# GMH3xxx – Serial Interface

(Please note: Document not applicable to GMH37xx!)

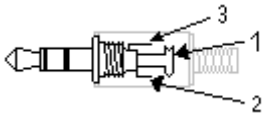
## Interfaceadapter GRS3100

To connect a GMH3xxx to a serial RS232 interface (e.g. COM2 of a PC)the adapter GRS3100 is available. This adapter converts the GMH3xxx levels to the RS232 standardised levels. Additionally the instrument is electrically isolated to the RS232 interface. The adapter is supplied by the RS232-interface. Please also consider the notes in the manual of the adapter. If more than one instrument should be connected to the same interface, the GRS3105 is the solution (up to 5 instruments, GRS3110 up to ten). In this case the referring base address setting has to be made in the instruments, to avoid collisions.
Restriction herein: The GMH34xx with an older Version number as V1.8 doesn't support base address settings, therefore only one instrument of this kind can be connected, base address is always 01.

## Interface Connection

The connection of the serial interface is done by a 3.5mm stereo audio plug.

Connections:




| | |
|---|---|
| 1 = GND | Ground |
| 2 = RxD | Receive Data |
| 3 = TxD | Transmit Data |

## Interface levels of the GMH3xxx

RxD – Input : 1 = +5V or tristate
: 0 = 0V
Input has an internal pullup (760 Ohm) to +5V

TxD – Output : 1 = 0V
: 0 = +5V
Output current is limited by a serial resistor (3.9KOhm)

## Interface settings

According to the RS232 agreements (idle state = logic 1)

- Baudrate 4800
- Parity none
- Databits 8
- Stopbits 1
- Handshake none

## Interface protocoll

According to EASYBus specification

# Simplified data transfer with the GMH3xxx-Series

The measuring data are collected in polling operation, i.e. a request is sent to the instrument, which answers with the desired data. The 3rd., 6th., 9th , ... byte of each transfer is a control byte (CRC). They won't be described further here. (please refer to appendix: **C-Code to calculate CRC-Bytes for EASYBus-Protocol (Bytes 2,5,8..)**)
In the following a simplified scan of the measuring values will be described. Assumption is the correct installed application with GMH3xxx and **GRS310x** interface adapter.

## *1 Opening The Interface / Interfaceparameters*

Standard Settings:

- Baudrate 4800
- Parity none
- Databits 8
- Stopbits 1
- Handshake none

Enhanced Settings:

- RTS=disabled
- DTR=enabled

By setting the RTS and DTR wires, the adapter will be switched on. Using Windows-C, this wires can be set e.g by means of the ‚DCB', where all interface settings are stored in (prt. Appendix).

## *2 Request*

Depending on the instrument there may be more than one addresses (measuring channels) available. To get the referring measuring value (prt. operating manual of the instrument) following string of the length of 3 bytes has to be sent to the instrument (without linefeed or carriage return) :

| GMH-Address | Request | | |
|---|---|---|---|
| | byte 0 | byte 1 | byte 2 |
| 1 | Chr$(254) | chr$(0) | Chr$(61) |
| 2 | Chr$(253) | chr$(0) | Chr$(2) |
| 3 | Chr$(252) | chr$(0) | Chr$(23) |
| 4 | Chr$(251) | chr$(0) | Chr$(124) |
| 5 | Chr$(250) | chr$(0) | Chr$(105) |
| 6 | Chr$(249) | chr$(0) | Chr$(86) |

The request string consists of Byte 0 & Byte 1 & Byte 2. (Byte 0 has to be sent first).
The request strings for instruments with other base addresses than 01 can be found in the Appendix.

## *3 Response*

If the corresponding address is available, all connections are correct and the instrument is switched on, the instrument will respond within much less than one second with 6 bytes. The measuring result can be calculated from this string as follows:

**response string** = Byte0 & Byte1 & Byte2 & **Byte3 & Byte4** & Byte5

**integer value** = 16383 AND {[256*(255-**Byte3**)] + **Byte4**} – 2048
**decimal point information** = 49152 AND [256*(255-**Byte3**)]

Depending on the decimal point information the measuring value can be calculated:

- decimal point information=49152 ⇒ measuring value = integer value/1000
- decimal point information=32768 ⇒ measuring value = integer value/100
- decimal point information=16384 ⇒ measuring value = integer value/10
- decimal point information=0 ⇒ measuring value = integer value

