Communications protocol

for the LEX - Manometer from KELLER

1 Introduction

This document describes the communications protocol for the LEX-Manometer from KELLER Druckmesstechnik. In addition to these transmitters, other devices such as data loggers or manometers are also offered. These products are distinguished by the designation CLASS. Within this device class, the individual device groups are differentiated by the designation GROUP. LEX-Manometer bear the CLASS designation 10.

The protocol itself is based on MODBUS, but incorporates optimised functions for the device.

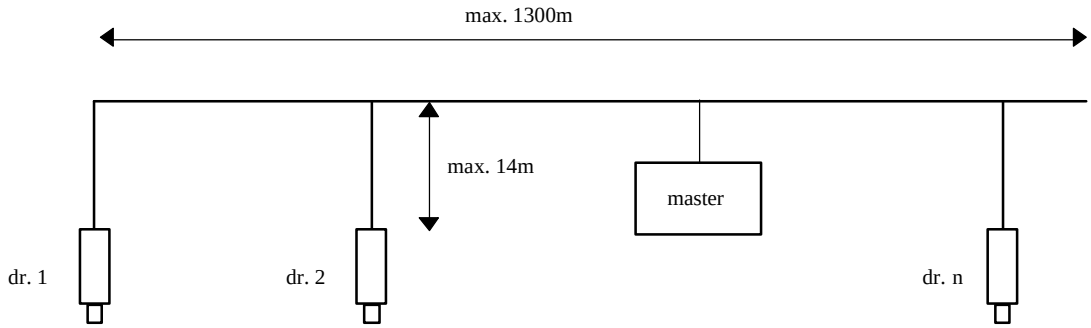
2 Bit transfer layer (physical layer)

2.1 Introduction

The physical connection is provided by the RS485 serial interface. This guarantees good interference immunity and enables a flexible bus structure, i.e. several devices can be administrated as slaves by a single master. In order to minimise the scope of cabling, the RS485 is used in half-duplex mode. This means that 2 wires are required for communications.

2.2 Characteristic

In order to operate several devices at one serial interface, they are simply all connected in parallel (RS485A, RS485B). Before incorporating the devices into the bus, each device must be programmed with a different address.
It is possible to configure a network up to a length of 1300 metres with a maximum of 128 devices. Each riser cable may be up to 14 m in length. The employed cable should correspond to specification EIA RS485. This means that the wires must be installed in twisted pairs, for example.



The following RS485 driver is used in the devices:
MAX3471

These drivers are slew rate-limited and do not require any bias resistors. Most applications do not require terminating resistors, either.
Voltage protection is installed on both lines (RS485A and RS485B) in the devices. The common mode voltage relative to GND is –7V ... + 12 V. This voltage must not be exceeded in any circumstances.

Further information on RS485: http://www.maxim-ic.com/MaximProducts/Interface/rs-485.htm
Information on wiring: http://www.maxim-ic.com/appnotes.cfm/appnote_number/763

3 Data-link layer

This section describes how data interchange is effected on this bus. The data and their check and control structures are grouped together to form messages. These constitute the smallest communication unit, i.e. only messages can be exchanged between the devices. As a half-duplex protocol is in use here, only one device can use the bus as a transmitter at any one time. All other devices are then in receiver mode. The master takes the form of a PC or microcontroller, for example, and the devices are the slaves. Each message exchange takes place under the control of the master. The message contains the address for the receiving slave.

This results in the following 2 options for data interchange :

a) Broadcasting	This mode of communication enables the master to transmit a message to all slaves simultaneously. The master does not receive a reply, however, and is thus unable to check whether the message has been correctly received by every slave.
b) Data interchange	This mode of communication enables the master to communicate with a single slave. This normally involves the transmission of two messages: the master transmits a request and the slave responds to this request. Only the master is permitted to request a response. The request is received by every slave, but only the selected slave responds. The response must be received within a stipulated time, otherwise the master will assess the attempt as failed and must transmit the request again.

3.1 Transmission format for the serial interface

The data are transmitted serially via the bus. The following format applies:

- 1 start bit
- 8 data bits (the least significant bit first)
- 1 stop bit
- 9600 baud

This results in 10 bits per transmission byte.

