class=4 0 0 Length	0 0 0 0 Class=4 1 1 Length ID Code ID Code :	0 0 0 0 Class=4 0 0 Length 1 or 4 byte INFO Data 1 or 4 byte INFO Data :
Class 5 Reference Values	0 0 0 0 Class=5 0 0 Length ID Code ID Code :	0 0 0 0 Class=5 0 0 Length Value Value :	0 0 0 0 Class=5 1 0 Length ID Code Value ID Code Value :	0 0 0 0 Class=5 0 0 Length	0 0 0 0 Class=5 1 1 Length ID Code ID Code :	0 0 0 0 Class=5 0 0 Length 1 or 4 byte INFO Data 1 or 4 byte INFO Data :
Class 7 ASCII Strings	0 0 0 0 Class=7 0 0 Length ID	0 0 0 0 Class=7 0 0 Length ASCII string (zero terminated)	***Illegal***		***Illegal***	

Figure 5*: Survey of possible APDUs for the various Data Classes. Complete telegram examples can be found in chapter 3.5*

3.3 CRC Generation

The figure below shows a hardware equivalent of the CRC generation. It can be a great help in trying to understand the mechanism. The dark shaded 16 bit register is called the *CRC-Accumulator*. When the whole telegram has been shifted into the machine the Accumulator will hold the CCITT version of the CRC-value. CCITT specifies an initialization of the CRC-Accumulator with all zeros. The Accumulator must be initialized with all 1s and the bits inverted just before transmitting to make the CRC resistant to leading erroneous zeros and to merged telegrams. A function that implements this behavior in software is written below:

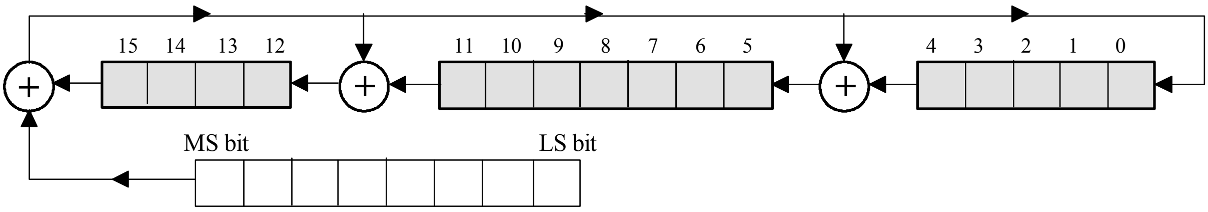


Figure 6*: A hardware equivalent CRC generator. GENIbus uses 0x1021 as generator polynomial.*

```
ushort crchware(ushort data, ushort accum)
 {
  uchar i;
  data <<=8;
  for (i=8; i>0; i--)               /* Do for each bit:         */
    {
    if ((data ^ accum) & 0x8000)    /* IF a 1 in feedback path */
      accum = (accum <<1)^genpoly; /*    feedback interaction */
    else                            /* ELSE                     */
      accum <<= 1;                  /*    transparent shift    */
    data <<=1;                      /* Make next bit ready     */
    }
 return accum;
 }
```

Each byte in the telegram which is to take part in the cyclic redundancy check is passed to the function one by one along with the value of the Accumulator. When the last byte of the telegram has been passed the return value will be the CRC-value. Data has been declared as a 16 bit value due to its presence in a 16 bit expression.

By studying the CRC circuit we can see that when applying a new byte to the CRC circuit the feedback path will not be influenced by the existing low order byte of the Accumulator. Only the high order Accumulator byte interacts with the data bits. We refer to the result of this XOR'ing as the *combining value*. This leads to the observation that the new Accumulator is equal to the CRC of the combining value XOR'ed with the unchanged half of the Accumulator. This relationship can be expressed in C.

```
comb_val = (accum >>8) ^ data;
tmp = crchware(comb_val,0);
accum = tmp ^ (accum <<8);
```

Since there are only 256 possible combining values, it would be a good idea to calculate their CRCs in advance and store them in a table, `crctab[256]`, thereby saving a great deal of run time computer power. The following piece of code uses `crchware` to generate the lookup table.

```
void main(void)
   {
   unsigned short i,j;
   printf("\nunsigned short crctab[256] = \n    {");
   for (j=0; j<=31; j++)
        {
        printf("\n    ");
        for (i=8*j; i<8*j+8; i++) printf("%6u,  ", crchware(i,0));
        }
   printf("\n    }");
```

By redirecting the output to a file the table is ready to paste into the protocol source code. The CRC calculation now takes this form.

```
comb_val = (accum >>8) ^ data;
tmp = crctab[comb_val];
accum = tmp ^ (accum <<8);
```

Combining this into a more compact form leads to the final CRC function. The Accumulator has been removed from the arguments and made a global variable, to avoid the overhead of passing it to the function for each byte to process.

