Description of the communication protocol

for interface converter

K-114

from KELLER Druckmesstechnik
Class.Group = 20.1

Version 1.0

CONTENT

1 Introduction

This document describes the communications protocol for the interface converter K-114 from KELLER Druckmesstechnik. In addition to these converter, 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.

The software version number consists of following components:

short-designator:	**Class**	**Group**	**Year**	**Week**
	Device group		SW-Version	
K-114	**20**	**1**	**12**	**11**

In this document, the software version is defined by **Class.Group-Year.Week**, e.g. 20.1 - 12.11.

The protocol itself is similar to MODBUS and incorporates optimised functions for the device, these functions are called Keller bus functions.

See Appendix for an overview of the different versions.

2 Bit transfer layer (physical layer)

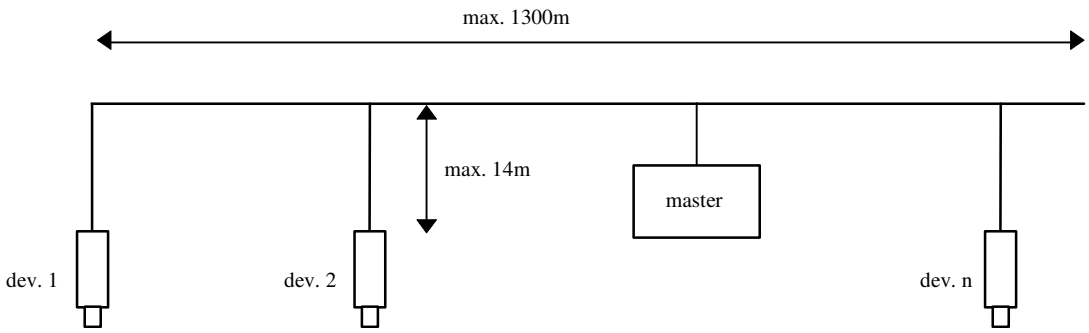
2.1 Introduction

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 and 2 wires for power infeed.

2.2 Characteristic

In order to operate several devices at one serial interface, they are simply all connected in parallel (RS485A, RS485B, GND and +Vcc). 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.

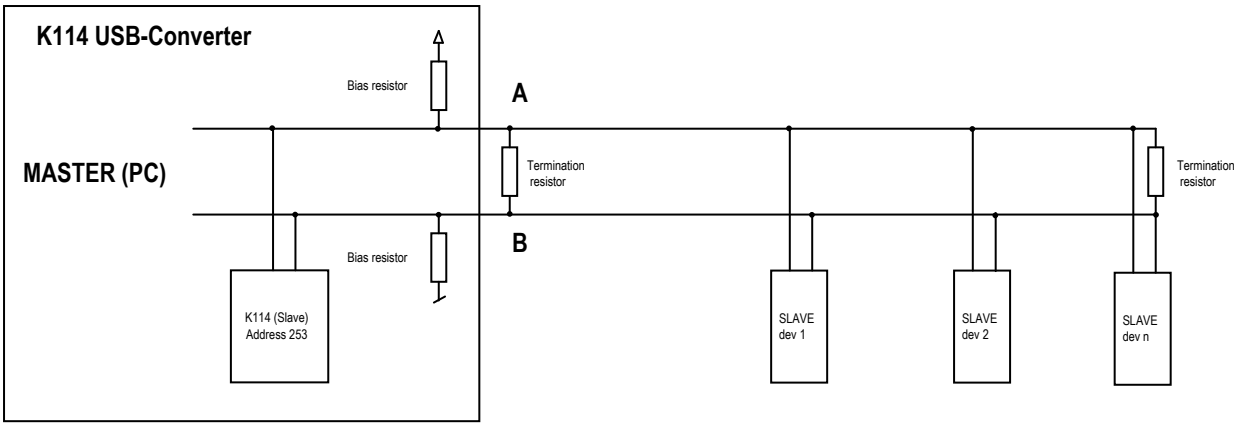


2.3 RS485 half-duplex details

To ensure best possible operation in an industrial environment Keller uses RS485 driver with tailored characteristics. To provide compatibility and get full advantage the bus driver of the master device has to support these specifications.

slew rate limited	In order to avoid oscillations and interference the signal slew rate is limited. This measure allows also usage of standard cables or non-standard topologies (e.g. level detectors or branch lines >> 14m). The more, termination is less critical and has not to be implemented compulsory at the line ends, a feature important for level detectors.
fail safe	Defined signal level – even in short or open circuit case. This is very important for half-duplex operation if all devices are in reception mode – here the line is open in case that no bias resistors are implemented at the master.

