V1.0 2016-01-11

GREISINGER DE

Schnittstellenbeschreibung für EASYBus Sensormodule und GMH Handmessgeräte

ab Version 1.0

Dokumentation

CE

☞ Vor Inbetriebnahme aufmerksam lesen!

☞ Beachten Sie die Sicherheitshinweise!

☞ Zum späteren Gebrauch aufbewahren!

Made in Germany

WEEE-Reg.-Nr. DE 93889386

GHM Messtechnik GmbH • Standort GREISINGER
Hans-Sachs-Str. 26 • D-93128 Regenstauf
☎ +49 (0) 9402 / 9383-0 🖷 +49 (0) 9402 / 9383-33 info@greisinger.de

Inhalt

1 Allgemeiner Hinweis

Lesen Sie dieses Dokument aufmerksam durch und machen Sie sich mit der Bedienung des Gerätes vertraut, bevor Sie es einsetzen. Bewahren Sie dieses Dokument griffbereit und in unmittelbarer Nähe des Gerätes auf, damit Sie oder das Fachpersonal im Zweifelsfalle jederzeit nachschlagen können.
Montage, Inbetriebnahme, Betrieb, Wartung und Außerbetriebnahme dürfen nur von fachspezifisch qualifiziertem Personal durchgeführt werden. Das Fachpersonal muss die Betriebsanleitung vor Beginn aller Arbeiten sorgfältig durchgelesen und verstanden haben.
Die Haftung und Gewährleistung des Herstellers für Schäden und Folgeschäden erlischt bei bestimmungswidriger Verwendung, Nichtbeachten dieser Betriebsanleitung, Einsatz ungenügend qualifizierten Fachpersonals sowie eigenmächtiger Veränderung am Gerät.
Der Hersteller haftet nicht für Kosten oder Schäden, die dem Benutzer oder Dritten durch den Einsatz dieses Geräts, vor allem bei unsachgemäßem Gebrauch des Geräts oder bei Missbrauch oder Störungen des Anschlusses oder des Geräts, entstehen.
Der Hersteller übernimmt keine Haftung bei Druckfehler.

2 Sicherheit

2.1 Bestimmungsgemäße Verwendung

Diese Dokumentation dient als Schnittstellenbeschreibung zur Kommunikation mit GMH Handmessgeräten. Es dürfen nur hier dokumentierte Befehle verwendet werden.

2.2 Sicherheitszeichen und Symbole

Warnhinweise sind in diesem Dokument wie folgt gekennzeichnet:

Warnung! Symbol warnt vor unmittelbar drohender Gefahr, Tod, schweren Körperverletzungen bzw. schweren Sachschäden bei Nichtbeachtung.

Achtung! Symbol warnt vor möglichen Gefahren oder schädlichen Situationen, die bei Nichtbeachtung Schäden am Gerät bzw. an der Umwelt hervorrufen.

Hinweis! Symbol weist auf Vorgänge hin, die bei Nichtbeachtung einen indirekten Einfluss auf den Betrieb haben oder eine nicht vorhergesehene Reaktion auslösen können.

2.3 Hinweise

Zum Schnittstellenbetrieb muss das Gerät eine Schnittstelle besitzen.
Die Schnittstelle im Gerät muss aktiviert sein.
Bei GMH 3xxx-Geräten muss der möglicherweise vorhandene Analogausgang deaktiviert werden. Siehe hierzu Betriebsanleitung des verwendeten Handmessgerätes.
Weitere Voraussetzung sind:

