

IMACS

controller communication 2.0

data log

Stand: 15

IMACS GmbH

Meß- und Steuerungstechnik

Mainzer Str. 139
D-55545 Bad Kreuznach

1. GENERAL	3
1.1 Example checksum calculation in C	3
2. COMMANDO FORMAT	4
3. COMMANDS	5
3.1 Request for version data	5
3.2 Request status block	6
3.3 Request status data	8
3.4 Request parameter block	9
3.5 Set parameter data	11
3.6 Set system parameter data	11
3.7 Set process data	12
3.8 Set flag data	12
4. USE RADCASE INFORMATION	13
4.1 Request for specific data value	14
5. IMPLEMENTATION EXAMPLE IN C	15
5.1 Send commando	15
5.2 Receive answer	16
5.3 Demand and receive status block	18

1. General

The control is defined as slave i. e. it receives commands from the host (e. g. PC/SPS) and returns an answer if necessary. The data exchange is based on blocks, the transmission errors are recognized by checksum. The checksum will be returned as confirmation in case of no transmission error. The data hardware indicates a correct transmission of the command/data block and answers with returning the checksum (always 0AAH). The size of a block is limited to maximal 240 bytes of reference data. Therefore larger reference data are split to an equal number of blocks.

1.1 Example checksum calculation in C

```
int i;
unsigned char sum;
unsigned char buf[256];

buf[0] = data_len;      // number of reference data byte
buf[1] = dst;           // destination
buf[2] = src;           // source
buf[3] = cmd1;          // commando 1
buf[4] = cmd2;          // commando 2
memcpy(&buf[5], data, data_len); // copy reference data into transciever
cache

// calculate checksum
for (sum = 0, i = 0; i < (data_len + 5); i++)
    sum = (unsigned char)(sum + buf[i]);

// write checksum into transciever cache
buf[i] = (unsigned char)(0xAA - sum);
```

2. Commando format

The basic format of a command block sent from the data hardware to the controller is described in the following.

name	offset dec	value hex	description
LEN	0	nn	Command-/reference data length byte (Length of block data) [0..240]
DST	1	dd	Destination adress (Bus-adress of target) [1...x]
SRC	2	ss	Source adress (Bus-adress of target, at PC = 0)
CMD1	3	##	Command byte 1
CMD2	4	##	Command byte 2
DATA1	5	##	Reference data byte 1
...
DATAnn	5+(nn-1)	##	Reference data byte nn
CHKSUM	5+nn	##	Checksum byte

- The block length byte contains the length of the block (without commad length byte, without adress bytes, without command bytes, without checksum byte).
- The checksum byte supplies the sum byte of the whole block (inclusive of all bytes) on 0AAh.
- The source- and destination adress has no function in case of communication with only one terminal (content is not used). If data bus competent devices are used at network (e. g. RS485) it is according to the target bus adress.
- The commando-/reference data length is never larger as 240 Bytes. If a larger pack of data has to be transmissioned, devide into several adequate frames/blocks.
- The transmission of numeral values (e. g. data, dimension- or offset indication), larger as one byte (16 or 32 bit), results with lower byte first (big endian)

Explanation:

Transmissions from Pc/Host to the controller/target are labeled as follows:

Host → Target (commano)

Returns from controller/target to the PC/Host are labeled as follows:

Host ← Target (answer)

3. Commands

A line represents a byte if not in the following chart especially noted. If a „...“ is noted, it stand for a indefinite number of bytes (contextual).

3.1 Request for version data

Host → Target (Version commando)

name	offset dec	value hex	description
LEN	0	00	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	94	Command byte 1
CMD2	4	00	Command byte 2
CHKSUM	5	15	Checksum byte (15h = AAh – [94h+01h])

Host ← Target 0AAh (= correct checksum) within 0.2s.