**Is the integer value > 16352, the value is an error message!**

| integer value | Meaning |
|---|---|
| 16352 | Error 1: measuring range overrun |
| 16353 | Error 2: measuring range underrun |
| 16362 | Error 11: calculation not possible |
| 16363 | Error 7: system error |
| 16364 | Error 8: battery empty |
| 16365 | Error 9: sensor defective |

## C-Code to calculate measuring values

```
// Input: HighByte = Byte3, LowByte=Byte4
//
// Rueckgabe = 0:  Decoding ok, Measuring value is in *Fliesspunkt
// Rueckgabe = -1: no valid value
//                 in *Fliesspunkt is errorcode or invalid value

short int FloatDekodieren (BYTE HighByte, BYTE LowByte, double *Fliesspunkt)
  {
  // Declarations
  long            CodierteDaten, LongPuffer;
  int             Dezimalpunkt;
  short int       Rueckgabe = 0;
  double          DoublePuffer;
  #define         GesperrterBereichMin         0x3EB1
  #define         GesperrterBereichMax         0x3FFF

  //Function
  CodierteDaten = HighByte^255;          //Bytes zusammenfügen und
  CodierteDaten = CodierteDaten << 8;    //Dezimalpunktinformation 'heraus-unden'
  CodierteDaten = CodierteDaten | LowByte;
  CodierteDaten = CodierteDaten & 0x3FFF;
  Dezimalpunkt  = (HighByte^255)>>6;     //Dezimalpunkt aus obersten Bits auslesen

  //Data check
  if((CodierteDaten>=GesperrterBereichMin)&&(CodierteDaten<=GesperrterBereichMax))
      {// Daten im verbotenen Wertebereich
      *Fliesspunkt = CodierteDaten;
      Rueckgabe = -1; //Hier Fehlerbehandlung umsetzen!!
      }
  else
      {
      LongPuffer = (long)CodierteDaten-(long)0x0800; //Umwandlung Ganzzahlwert mit
                                                     // Vorzeichen, Offset abziehen
      DoublePuffer = (double)LongPuffer;             //Umwandlung in Fliesspunktzahl
      switch (Dezimalpunkt)                          //Setzen des Dezimalpunktes
                {
                case 0:     break;
                case 1:     DoublePuffer=DoublePuffer/10;     break;
                case 2:     DoublePuffer=DoublePuffer/100;    break;
                case 3:     DoublePuffer=DoublePuffer/1000;   break;
                }
      *Fliesspunkt = DoublePuffer;        //An Returnwert übergeben
      }
  return(Rueckgabe);
}
```

## C-Code to open the interface

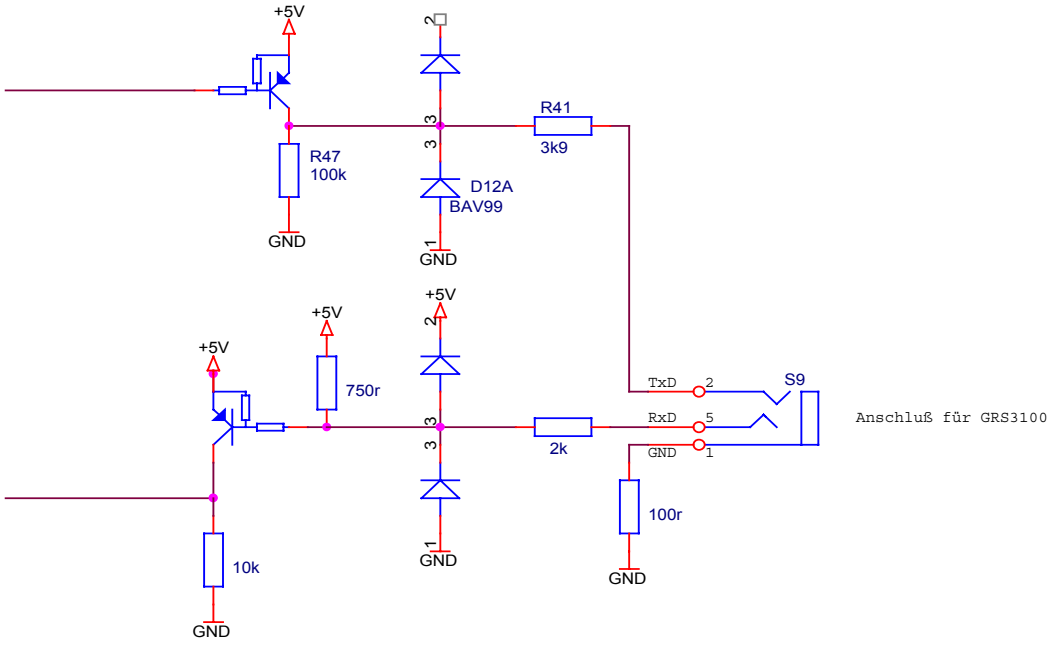
```
   HANDLE    hport;                                          // Decl.Interface-Handle
   DCB       dcb;                                            // Declaration DCB-Block
   BOOL      ok;                                             // Declaration return value
   FillMemory(&dcb, sizeof(dcb),0);                          // clear old DCB settings
   dcb.DCBlength = sizeof(dcb);                              // and determine length
   BuildCommDCB("baud=4800 parity=N data=8 stop=1",&dcb);    // 'pre-setting'
   dcb.fRtsControl = RTS_CONTROL_DISABLE;                    // RTS low:  switch on GRS3xxx
   dcb.fDtrControl = DTR_CONTROL_ENABLE;                     // DTR high: switch on GRS3xxx
   ok = SetupComm(hport, 1024,100);                          // set in/out buffer
   ok = SetCommState(hport, &dcb);                           // set com parameters
```