Final GENIbus CRC algorithm

```
ushort accum;
void crc_update(uchar data)
      {
      accum = (accum <<8)^ crctab[(accum >>8) ^ data];
      }
```

Transmitter: The CRC-Accumulator is initialized to 'all ones' and each byte, <u>except</u> the *Start Delimiter*, is processed through the `crc_update` function before being sent to the Drivers. Finally the CRC-Accumulator is inverted and its two bytes are appended to the telegram with high order byte first. These two bytes are what we define as the *CRC-Value*.

Receiver: Performs a similar procedure. Initializes the CRC-Accumulator to 'all ones'. Then, each byte received, <u>except</u> the *Start Delimiter*, is processed through the `crc_update` function. When the CRC-Value bytes arrive they are inverted and then also processed through `crc_update`. If the CRC-Accumulator hereafter is equal to zero the received telegram is considered as sound.

3.4 Connection Request Mechanism

The GENIbus protocol offers an effective mechanism for a master device to recognize all units connected to the bus. The master can use a Connection Request Telegram. This telegram is characterized by the usage of the destination address DA=254 (0xFE). The Connection Reply which results is characterized by

- only generated if the unit has not been requested (polled) by using its unit address within the last 20 seconds
- random reply delay [3ms; 43ms] to minimize the probability of several units replying simultaneously

When a network is powered on, and the master has no previous knowledge of which units are connected, it can use Connection Requests (instead of polling through all possible addresses) to recognize the units. When all units have been recognized (no more replies to connection requests), the master can use Connection Requests occasionally as a simple means to detect if new units are connected (or units that have been disconnected or switched off are being reconnected or switched on).

Connection Request

Start Delimiter	0x27
Length	0x0E
Destination Address	0xFE
Source Address	0x01
Class 0: Protocol Data	0x00
OS=0 (GET), Length=2	0x02
`df_buf_len` = ID 2	0x02
`unit_bus_mode` = ID 3	0x03
Class 4: Configuration Parameters	0x04
OS=0 (GET), Length=2	0x02
`unit_addr` = ID 46	0x2E
`group_addr` = ID 47	0x2F
Class 2: Measured Data	0x02
OS=0 (GET), Length=2	0x02
`unit_family` = ID 148	0x94
`unit_type` = ID 149	0x95
CRC high	0xA2
CRC low	0xAA

Connection Reply

Start Delimiter	0x24
Length	0x0E
Destination Address	0x01
Source Address	0x20
Class 0: Protocol Data	0x00
Ack=0, Length=2	0x02
Value example of `df_buf_len`	0x46
Value example of `unit_bus_mode`	0x0E
Class 4: Configuration Parameters	0x04
Ack=0, Length=2	0x02
Value example of `unit_addr`	0x20
Value example of `group_addr`	0xF7
Class 2: Measured Data	0x02
Ack=0, Length=2	0x02
Value example of `unit_family`	0x03
Value example of `unit_type`	0x01
CRC high	0x00
CRC low	0x04

Figure 7: *Connection Request/Reply example. In this example the master has a Unit Address of 1 and the slave is a CU3 unit with unit address 32 (0x20), corresponding to No. 1 given with the infra red remote controller R100.*
The Class 0 Data Items need not to be considered.

3.5 Telegram Examples

Both examples assume communication with a Control Unit CU3 with unit address 0x20 (No. 1 given with R100). Compare the examples with the general telegram format in figure 2 and the APDU survey in figure 5.