Host ← Target (Version answer)

name	offset dec	value hex	description
LEN	0	0C	Reference data length byte (of this frame)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	94	Command byte 1
CMD2	4	00	Command byte 2
DEV-ID	5	iiii	Component software-ID
VERS-SOFT	7	ssss	Software version (SHORT)
VERS-KALI	9	aaaa	Calibration data version (SHORT)
VERS-SYS	11	bbbb	System data version (SHORT)
VERS-PAR	13	cccc	Parameter data version (SHORT)
VERS-PROC	15	dddd	Process data version (SHORT)
CHKSUM	17	##	Checksum byte

3.2 Request status block

The status data contain a survey of sizes of all data files as well as changing data (IO-data, flag-data, process-data). The communication proceeds as follows:

Host → Target (demand for primary block)

name	offset dec	value hex	description
LEN	0	00	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	12	Command byte 1
CMD2	4	0	Command byte 2, blocknumber 0, according to the number of data several blocks (max 240 bytes) must be requested.
CHKSUM	5	97	Checksum byte (97h = AAh – [12h+01h])

Host ← Target 0AAh (= correct checksum) within 0.2s.

Host ← Target (answer primary block)

name	offset dec	value hex	description
LEN	0	nn	Reference data length byte (of this frame)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	12	Command byte 1
CMD2	4	0	Command byte 2, number of the required block
C.-VERS0	5	00	Communication version byte 0
C.-VERS1	6	01	Communication version Byte 1
IO-LEN	7	aaaa	Number of I/O-data in bytes (SHORT)
FLAG-LEN	9	bbbb	Number of Flag-data in bytes (SHORT)
PAR-LEN	11	0000	Number of Parameter-daten in bytes (SHORT, here = 0)
PROC-LEN	13	cccc	Number of Process data in bytes (SHORT)
HMI-LEN	15	0000	HMI-data (SHORT, here = 0)
IO-DATA	ab 17	##	I/O-data
...
FLAG-DATA	##	##	Flag-data
...
PROC-DATA	##	##	Process-data
...
CHKSUM	5+nn	##	Checksum byte

If the sum of I/O-, flag- and process-data is larger than 240 bytes, the cycle-status block must be demanded several times. If an error occurs, the procedure must be repeated completely with block number 0!

Host → Target (demand status-cycle block)

name	offset dec	value hex	description
LEN	0	00	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	12	Command byte 1
CMD2	4	blknr	Command byte 2, block number blknr=1..nn
CHKSUM	5	##	Checksum byte

Host ← Target: 0AAh (= correct checksum) within 0.2s.

Host ← Target (answer status-cycle block)

name	offset dec	value hex	description
LEN	0	nn	Reference data length byte (of this frame)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	12	Command byte 1
CMD2	4	blknr	Command byte 2, number of the required block
IO-DATA	##	##	I/O-data
...
FLAG-DATA	##	##	Flag-data
...
PROC-DATA	##	##	Process-data
...
CHKSUM	5+nn	##	Checksum byte

IO-DATA:

The configuration of is documented in the file "\DEVELOP\ memio.dok" (memio_eng.dok).

FLAG-DATA:

The configuration of these data is corresponding to the structure "FLAG" from the file "\APPL\source2.h". The number of the transmitted information is obvious via "NUM_FLAGCOM_BYTES". This information is listed on the top of the file.

The data are also listed in the file "\DEVELOP\ memio.dok".

PROC-DATA:

The configuration of these data is corresponding to the structure "RD_PROC" from the file "\APPL\source2.h".

The data are also listed in the file "\DEVELOP\ memio.dok".

3.3 Request status data

This order allows the specific retransfer (demand) of selected parts of the IO-, flag-, system- and proces data. Via this offset and the length can the start and the number of bytes be forced. These details refer to the data from the file "memio.dok".

