Table of Contents

1 Typographical Conventions 2
1.1 Employed data types 2
2 General Specification 3
2.1 Address calculation 3
2.1.1 For zones 3
2.1.2 For system parameters 3
2.2 Content of TCP/IP telegram 4
2.2.1 Telegram structure for commands 0x0000, 0x0001, 0x0003, 0x0004, 0x000D, 0x000E 4
2.2.2 Telegram structure for command 0x000A 5
2.3 Calculation of check sum 6
2.4 Command 0x0000 – Establish connection 7
2.5 Command 0x0001 – Read software version 8
2.6 Command 0x0003 – Read bytes 9
2.7 Command 0x0004 – Write bytes 10
2.8 Command 0x000D – Read bytes of several zones 11
2.9 Command 0x000E – Write bytes of several zones 12
2.10 Command 0x000A – Send CAN Command 13
3 Appendix 14
3.1 Version History 14

1 Typographical Conventions

Symbols and conventions are used in this manual for faster orientation for you.

	Caution	With this symbol, references and information are displayed which are decisive for the operation of the device. In case of non-compliance with or inaccurate compliance there can result damage to the device or injuries to persons.
	Note	The symbol refers to additional information and declarations, which serve for improved understanding.
	Example	With the symbol, a function is explained by means of an example.
	Reference	With this symbol, information in another document is referred to.
	FAQ	Here FAQ (Frequently Asked Questions) are answered.
	`Equations`	Calculation specifications and examples are represented in this way.

1.1 Employed data types

Size (Bit)	Identification
8	Byte
16	Word
32	Long

2 General Specification

The „Ethernet Binary" Protocol is based on a TCP/IP connection.
First a socket connection to the IP address of the controller on port 5000 is established. Thereafter the connection must be activated by command „CMD_CONNECT“ (see chapter 2.4). The described commands can then be executed.

2.1 Address calculation

The specified addresses in the commands are basically absolute addresses.

The offset in the zones can be taken from the documentation of the controller specific parameter and object lists for protocol PSGII...

2.1.1 For zones

Should data be read from the zones, the absolute addresses are calculated in the following way:

For flexoTEMP PCU, PCU PNIO und MCU
Absolute address = 0xC0000 + zone[0...127] * 0x800 + Offset

For flexoTEMP PCU NEXT
Absolute address = 0xC0000 + zone[0...250] * 0x400 + Offset

2.1.2 For system parameters

Should data be read from system parameters, the absolute addresses are calculated in the following way:

Absolute address = 0xA0000 + offset;

2.2 Content of TCP/IP telegram

2.2.1 Telegram structure for commands 0x0000, 0x0001, 0x0003, 0x0004, 0x000D, 0x000E

```
typedef struct
{
    word        HEAD        // Telegram identification
                            // At request to controller:                    0xA5EF
    byte        RESERVE     // Always set to 0
    word        COMMAND     // Command
    byte        RESERVE     // Always set to 0
    long        ADDRESS     // Access address
    byte        RESERVE     // Always set to 0
    word        LEN         // Length of command
                            // Number of data bytes in BUFF + 15 + 1 byte check sum
    word        NUM         // Number of data to read / write
                            // (per zone for zone commands)
    byte        BUFF[1020]  // Data bytes + 1 byte check sum after last data
} tx;
```

In the TCP/IP telegram the following data structure is sent by controller (response):

```
typedef struct
{
    word        HEAD        // Telegram identification
                            // Response by controller:                      0x4143
    byte        RESERVE     // Always set to 0
    word        Status      // Status of response
    byte        RESERVE     // Always set to 0
    long        ADDRESS     // Access address
    byte        RESERVE     // Always set to 0
    word        LEN         // Length of command
                            // Number of data bytes in BUFF + 15 + 1 byte check sum
    word        NUM         // Number of all user data
    byte        BUFF[1020]  // Data bytes + 1 byte check sum after last data
} rx;
```