3.2 Format of a message

3.2.1 Format of the message sent by the master

Note on the presentation of messages: Each box presents 1 data byte consisting of 8 bits, unless otherwise stated.

Each message sent by the master possesses the following format:

DevAddr	0	Function code	n byte parameters (optional)	CRC16_H	CRC16_L

- `DevAddr`: Address of the device.
  Address 0 is reserved for broadcasting.
  Addresses 1...249 can be used for bus mode.
  Address 250 is transparent and reserved for non-bus mode. Every device can be contacted with this address.
  Addresses 251...255 are reserved for subsequent developments.
- `Function code`: Function number
  A function is selected and executed by the device via the function number. The function number is encoded in 7 bits. Bit 7 is always 0. The functions are described further below.
- `Parameters`:
  The parameters required by the function (n = 0 .. 6, according to function)
- `CRC16`: 16-bit checksum
  These two check bytes serve to verify the integrity of the received data. If an error is established, the entire message will be discarded. The principle employed for CRC16 calculation is described in the appendix. The CRC16 standard is applied here.

Note: The length of a message from the master is at least **4** bytes.

3.2.2 Format of the message sent by the slave

A message transmitted by the slave possesses the following format:

DevAddr	X	Function code	n byte data (optional)	CRC16_H	CRC16_L

- `DevAddr`: Address of the device.
  This address corresponds to the address of the responding device.
- `Function code`:
  The function number is identical to the function number sent by the master. If the most significant bit is X = 0, this indicates that the function has been executed correctly. If bit X = 1, an exception error has occurred.
- `Data`:
  Any data requested via the function follow here.
- `CRC16`:
  See above.

Note: A message from the slave has a minimum length of **5** bytes, and a maximum length of **10** bytes.

3.3 Principle of message interchange

3.3.1 General rules

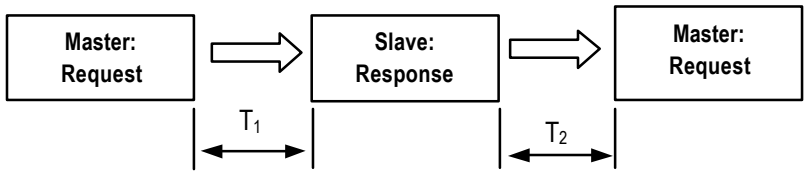
- An address may only be allocated to **one** device connected to the bus. If two devices on the bus have the same address, both will respond, leading to a conflict.
- Every data interchange is initiated by the master. This means that a device may only transmit data if requested to do so by the master.
- A message consists of several bytes. These bytes are transmitted **without any timedelay between eache Byte**.
- The addressed device must respond within time T_1, otherwise the message will be invalid.

Bit frame:

ST	D0	D1	D2	D3	D4	D5	D6	D7	SP	ST	D0	…	D6	D7	SP

Whereby: ST: start bit, SP: stop bit. A parity bit (if active) is inserted before the SP, D0 .. D7: 8 data bits

Message frame:



Response times:

- T_1: Time between receipt of inquiry and beginning of response.
  Min. 100ms to max. 500ms for all functions and devices.
  T_2: Time to ready-to-receive state for the slave.
  min. 2ms

3.3.2 Treatment of errors

2 types of errors may occur during the interchange of messages between master and slave: transmission errors and exception errors.

3.3.2.1 Transmission errors

These errors are primarily accountable to line faults. The message format is incorrect. The following problems are possible :

- A received message is too short.
- A message is longer than the internal transmission buffer permits.
- The word length cannot be interpreted correctly.
- The CRC16 checksum is incorrect.

In response to a transmission error, all received data are ignored. The slave remains in receive mode while the master is required to initiate a new data interchange.

3.3.2.2 Exception errors

The message has been received correctly (no transmission error has occurred), but the transmitted function number and/or the parameters are invalid. The slave responds with an exception error, unless the message has been received in broadcasting mode.
The message transmitted as a response by the slave has the following format:

DevAddr	1	Function code	Exception code	CRC16_H	CRC16_L

4 types of exception errors are defined :

- non-implemented function 1
- incorrect parameters 2
- erroneous data 3
- initialisation 32

Exception error 32 occurs when the device is started up anew and initialisation has not been carried out. This happens every time the device is connected anew after a break in the power supply.