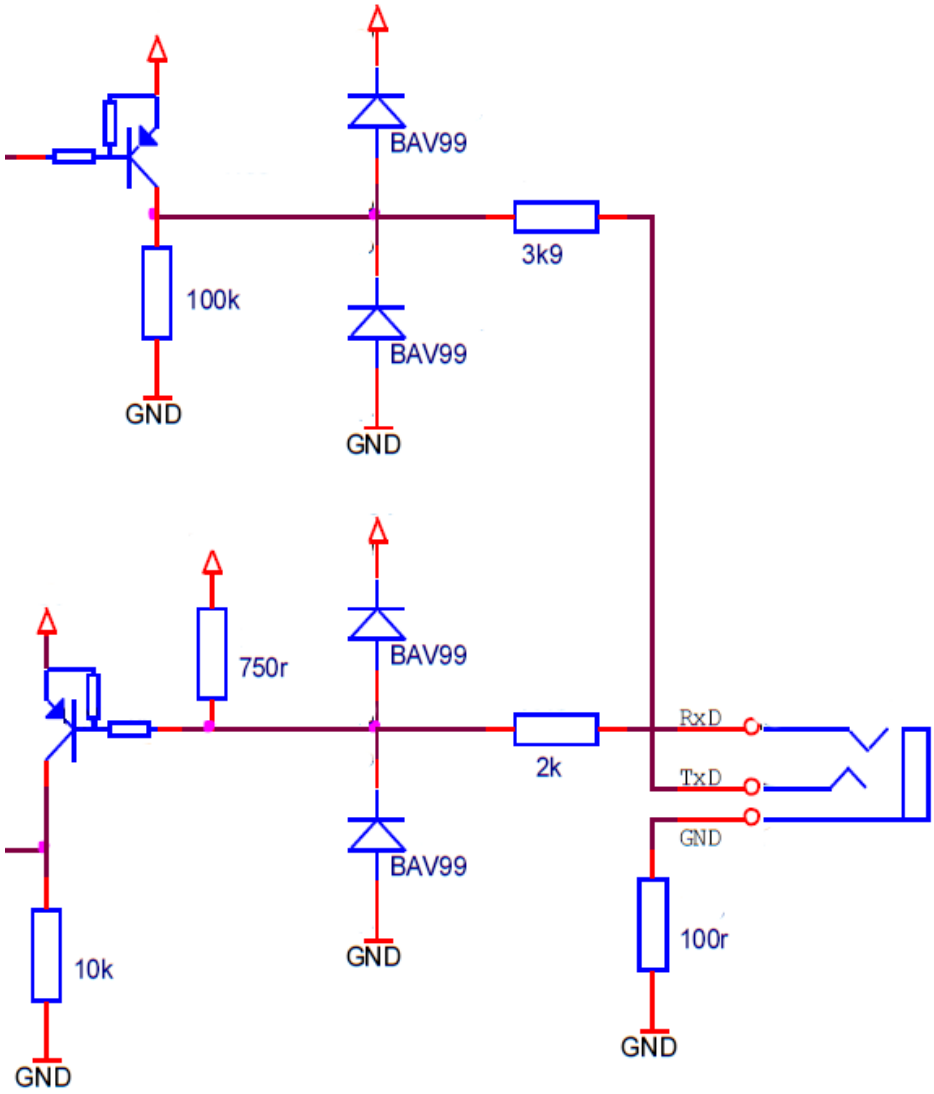
- fehlerfrei arbeitender Aufbau.
- original GMH Schnittstellenadapter (GRS 3100, GRS 3105, USB 3100, USB 3105 oder USB 5100).
- original EASYBus Schnittstellenadapter (EBW 1, EBW 3, EBW 64, EBW 240)
- korrekt initialisierte und geöffnete serielle Schnittstelle
- aktuellste USB-Treiber

3 GRS 3100, GRS 3105, USB 3100, USB 3100 N

Für den Anschluss des GMH 3xxx an einen PC sind die Schnittstellenadapter USB 3100, USB 3100 N, GRS 3100, GRS 3105 vorgesehen. Der entsprechende Adapter sorgt für eine galvanische Trennung zwischen PC und GMH 3xxx. Die USB Versionen verwenden einen Standard USB zu UART Chip um die Gerätekommunikation zu ermöglichen. Bitte beachten Sie die Hinweise in der Bedienungsanleitung des Schnittstellenadapters.

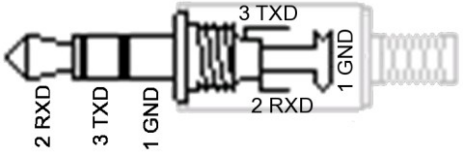
3.1 Hardware (Geräte Sende- und Empfangsstufe)

Beispielbeschaltung. Ein direkter Anschluss an die RS232-Leitung des Computers oder einen RS232-Pegelwandler (z.B. MAX 232) ist nicht zulässig!