2.2.2 Telegram structure for command 0x000A

```
typedef struct
{
    word        HEAD        // Telegram identification
                            // At request to controller:                    0xA5FE
    byte        RESERVE     // Always set to 0
    word        COMMAND     // Command
    byte        LEN         // Length of command
                            // Number of data bytes in BUFF + 11 + 1 byte check sum
    word        TXNode      // Sending-ID
    word        RXNode      // Receiving-ID
    byte        NUM         // Number of data bytes
    byte        BUFF[1024]  // CAN-Telegram + BCC
} tx_can;
```

In the TCP/IP telegram the following data structure is sent by controller (response):

```
typedef struct
{
    word        HEAD        // Telegram identification
                            // Response by controller:                      0x4142
    byte        RESERVE     // Always set to 0
    word        Status      // Status of response
    byte        LEN         // Length of command
                            // Number of data bytes in BUFF + 11 + 1 byte check sum
    word        TXNode      // Sending-Node-ID
    word        RXNode      // Receiving-Node-ID
    byte        NUM         // Number of data bytes
    byte        BUFF[1024]  // CAN-Telegram + BCC
} rx_can;
```

2.3 Calculation of check sum

The check sum can be calculated in the following way:

```
// Calculate check sum

byte ps=0;
byte* Ptr;

// Set pointer on Send/Receive structure

Ptr = &tx;

for (cnt=0; cnt< tx.LEN-1; cnt++)
{
        ps += Ptr[cnt];
}

// Enter check sum in sending string

Ptr[tx.LEN-1] = 0-ps;
```

2.4 Command 0x0000 – Establish connection

Request to controller
(Telegram structure see 2.2.1)

tx.HEAD	=	0xA5EF;	Telegram identification
tx.COMMAND	=	**0x0000**;	Command
tx.ADDRESS	=	0x5555AAAA;	Access address
tx.LEN	=	0x0010;	Length of command
tx.NUM	=	0x0000;	Number of data to read / write
tx.BUFF[0]	=	0x5B;	Data bytes + 1 byte check sum after last data

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.ADDRESS	=	0;
rx.LEN	=	19;
rx.NUM	=	3;
rx.BUFF[0]	=	„OK“;
rx.BUFF[3]	=	Check sum;

2.5 Command 0x0001 – Read software version

Request to controller
(Telegram structure see 2.2.1)

tx.HEAD	=	0xA5EF;	Telegram identification
tx.COMMAND	=	**0x0001**;	Command
tx.ADDRESS	=	0x00000000;	Access address
tx.LEN	=	0x0010;	Length of command
tx.NUM	=	0x0000;	Number of data to read / write
tx.BUFF[0]	=	0x5A;	Data bytes + 1 byte check sum after last data

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.LEN	=	29;
rx.NUM	=	13;
rx.BUFF[0]	=	Version number;
rx.BUFF[13]	=	Check sum;

2.6 Command 0x0003 – Read bytes

Request to controller
(Telegram structure see 2.2.1)

tx.HEAD	=	0xA5EF;
tx.COMMAND	=	**0x0003**
tx.ADDRESS	=	Address read from;
tx.LEN	=	16;
tx.NUM	=	Number of bytes to read;
tx.BUFF[0]	=	Check sum;

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.ADDRESS	=	0;
rx.LEN	=	16 + Number of bytes read;
rx.NUM	=	Number of bytes read;
rx.BUFF[0]	=	Data;
rx.BUFF[Number of bytes read]	=	Check sum;

2.7 Command 0x0004 – Write bytes

Request to controller
(Telegram structure see 2.2.1)

tx.HEAD	=	0xA5EF;
tx.COMMAND	=	**0x0004**;
tx.ADDRESS	=	Address, written to;
tx.LEN	=	16 + Number of data to write;
tx.NUM	=	Number of bytes to write;
tx.BUFF[0]	=	Data;
tx.BUFF[Number of bytes to write]	=	Check sum;

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.ADDRESS	=	0;
rx.LEN	=	19;
rx.NUM	=	3;
rx.BUFF[0]	=	„OK“;
rx.BUFF[3]	=	Check sum;

2.8 Command 0x000D – Read bytes of several zones

Request to controller
(Telegram structure see 2.2.1)