Host → Target (request status data partial)

name	offset dec	value hex	description
LEN	0	3	Command-/reference data length byte
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	13	Command byte 1
CMD2	4	tt	Command byte 2, tt = 01h: from IO-data tt = 02h: from flag-data tt = 03h: from parameter data tt = 04h: from system data tt = 05h: from process-data
OFFSET_L	5	OI	Offset, low order byte
OFFSET_H	6	Oh	Offset, high order byte
LENGTH	7	len	Number of retransferred bytes (<240)
CHKSUM	8	##	Checksum byte

Host ← Target

0AAH (within 0.2s, if correct checksum is acquired)

Host ← Target

name	offset dec	value hex	description
LEN	0	nn	Reference data length byte (of this telegram)
DST	1	00	Destination adresss
SRC	2	01	Source adress
CMD1	3	13	Command byte 1
CMD2	4	tt	Command byte 2
REQ-DATA	ab 5	##	Demanded databytes
CHKSUM	nn+5	##	Checksum byte

3.4 Request parameter block

Communication procedure:

Host → Target (ask for first parameter block)

name	offset dec	value hex	description
LEN	0	00	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	99	Command byte 1
CMD2	4	00	Command byte 2, blocknumber 0, according to the number of data several blocks (max 240 bytes) must be requested.
CHKSUM	5	10	Checksum byte (10h = AAh – [99h+01h])

Host ← Target

0AAh (within 0.2s if correct checksum is acquired)

Host ← target (answer on first block)

Name	offset dec	value hex	description
LEN	0	nn	Reference data length byte (of this frame)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	99	Command byte 1
CMD2	4	0	Command byte 2, number of the required block
C.-VERS0	5	00	Communication version byte 0
C.-VERS1	6	01	Communication version byte 1
IO-LEN	7	0000	Number of I/O-data in bytes (SHORT)
FLAG-LEN	9	0000	Number of Flag-data in bytes (SHORT)
PAR-LEN	11	aaaa	Number of Parameter-Data in bytes (SHORT)
SYS-LEN	13	bbbb	Number of System data in byta (SHORT)
HMI-LEN	15	0000	HMI-Data (SHORT, here = 0)
PAR-DATA	ab 17	##	Parameter-data
...
SYS-DATA	##	##	System parameter-data
...
CHKSUM	nn+5	##	Checksum byte

If the sum of I/O-, flag- and process-data is larger than 240 bytes, the cycle-status block must be demanded several times. If an error occurs, the procedure must be repeated completely with block number 0!

Host → Target (demand parameter cycle block)

name	offset dec	value hex	description
LEN	0	00	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	99	Command byte 1
CMD2	4	blknr	Command byte 2, block number blknr=1..nn
CHKSUM	5	##	Checksum byte

Host ← Target 0AAh (= correct checksum) within 0.2s.

Host ← Target (parameter cycle block)

name	offset dec	value hex	description
LEN	0	nn	Reference data length byte (of this frame)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	99	Command byte 1
CMD2	4	blknr	Command byte 2, number of the required block
PAR-DATA	ab 5	##	Parameter-data
...
SYS-DATA	##	##	System parameter-data
...
CHKSUM	nn+5	##	Checksum byte

Explanation of data-range:

PAR-DATA:

The configuration of these data is corresponding to the structure "PARAM" from the file „APPL\source2.h“.

The data are also listed in the file “\DEVELOP\ memio.dok“ (memio_eng.dok).

SYS-DATA:

The configuration of these data is corresponding to the structure „SYSTEM“ from the file “APPL\source2.h“.

The data are also listed in the file “\DEVELOP\ memio.dok“.