3.2 Schnittstellenanschluss

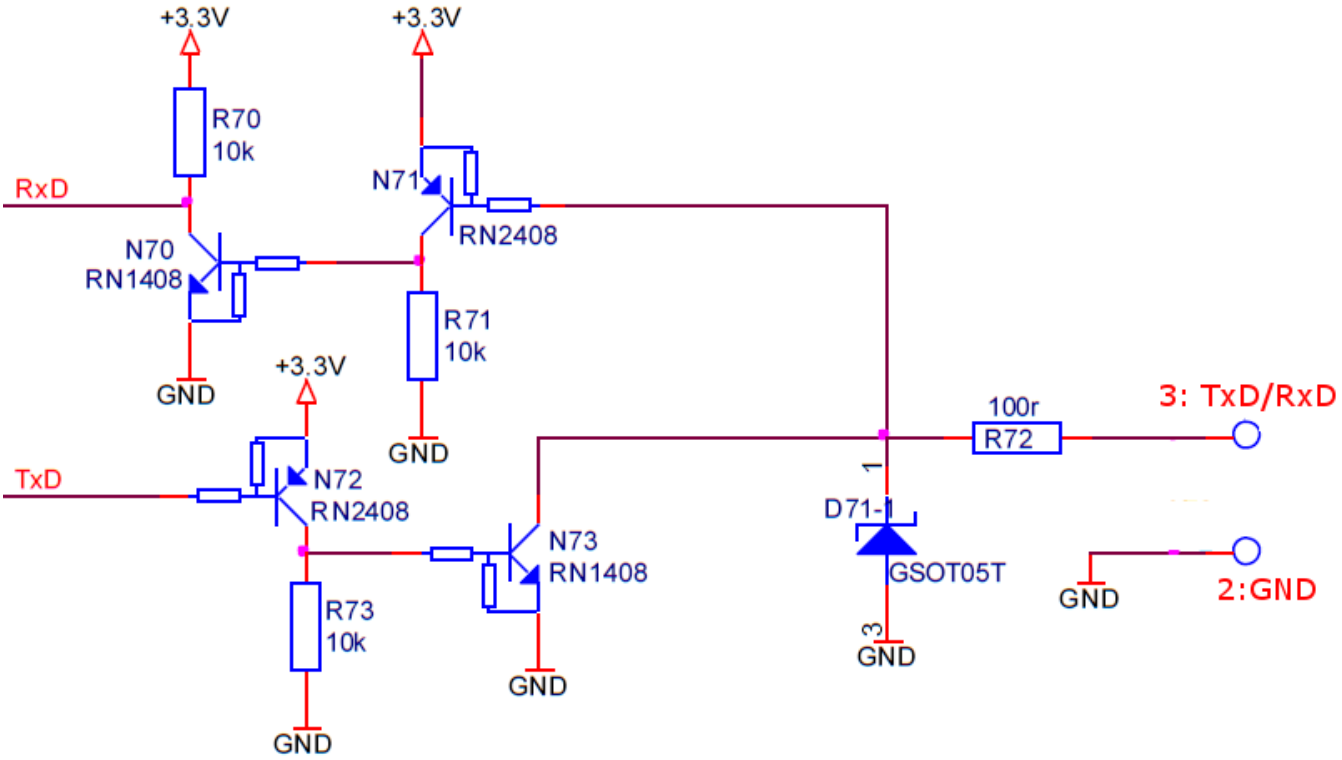
Der Anschluss an die serielle Schnittstelle des GMH3xxx erfolgt mittels 3.5mm Stereo-Klinkenstecker.

GMH 3xxx Geräte-Anschluss	Terminal	Gerät (siehe 3.1)	Geräteseitige Beschreibung Pegel, Hinweise
	1 GND	GND	Masseanschluss Verbindung GND
	2 RxD	TxD	Ausgang, Sendeleitung (Terminal: 2 RxD) Stromquelle mit Serienwiderstand Low Pegel: GND High Pegel: >1 VDC, max. 3,5 mA
	3 TxD	RxD	Eingang, Empfangsleitung (Terminal: 3 TxD) Pullup an Gerätespannung (3,3 oder 5,0 VDC) Low Pegel: 3,3 / 5 VDC High Pegel: GND

4 USB 5100

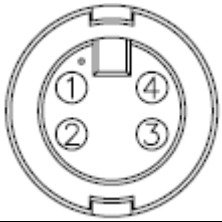
Für den Anschluss des GMH 5xxx an einen PC ist der Schnittstellenadapter USB 5100 vorgesehen. Dieser Adapter sorgt für eine galvanische Trennung zwischen PC und GMH 5xxx und verwendet einen Standard USB zu UART Chip um die Gerätekommunikation zu ermöglichen. Bitte beachten Sie die Hinweise in der Bedienungsanleitung des Schnittstellenadapters.