Termination resistor and bias resistors



Location of termination resistors: between A and B. At the beginning (Master) and at the end of the transmission line (Slave) Value: the same as the line-impedance. Typ. 120Ohm.

Location of the bias resistors: Resistor connected between A and power supply of the RS485 driver (+5V). Resistor between B and GND of the RS485 driver. Value :Typ. 560Ohm

In case of a fail-safe master driver (interface converter to the PC) and a noise-free environment the termination resistor and bias resistors are not mandatory. To reduce current peaks the resistor values can be chosen higher (termination resistor 1kOhm) or omitted (while transmitting the current increases about 50mA (termination 2x120Ohm).

To guarantee a stable communication in challenging environment (EMC) terminal resistor is recommended!
The communication will work also without a termination, if the environment is free of interference and the cable is held short.

If the connected device (transmitter) has additionally an analogue 4...20mA (two wire) output which will be used simultaneously with the serial communication, it could be useful to communicate without terminal resistor. Otherwise the analogue current signal will have heavy interferences. See application note: ***App. Note S30X-011 RS485 and current loop.pdf***

Bias-resistors: The function of the bias resistors are to keep up having always defined voltages on the communication lines.

Common Mode: The common-mode of the data circuit line is +12 / -7V down to GND. It is essential to keep up with this. Always connect the GND of the master (RS485 converter) with GND of the device (slave)!

Definition of data circuit line assignments

signal	Designation of Keller and diver manufacturers…	Designation of the EIA Standard
inverted (-)	B	A
non-inverted (+)	A	B

Further information on RS485 driver: http://www.maxim-ic.com/MaximProducts/Interface/rs-485.htm

Battery powered devices

It is not recommended to use the internal bias network when the external device is battery powered. The increased current consumption while the slave device is sending can cause a high voltage drop at battery and reset the slave device.

K114 Slave & termination resistor and bias resistors built in K114

The internal bias network or termination resistor of K-114 can be activated over the Software K-114Config.
To configure the K114 and for additional function like voltage and current measurement, the K114 includes a SLAVE device with a fixed BUS-Address of 253.

Devices from KELLER Druckmesstechnik will never have a terminal resistor or bias resistors built in internally (except the K114).

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 receive 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.

The converter K-114 has two basic types of communication :

a) **COM-Modus:** The master (PC) communicates via the RS-485 interface with external devices (address range 0 … 250).
b) **CONFIG-Modus:** The master (PC) communicates with the controller of the interface converter K-114 (fixed address 253). This way the converter can be configured or additional information (voltage and current measurement) can be retrieved from the K114.

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
- no parity
- 9600 baud (default) or 115’200 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)	KELLER:CRC16_H	KELLER: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.

  Address 253 is the device address of the interface converter K-114 (address can not be changed)
  Addresses 251, 252, 254 and 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)	KELLER:CRC16_H	KELLER: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

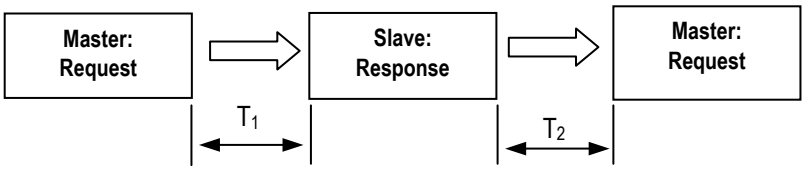
3.3.1 General rules

- 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 interruption**.
  Maximal time between two bytes:
  1.5ms @ 9600 baud (1.5 byte length)
  0.20 ms @ 115200 baud (2.3 byte length)
  If the time between two bytes exceed the specified time, the slave ignores the received data, because of wrong message length or CRC value. In that case the answer is omitted.
- The addressed device must respond within time T_1, otherwise the message will be invalid.