3.5 Set parameter data

This command allows the controlled manipulation of a selected parameter files on the target. Via the offset and the length can the start and the number of datas to manipulate be forced. This information corresponds to the data in the file "memio.dok" (memio_eng.dok). (Only one value of an element can be set)

Host → Target (set parameter commando)

name	offset dec	value hex	description
LEN	0	10+nn	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	8F	Command byte 1
CMD2	4	00	Command byte 2, number of the required block
Offset (long)	5-8	## ## ## ##	32bit-number, defines the offset within the parameter block
Value (long)	9-12	## ## ## ##	32bit-number, contains the value to set
Length (long)	13-16	## ## ## ##	32bit-number, defines the length of the value (byte = 1, short = 2, long = 4)
Length of the extended informaton	17-20	nn nn nn nn (00 00 00 00)	Number of bytes of the extended information (regularly = 0)
CHKSUM	nn+21	##	Checksum byte

Host ← Target 0AAh (= correct checksum) within 0.2s.
(no reponse)

3.6 Set system parameter data

This command allows the controlled manipulation of a selected system parameter files on the target. Via the offset and the length can the start and the number of datas to manipulate be forced. This information corresponds to the data/lines in the file "memio.dok" (memio_eng.dok).

Host → Target (set parameter commando)

name	offset dec	value hex	description
LEN	0	10+nn	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	95	Command byte 1
CMD2	4	00	Command byte 2, number of the required block
Offset (long)	5-8	## ## ## ##	32bit-number, defines the offset within the parameter block
Value (long)	9-12	## ## ## ##	32bit-number, contains the value to set
Length (long)	13-16	## ## ## ##	32bit-number, defines the length of the value (byte = 1, short = 2, long = 4)
Length of the extended informaton	17-20	nn nn nn nn (00 00 00 00)	Number of bytes of the extended information (regularly = 0)
CHKSUM	nn+21	##	Checksum byte

Host ← Target 0AAh (= correct checksum) within 0.2s.
(no reponse)

3.7 Set process data

This command allows the controlled manipulation of a selected process data on the target. Via the offset and the length can the start and the number of datas to manipulate be forced. This information corresponds to the data in the file "memio.dok" (memio_eng.dok). (Only one value of an element can be set)

Host → Target (set parameter command)

name	offset dec	value hex	description
LEN	0	0C	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	91	Command byte 1
CMD2	4	00	Command byte 2, number of the required block
Offset (long)	5-8	## ## ## ##	32bit-number, defines the offset within the parameter block
Value (long)	9-12	## ## ## ##	32bit-number, contains the value to set
Length (long)	13-16	## ## ## ##	32bit-number, defines the length of the value (byte = 1, short = 2, long = 4)
CHKSUM	nn+21	##	Checksum byte

Host ← Target 0AAh (= correct checksum) within 0.2s.
(no reponse)

3.8 Set flag data

This command allows the controlled manipulation of a selected flag data on the target (**ATTENTION – the application flow of the software can be disturbed**). Via the offset and the length can the start and the number of datas to manipulate be forced. This information corresponds to the data in the file "memio.dok" (memio_eng.dok). (Only one value of an element can be set)

Host → Target (set parameter command)

name	offset dec	value hex	description
LEN	0	0C	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	89	Command byte 1
CMD2	4	00	Command byte 2, number of the required block
Offset (long)	5-8	## ## ## ##	32bit-number, defines the offset within the parameter block
Value (long)	9-12	## ## ## ##	32bit-number, contains the value to set
Length (long)	13-16	## ## ## ##	32bit-number, defines the length of the value (byte = 1, short = 2, long = 4)
CHKSUM	nn+21	##	Checksum byte

Host ← Target 0AAh (= correct checksum) within 0.2s.
(no reponse)

4. Use radCASE information

For each radCASE-project, a file "memio.dok" is generated. It contains measurement data (IO's), work variable/marker (flags), fixed setting data (parameter), fixed data (process data) and fixed system data (system parameter).

The **I/O-, flag- and process-data** can be completely demanded by the commando **request for status block (3.2)** from the controller.

The **parameter- and system-parameter-data** can be read completely by the commando **request for parameter block (3.4)**.