4.1 Hardware (Geräte Sende- und Empfangsstufe)



4.2 Schnittstellenanschluss

Der Anschluss an die serielle Schnittstelle des GMH 5xxx erfolgt mittels wasserdichtem Steckverbinder

GMH 5xxx Geräte-Anschluss	Bedeutung	Pegel, Hinweise
	1: externe Versorgung	+5 V DC (zulässig 4,5 … 5,5 V DC)
	2: GND	0 V / GND
	3: TxD/RxD	High Pegel: 3,3 VDC Low Pegel: 0 V / GND
	4: $+U_{DAC}$	Analogausgang: 0 … 1,0 V DC

5 EASYBus

5.1 Physikalische Datenübertragung

Die Datenübertragung wird immer vom EASYBus Master gestartet.

5.1.1 Senden

High Pegel 30 V DC
Low Pegel 24 V DC
The EASYBus master modulates between high and low level

5.1.2 Empfangen

High Pegel statische Stromaufnahme
Low Pegel + 16 mA Stromaufnahme
Das EASYBus Sensormodul moduliert indem die Stromaufnahme um mindestens 16 mA erhöht wird.

Da Sensormodul und Master nur einen Spannungs- oder Stromunterschied detektieren ist die EASYBus-Leitung nicht verpolbar. Die meißten Sensormodule können direkt über den EASYBus versorgt werden, diese Module benötigen dann keine separate Spannungsversorgung.

6 Schnittstelleneinstellungen

(In Programm oder Terminal einzustellen, nicht über Windows-Gerätemanager)
Entsprechend den RS232-Festlegungen (Ruhezustand = logische '1')

Baudrate	Datenbit	Parität	Stoppbits	Flusskontrolle
EASYBus / GMH 3xxx: 4800 GMH 5xxx: 38400	8	keine	1	keine

Erweiterte COM-Schnittstellen Einstellungen
(Versorgung der Galvanischen Trennung bei GRS 3100)
DTR Aktiviert
RTS Deaktiviert

6.1 Code zum Initialisieren der Schnittstelle (C++ mit Windows API)

```
HANDLE hPort;                    /* Deklaration Schnittstellen Handle */;
DCB dcb;                         /* Deklaration DCB Block */
BOOL ok;
FillMemory(&dcb, sizeof(dcb), 0);     /* Alte DCB-Einstellungen löschen */
dcb.DCBlength = sizeof(dcb);          /* DCB-Länge festlegen */
dcb.BaudRate = CBR_4800; /* CBR_38400 Baudrate auf '4800'/'38400' festlegen
*/
dcb.Parity = PARITY_NONE;        /* Parität auf 'keine' festlegen */
dcb.ByteSize = DATABITS_8;       /* Datenbits auf '8' festlegen */
dcb.StopBits = STOPBITS_10;           /* Stoppbits auf '1' festlegen */
dcb.fInX = false;
dcb.fOutX = false;
dcb.fOutxCtsFlow = false;
dcb.fOutxDsrFlow = false;
dcb.fDsrSensitivity = false;
dcb.fAbortOnError = false;
dcb.fBinary = true;
dcb.fDtrControl = DTR_CONTROL_ENABLE;
dcb.fRtsControl = RTS_CONTROL_DISABLE;
ok = SetupComm(hPort, 1024, 100);/* Ein- und Ausgangspuffer setzen */
ok = SetupCommState(hPort, &dcb);/* Parameter setzen */
```

6.2 Code zum Initialisieren der Schnittstelle (C# mit .NET Framework)

```
System.IO.Ports.SerialPort spSeriellerPort = new SerialPort("COM1");
spSeriellerPort.BaudRate = 4800; /* Baudrate '4800' oder '38400' */
spSeriellerPort.Parity = System.IO.Ports.Parity.None;
spSeriellerPort.DataBits = 8;
spSeriellerPort.StopBits = System.IO.Ports.StopBits.One;
spSeriellerPort.DtrEnable = true;
spSeriellerPort.RtsEnable = false;
```