4 Description of functions

This section describes the functions of the bus protocol for LEX-Manometer.

Overview:

F30: Read scaling values
F31: Write scaling values
F48: Initialise devices, whereby the device ID is returned
F66: Programme bus address
F69: Read serial number
F73: Read out current pressure and temperature values in floating-point format
F95: Zeroing functions and commands
F100: Read configurations

4.1 Function 30: Read coefficient

Request:

DevAddr	30	Nr.	CRC16_H	CRC16_L

Response:

DevAddr	30	B3	B2	B1	B0	CRC16_H	CRC16_L

Exception errors:

2 if no. > 111
3 if message length incorrect
32 if device is not yet initialised

Note:
Every coefficient can be read in IEEE754 format (floating-point format 4-byte B0 .. B3) via this function.
→ Information on IEEE754: see appendix.

4.1.1 Calibration values

No.	Description of coefficient	Unit
52	Offset of pressure sensor P1 (by Key or Function 95)	bar
53	Offset of pressure sensor P2 (by Key or Function 95)	bar
64	Offset of pressure sensor P1 (Calibration)	bar
65	Gain factor of pressure sensor P1 (Calibration)	
66	Offset of pressure sensor P2 (Calibration)	bar
67	Gain factor of pressure sensor P2 (Calibration)	
112	Custom-Unit-Factor of Unit 1	
113	Custom-Unit-Factor of Unit 2	
114	Custom-Unit-Factor of Unit 3	
115	Custom-Unit-Factor of Unit 4	
116	Custom-Unit-Factor of Unit 5	

The calibration values can be read and written.

4.1.2 Information values

No.	Description of the coefficient	Unit
80	Minimum pressure of sensor P1	bar
81	Maximum pressure of sensor P1	bar
82	Minimum pressure of sensor P2	bar
83	Maximum pressure of sensor P2	bar
84	Minimum temperature of temperature sensor	°C
85	Maximum temperature of temperature sensor	°C
86	Minimum temperature of sensor P1	°C
87	Maximum temperature of sensor P1	°C
88	Minimum temperature of sensor P2	°C
89	Maximum temperature of sensor P2	°C

4.1.3 Scaling of channels P1 and P2

P1 and P2 are linearly scalable with zero point and gain factor: **Value = gain factor * value + offset**
Standard values: Offset = 0, gain factor = 1.0

The gain factor should be used **for calibration purposes only**, and not to alter pressure units. The latter operation should always be carried out by the master!

4.2 Function 31: Write coefficient

Request:

DevAddr	31	Nr.	B3	B2	B1	B0	CRC16_H	CRC16_L

Response:

DevAddr	31	0	CRC16_H	CRC16_L

Exception errors:

- **2** If Nr. Is not valide for writeoperation
- **3** If message length is incorrect
- **32** If device has not yet been initialised

Note:
Information on scaling of the channels: See functions 73 and 95. Information on which channels are active: See function 100.

4.3 Function 48 : Initialise and release

Request:

DevAddr	48	CRC16_H	CRC16_L

Response:

DevAddr	48	CLASS	GROUP	YEAR	WEEK	BUF	STAT	CRC16_H	CRC16_L

Exception error:

3 If message length incorrect

Note:
Each time the device is switched on by applying the supply voltage or after a break in the power supply, the device must be initialised via this function. Calling a different function will lead to **exception error 32**.

The following information is returned:

CLASS Device ID code
10: for LEX-Manometer

GROUP Subdivision within a device class

YEAR, WEEK Firmware version

BUF Length of the internal receive buffer

STAT Status information
0: Device addressed for first time after power on (Insert Battery).
1: Device was already initialised

4.4 Function 66 : Write new device address or read actual address

Request:

DevAddr	66	NewAddr	CRC16_H	CRC16_L

Response:

DevAddr	66	ActAddr	CRC16_H	CRC16_L

Exception error:

3 If message length is incorrect
32 If device is not yet initialised

Note:
This function programmes the device addresses to NewAddr. The address is returned in ActAddr as confirmation. It is to be ensured that the new address NewAddr is not already in use by another bus user.