When **only certain values or categorys** are of interest, you can find out via the commando **request for status data (3.3)**

Following information is in the file "memio.dok":

	Element	Offset	Length/Bit	Description	Unit/Selection Format
- IOs	Element	Offset	Length/Bit	Description	Unit/Selection Format
- flags	Element	Offset	Length/Bit	Description	Unit/Selection Format
- parameter	Element	Offset	Length/Bit	Description	Unit/Selection Format
- process data	Element	Offset	Length/Bit	Description	Unit/Selection Format
- system parameter	Element	Offset	Length/Bit	Description	Unit/Selection Format

Element:

The (radCASE-) name for the data value

Offset:

The location of the data value within his specific block (IO/Flag/Process/Parameter/System)

Length/Bit:

Either the length of the date value or the bit number ("Bit n"), if it is a binary data value

Description:

Short description of the element

Unit/Selection:

Has the value a unit ("unit", i. e. "°C"), it is notet there. If the value has a specific denotation than are there several possibilities too choose → example: "0 = UNDEF, 1 = DISABLE, 2 = ENABLE".

Format:

If it is a numerical value then "Format" indicates the kind of fixed-point number (Digits + Dezimalen) → example: 123.45 accords to 6 Digits and 2 Dezimale, 9876 accords to 4 Digits and 0 Dezimale. (Fixed-point number: 123.45 is managed as 12345)

4.1 Request for specific data value

If you want to know i. e. an I/O-value, can it be as following:

Example: The project contains an analogue input with following entry in the "memio.dok" (memio_eng.dok) in the segment "IOs":

Wasserkrei_Wassertemp_AL_TVorlaufRohEx (element)
 32 (offset → 0020hex)
 2 (length/bit)
 Outlet-raw (Description)
 °C (Unit/Selections)
 6 Digits, 2 Dezimalen (Format)

→ Send to controller:

Commando ask for status data

name	offset dec	value hex	description
LEN	0	03	Command-/reference data length byte
DST	1	01	Destination adress
SRC	2	00	Source adress
CMD1	3	13	Command byte 1
CMD2	4	01	Command byte 2, tt = 01h: from IO-data tt = 02h: from flag-data tt = 03h: from parameter data tt = 04h: from system data tt = 05h: from process-data
OFFSET_L	5	20	Offset, low order byte
OFFSET_H	6	00	Offset, high order byte
LENGTH	7	02	Number of retransfered bytes (<240)
CHKSUM	8	70	Checksum byte

You get as answer the checksum (regularly 0xAA) first and then the asked data.

name	offset dec	value hex	description
LEN	0	02	Reference data length byte (of this telegram)
DST	1	00	Destination adress
SRC	2	01	Source adress
CMD1	3	13	Command byte 1
CMD2	4	01	Command byte 2
REQ-DATA0	5	30	Low, asked data bytes (Wasserkrei_Wassertemp_AL_TVorlaufRohEx)
REQ-DATA1	6	06	High, asked data bytes (Wasserkrei_Wassertemp_AL_TVorlaufRohEx)
CHKSUM	2+5	5D	Checksum byte

The value for Wasserkrei_Wassertemp_AL_TVorlaufRohEx is 0630hex (→ 1584 dez) = "015.84 °C".