Bit frame:

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. 1ms to max. 100ms for all functions and devices.
  Most functions (except those with EEPROM access like information values):T_1 min. 1.2… 3ms
- T_2: Time to ready-to-receive state for the slave:
  min 1 ms @ 9600 baud (1 byte length)
  min 0.10 ms @ 115200 baud (1 byte length)

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. → e.g. too much of time gap in frame between the bytes.
- A message is longer than the internal transmission buffer permits.
- The word length cannot be interpreted correctly.
- The CRC16 checksum is incorrect.

In these cases the slave denies the request and will therefore **not reply**. This will lead to a timeout at the master. → the request has to be repeated again. 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	KELLER:CRC16_H	KELLER:CRC16_L

4 types of exception errors are defined :

- non-implemented function 1
- incorrect parameters 2
- erroneous data 3
- initialisation missing (only KELLER bus) 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.

3.3.2.3 Slave address of converter K-114

The interface converter K-114 has a fixed bus address. Address 253 is used to communicate with the K114 slave.
The K-114 can be configured over the same address.
All communication (addresses) are transferred via RS-485 interface to the external connected device.

4 Description of Keller bus functions

This section describes the functions of the bus protocol for the K114 device (device *Class.Group* 20.1) using the Keller bus functions (not MODBUS).

Overview:

F30: Read out calibration (scaling) and information in floating-point values
F31: Write calibration floating-point values
F48: Initialise device
F69: Read out serial number
F73: Read out current Voltage or current values in floating-point format
F100: Read configuration (Termination, Bias, …)
F101: Write configuration (Termination, Bias, …)

4.1 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 is 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
20: K114 Device

Group — Subdivision within a device class
1: standard K114

Year, Week — Firmware version

BUF — Length of the internal receive buffer

STAT — Status information
0: Device addressed for first time after switching on.
1: Device was already initialised

4.2 Function 30: Read float value

Request:

DevAddr	30	Index	CRC16_H	CRC16_L

Response:

DevAddr	30	B3	B2	B1	B0	CRC16_H	CRC16_L

Exception errors:

2 Index not supported
3 message length incorrect
32 device is not yet initialised

Note:
Every float value can be read in IEEE754 format (floating-point format 4-byte B0 .. B3) via this function.
→ Information on IEEE754: see appendix. Unused coefficients contain undefined values (NaN).

4.3 Function 31: Write float

Request:

DevAddr	31	Index	B3	B2	B1	B0	CRC16_H	CRC16_L

Response:

DevAddr	31	0	CRC16_H	CRC16_L

Exception errors:

2 Index not supported
3 message length is incorrect
32 device has not yet been initialised

Index	Float Value	Index	Float Value	Index	Float Value	Index	Float Value
64	**U-IN OFFS**	65	**U-IN GAIN**	66	**I-OUT OFFS**	67	**I-OUT GAIN**
80	U-IN_MIN	81	U-IN_MAX	82	I-OUT_MIN	83	I-OUT_MAX
84	U-OUT_MIN	85	U-OUT_MAX	86	U-USB_MIN	87	U-USB_MAX

Nr. 64 .. 65: Offset/Gain -Calibration in Volt, changeable with function F31.
Nr. 66 .. 67: Offset/Gain -Calibration in mA, changeable with function F31.
Nr. 80... 87: Read only (measuring range information)

Scaling of channels U-IN & I-OUT

U-IN and I-OUT are linearly scalable with zero point and gain factor: **Value = GAIN * value + OFFS**
Standard values: Offset = 0.0, gain factor = 1.0
The gain factor should be used **for calibration purposes only**, and not to alter units.

4.4 Function 66 : Read device 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 reads the device addresse. The address is returned in ActAddr.
The BUS address of the K114 can not be changed. He address is fixed to 253.

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 unsigned integer 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 > 4
3 If message length is incorrect
32 If device is not yet initialised

Note:
The interface converter K-114 can measure up to four signals (channels):
A voltage input (U-IN) to measure analog signals, the current supply of external connected consumer (I-OUT), supply voltage of external connected consumer (U-OUT) and the USB supply voltage of K-114 (U-USB).

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

CH	Name	Description	Unit
0	---		
1	U-IN	Voltage input (transmitter signal)	V
2	I-OUT	Current supply Current supply – external consumer	mA
3	U-OUT	Voltage supply – external consumer	V
4	U-USB	USB voltage supply ot K-114	V
5	---		

The **STAT** byte contains the current error status of the different channels.

