HND-Z031, HND-Z032, HND-Z033
Interface adapter
Interface description for
HND handheld instruments

from Version 1.0

Inhalt

manufactured and sold by:

Kobold Messring GmbH
Nordring 22-24
D-65719 Hofheim
Tel.: +49 (0)6192-2990
Fax: +49(0)6192-23398
E-Mail: info.de@kobold.com
Internet: www.kobold.com

version K01/0118

1 General Note

Read this document carefully and get used to the operation of the device before you use it. Keep this document within easy reach near the device for consulting in case of doubt.

Mounting, start-up, operating, maintenance and removing from operation must be done by qualified, specially trained staff that have carefully read and understood this manual before starting any work.

The manufacturer will assume no liability or warranty in case of usage for other purpose than the intended one, ignoring this manual, operating by unqualified staff as well as unauthorized modifications to the device. The manufacturer is not liable for any costs or damages incurred at the user or third parties because of the usage or application of this device, in particular in case of improper use of the device, misuse or malfunction of the connection or of the device.

The manufacturer is not liable for misprints.

2 Safety

2.1 Intended Use

This documentation describes the communication protocol of HND handheld devices.
You must not use functions that are not documented here.

2.2 Safety signs and symbols

Warnings are labeled in this document with the followings signs:

DANGER

Caution! This symbol warns of imminent danger, death, serious injuries and significant damage to property at non-observance.

Attention! This symbol warns of possible dangers or dangerous situations which can provoke damage to the device or environment at non-observance.

Note! This symbol point out processes which can indirectly influence operation or provoke unforeseen reactions at non-observance.

2.3 Hints

The device needs an interface for communication.
The interface has to be activated.
The analog output (if available) needs to be deactivated. Refer to the device's manual for more information.
Prerequisites:

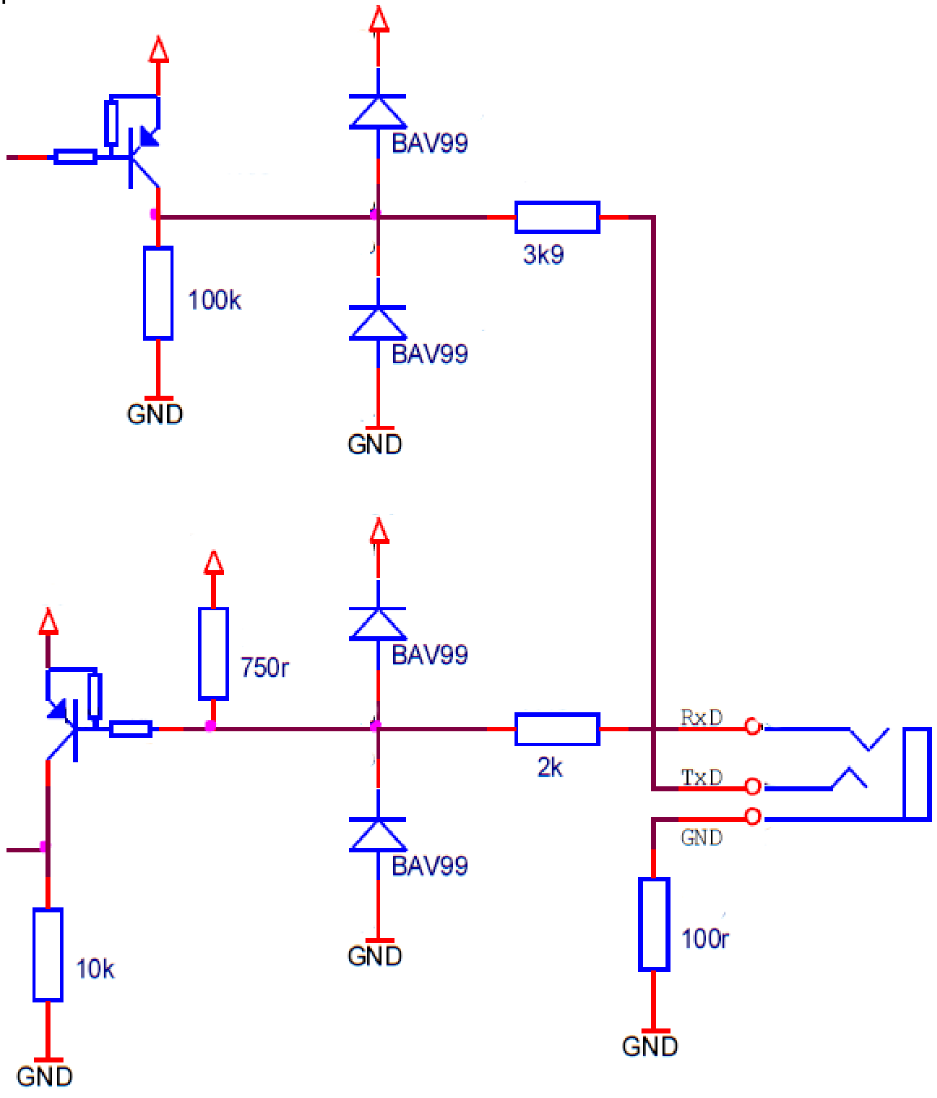
- error free and working installation
- genuine HND interface converter (HND-Z031, HND-Z032, HND-Z033).
- correctly initialized and opened serial interface
- most recent USB drivers