Permissible addresses: 1 .. 249. Address 250 is transparent. This means that every device, irrespective of the set address, will respond to address 250. Consequently, *transparent* DevAddr = 250 may only be used in stand-alone operating mode!

For the purpose of **reading the device address** when the address is not known, for example, the value 250 is transferred as DevAddr and the value 0 is transferred as NewAddr. The current address is then returned in response.

4.5 Function 69 : Read serial number

Request:

DevAddr	69	CRC16_H	CRC16_L

Response:

DevAddr	69	SN3	SN2	SN1	SN0	CRC16_H	CRC16_L

Exception errors:

3 If message length is incorrect
32 If device is not yet initialised.

Note:
The serial number is allocated at the factory. It consists of 4 bytes and is calculated as follows :

$SN = 256^3 * SN3 + 256^2 * SN2 + 256 * SN1 + SN0$

4.6 Function 73 : Read value of a channel (floating point)

Request:

DevAddr	73	CH	CRC16_H	CRC16_L

Response:

DevAddr	73	B3	B2	B1	B0	STAT	CRC16_H	CRC16_L

Exception errors:

2 If CH > 7
3 If message length is incorrect
32 If device is not yet initialised

Note:
A device can measure up to five signals (channels):
Two independent pressure sensors, P1 and P2. Plus the temperatures of pressure sensors TOB1 and TOB2 respectively. The temperatures of the pressure sensors (TOB1, TOB2) are required for temperature compensation of the pressure signal. A temperature sensor (T) can also be measured.

On a standard LEX-Manometer, only channels P1 and TOB1 are available. You can read out which channels are available via function 100.

The measured value is returned in IEEE754 format (4-byte B0 ... B3).

CH	Name	Description	Unit
0	P1-P2	Pressure difference	bar
1	P1	Pressure from pressure sensor 1	bar
2	P2	Pressure from pressure sensor 2	bar
3	T	Additional temperature sensor	°C
4	TOB1	Temperature of pressure sensor 1	°C
5	TOB2	Temperature of pressure sensor 2	°C
6	P1 Min	Maximal Pressure (same like on lower display)	bar
7	P1 Max	Maximal Pressure (same like on lower display)	bar

* Dependent on definition in function 100.

The **STAT** byte contains the current status.

Bit position	.7	.6	.5	.4	.3	.2	.1	.0
Name	/STD	---	TOB2	TOB1	T	P2	P1	CH0

A set **/STD** bit indicates whether the transmitter is in Power-up mode, otherwise it is in Standard mode.

A set **P1**, **P2**, **T**, **TOB1**, **TOB2** bit indicates that a measuring or computation error has occurred in the channel concerned.

Function 95 : Commands for setting the zero point and reset Min / Max

Requests:

Request a:

DevAddr	95	CMD	CRC16_H	CRC16_L

Request b with setpoint:

DevAddr	95	CMD	*B3*	*B2*	*B1*	*B0*	CRC16_H	CRC16_L

Response:

DevAddr	95	0	CRC16_H	CRC16_L

Exception errors:

1	If in Power-up mode
2	If CMD invalid
3	If message length incorrect
32	If device is not yet initialised

Note:
The following actions can be carried out with this function:

CMD	Meaning
0	Set zero point of P1
1	Reset zero point of P1 to standard value
2	Set zero point of P2
3	Reset zero point of P2 to standard value
4	
5	Reset Min/Max --> Max / Min Value to actual Value
6	
7	

CMD 0, 2:
Zero point values for pressure channels P1 and P2. These values can also be read via function 30 and written via function 31.

Request a: The zero point is calculated such that the current measured value = 0.
Request b: The zero point is calculated such that the current measured value equals the setpoint.

CMD 1, 3, 7: Reset zero point to factory setting
The zero point values are reset to 0.