# C-Code to calculate CRC-Bytes for EASYBus-Protocol (Bytes 2,5,8..)

```
//*****************************************************************************
//Function: Calculates CRC-BYTE
//Input:          _ubByte1 = 1. Byte, must be inverted before
//                _ubByte1 = 2. byte
//Return:   CRC-BYTE
//!! Integer have to be 16bit !!
//*****************************************************************************
unsigned char CrcCalculate(unsigned char _ubByte1, unsigned char _ubByte2)
 {
 unsigned int _uiHilf, _uiLoop     ;

      _uiHilf = (_ubByte1 << 8)+ _ubByte2;
      for (_uiLoop=0; _uiLoop<16; _uiLoop++)
      {
            if    (_uiHilf & 0x8000)      _uiHilf=(_uiHilf << 1)^ 0x700;
            else                          _uiHilf=_uiHilf << 1;
      }
      return ~(_uiHilf >> 8);
 }
```

# Hardware - Serial Interface of GMH3xxx-Series




Level: RxD: idle state = +5V, corresponds to –12V of RS232
TxD: idle state = +0V, corresponds to –12V of RS232

Please keep in mind: a logic 1 is displayed by –12V at RS232 interfaces

# Measuring Value Read Out Codes

Instruction 0 – read measuring value

base address 01

| addr | Byte0 | Byte1 | Byte2 |
|---|---|---|---|
| 01 | 254 | 0 | 061 |
| 02 | 253 | 0 | 002 |
| 03 | 252 | 0 | 023 |
| 04 | 251 | 0 | 124 |
| 05 | 250 | 0 | 105 |
| 06 | 249 | 0 | 086 |
| 07 | 248 | 0 | 067 |
| 08 | 247 | 0 | 128 |
| 09 | 246 | 0 | 149 |
| 10 | 245 | 0 | 170 |

base address 11

| addr | Byte0 | Byte1 | Byte2 |
|---|---|---|---|
| 11 | 244 | 0 | 191 |
| 12 | 243 | 0 | 212 |
| 13 | 242 | 0 | 193 |
| 14 | 241 | 0 | 254 |
| 15 | 240 | 0 | 235 |
| 16 | 239 | 0 | 127 |
| 17 | 238 | 0 | 106 |
| 18 | 237 | 0 | 085 |
| 19 | 236 | 0 | 064 |
| 20 | 235 | 0 | 043 |

base address 21

| addr | Byte0 | Byte1 | Byte2 |
|---|---|---|---|
| 21 | 234 | 0 | 062 |
| 22 | 233 | 0 | 001 |
| 23 | 232 | 0 | 020 |
| 24 | 231 | 0 | 215 |
| 25 | 230 | 0 | 194 |
| 26 | 229 | 0 | 253 |
| 27 | 228 | 0 | 232 |
| 28 | 227 | 0 | 131 |
| 29 | 226 | 0 | 150 |
| 30 | 225 | 0 | 169 |