6.3 Code zum Initialisieren der Schnittstelle (POSIX C z.B. Linux)

```
struct termios* Port;
Port = (termios*)malloc(sizeof(termios));
memset(Port, 0, sizeof(struct termios));
cfsetispeed(Port, B4800); /* oder 'B38400' */
Port->c_cflag &= ~PARENB;
Port->c_cflag &= ~CSTOPB;
Port->c_cflag &= ~CSIZE;
Port->c_cflag |= (CS8 | CREAD | CLOCAL);
Port->c_cflag &= ~CRTSCTS;
Port->c_lflag |= ~(ISIG | ICANON | XCASE | ECHO | ECHOE | ECHOK | ECHONL |
NOFLSH | IEXTEN | ECHOCTL | ECHOPRT | ECHOKE | FLUSHO | PENDIN | TOSTOP);
Port->c_iflag |= (IGNPAR);
Port->c_oflag &= ~OPOST;
Port->c_cc[VTIME]    = 0;
Port->c_cc[VMIN]     = 1;
```

7 Schnittstellenprotokoll

Entsprechend der EASYBus Spezifikation.

7.1 Grundsätzlicher Nachrichtenaufbau einer 6 Byte Nachricht/Antwort

	Bye0	Byte1				Byte2	Byte3	Byte4	Byte5
	Adresse	Header				CRC	DB0	DB1	CRC
Bit	7 ... 0	7 ... 4	3	2 ... 1	0	7 ... 0	7 ... 0	7 ... 0	7 ... 0
		F1-Code	Priorität	Länge	Richtung				
		Aufruf code	1 = Priorität 0 = keine Priorität	00 = 3 Byte 01 = 6 Byte 10 = 9 Byte 11 = variabel	1 = vom Slave 0 = vom Master				

7.2 Header-Aufbau

Bit 7...4: F1-Code = Aufrufcode (z.B. 0 für Anzeigewert lesen)
Bit 3: Prioritäts-Bit. Liegt eine Priorität vor so ist dieses Bit '1'. Eine Priorität liegt z.B. dann vor, wenn die Min.-Alarmgrenze unterschritten wurde. Dieses Bit ist bei einer Anfrage (Master → Slave) auf '0' zu setzen und nur bei einer Antwort (Slave → Master) relevant.
Bit 2...1: Nachrichtenlänge in Bytes.
Bit 0: Nachrichten-Richtung '0': Master → Slave, '1': Slave → Master

7.3 Schnittstellenkommunikation

Die Daten werden im Polling-Betrieb abgefragt. Man erhält nur auf eine Anfrage gesendet Daten vom Gerät.

Übertragung von Datenbytes
Byte0 wird zuerst übertragen.
Es dürfen keine Steuerzeichen (Zeilenende, ...) übertragen werden.
Die Datenübertragung findet im Raw-Modus statt (auch nicht darstellbare Zeichen).

Invertierung von Datenbytes
Jedes erste Bytes vom 3er-Byteblock (Byte0, Byte3, ...) wird invertiert übertragen.
($Byte0_{Übertragen}$ = 255 - Byte0)

Kontrollbytes
Jedes dritte Bytes vom 3er-Byteblock (Byte2, Byte5, ...) ist ein Kontrollbyte (CRC).
Die Berechnung des CRC-Bytes befindet sich im Anhang.

Decodierung von Antworten auf Messwert-Anfragen
Antworten mit 6 Bytes Länge: Decodierung mit Messwert16Decodieren(...)
Antworten mit 9 Bytes Länge: Decodierung mit Messwert32Decodieren(...)

Besonderheiten der GMH 5xxx-Serie
Geräteantworten enthalten zuerst immer die Anfrage (Echo-Betrieb). Alle Beispiele und Tabellen beziehen sich auf die Daten nach den Echo-Daten.