3 HND-Z031, HND-Z032, HND-Z033

The HND-Z031, HND-Z032 and HND-Z033 are used for connection a HND to a PC. This interface converter uses an electrical insulation circuitry and for the USB types a standard USB to UART chipset for device communication. Please refer to the manual of the interface converter.

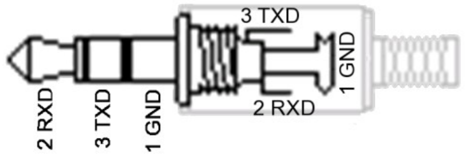
3.1 Hardware (device transceiver circuit)

Example circuit. Direct connection to RS 232 lines of a PC or to RS232 level converter (e.g. MAX 232) is not permissible!



3.2 Interface connection

The connection to the serial interface of the HND is established by a 3.5 mm stereo jack.

HND device connector	Terminal	Device (ref. 3.1)	Description (from device side) Level, hints
	1 GND	GND	Ground Connect GND
	2 RxD	TxD	Output, transmit line (Terminal: 2 RxD) Current source with series resistor Low level: GND High level: >1 VDC, max. 3,5 mA
	3 TxD	RxD	Input, receive line (Terminal: 3 TxD) Pullup to device supply (3.3 or 5.0 VDC) Low level: 3.3 / 5 VDC High level: GND

4 Interface settings

(Needs to be set in Program or Terminal not in the Windows Device Manager)
According to RS232-Standards (idle state = logical '1')

Baudrate	Databits	Parity	Stopbits	Flow control
4800	8	none	1	none

☞ **Extended Interface settings**
(Supply for electric insulation circuit of HND-Z031)

DTR	enabled
RTS	disabled

4.1 Code for initializing the serial interface (C++ with Windows API)

```
HANDLE hPort;                     /* Declare interface handle */;
DCB dcb;                          /* Declare DCB block */
BOOL ok;
FillMemory(&dcb, sizeof(dcb), 0);     /* Delete old DCB settings */
dcb.DCBlength = sizeof(dcb);          /* Set DCB-length */
dcb.BaudRate = CBR_4800; /* Set baudrate to '4800' or '38400' */
dcb.Parity = PARITY_NONE;         /* Set parity to 'none' */
dcb.ByteSize = DATABITS_8;        /* Set databits to '8' */
dcb.StopBits = STOPBITS_10;           /* Set stopbits to '1' */
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);/* Configure input- and outputbuffer */
ok = SetupCommState(hPort, &dcb);/* Set parameters */
```

4.2 Code for initializing the serial interface (C# with .NET)

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

4.3 Code for initializing the serial interface (POSIX C e.g. Linux)

```
struct termios* Port;
Port = (termios*)malloc(sizeof(termios));
memset(Port, 0, sizeof(struct termios));
cfsetispeed(Port, B4800); /* or '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;
```

5 Protocol specification

According to Bus specification.

5.1 Basic message organization of a 6 byte query/response

	Bye0	Byte1				Byte2	Byte3	Byte4	Byte5
	Address	Header				CRC	DB0	DB1	CRC
Bit	7 ... 0	7 ... 4	3	2 ... 1	0	7 ... 0	7 ... 0	7 ... 0	7 ... 0
		F1-Code	Priority	Length	Direction				
		Query code	1 = Priority 0 = no Priority	00 = 3 Byte 01 = 6 Byte 10 = 9 Byte 11 = variable	1 = from Slave 0 = from Master				

5.2 Header organization

Bit 7...4: F1-Code = Query code (e.g. 0 for read display value)
Bit 3: Priority-Bit. Will be '1' for priority response. Priority bit is set e.g. on exceeding the alarm limits. This bit has to be '0' for queries (Master → Slave) and is only used for answers (Slave → Master).
Bit 2...1: Message length in Bytes.
Bit 0: Message direction '0': Master → Slave, '1': Slave → Master

5.3 Communication

Data transmission is in polling operation. The device will only respond after a valid query.

Transmission of data bytes
Byte0 is send first.
There must not be any control characters (line feed, ...) at the end of the transmission.
Transmission is done in raw-mode (also non printable characters).