4.7 Function 100 : Read configuration

Request:

DevAddr	100	Index	CRC16_H	CRC16_L

Response:

DevAddr	100	PARA0	PARA1	PARA2	PARA3	PARA4	CRC16_H	CRC16_L

Exception errors:

2 If index > 8
3 If message length is incorrect
32 If device is not yet initialised

Note:
This function supplies the information as to how the device has been configured. This configuration is carried out at the factory and cannot be altered by the customer.
A pressure transmitter can read two independent pressure sensors (P1 and P2), plus the temperatures of the respective pressure sensors (TOB1 and TOB2) and an independent temperature (T).

Index	**Para0**	**Para1**	**Para2**	**Para3**	**Para4**
2	CFG_P	CFG_T	---	---	CNT_T

CFG_P: Channels which are measured with maximum priority (typically pressure P1)
CFG_T: Channels which are measured every **CNT_T** seconds (typically temperature TOB1)
After **CNT_T** seconds the temperature correction for the pressure sensor is calculated anew.

CFG_P and **CFG_P** are encoded as follows:

Bit position:	.7	.6	.5	.4	.3	.2	.1	.0
Description:	-	-	TOB2	TOB1	T	P2	P1	-

5 Appendix

5.1 Interface converter

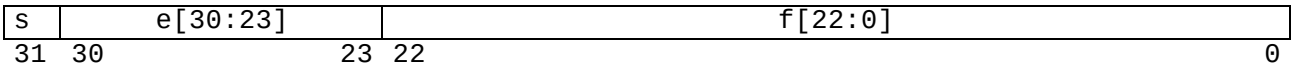
The serial RS232 interface or the USB interface can be used for connection to a PC. KELLER offers converters for this purpose. Various other products are commercially available, however. The following requirements apply when working with KELLER software:

- The converter must control transmit / receive switch-over automatically.
- KELLER converters feature a hardware echo, i.e. the transmitted message is received again immediately as an echo. This echo is required by some KELLER software programmes.

5.2 Floating-point format IEEE754

As data transmission is effected byte-wise (8-bit data), the floating-point values are represented as follows :
B0: Bit 0..7; B1: Bit 8..15, B2: Bit 16..23, B3: Bit 24..31

Representation in accordance with IEEE754:



If you use the DLL which is available from KELLER, you do not need to carry out conversion, as this is encapsulated in the DLL. If you wish to address the devices directly, however, you must convert the individual bytes into a floating-point value.
To obtain a floating-point value from the individual bytes, proceed as follows:

1. Define data structure in which an array of 4 bytes and a 32-bit floating-point value is defined at the same memory location.
2. Write the bytes into the byte array.
3. Read out the floating-point value.

You do not need to carry out any actions, therefore, as the computer attends to interpretation. Some microcontrollers have a different data structure for floating-point values. In such cases, adaptation is necessary.

Further information is to be found at:
http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html - 556