F1-Code (Aufrufcode)	Bedeutung
0x0	Anzeigewert lesen. 3-Byte-Anfrage, 6 oder 9 Byte-Antwort Antwort mit Funktion Messwert16Decodieren(...) oder Messwert32Decodieren(...) ermitteln
0x3	Systemstatus lesen. 3-Byte-Anfrage, 6 Byte-Antwort Antwort mit Funktion UInt16Decodieren(Byte3, Byte4) ermitteln
0x5	Anfrage nicht unterstützt (nur als Geräteantwort möglich)
0x6	Minwertspeicher lesen. 3-Byte-Anfrage, 6 oder 9 Byte-Antwort Antwort mit Funktion Messwert16Decodieren(...) oder Messwert32Decodieren(...) ermitteln
0x7	Maxwertspeicher lesen. 3-Byte-Anfrage, 6 oder 9 Byte-Antwort Antwort mit Funktion Messwert16Decodieren(...) oder Messwert32Decodieren(...) ermitteln
0xC	Seriennummer lesen. 3-Byte-Anfrage, 9 Byte-Antwort Antwort mit Funktion UInt16Decodieren(Byte3, Byte4), UInt16Decodieren(Byte6, Byte7) und UInt32Decodieren(...) ermitteln. ID-Nummer als HEX-Wert
0xF	Erweiterter Aufrufcode, Bytelänge = 6 Bytes

F1-Code (Aufrufcode)	Byte3 (DB0)	Byte4 (DB1)	Bedeutung
0xF	0xCA	0x00	Anzeigeeinheit lesen. 6-Byte-Anfrage, 9 Byte-Antwort Antwort mit Funktion UInt16Decodieren(Byte6, Byte7) ermitteln
0xF	0xD0	0x00	Kanalzahl lesen. 6-Byte-Anfrage, 9 Byte-Antwort Byte6 = 0x00: Adressweise Kanaladressierung Byte6 = 0x01: Seriennummernweise Kanaladressierung Byte7 = positiv: Kanalzahl Byte7 = negativ: Kanalnummer

7.4 Beispiel Anfragen

Anzeigewert von Adresse 1 lesen:

Byte0	Byte1	Byte2
0xFE (254_D)	0x00 (0_D)	0x3D (61_D)

Byte 0 = 0xFF - Adresse = 0xFE
Byte 1 = F1-Code Anzeigewert lesen = 0x0, Priorität = 0, Länge = 0 (3 Byte), Master-Anfrage = 0 => 0x00
Byte2 = CRC Berechnung von Byte0 und Byte1 = 0x3D

Statuscode von Adresse 2 lesen:

Byte0	Byte1	Byte2
0xFD (253_D)	0x30 (48_D)	0x92 (146_D)

Byte 0 = 0xFF - Adresse = 0xFD
Byte 1 = F1-Code Statuscode lesen 0x3, Priorität = 0, Länge = 0 (3 Byte) , Master-Anfrage = 0 => 0x00
Byte2 = CRC Berechnung von Byte0 und Byte1 = 0x92

Anzeigeeinheit von Adresse 3 lesen:

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
0xFC (252_D)	0xF2 (242_D)	0xC7 (199_D)	0x35 (53_D)	0x00 (0_D)	0x47 (71_D)

Byte 0 = 0xFF - Adresse = 0xFC
Byte 1 = F1-Code Statuscode lesen 0xF, Priorität = 0, Länge = 1 (6 Byte) , Master-Anfrage = 0 => 0xF2
Byte2 = CRC Berechnung von Byte0 und Byte1 = 0xC7
Byte3 = 0xFF - 0xCA, Byte4 = 0x00, Byte5 = CRC Berechnung von Byte3 und Byte4 = 0x47

7.5 Beispiel Antworten

Wenn die entsprechende Adresse vorhanden ist, alle Verbindungen richtig angeschlossen wurden und das Gerät eingeschaltet ist, wird es innerhalb < 1 Sekunde je nach Anfrage mit 3, 6 oder 9 Bytes antworten.
Die ersten 3 Bytes sind der Header, aus diesem kann die Nachrichtenlänge ermittelt werden.
Beispielantwort von Gerät auf Adresse 1 nach der Anfrage Anzeigewert lesen (Anzeigewert -0,04)

	Empfangene Datenbytes								
	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Beispiel für -0.04	0xFE	0x0D	0x10	0x72	0xFF	0x84	0x00	0xFC	0x05
	Header		CRC	Nutzdaten0		CRC	Nutzdaten1		CRC