Data Request

Start Delimiter	0x27
Length	0x07
Destination Address	0x20
Source Address	0x01
Class 2: Measured Data	0x02
OS=3 (INFO), Length=3	0xC3
`i_rst` = ID 2	0x02
`t_mo` = ID 16	0x10
`p_hi` = ID 26	0x1A
CRC high	0x90
CRC low	0x1C

Data Reply

Start Delimiter	0x24
Length	0x10
Destination Address	0x01
Source Address	0x20
Class 2: Measured Data	0x02
Ack=0, Length=12	0x0C
Value example of `i_rst` INFO head	0x82
Value example of `i_rst` UNIT Index	0x3E
Value example of `i_rst` ZERO	0x00
Value example of `i_rst` RANGE	0x39
Value example of `t_mo` INFO head	0x82
Value example of `t_mo` UNIT Index	0x15
Value example of `t_mo` ZERO	0x00
Value example of `t_mo` RANGE	0x64
Value example of `p_hi` INFO head	0x82
Value example of `p_hi` UNIT Index	0x09
Value example of `p_hi` ZERO	0x00
Value example of `p_hi` RANGE	0xFA
CRC high	0x91
CRC low	0x0A

Figure 8: *Request of scaling information of 3 data items by using the INFO operation. Notice that data items with scaling information return INFO head, UNIT index, ZERO and RANGE. The values in the reply are examples, your reply might be different.*

Data Request

Start Delimiter	0x27
Length	0x0F
Destination Address	0x20
Source Address	0x01
Class 2: Measured Data	0x02
OS=0 (GET), Length=4	0x04
`i_rst` = ID 2	0x02
`t_mo` = ID 16	0x10
`p_hi` = ID 26	0x1A
`p_lo` = ID 27	0x1B
Class 4: Configuration Parameters	0x04
OS=0 (GET), Length=2	0x02
`t_mo_stop` = ID 4	0x04
`i_rst_max_stop` = ID 5	0x05
Class 3: Commands	0x03
OS=2 (SET), Length=1	0x81
`START` = ID 6	0x06
CRC high	0x80
CRC low	0x2A

Data Reply

Start Delimiter	0x24
Length	0x0E
Destination Address	0x01
Source Address	0x20
Class 2: Measured Data	0x02
Ack=0, Length=4	0x04
Value example of `i_rst`	0x7A
Value example of `t_mo`	0x42
Value example of `p_hi`	0x39
Value example of `p_lo`	0x80
Class 4: Configuration Parameters	0x04
Ack=0, Length=2	0x02
Value example of `t_mo_stop`	0xB5
Value example of `I_rst_max_stop`	0xC8
Class 3: Commands	0x03
Ack=0, Length=0	0x00
CRC high	0xF2
CRC low	0xD7

Figure 9: *The example shows how a reading of 4 data items from class 2 (`p_hi` and `p_lo` combines to a 16 bit value), and 2 data items from class 4 and writing of a command (class 3) can be done in one telegram. The values in the reply are examples, your reply might be different.*

4. Scaling of values

The purpose of scaling is to map the value of a variable *x*, representing some physical entity, into its computer representation *X*. The *GENIbus* value representation is based on 8 bit quantities. This means, that scaling maps an interval of real numbers with any chosen length and from any chosen decade linearly into an 8 bit integer representation:

$$x \in [a;b] \quad \rightarrow \quad X \in [0;254] \qquad ;(a,b) \in \Re$$

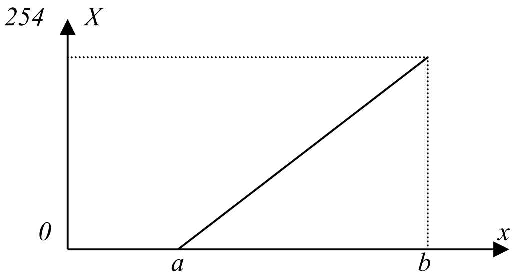


Figure 10: *Mapping real numbers into a byte value.*

This mapping is shown graphically in the figure above. The value *255* is reserved for indication of "data not available". What is needed now is a mathematical expression for the mapping and its inverse. The use of the symbols in the figure and the straight line relationship give this equation to start with:

$$(1) \qquad X(x) = -254 \cdot \frac{a}{b-a} + 254 \cdot \frac{1}{b-a} x = \frac{254}{b-a}(-a+x)$$

a is the *"zero"* for the interval to be mapped, and *(b-a)* is its *"range"*. Both occur directly in the equation. Contained in this *"zero"* and *"range"*, however is the physical unit with some prefix factor to take the decade into consideration. To isolate this, a multiplier called *UNIT* is introduced in the following way:

$$\begin{aligned} b-a &= \text{"range"} &&= RANGE \cdot UNIT \\ a &= \text{"zero"} &&= ZERO \cdot UNIT \end{aligned}$$