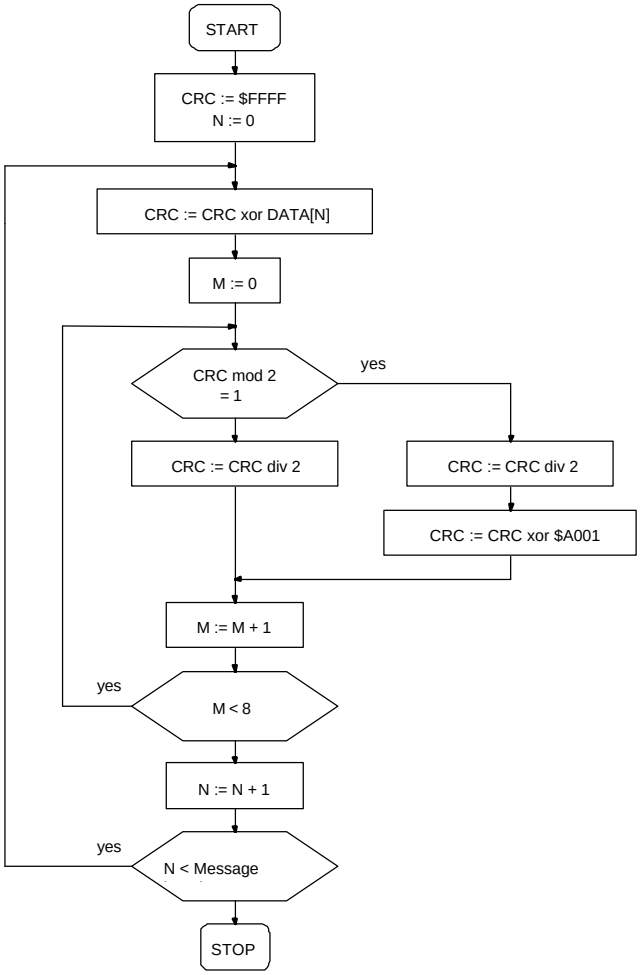
5.3 Calculation of the CRC16 checksum

The checksum can either be calculated or derived from a table.
Here is an example of CRC16 calculation in C:

```
//////////////////////////////////////////////////////////////////////////
// CRC-16 calculation in C
//
// Calculation of CRC-16 checksum over an amount of bytes in the serial buffer.
// The calculation is done without the 2byte from crc16 (receive-mode).
// SC_Buffer[]: Byte-Buffer for the serial interface. Type: unsigned char (8bit)
// SC_Amount : Amount of Bytes which should be transmitted or are received (without CRC16)
//
//////////////////////////////////////////////////////////////////////////
void CalcCRC16(unsigned char* CRC_H, unsigned char* CRC_L)
{
    // locals
    unsigned int Crc;
    unsigned char  n, m, x;

    // initialisation
    Crc= 0xFFFF;
    m= SC_Amount;
    x= 0;

    // loop over all bits
    while(m>0)
    {
        Crc^= SC_Buffer[x];
        for(n=0; n<8; n++)
        {
           if(Crc&1)
           {
               Crc>>= 1;
               Crc^= 0xA001;
           }
           else
               Crc>>= 1;
        }
        m--;
        x++;
    }
    // result
    *CRC_H= (Crc>>8)&0xFF;
    *CRC_L= Crc&0xFF;
}// end CalcCRC16
```



This results in the following calculation for function 48 with device address 250: CRC16_H= 4, CRC16_L= 67.

Examples showing use based on a table are to be found in the MODBUS documentation at:
http://www.modbus.org

5.4 Description of the software driver (DLL)

5.4.1 General

The available DLL ***s30c.dll*** has been tested on the Windows 95, 98, NT and 2000 operating systems.
Examples of the use of this DLL are available for the following programming languages:

- LabVIEW
- C++
- Delphi
- VB
- VBA

The call convention **stdcall** is used for assigning the parameters to the functions. This means that:

– all parameters are passed via the stack,
– the parameter furthest to the right is calculated and passed first, the parameter furthest to the left is calculated and passed last,
– the function itself deletes the parameters from the stack.

As the declarations for the functions presented below show, many variables are declared with the prefixed word *var*. This means that these variables are passed as pointers and not as values.
The types employed for declaration purposes are described below:

Type	**Range**	**Format**
Byte	0..255	8-bit without sign
Word	0..65535	16-bit without sign
Smallint	-32768..32767	16-bit with sign
Longint	-2147483648.. 2147483647	32-bit with sign
Pbyte		Pointer to byte
Single	+/- $1.5x10^{-45}$..$3.4x10^{38}$	32-bit

5.4.2 The functions of the DLL

Each function returns a value which indicates whether the desired function has been successfully executed or not. All the possible return values are specified below. The returned parameters are only valid and may only be processed if the function concerned has been successfully executed.

Return value		**Description**
RS_OK	0	Function successfully executed; return parameters are valid
RS_EX1	1	Function successfully executed; but exception error 1 has occurred
RS_EX2	2	Function successfully executed; but exception error 2 has occurred
RS_EX3	3	Function successfully executed; but exception error 3 has occurred
RS_EX32	32	Function successfully executed; but exception error 32 has occurred
RS_BROADCAST	100	Broadcast
RS_ERROR	-1	General error
RS_TXERROR	-2	Transmit error
RS_RXERROR	-3	Receive error in UART
RS_TIMEOUT	-4	No data or insufficient data received
RS_BADDATA	-5	Data erroneous (e.g. CRC16 erroneous)