base address 31

| addr | Byte0 | Byte1 | Byte2 |
|---|---|---|---|
| 31 | 224 | 0 | 188 |
| 32 | 223 | 0 | 134 |
| 33 | 222 | 0 | 147 |
| 34 | 221 | 0 | 172 |
| 35 | 220 | 0 | 185 |
| 36 | 219 | 0 | 210 |
| 37 | 218 | 0 | 199 |
| 38 | 217 | 0 | 248 |
| 39 | 216 | 0 | 237 |
| 40 | 215 | 0 | 046 |

base address 41

| addr | Byte0 | Byte1 | Byte2 |
|---|---|---|---|
| 41 | 214 | 0 | 059 |
| 42 | 213 | 0 | 004 |
| 43 | 212 | 0 | 017 |
| 44 | 211 | 0 | 122 |
| 45 | 210 | 0 | 111 |
| 46 | 209 | 0 | 080 |
| 47 | 208 | 0 | 069 |
| 48 | 207 | 0 | 209 |
| 49 | 206 | 0 | 196 |
| 50 | 205 | 0 | 251 |

# Reading Display Unit

The instruments are supporting a function to read the unit refering zu the read value.
**Code** 202 – read display unit, 6Byte request, 9Byte answer
Base address 01

| Addr | Byte0 | Byte1 | Byte2, | Byte0 | Byte1 | Byte2 |
|---|---|---|---|---|---|---|
| 01 | 254 | 242 | 237 | 053 | 000 | 071 |
| 02 | 253 | 242 | 210 | 053 | 000 | 071 |
| 03 | 252 | 242 | 199 | 053 | 000 | 071 |
| 04 | 251 | 242 | 172 | 053 | 000 | 071 |
| 05 | 250 | 242 | 185 | 053 | 000 | 071 |
| 06 | 249 | 242 | 134 | 053 | 000 | 071 |
| 07 | 248 | 242 | 147 | 053 | 000 | 071 |
| 08 | 247 | 242 | 080 | 053 | 000 | 071 |
| 09 | 246 | 242 | 069 | 053 | 000 | 071 |
| 10 | 245 | 242 | 122 | 053 | 000 | 071 |

**9 Byte response decoding**:

```
// Input: HighByte = Byte6, LowByte=Byte7 of  9Byte response
// Rueckgabe = 0:  Decoding ok, Unit Nr. in *int_dat
short int EinheitNrDekodieren (BYTE HighByte, BYTE LowByte, long *int_dat)
  {
  // Declarations
  short int          Rueckgabe = 0;
// Code
  *int_dat = 0;
  *int_dat = *int_dat | (HighByte^255);
  *int_dat = *int_dat << 8;
  *int_dat = *int_dat | LowByte[7];
  return (Rueckgabe);
  }
```

**Meaning of Unit Nrs:**

| | | | | | |
|---|---|---|---|---|---|
| °C | 1 | U/min | 50 | V | 105 |
| °F | 2 | Hz | 53 | mV | 106 |
| K | 3 | Impuls(e) | 55 | uV | 107 |
| | | | | | |
| % r.F. | 10 | m/s | 60 | W | 111 |
| | | km/h | 61 | kW | 112 |
| bar | 20 | | | | |
| mbar | 21 | | | Wh | 115 |
| Pascal | 22 | mm | 70 | kWh | 116 |
| hPascal | 23 | m | 71 | | |
| kPascal | 24 | inch | 72 | Wh/m2 | 119 |
| MPascal | 25 | ft | 73 | | |
| mmHg | 27 | | | mOhm | 120 |
| PSI | 28 | | | Ohm | 121 |
| mm H20 | 29 | l/h | 80 | kOhm | 122 |
| | | l/min | 81 | MOhm | 123 |
| | | m^3/h | 82 | kOhm/cm | 125 |
| | | m^3/min | 83 | | |
| S/cm | 30 | | | % | 150 |
| mS/cm | 31 | | | ° | 151 |
| uS/cm | 32 | g | 90 | ppm | 152 |
| | | kg | 91 | g/kg | 160 |
| | | N | 92 | kJ/kg | 170 |
| pH | 40 | Nm | 93 | kcal/kg | 171 |
| rH | 42 | | | mg/l | 172 |
| | | | | dB | 175 |
| mg/l O2 | 45 | A | 100 | dBm | 176 |
| % Sat O2 | 46 | mA | 101 | dBA | 177 |