Inverting of data bytes
Each first byte from the byte-triple (Byte0, Byte3, ...) needs to be inverted.
($\text{Byte0}_{\text{Transmit}}$ = 255 - Byte0)

Control bytes
Each third bytes from the byte-triple (Byte2, Byte5, ...) is a control byte (CRC).
The calculation can be found in the appendix.

Decoding of responses to measurement queries
A response with 6 bytes length: has to be decoded with DecodeMeasurement16(...)
A response with 9 bytes length: has to be decoded with DecodeMeasurement32(...)

Characteristic of the HND-series
The response starts with the received query (echo). All examples and tables refer to the data after the echo-data.

F1-Code (Query code)	Description
0x0	Read display value. 3 byte query, 6 or 9 byte response Use function DecodeMeasurement16(...) or DecodeMeasurement32(...).
0x3	Read system state. 3 byte query, 6 byte response Use function UInt16Decode(Byte3, Byte4)
0x5	Query not supported (possible only for device response)
0x6	Read min measured value. 3 byte query, 6 or 9 byte response Use function DecodeMeasurement16(...) or DecodeMeasurement32(...).
0x7	Read max measured value. 3 byte query, 6 or 9 byte response Use function DecodeMeasurement16(...) or DecodeMeasurement32(...).
0xC	Read serial number. 3 byte query, 9 byte response Use function UInt16Decode(Byte3, Byte4), UInt16Decode(Byte6, Byte7) and UInt32Decode(...). Serial number in HEX
0xF	Extended query code byte length = 6 bytes

F1-Code (Query code)	Byte3 (DB0)	Byte4 (DB1)	Description
0xF	0xCA	0x00	Read display unit. 6 byte query, 9 byte response Use function UInt16Decode(Byte6, Byte7)
0xF	0xD0	0x00	Read number of channels. 6 byte query, 9 byte response Byte6 = 0x00: Channel addressing via addresses Byte6 = 0x01: Channel addressing via serial number Byte7 = positive: Channel count Byte7 = negative: Channel number

5.4 Example queries

Read display value of device with address 1:

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

Byte 0 = 0xFF - Address = 0xFE
Byte 1 = F1-Code read display value = 0x0, Priority = 0, Length = 0 (3 Byte), Master->Slave = 0 => 0x00
Byte2 = CRC calculation of Byte0 and Byte1 = 0x3D

Read status code of device with address 2:

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

Byte 0 = 0xFF - Address = 0xFD
Byte 1 = F1-Code read status code 0x3, Priority = 0, Length = 0 (3 Byte) , Master->Slave = 0 => 0x00
Byte2 = CRC calculation of Byte0 and Byte1 = 0x92

Read display unit of device with address 3:

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 - Address = 0xFC
Byte 1 = F1-Code read status code = 0xF, Priority = 0, Length = 1 (6 Byte) , Master->Slave = 0 => 0xF2
Byte2 = CRC calculation of Byte0 and Byte1 = 0xC7
Byte3 = 0xFF - 0xCA, Byte4 = 0x00, Byte5 = CRC calculation of Byte3 and Byte4 = 0x47

5.5 Example response

The device will answer in less than 1 second with 3, 6 or 9 bytes. The first 3 bytes are the header, the length of the response can be extracted from the header. Example for a device with address 1 responding to the query read display value (the device displays -0,04)

	Received data bytes								
	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Example for -0.04	0xFE	0x0D	0x10	0x72	0xFF	0x84	0x00	0xFC	0x05
	Header		CRC	Payload0		CRC	Payload1		CRC