5. Implementation example in C

5.1 Send commando

```

int SendCommand(unsigned char src, unsigned char dst, unsigned char cmd1, un-
signed char cmd2, char*
data, unsigned char data_len)
{
    int i, count;
    unsigned char sum;
    unsigned char buf[256];
    unsigned char rec_byte=0;

    if ( data_len > 240 )
        return -1;

    buf[0] = data_len; // number of reference data bytes
    buf[1] = dst; // target
    buf[2] = src; // source
    buf[3] = cmd1; // commando 1
    buf[4] = cmd2; // commando 2
    memcpy(&buf[5], data, data_len); // copy reference date into cache

    // find out checksum
    for (sum = 0, i = 0; i < (data_len + 5); i++)
        sum = (unsigned char)(sum + buf[i]);

    // write checksum
        buf[i] = (unsigned char)(0xAA - sum);

    // try max 5 time to dispose commande
    count = 0;
    do
    {
        int anz;

        // delete serial read and write cache
        g_serial.Clear();

        // send complete data block
        g_serial.Write(buf, data_len + 6, 100);

        // read answer from controller (AAh = correct received), wait max. Re-
        sponse-time

        rec_byte = 0;
        anz = g_serial.Read(&rec_byte, sizeof(rec_byte), g.tResponse + 300);
        if ( anz < 0 )
            {
                char text[80];
                sprintf(text, "SendCommand %x-Lesefehler: %d\n", (int)cmd1, anz);
                OutputDebugString(text);
                Sleep(100);
            }

        // no answer?
        if ( !anz )
            {
                char text[120];
                sprintf(text, "Keine Antwort auf SendCommand dst:%d, cmd1:%d,
                cmd2:%d\n",
                dst, cmd1, cmd2);
                OutputDebugString(text);
                Sleep(g.tError);
            }
    }
}

```

```

        if ( count == 0 )
            count += 1;
        }

// got wrong answer?
if ( (rec_byte != 0xAA) && (anz > 0) )
    {
        char text[120];
        sprintf(text, "SendCommand: Antwort %d != 0xAA, dst:%d,
cmd1:%d, cmd2:%d\n",
(int)rec_byte, dst, cmd1, cmd2);
        OutputDebugString(text);

        Sleep(g.tError);
        g_serial.Clear();
    }

    count++;
}while ( (rec_byte != 0xAA) && (count < 5) );

// after x tries to dispose command block no correct answer?

if ( rec_byte != 0xAA )
    {
        // yes, report error
        char szText[160];
        sprintf(szText, "SendCommand-Abbruch: Fehler bei Uebertragung:
%d\n", (int)rec_byte);
        OutputDebugString(szText);
        return -1;
    }
return 0;
}

```

5.2 Receive answer

```

int ReceiveAnswer(unsigned char* buffer)
{
    unsigned char rec_byte, anz;
    int i;
    unsigned char sum, chksum, src, dst;

    sum = 0;
    int ret;

    // read number of reference data bytes
    ret = g_serial.Read(&anz, sizeof(anz), 1000);
    if ( ret <= 0 )

        {
            OutputDebugString("ReceiveAnswer: Nichts empfangen.\n");
            return -1;
        }
    sum = (unsigned char)(sum + anz);

    // target dst
    if ( !g_serial.Read(&dst, sizeof(dst), TIMEOUT_PER_BYTE) )
        {
            OutputDebugString("ReceiveAnswer: Keine Dst empfangen\n");
            return -1;
        }
    sum = (unsigned char)(sum + dst);

    // source src
    if ( !g_serial.Read(&src, sizeof(src), TIMEOUT_PER_BYTE) )
        {
            OutputDebugString("ReceiveAnswer: Keine Src empfangen\n");