Bit position	.7	.6	.5	.4	.3	.2	.1	.0
Name	/STD	-	-	U-USB	U-OUT	I-OUT	U-IN	-

4.7 Function 100 : Read configuration

Request:

DevAddr	100	Index	CRC16_H	CRC16_L

Response:

DevAddr	100	Index	B0	B1	B2	B3	CRC16_H	CRC16_L

Exception errors:

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

Note: Function 100 199 serves only as a quick reference. Currently set settings can be read out

Request communication speed setting (INDEX 0)

DevAddr	100	Index	CRC16_H	CRC16_L
253	100	0		

Response:

DevAddr	100	Index	B0	B1	B2	B3	CRC16_H	CRC16_L
253	100	0	170	BaudRate	0	0		

Baudrate = 0 --> 9600 Baud Baudrate <> 0 --> 115200 Baud

Request ALL actual settings (INDEX 199)

DevAddr	100	Index	CRC16_H	CRC16_L
253	100	199		

Response:

DevAddr	100	Index	B0	B1	B2	B3	CRC16_H	CRC16_L
253	100	199	ECHO	TERMINATION	BIAS	HIGHSPEED		

Request single values (INDEX 200-203)

DevAddr	100	Index	CRC16_H	CRC16_L
253	100	200 (ECHO) 201 (TERMINATION) 202 (BIAS) 203 (HIGHSPEED)		

Response:

DevAddr	100	Index	TEMPORARY	EEPROM	0	CODE	CRC16_H	CRC16_L
253	100	200 – 203	(0/1) off/on	(0/1) off/on	0	204		

The TEMPORARY Byte shows the current setting.
The EEPROM Byte shows the saved settings.
Where Byte = 0 function is OFF and Byte <> 0 function is ON

4.8 Function 101: Write configuration

Request :

DevAddr	101	Index	B0	B1	B2	B3	CRC16_H	CRC16_L

Response:

DevAddr	101	Index	CRC16_H	CRC16_L

Exception errors:

2 If index not allowed / wrong CODE
3 If message length is incorrect
32 If device is not yet initialised

The settings of the functions (ECHO,TERMINATION,BIAS,HIGHSPEED) can be set permanently or temporary.
The temporary settings are lost after powering off the converter K114 (plugging out from USB Port with disconnected external supply).
The Byte **EEPROM** defines if the function are set temporary (EEPROM <> 54) or permanently (EEPROM = 51).

SET communication speed (INDEX 0)

DevAddr	101	Index	void	Set BaudRate	void	void	CODE	CRC16_H	CRC16_L
253	101	0	0	(0/1) 9600/115200	0	0	204		

This setting becomes active right after the K114 has responded to that command.

SET Factory settings (INDEX 199)

DevAddr	101	Index	temporary/permanently	void	void	void	CODE	CRC16_H	CRC16_L
253	101	199	(0/1)	0	0	0	204		

Factory settings:

ECHO = on TERMINATION = off BIAS = off HIGHSPEED = off

SET ECHO (INDEX 200)

DevAddr	101	Index	Set ECHO	EEPROM	void	void	CODE	CRC16_H	CRC16_L
253	101	200	(0/1) off/on	(0/51) temporary/permanently	0	0	204		

SET TERMINATION(INDEX 201)

DevAddr	101	Index	TERMINATION	EEPROM	void	void	CODE	CRC16_H	CRC16_L
253	101	201	(0/1) off/on	(0/51) temporary/permanently	0	0	204		

SET BIAS(INDEX 202)

DevAddr	101	Index	BIAS	EEPROM	void	void	CODE	CRC16_H	CRC16_L
253	101	202	(0/1) off/on	(0/51) temporary/permanently	0	0	204		

SET HIGHSPEED (INDEX 203)

DevAddr	101	Index	HIGHSPEED	EEPROM	void	void	CODE	CRC16_H	CRC16_L
253	101	203	(0/1) off/on	(0/51) temporary/permanently	0	0	204		

5 Appendix

5.1 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:

B3 DATA H (Reg. 0)	**B2** DATA L (Reg. 0)	**B1** DATA H (Reg. 1)	**B0** DATA L (Reg. 1)	
b01000001 (0x41)	b00101001 (0x29)	b00000010 (0x02)	b11011110 (0xDE)	Valid Number
b01111111 (0x7F)	b10000000 (0x80)	b00000000 (0x00)	b00000000 (0x00)	∞ / Overflow
b11111111 (0xFF)	b10000000 (0x80)	b00000000 (0x00)	b00000000 (0x00)	-∞ / Underflow
bx1111111 (0xFF)	b11111111 (0xFF)	b11111111 (0xFF)	b11111111 (0xFF)	Not a Number