6 Appendix

6.1 Codes for decoding received data

```
/* Combinde 2 bytes to an unsigned 16 bit integer */
UInt16 UInt16Decode(byte inbyByteA, byte inbyByteB)
{
      return (UInt16)((UInt16)((255 - inbyByteA) << 8) | inbyByteB);
}

/* Combind 2 unsigned 16 Bit integer to an unsigned 32 bit integer */
UInt32 UInt32Decode(UInt16 inui16Payload0, UInt16 inui16Payload1)
{
      return (UInt32)( inui16Payload0<< 16 | inui16Payload1);
}

Byte[] Exampledata = {0xFE, 0x0F, 0x10, 0x72, 0xFF, 0x84, 0x00, 0xFC, 0x05}; /* => -0.04 */
/* Calculate measuring value from 4 bytes of received data */
Int16 DecodeMeasurement32(byte inbyByte3, byte inbyByte4, byte inbyByte6,
byte inbyByte7, out double outdblFloatValue, out Int16
outi16DecimalPointPosition)
{
      outdblFloatValue = 0;
      outi16DecimalPointPosition = 0;
      UInt16 ui16Integer1, ui16Integer2;
      ui16Integer1 = UInt16Decode (inbyByte3, inbyByte4);
      ui16Integer2 = UInt16Decode (inbyByte6, inbyByte7);
      UInt32 ui32Integer = UInt32Decode (ui16Integer1, ui16Integer2);
      /* Combine bytes, with Exampledata: 0x8DFFFFFC */
      outi16DecimalPointPosition = (Int16)(((0xFF - inbyByte3) >> 3) - 15);
      /* Get decimal point, with Exampledata: 0x0002*/
      ui32Integer = ui32Integer & 0x07FFFFFF;
      /* Decode raw value, with Exampledata: 0x05FFFFFC */
      if ((100000000 + 0x2000000) > ui32Integer)
      {
            /* Data is a valid value */
            if (0x04000000 == (ui32Integer & 0x04000000))
            {
                  ui32Integer = (ui32Integer | 0xF8000000);
                 /* With Exampledata: 0xFDFFFFFC */
            }
            ui32Integer = (UInt32)((UInt64)ui32Integer + 0x02000000);
           /* with Exampledata: 0xFFFFFFFC */
      }
      else
      {
            /* Data contains error code, decode error code */
            outdblFloatValue = (double)(ui32Integer - 0x02000000 - 16352.0);
            outi16DecimalPointPosition = 0;
            return -36; /* Return value is error code */
      }
      /* Convert to floating point value, with Exampledata: -4f */
      outdblFloatValue = (double)(Int32)ui32Integer;
      outdblFloatValue = outdblFloatValue /
                        (Math.Pow(10.0f, (double)outi16DecimalPointPosition));
      return 0; /* Return value is OK, with Exampledata: -0,04 */
}
```

```
/* Calculate measuring value from 2 bytes of received data */
Int16 DecodeMeasurement16(byte inbyByte3, byte inbyByte4, out double
outdblFloatValue, out Int16 outi16DecimalPointPosition)
{
      Int16 ui16Integer = UInt16Decodieren(inbyByte3, inbyByte4);
      outi16DecimalPointPosition = (Int16)((ui16Integer& 0xC000) >> 14);
      ui16Integer = (UInt16)(ui16Integer& 0x3FFF);
      if ((ui16Integer >= 0x3FE0) && (ui16Integer <= 0x3FFF))
      {
            outdblFloatValue = (double)ui16Integer - (double)16352.0;
            return -36; /* Return value is error code*/
      }
      Int32 i32Nenner = (Int32)System.Math.Pow((double)10.0,
(double)i16DezimalPunktPosition);
      Int32 i32Zaehler = (Int32)((double)ui16Integer - (double)2048.0);
      outdblFloatValue = (double)((double)i32Zaehler / (double)i32Nenner);
      return 0; /* Return value is valid, OK */
}
```

6.2 Error codes

Error	Description
16352	measuring range overrun
16353	measuring range underrun
16362	no value
16363	system error
16364	battery empty
16365	no sensor
16366	Recording error: EEPROM error
16367	EEPROM checksum error
16368	Recording error, Error 6: System restarted
16369	Recording error: data pointer
16370	Recording error: marker, data invalid
16371	data invalid
others	unknown EASYBus error code

6.3 Status codes

Bit number	Description
0	Max. Alarm
1	Min. Alarm
2	Display range overrun
3	Display range underrun
4	- (reserved for future changes)
5	- (reserved for future changes)
6	- (reserved for future changes)
7	- (reserved for future changes)
8	Measuring range overrun
9	Measuring range underrun
10	Sensor error
11	- (reserved for future changes)
12	System fault
13	Calculation not possible
14	- (reserved for future changes)
15	Low battery

6.4 Unit codes

Code	Unit	Code	Unit	Code	Unit	Code	Unit
1	°C	61	km/h	122	kOhm		
2	°F	62	mph	123	MOhm		
3	K	63	Knots				
				125	kOhm*cm		
				126	MOhm*cm		
10	% RH	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	Pulses	116	kWh	172	mg/l		
		117	mW/cm^2	173	g/l		
				175	dB		
		119	Wh/m2	176	dBm		
		120	mOhm	177	dBA		
60	m/s	121	Ohm				

6.5 Code for CRC-Calculation

```
Byte[] Exampledata = {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 CRCCalculation(byte[] inByteArray, UInt16 inLength)
{
        if (inLength >= 3)
        {
                if (inByteArray[2] != CRC(inByteArray[0], inByteArray[1]))
                {
                        return false;
                }
        }
        if (inLength >= 6)
        {
                if (inByteArray[5] != CRC(inByteArray[3], inByteArray[4]))
                {
                        return false;
                }
        }
        if (inLength >= 9)
        {
                if (inByteArray[8] != CRC(inByteArray[6], inByteArray[7]))
                {
                        return false;
                }
        }
        if (inLength >= 12)
        {
                if (inByteArray[11] != CRC(inByteArray[9], inByteArray[10]))
                {
                        return false;
                }
        }
        return true;
}
```