8 Anhang

8.1 Codes zur Decodierung

```
/* 2 Übertragungs-Bytes zu unsigned 16 Bit Integer zusammenfassen */
UInt16 UInt16Decodieren(byte inbyByteA, byte inbyByteB)
{
      return (UInt16)((UInt16)((255 - inbyByteA) << 8) | inbyByteB);
}

/* 2 unsigned 16 Bit Integer zu unsigned 32 Bit Integer zusammenfassen */
UInt32 UInt32Decodieren(UInt16 inui16Wert1, UInt16 inui16Wert2)
{
      return (UInt32)(inui16Wert1 << 16 | inui16Wert2);
}

Byte[] Beispieldaten = {0xFE, 0x0F, 0x10, 0x72, 0xFF, 0x84, 0x00, 0xFC, 0x05}; /* => -0.04 */
/* 4 Bytes aus Übertragung in Messwert oder Fehlercode umrechnen */
Int16 Messwert32Decodieren (byte inbyByte3, byte inbyByte4, byte inbyByte6,
byte inbyByte7, out double outdblFloatWert, out Int16
outi16DezimalPunktPosition)
{
      outdblFloatWert = 0;
      outi16DezimalPunktPosition = 0;
      UInt16 ui16Integer1, ui16Integer2;
      ui16Integer1 = UInt16Decodieren(inbyByte3, inbyByte4);
      ui16Integer2 = UInt16Decodieren(inbyByte6, inbyByte7);
      UInt32 ui32Integer = UInt32Decodieren(ui16Integer1, ui16Integer2);
      /* Bytes zusammenfassen, mit Beispieldaten: 0x8DFFFFFC */
      outi16DezimalPunktPosition = (Int16)(((0xFF - inbyByte3) >> 3) - 15);
      /* Dezimalpunkt dekodieren, mit Beispieldaten: 0x0002*/
      ui32Integer = ui32Integer & 0x07FFFFFF;
      /* Rohwert dekodieren, mit Beispieldaten: 0x05FFFFFC */
      if ((100000000 + 0x2000000) > ui32Integer)
      {
            /* Daten sind gültige Werte */
            if (0x04000000 == (ui32Integer & 0x04000000))
            {
                  ui32Integer = (ui32Integer | 0xF8000000);
                 /* Mit Beispieldaten: 0xFDFFFFFC */
            }
            ui32Integer = (UInt32)((UInt64)ui32Integer + 0x02000000);
           /* Mit Beispieldaten: 0xFFFFFFFC */
      }
      else
      {
            /* Daten sind Fehlercodes, Fehlercode auscodieren */
            outdblFloatWert = (double)(ui32Integer - 0x02000000 - 16352.0);
            outi16DezimalPunktPosition = 0;
            return -36; /* Rückgabewert ist Fehlercode */
      }
      /* Umwandlung in Fliesspunkt Zahl, mit Beispieldaten: -4f */
      outdblFloatWert = (double)(Int32)ui32Integer;
      outdblFloatWert = outdblFloatWert /
                        (Math.Pow(10.0f, (double)outi16DezimalPunktPosition));
      return 0; /* Rückgabewert ist OK, mit Beispieldaten: -0,04 */
}
```

```
/* 2 Bytes aus Übertragung in Messwert oder Fehlercode umrechnen */
Int16 Messwert16Decodieren (byte inbyByte3, byte inbyByte4, out double
outdblFloatWert, out Int16 outi16DezimalPunktPosition)
{
      Int16 ui16Integer = UInt16Decodieren(inbyByte3, inbyByte4);
      outi16DezimalPunktPosition = (Int16)((ui16Integer& 0xC000) >> 14);
      ui16Integer = (UInt16)(ui16Integer& 0x3FFF);
      if ((ui16Integer >= 0x3FE0) && (ui16Integer <= 0x3FFF))
      {
            outdblFloatWert = (double)ui16Integer - (double)16352.0;
            return -36; /* Rückgabewert ist Fehlercode */
      }
      Int32 i32Nenner = (Int32)System.Math.Pow((double)10.0,
(double)i16DezimalPunktPosition);
      Int32 i32Zaehler = (Int32)((double)ui16Integer - (double)2048.0);
      outdblFloatWert = (double)((double)i32Zaehler / (double)i32Nenner);
      return 0; /* Rückgabewert ist kein Fehler, OK */
}
```

8.2 Fehlercodes

Fehler	Bedeutung
16352	Meßbereich überschritten
16353	Meßbereich unterschritten
16362	Berechnung nicht möglich
16363	Systemfehler
16364	Batterie leer
16365	Kein Sensor
16366	Aufzeichnungsfehler, EEPROM-Fehler
16367	EEPROM-Checksumme falsch
16368	Aufzeichnungsfehler, Fehler 6: Systemneustart
16369	Aufzeichnungsfehler: Datenzeiger
16370	Aufzeichnungsfehler: Markierung, Daten ungültig
16371	Daten ungültig
sonstige	unbekannter EASYBus Fehlercode