Substituting this in (1) gives the final conversion formulas:

$$(2a) \qquad X = \frac{254}{RANGE \cdot UNIT}(-ZERO \cdot UNIT + x)$$

$$(2b) \qquad x = \left(ZERO + X\frac{RANGE}{254}\right) \cdot UNIT$$

ZERO and *RANGE* can be represented in 8 bit, consequently they are suitable for *GENIbus*. *UNIT* is an index to a standard table of defined physical units with a prefix factor. This table, from now on called *The Unit Table*, although small, can cover a substantial amount of different scalings.
UNIT is chosen to be in 8 bit (not surprisingly). Bit 7 is reserved for one problem that was left over: the sign of *ZERO*. Left are 7 bits giving 128 possible entries in The Unit Table. The job of converting a scaled *GENIbus* data item value into its real number representation with physical units, now means to read (with INFO Operation) *ZERO, RANGE* and *UNIT* for that data item, and then process the value through formula (2b).

The scaling formulas (2a-b) are only valid for 8 bit values, but it is only natural to extend them to count for 16 bit values as well. Understanding X_{16} as a 16 bit computer representation leads directly to the 16 bit version of the conversion formulas:

$$(3a) \qquad X_{16} = \frac{254 \cdot 256}{RANGE \cdot UNIT}(-ZERO \cdot UNIT + x)$$

$$(3b) \qquad x = \left(ZERO + X_{16}\frac{RANGE}{254 \cdot 256}\right) \cdot UNIT = \left(ZERO + X_{hi}\frac{RANGE}{254} + X_{lo}\frac{RANGE}{254 \cdot 256}\right) \cdot UNIT$$

Notice that "full range" for 16 bit values equals 254·256 = 0xFE00, and that "data not available" is indicated with X_{16} =0xFFFF.

Because *GENIpro* handles single byte *Data Items, X_{hi}* and *X_{lo}* must have one ID code each. Per definition, the scaling information *ZERO, RANGE* and *UNIT* is connected with the high order byte, *X_{hi}*. The low order byte *X_{lo}* has no scaling information because this comes implicit from (3a-b). Using the *INFO* operation on a low order byte will however return an *INFO head,* where the *BO* bit is set to indicate that this is a low order byte (See figure 4). The functional profile for the device in question specifies which data items are split in a high/low pair.