```

```
        return -1;
    }
    sum = (unsigned char)(sum + src);

// comando 1
if ( !g_serial.Read(&g_cmd1, sizeof(g_cmd1), TIMEOUT_PER_BYTE) )
    {
    OutputDebugString("ReceiveAnswer: Kein Cmd1 empfangen\n");
    return -1;
    }
    sum = (unsigned char)(sum + g_cmd1);

// comando 2
if ( !g_serial.Read(&g_cmd2, sizeof(g_cmd2), TIMEOUT_PER_BYTE) )
    {
    OutputDebugString("ReceiveAnswer: Kein Cmd2 empfangen\n");
    return -1;
    }
sum = (unsigned char)(sum + g_cmd2);

// maybe read extended info
if ( g_cmd2 & 0x80 )
    {
    // because bit 7 is set im cmd2, an additional byte is read like the
    // following info
    if ( !g_serial.Read(&ext_info_types, sizeof(ext_info_types), TIME-
    OUT_PER_BYTE) )
        {
        OutputDebugString("ReceiveAnswer: Kein ext_info_types empfan-
        gen\n");
        return -1;
        }
    sum = (unsigned char)(sum + ext_info_types);

    // read the extended info according to the type/bit
    if ( ext_info_types & EIT_BURST )
        {
        if ( !g_serial.Read(&eit_burst_block_id,
        sizeof(eit_burst_block_id), TIMEOUT_PER_BYTE) )
            {
            OutputDebugString("ReceiveAnswer: Kein
            eit_burst_block_id empfangen\
            n");
            return -1;
            }
        sum = (unsigned char)(sum + eit_burst_block_id);
        }
    }
else
    ext_info_types = 0; // es werden keine ext_info_types von der Steue-
    rung gesendet

// read reference data byte
if ( anz > 0 )
    {
    if ( anz > 240 )
        return -23;
    int read_bytes = g_serial.Read(buffer, anz, 750);

    if ( read_bytes != anz )
        {
        OutputDebugString("ReceiveAnswer: Keine/zu wenig Nutzdaten
        empfangen\n");
        return -1;
        }

    for ( i = 0; i < (int)anz; i++)
```

```

        sum = (unsigned char)(sum + buffer[i]);
    }

    // read checksum
    if ( !g_serial.Read(&rec_byte, sizeof(rec_byte), 50) )
    {
        OutputDebugString("ReceiveAnswer: Keine Checksumme empfangen\n");
        return -1;
    }
    chksum = (unsigned char)(rec_byte + sum);

    if ( chksum != 0xAA )
    {
        OutputDebugString("ReceiveAnswer: Falsche Checksumme empfangen\n");
        return -1;
    }

    return (int)anz;
}

```

5.3 Demand and receive status block

```

#define VCM_GET_STATUS 0x12

char _ioBlock[IOBLOCKLEN];
char _flagsBlock[OTHERBLOCKLEN];
char _paramBlock[PARAMBLOCKLEN];
char _systemBlock[OTHERBLOCKLEN];
char _procBlock[OTHERBLOCKLEN];

ret = SendCommand(0, 1, VCM_GET_STATUS, 0, 0, 0); // Status-Block anfordern und
holen
// could demand be sent?
if (ret >= 0)
{
    // yes, wait for possible controller delay
    Sleep(15);

    int anz = cclReceiveAnswer(RecBuf);
    if (anz > 0)
    {
        // communication version 1 / newest communication?
        if ( (RecBuf[0] == 0) && (RecBuf[1] == 1) )
        {
            if ( RecvMultiBlock(RecBuf) < 0 )
            {
                // Error
            }
        }
    }
}

int RecvMultiBlock(unsigned char* RecBuf)
{
    unsigned short IoToRecv, FlagToRecv, ParamToRecv, ProcToRecv, SystemTo-
Recv, ToRecv;
    unsigned short IoRecvd=0, FlagRecvd=0, ParamRecvd=0, ProcRecvd=0, System-
Recv=0;
    unsigned short* usBuf;
    unsigned char blk_nr=1;
    unsigned char recvd=12; // CommVersion, IoToRecv, FlagToRecv, ParamToRecv,
ProcToRecv, HMITo-
Recv ueberspringen ...
    unsigned long par_align_corr, proc_align_corr;
    /*** in paramLenSum the length of the parameter blocks (which came from
    /*** different nodes) are accumulated
    static short paramLenSum = 0;
    bool bParams = false;