5.4.2.1 Port functions

The devices are connected to the PC via a serial interface. The port functions serve to open and close this interface. Ports 1 to 9 (COM1..COM9) are valid. The standard setting should be used for the timeout time (Timeout = 0). When the desired port has been successfully opened, the **OpenComPort** function returns the value RS_OK, otherwise RS_ERROR.
An open port is closed automatically on ending the programme.
It is additionally possible to set the baud rate and the data format via the **OpenComExt** function. KELLER devices only support 9600 baud. Exception: Transmitters with firmware 5.20 can also be operated at 115'200 baud. Use the READ30 software from KELLER to change the transmitter's baud rate.
As a standard setting, **no** parity is used (none). This results in a data format of 10 bits per byte. If parity is active, the data format is 11 bits per byte.

```
function OpenComPort( intPort, intTimeout: Smallint ): Smallint; stdcall; export;
```

```
function OpenComExt( intPort, intTimeout: Smallint; longBaud: Longint; intParity:Smallint
): Smallint; stdcall; export;
```

`intParity`: 0: no parity bit (sStandard), 1: odd parity bit, 2: even parity bit
`longBaud`: 9600 for 9600 baud, 115'200 for 115'200 baud

```
function CloseComPort : Smallint; stdcall; export;
```

5.4.2.2 Echo function

Interface converters from KELLER Druckmesstechnik always supply an echo of the message transmitted by the PC.
This function has the standard value 1 (Echo On), to enable operation with the converters supplied by KELLER. If other converters are used which do not supply a hardware echo, the function must be set to 0 = Echo Off .

```
function EchoOn(  bteEcho: Byte ): Smallint; stdcall; export;
```

5.4.2.3 Protocol functions

The following functions encapsulate the above-described bus functions. The parameter sequences are identical. The CRC16 checksum is not included here, as it is calculated and checked in the DLL. Some parameters consist of several bytes. These are grouped together for the sake of clarity. The different requests a and b pertaining to function 95 are split into two functions: F95 and F95val.
Functions F34, F35, F64, F65 and F101 are only listed here for the sake of completeness, and are of no relevance in these devices.

```
function F30(  bteDeviceAddr, bteCoeffNo: Byte; var sinCoeff: Single
): Smallint; stdcall; export;

function F31(  bteDeviceAddr, bteCoeffNo: Byte; sinCoeff: Single
): Smallint; stdcall; export;

function F48(
  bteDeviceAddr: Byte; var bteClass, bteGroup, bteYear, bteWeek, bteBuffer, bteState: Byte
): Smallint; stdcall; export;

function F64(  bteDeviceAddr: Byte; wrdAddr: Word; bteAmount: Byte; pbteData: PByte
): Smallint; stdcall; export;

function F65(  bteDeviceAddr: Byte; wrdAddr: Word; bteAmount: Byte; pbteData: PByte
): Smallint; stdcall; export;

function F66(  bteDeviceAddr, bteNewAddr: Byte; var bteActualAddr: Byte
): Smallint; stdcall; export;

function F69(  bteDeviceAddr: Byte; var linSN: Longint
): Smallint; stdcall; export;

function F73(  bteDeviceAddr, bteChannel: Byte; var sinValue: Single; var bteStat: Byte
): Smallint; stdcall; export;

function F95(  bteDeviceAddr, bteCmd: Byte
): Smallint; stdcall; export;

function F95val(  bteDeviceAddr, bteCmd: Byte; sinVal: Single
): Smallint; stdcall stdcall; export;

function F100(
  bteDeviceAddr, bteIndex: Byte; var btePara0, btePara1, btePara2, btePara3, btePara4: Byte
): Smallint; stdcall stdcall; export;
```

5.5 Changes

5.6 Support

We are pleased to offer you support in implementing the protocol.

For implementation under Windows, a DLL with diverse reference implementations is available.

The READ30 software is available free of charge for the configuration and read-out of up to 16 devices.
Download: http://www.keller-druck.com

KELLER AG für Druckmesstechnik
St. Gallerstrasse 119 • CH-8404 Winterthur
Tel: ++41 52 235 25 25
http://www.keller-druck.com