8.3 Statuscodes

Bitnummer	Bedeutung
0	Max. Alarm
1	Min. Alarm
2	Darstellbarer Bereich überschritten
3	Darstellbarer Bereich unterschritten
4	- (für zukünftige Änderungen reserviert)
5	- (für zukünftige Änderungen reserviert)
6	- (für zukünftige Änderungen reserviert)
7	- (für zukünftige Änderungen reserviert)
8	Messbereich überschritten
9	Messbereich unterschritten
10	Sensorfehler
11	- (für zukünftige Änderungen reserviert)
12	Systemfehler
13	Berechnung nicht möglich
14	- (für zukünftige Änderungen reserviert)
15	Batteriezustand niedrig

8.4 Einheitencodes

Code	Einheit	Code	Einheit	Code	Einheit	Code	Einheit
1	°C	61	km/h	122	kOhm		
2	°F	62	mph	123	MOhm		
3	K	63	Knoten				
				125	kOhm*cm		
				126	MOhm*cm		
10	% r.F.	70	mm				
		71	m				
		72	inch	130	cd		
		73	ft	131	lx		
		74	cm	132	lm		
		75	km			190	sone
						191	phon
						192	µPa
18	inHg(0°C)					193	dB(SPL)
19	inHg(60°F)	79	l/s				
20	bar	80	l/h				
21	mbar	81	l/min				
22	Pascal	82	m^3/h				
23	hPascal	83	m^3/min				
24	kPascal	84	nm^3/h				
25	MPascal	85	ml/s				
26	kg/cm^2	86	ml/min				
27	mmHg	87	ml/h				
28	PSI	88	m^3/s				
29	mm H20						
30	S/cm	90	g				
31	mS/cm	91	kg				
32	µS/cm	92	N	150	%		
		93	Nm	151	°		
		94	t	152	ppm		
				153	ppb		
40	pH	100	A				
42	rH	101	mA				
		102	µA	160	g/kg		
				161	g/m^3		
45	mg/l O2	105	V	162	mg/m^3		
46	% Sat O2	106	mV	163	µg/m^3		
47	% O2	107	µV				
50	U/min						
		111	W				
		112	kW				
53	Hz			170	kJ/kg		
		115	Wh	171	kcal/kg		
55	Impulse	116	kWh	172	mg/l		
		117	mW/cm²	173	g/l		
				175	dB		
		119	Wh/m2	176	dBm		
		120	mOhm	177	dBA		
60	m/s	121	Ohm				

8.5 Code zur CRC-Berechnung

```
Byte[] Beispieldaten = {0xFE, 0x0F, 0x10, 0x72, 0xFF, 0x84, 0x00, 0xFC, 0x05};
byte CRC(byte inbyByte0, byte inbyByte1)
{
      byte byCRCByte = 0;
      UInt16 ui16Integer = (UInt16)((inbyByte0 << 8) | inbyByte1);
      for (UInt16 ui16Zaehler = 0; ui16Zaehler < 16; ui16Zaehler++)
      {
            if ((ui16Integer & 0x8000) == 0x8000)
            {
                  ui16Integer = (UInt16)((ui16Integer << 1) ^ 0x0700);
            }
            else
            {
                  ui16Integer = (UInt16)(ui16Integer << 1);
            }
      }
      byCRCByte = (byte)(255 - (ui16Integer >> 8));
      return byCRCByte;
}
bool CRCBerechnen(byte[] inByteArray, UInt16 inLaenge)
{
      if (inLaenge >= 3)
      {
            if (inByteArray[2] != CRC(inByteArray[0], inByteArray[1]))
            {
                  return false;
            }
      }
      if (inLaenge >= 6)
      {
            if (inByteArray[5] != CRC(inByteArray[3], inByteArray[4]))
            {
                  return false;
            }
      }
      if (inLaenge >= 9)
      {
            if (inByteArray[8] != CRC(inByteArray[6], inByteArray[7]))
            {
                  return false;
            }
      }
      if (inLaenge >= 12)
      {
            if (inByteArray[11] != CRC(inByteArray[9], inByteArray[10]))
            {
                  return false;
            }
      }
      return true;
}
```