```

```
long IoOffset = 0, FlagOffset=0, ParamOffset=0, ProcOffset=0, SystemOff-
set=0;

par_align_corr = 0;
proc_align_corr = 0;

ParamOffset -= 0;//par_align_corr;
ProcOffset -= 0;//proc_align_corr;

usBuf = (unsigned short*)RecBuf;

IoToRecv = usBuf[1];
FlagToRecv = usBuf[2];
ParamToRecv = usBuf[3];
ProcToRecv = usBuf[4];

// when param-data are received, in usBuf[4] ist he system data lenght no-
// ted (if sent)
if ( ParamToRecv && ProcToRecv )
{
    SystemToRecv = ProcToRecv;
    ProcToRecv = 0;
}
else
    SystemToRecv = 0;

ToRecv = IoToRecv + FlagToRecv + ParamToRecv + ProcToRecv + SystemToRecv;

if ( ParamToRecv )
{
    bParams = true;

    paramLenSum = 0;

    if ( _MemoryFormat == 1 )
        paramLenSum += Swap(usBuf[6]); // 6 -> size aus Param-Struktur
        (5=HMI-Laenge)
    else
        paramLenSum += usBuf[6];
}

while ( ToRecv > 0 )
{
    // I/O-Daten
    if ( IoToRecv > 0 )
    {
        unsigned short n;

        if ( (IoToRecv + recvd) >= 240 )
            n = 240 - recvd;
        else
            n = IoToRecv;

        ToRecv -= n;
        IoToRecv -= n;

        memcpy(&_ioBlock[IoRecvd+IoOffset], &RecBuf[recvd], n);
        IoRecvd += n;
        recvd += n;
    }

    // Flag-Daten
    if ( (recvd < 240) && (FlagToRecv > 0) )
    {
        unsigned short n;

        if ( (FlagToRecv + recvd) >= 240 )
```

```
        n = 240 - recvd;
    else
        n = FlagToRecv;

    ToRecv -= n;
    FlagToRecv -= n;

    memcpy(&_flagsBlock[FlagRecv+FlagOffset], &RecBuf[recvd], n);
    FlagRecv += n;
    recvd += n;
}

// Parameter-data
if ( (recvd < 240) && (ParamToRecv > 0) )
{
    unsigned short n;

    if ( (ParamToRecv + recvd) >= 240 )
        n = 240 - recvd;
    else
        n = ParamToRecv;

    ToRecv -= n;
    ParamToRecv -= n;

    int corrOff = 0, corrLen = 0;

    memcpy(&_paramBlock[ParamRecv+ParamOffset],
        &RecBuf[recvd+corrOff], n -corrLen);
    ParamRecv += n - corrLen;
    recvd += n;
}

// System-Data
if ( (recvd < 240) && (SystemToRecv > 0) )
{
    unsigned short n;

    if ( (SystemToRecv + recvd) >= 240 )
        n = 240 - recvd;
    else
        n = SystemToRecv;

    ToRecv -= n;
    SystemToRecv -= n;

    int corrOff = 0, corrLen = 0;

    memcpy(&_systemBlock[SystemRecv+SystemOffset],
        &RecBuf[recvd+corrOff], n -corrLen);
    SystemRecv += n - corrLen;
    recvd += n;
}

// Process-Data
if ( (recvd < 240) && (ProcToRecv > 0) )
{
    unsigned short n;

    if ( (ProcToRecv + recvd) >= 240 )
        n = 240 - recvd;
    else
        n = ProcToRecv;

    ToRecv -= n;
    ProcToRecv -= n;
}
```

```
int corrOff = 0, corrLen = 0;

memcpy(&_procBlock[ProcRecvd+ProcOffset],
&RecBuf[recvd+corrOff], n -corrLen);

ProcRecvd += n - corrLen;
recvd += n;
}
if ( ToRecv <= 0 )
{
if ( _MemoryFormat == 1 )
*(short*)&_paramBlock[0] = Swap(paramLenSum);
else
*(short*)&_paramBlock[0] = paramLenSum;

return 0;
}

recvd = 0;

// ask for statusblock and receiving
int ret = cclSendCommand(0, 1, VCM_GET_STATUS, blk_nr, 0, 0);
// Konnte Anforderung abgesetzt werden?
if (ret >= 0)
{
int anz = cclReceiveAnswer(RecBuf);
if (anz <= 0)
return -1; // FEHLER!
blk_nr++;
}
else
return -2; // FEHLER!
} // while

return 0;
}
```