The Unit Table								
Index	**Physical entity**	**Unit with prefix**	**Index**	**Physical entity**	**Unit with prefix**	**Index**	**Physical entity**	**Unit with prefix**
15	Electrical current	1 μA	**93**	Flow	100 m^3/h	**90**	Velocity	1 mm/s
112	Electrical current	1 mA	**114**	Flow	0.1 l/h	**34**	Ang. velocity	2 rad/s
1	Electrical current	0.1 A	**73**	Flow	0.5 l/h	**100**	Time	1 day
42	Electrical current	0.2 A	**52**	Flow	1 l/s	**39**	Time	1024 h
62	Electrical current	0.5 A	**63**	Flow	0.1 l/s	**81**	Time	100 h
2	Electrical current	5 A	**53**	Flow	1 m^3/s	**72**	Time	1024 min
3	Voltage	0.1 V	**54**	Flow	1 gpm	**13**	Time	2 h
4	Voltage	1 V	**58**	Flow	10 gpm	**35**	Time	1 h
104	Voltage	2 V	**82**	Flow	0.1 l/min	**36**	Time	2 min
5	Voltage	5 V	**91**	Head/Distance	0.0001 m	**80**	Time	1 min
6	Elec. resistance	1 Ω	**83**	Head/Distance	0.01 m	**14**	Time	30 s
43	Elec. resistance	10 kΩ	**24**	Head/Distance	0.1 m	**78**	Time	10 s
89	Elec. resistance	100 kΩ	**25**	Head/Distance	1 m	**37**	Time	1 s
10	Elec. capacitance	1 μF	**26**	Head/Distance	10 m	**79**	Time	0.1 s
7	Power (active)	1 W	**56**	Head/Distance	1 ft	**108**	Time	Unix time
8	Power (active)	10 W	**59**	Head/Distance	10 ft	**49**	Ang. degrees	1 º
9	Power (active)	100 W	**51**	Pressure	0.001 bar	**77**	Gain	0.01
44	Power (active)	1 kW	**27**	Pressure	0.01 bar	**96**	Gain	0.1
45	Power (active)	10 kW	**28**	Pressure	0.1 bar	**50**	Gain	1
105	Frequency	0.01 Hz	**29**	Pressure	1 bar	**71**	Volume	1 nl
11	Frequency	0.5 Hz	**61**	Pressure	1 kPa	**106**	Volume	1 μl
16	Frequency	1 Hz	**55**	Pressure	1 psi	**70**	Volume	0.1 ml
38	Frequency	2 Hz	**60**	Pressure	10 psi	**88**	Volume	1 ml
17	Frequency	2.5 Hz	**113**	Percentage	1 ppm	**109**	Volume	0.1 ltr
98	Rot. velocity	1 rpm	**107**	Percentage	0.01 %	**99**	Volume	1 ltr
18	Rot. velocity	12 rpm	**12**	Percentage	0.1 %	**64**	Volume	0.1 m^3
19	Rot. velocity	100 rpm	**30**	Percentage	1 %	**86**	Volume	1 m^3
20	Temperature	0.1 ºC	**76**	Percentage	10 %	**67**	Volume	256 m^3
21	Temperature	1 ºC	**87**	Energy	1 Ws	**65**	Volume	1000 m^3
57	Temperature	1 ºF	**94**	Energy	1 Wh	**66**	Energy pr vol.	10 kWh/m^3
84	Temperature	0.01 K	**103**	Energy	0.1 kWh	**74**	Energy pr vol.	1 Wh/m^3
110	Temperature diff.	0.01 K	**31**	Energy	1 kWh	**115**	Spec. heat cap.	1 J/kg/K
111	Temperature diff.	1 K	**85**	Energy	2 kWh	**68**	Area	1 m^2
69	Flow	0.1 ml/h	**32**	Energy	10 kWh	**75**	Torque	1 Nm
95	Flow	0.01 m^3/h	**33**	Energy	100 kWh	**97**	Torque	0.1 Nm
22	Flow	0.1 m^3/h	**40**	Energy	512 kWh	**101**	Acceleration	0.01 m/s^2
23	Flow	1 m^3/h	**46**	Energy	1 MWh	**102**	Mass density	0.1 kg/m^3
41	Flow	5 m^3/h	**47**	Energy	10 MWh			
92	Flow	10 m^3/h	**48**	Energy	100 MWh			

Table 3: *The Unit Table*

Scaling example for 8 bit data item

Assume that a GET request for the data item `t_m` (Motor temperature, Class 2, ID 29) in a UPE pump (or E-pump) returns the value 163 and an INFO request for the same data item returns UNIT=21, RANGE=90, ZERO=10, then

$$T_m = \left(ZERO + t\_m \frac{RANGE}{254} \right) \cdot UNIT = \left(10 + 163 \cdot \frac{90}{254} \right) \cdot 1^oC = 68^oC$$

Scaling example for 16 bit data item

Assume that a GET request for the data items `p_hi` and `p_lo` which constitute a 16 bit high/low data item pair (Power consumption, Class 2, ID 26/27) in a CU3 control unit returns the values 16/214 and an INFO request for `p_hi` returns UNIT=44, RANGE=120, ZERO=0, then

$$P = \left(ZERO + p\_hi \frac{RANGE}{254} + p\_lo \frac{RANGE}{254 \cdot 256} \right) \cdot UNIT = \left(0 + 16 \cdot \frac{120}{254} + 214 \cdot \frac{120}{254 \cdot 256} \right) \cdot 1kW = 7.95kW$$

It can generally be assumed that no GENIbus device changes the scaling of the data items dynamically. This means that once ZERO, RANGE and UNIT are known for all the required data items (by using INFO requests when starting) it is now only necessary to request the data item values and process the reply through the scaling formula with the value of the scaling parameters inserted.