1 bit Sign + 8 bit Exponent + 23 bit Mantis = 32 bit

Calculation of the value transmitted:

$$V = (-1)^S \cdot (1.0 + \frac{M}{2^{23}}) \cdot 2^{E-127}$$

0 = 0
10000010 = 130
01010010000001011011110 = 2687710

$-1^0 * (1.0 + 2687710/8388608) * 2^{130-127} = 10.5631999969482421875$

These values directly show the value in the requested unit [bar] or [°C].

⇨ 10.5632 bar

Usage of Keller software:
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.

For more informations, please visit:
http://cch.loria.fr/documentation/IEEE754/numerical_comp_guide/ncg_math.doc.html - 556

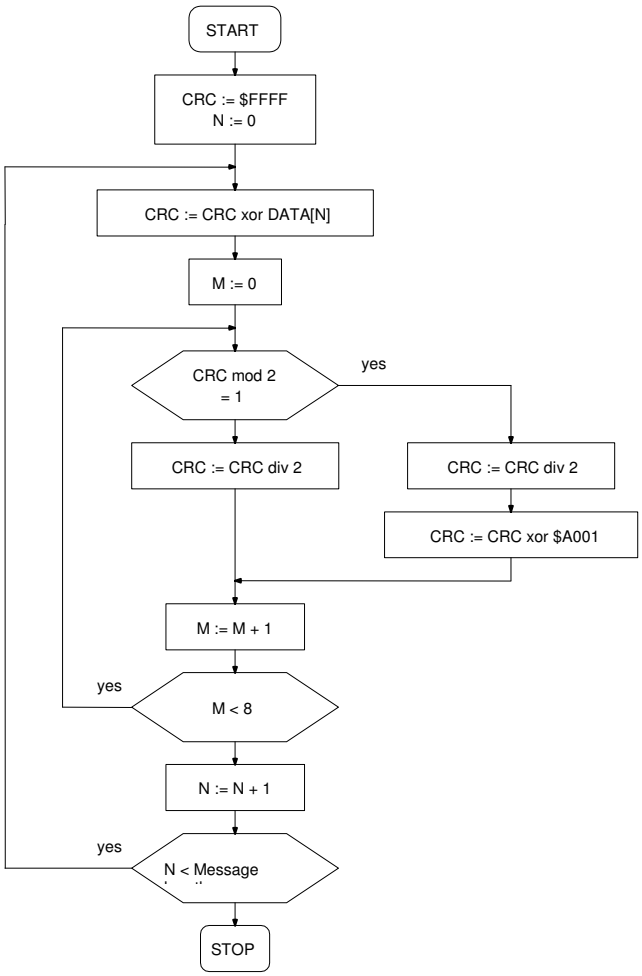
5.2 Calculation of the CRC16 checksum

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

5.3 Description of the software driver (DLL)

5.3.1 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

5.3.2 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)

5.3.2.1 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 CCS30 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 (devices with firmware 5.20)

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

5.3.2.2 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;
```

5.3.2.3 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 F32 and F33 are new functions which are aviable in the s30c.dll from the 12.9.2005 and later.

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

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

function F32(  bteDeviceAddr, bteCoeffNo: Byte; var sinCoeff: Byte
): Smallint; stdcall; export;

function F33(  bteDeviceAddr, bteCoeffNo: Byte; sinCoeff: Byte
): Smallint; stdcall; export;

function F34(  bteDeviceAddr: Byte; wrdAddr: Word; bteAmount: Byte; pbteData: PByte
): Smallint; stdcall; export;

function F35(  bteDeviceAddr: Byte; wrdAddr: Word; bteAmount: Byte; pbteData: PByte
): 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;

function F101(
  bteDeviceAddr, bteIndex: Byte; btePara0, btePara1, btePara2, btePara3, btePara4: Byte
): Smallint; stdcall stdcall; export;
```

5.4 Support

We are pleased to offer you support in implementing the protocol. Use our free PC-software CCS30 for communication and configuration. Also divers for LabView, C#, etc are aviable on our website: 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