Example Read zone parameters setpoint value and actual value (for 80 zones from first zone)

tx.HEAD	=	0xA5EF;	Telegram identification
tx.COMMAND	=	**0x000D**;	Command
tx.ADDRESS	=	0x000C0000;	Address read from (address results from the first zone to read);
tx.LEN	=	0x0011;	Length of command;
tx.NUM	=	0x0004	Number of bytes to read per zone;
tx.BUFF[0]	=	0x50;	Number of zones;
tx.BUFF[1]	=	0xEC;	Check sum;

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.ADDRESS	=	0;
rx.LEN	=	17+ Number of all bytes read;
rx.NUM	=	Number of all bytes read;
rx.BUFF[0]	=	Number of zones;
rx.BUFF[1]	=	Data (data zone 1, data zone 2, …)
rx.BUFF[Number of all bytes read + 1]	=	Check sum;

2.9 Command 0x000E – Write bytes of several zones

Request to controller
(Telegram structure see 2.2.1)

tx.HEAD	=	0xA5EF;
tx.COMMAND	=	**0x000E**;
tx.ADDRESS	=	Address written on (address results from the first zone to write);
tx.LEN	=	17 + Number of all bytes to write;
tx.NUM	=	Number of all bytes to write;
tx.BUFF[0]	=	Number of zones;
tx.BUFF[1]	=	Data (data zone 1, data zone 2, …)
rx.BUFF[Number of all bytes read + 1]		Check sum;

Response by controller
(Telegram structure see 2.2.1)

rx.HEAD	=	0x4143;
rx.STATUS	=	**0x0000**;
rx.ADDRESS	=	0;
rx.LEN	=	19;
rx.NUM	=	3;
rx.BUFF[0]	=	„OK“;
rx.BUFF[3]	=	Check sum;

2.10 Command 0x000A – Send CAN Command

Request to controller
(Telegram structure see 2.2.2)

tx_can.HEAD	=	0xA5FE;
tx_can.COMMAND	=	**0x000A;**
tx_can.TXNode	=	Sending-ID;
tx_can.RXNode	=	Receiving-ID;
tx_can.LEN	=	12 + Number of data bytes;
tx_can.NUM	=	Number of data bytes;
tx_can.BUFF[0]	=	Data bytes (Data byte [0], Data byte [1], ..) ;
tx_can.BUFF[Number of data bytes]	=	Check sum;

Response by controller
(Telegram structure see 2.2.2)

rx_can.HEAD	=	0x4142;
rx_can.STATUS	=	**0x0000;**
rx_can.TXNode	=	Sending-ID;
rx_can.RXNode	=	Receiving-ID;
rx_can.LEN	=	20;
rx_can.NUM	=	8;
rx_can.BUFF[0]	=	Data bytes [0] ;
rx_can.BUFF[1]	=	Data bytes [1] ;
rx_can.BUFF[2]	=	Data bytes [2] ;
rx_can.BUFF[3]	=	Data bytes [3] ;
rx_can.BUFF[4]	=	Data bytes [4] ;
rx_can.BUFF[5]	=	Data bytes [5] ;
rx_can.BUFF[6]	=	Data bytes [6] ;
rx_can.BUFF[7]	=	Data bytes [7] ;
rx_can.BUFF[8]	=	Check sum;

3 Appendix

3.1 Version History

Version	Date	Changes
1.00.05	2/15/024	Address calculation for PCU NEXT added
1.00.04	10/17/2013	Designation unsigned int substituted by word. Checksum calculation corrected. Designation of variables standardized.
1.00.03	8/12/2013	Employed data types added; Command 0x0000, 0x0001 check sum added; Command 0x000D example;
1.00.02	7/5/2013	Address calculation system parameters, write command, CAN command added.
1.00.01	1/20/2011	Correction of typing errors
1.00.00	9/23/2010	First edition Meusburger Deutschland GmbH Voltastraße 2 68519 Viernheim Germany Tel. +49 6204 6069 2 0 www.meusburger.com office-de@meusburger.com