Extended precision

To be able to deal with data that spans a range of several decades and which at the same time must preserve the same resolution in the high decade as in the low decade, the GENIbus protocol supports a data format with extended precision using 16 bit, 24 bit or 32 bit (see also figure 4 in chapter 3.2). This format differs from the standard scaling format in two ways: it does not include a RANGE specifier, and the ZERO specifier is in 16 bit. The conversion formulas for 8 bit, 16 bit, 24 bit and 32 bit respectively are shown below:

(4.a) $$X_8 = \frac{x}{UNIT} - ZERO_{16}$$

(4.b) $$x = (ZERO_{16} + X_8) \cdot UNIT$$

(5.a) $$X_{16} = \frac{x}{UNIT} - ZERO_{16}$$

(5.b) $$x = (ZERO_{16} + X_{16}) \cdot UNIT$$

(6.a) $$X_{24} = \frac{x}{UNIT} - 256 \cdot ZERO_{16}$$

(6.b) $$x = (256 \cdot ZERO_{16} + X_{24}) \cdot UNIT$$

(7.a) $$X_{32} = \frac{x}{UNIT} - 256^2 \cdot ZERO_{16}$$

(7.b) $$x = (256^2 \cdot ZERO_{16} + X_{32}) \cdot UNIT$$

Note that X_{16} consists of 2 data items: $X_{16} = X_{hi} \cdot 256 + X_{lo}$

that X_{24} consists of 3 data items: $X_{24} = X_{hi} \cdot 256^2 + X_{lo1} \cdot 256 + X_{lo2}$

that X_{32} consists of 4 data items: $X_{32} = X_{hi} \cdot 256^3 + X_{lo1} \cdot 256^2 + X_{lo2} \cdot 256 + X_{lo3}$

and that $ZERO_{16} = 256 \cdot ZERO_{hi} + ZERO_{lo}$

It is also worth observing that when ZERO is 0 (which is normally the case) the formulas 4, 5, 6 and 7 become identical.

From the INFO head it can be seen if a data item is scaled according to the extended precision specification or not. However it will also be mentioned explicitly in the function profile of the product in question.

Notice that "data not available" is indicated with X_8=0xFFFF, X_{16}=0xFFFF, X_{24}=0xFFFFFF and X_{32}=0xFFFFFFFF.

Extended precision scaling example for 8 bit data item

Extended precision with 8 bit is typically used for data items where the increments (1 bit) should have a nice value (e.g. 1W instead of 1.18W) or, if there are multi-byte data items using extended precision, to keep all scaling according to the extended format. Assume that a GET request for the data item `t_mo` (Class 2, ID 94) in an MP 204 motor protection unit, returns the value 87, and an INFO request for `t_mo` returns UNIT=21 (positive ZERO and UNIT index = 21 corresponding to 1 ºC), $ZERO_{hi}$=0, $ZERO_{lo}$=0, then

$$T_{motor} = (ZERO_{16} + t\_mo_8) \cdot UNIT$$

Substituting:

$t\_mo_8 = t\_mo$

$ZERO_{16} = 256 \cdot ZERO_{hi} + ZERO_{lo}$

$UNIT = 1\,°C$

results in:

$$T_{motor} = ((256 \cdot 0 + 0) + 87) \cdot 1\,°C = 87\,°C$$

Extended precision scaling example for 16 bit data item

Extended precision with 16 bit is typically used for data items which need a higher precision than 8 bit and where the increments (1 bit) should have a nice value (e.g. 1W instead of 1.18W). Assume that a GET request for the data items `water_level_hi` and `water_level_lo` which constitute a 16 bit data item pair

(Class 2, ID 201/202) in an SEE sewage pump, returns the values 18, 12, and an INFO request for `water_level_hi` returns UNIT=179 (negative ZERO and UNIT index = 51), $ZERO_{hi}$=3, $ZERO_{lo}$=245, then

$$L_{water} = \left(ZERO_{16} + water\_level_{16}\right) \cdot UNIT$$

Substituting:

$$water\_level_{16} = water\_level\_hi \cdot 256 + water\_level\_lo$$

$$ZERO_{16} = 256 \cdot ZERO_{hi} + ZERO_{lo}$$

$$UNIT51 = 1\,mbar$$

results in:

$$L_{water} = \left(-\left(256 \cdot 3 + 245\right) + 18 \cdot 256 + 12\right) \cdot 1\,mbar = -1013\,mbar + 4620\,mbar = 3607\,mbar$$

Extended precision scaling example for 24 bit data item

Extended precision with 24 bit is typically used for counters which count events or time. Assume that a GET request for the data items `power_on_time_hi`, `power_on_time_lo1` and `power_on_time_lo2` which constitute a 24 bit data item trio (Class 2, ID 192/193/194) in a UPE pump, returns the values 7, 108, 32, and an INFO request for `power_on_time_hi` returns UNIT=36, $ZERO_{hi}$=0, $ZERO_{lo}$=0, then

$$t_{power\_on} = \left(256 \cdot ZERO_{16} + power\_on\_time_{24}\right) \cdot UNIT$$

Substituting:

$$power\_on\_time_{24} = power\_on\_time\_hi \cdot 256^2 + power\_on\_time\_lo1 \cdot 256 + power\_on\_time\_lo2$$

$$ZERO_{16} = 256 \cdot ZERO_{hi} + ZERO_{lo}$$

$$UNIT36 = 2\,\min$$

results in:

$$T_{power\_on} = (256 \cdot (256 \cdot 0 + 0) + 7 \cdot 256^2 + 108 \cdot 256 + 32) \cdot 2min = 972864\ min = 675d\ 14h\ 24min$$

Extended precision scaling example for 32 bit data item

Extended precision with 32 bit is typically used for physical values (measured or calculated). Assume that a GET request for the data items `dosing_flow_hi`, `dosing_flow_lo1` `dosing_flow_lo2` and `dosing_flow_lo3` which constitute a 32 bit data item quartet (Actual dosing flow, Class 2, ID 39/40/41/42) in a DME dosing pump, returns the values 23, 216, 42, 214 and an INFO request for `dosing_flow_hi` returns UNIT=69, $ZERO_{hi}$=0, $ZERO_{lo}$=0, then

$$Q_{dosing} = \left(256^2 \cdot ZERO_{16} + dosing\_flow_{32}\right) \cdot UNIT$$

Substituting:

$$dosing\_flow_{32} = dosing\_flow\_hi \cdot 256^3 + dosing\_flow\_lo1 \cdot 256^2 + dosing\_flow\_lo2 \cdot 256 + dosing\_flow\_lo3$$

$$ZERO_{16} = 256 \cdot ZERO_{hi} + ZERO_{lo}$$

$$UNIT69 = 0.1\,ml/h$$

results in:

$$Q_{dosing} = \left(256^2 \cdot \left(256 \cdot 0 + 0\right) + 23 \cdot 256^3 + 216 \cdot 256^2 + 42 \cdot 256 + 214\right) \cdot 0.1 ml/h = 40.0043 \cdot 10^6\ ml/h$$

5. References

Functional Profiles for GENIbus devices	
CIU xx2 AutoAdapt.pdf	Operating the CIU xx2 AutoAdapt via GENIbus
CIU xx3 SQFlex.pdf	Operating the CIU xx3 SQFlex via GENIbus
CR-monitor.pdf	Operating the CR-Monitor via GENIbus
CU 36x – Dedicated Controls R3.pdf	Operating the CU 36x Dedicated Control unit via GENIbus
CU 3.pdf	Operating the CU3 control unit via GENIbus or G100
CU 300.pdf	Operating the CU300 control unit via GENIbus or G100
CU 323 – Multi-B.pdf	Operating the Hydro Multi-B booster via GENIbus
CU 35x – MPC R3.pdf	Operating the Hydro MPC booster via GENIbus
CU 35x – MPC R3 - DDD.pdf	Operating the CU 354 DDD controller via GENIbus
DME.pdf	Operating the DME dosing pump via GENIbus or G100
DDA.pdf	Operating the DDA dosing pump via GENIbus
MGE.pdf	Operating the 3 phase MGE motor via GENIbus or G100
MP204.pdf	Operating the MP 204 motor protection unit via GENIbus or G100
Multi-E.pdf	Operating the Hydro Multi-E boosting system via GENIbus or G100
IO 351.pdf	Operating the MPC module IO351 via GENIbus or G100
IO 113.pdf	Operating the IO113 via GENIbus
Large MGE and CUE.pdf	Operating the Large MGE and CUE drive via GENIbus
RedWolf-SaVer.pdf	Operating RedWolf/SaVer pumps via GENIbus
UPE.pdf	Operating the UPE pump via GENIbus or G100
UPS.pdf	Operating the UPS pump via GENIbus or G100