

IMACS

Steuergerätekommunikation

Datenprotokoll 2.0

Rev.-Stand: 15

IMACS GmbH

Meß- und Steuerungstechnik

Mainzer Str. 139
D-55545 Bad Kreuznach

1. ALLGEMEINES	3
1.1 Beispiel Prüfsummenberechnung in C	3
2. KOMMANDOBLOCKFORMAT	4
3. BEFEHLE	5
3.1 Versionsdaten anfordern.....	5
3.2 Statusblock anfordern	6
3.3 Statusdaten partiell anfordern.....	8
3.4 Parameterblock anfordern	9
3.5 Parameterdaten setzen.....	11
3.6 Systemparameterdaten setzen.....	12
3.7 Prozeßdaten setzen.....	13
3.8 Flagdaten setzen	13
4. RADCASE-INFORMATIONEN NUTZEN	14
4.1 Bestimmten Datenwert anfordern	15
5. IMPLEMENTIERUNGSBEISPIEL IN C	16
5.1 Kommando senden.....	16
5.2 Antwort empfangen	17
5.3 Statusblock anfordern und empfangen.....	18

1. Allgemeines

Die Steuerung arbeitet kommunikationstechnisch als Slave, d.h. sie empfängt vom Host (z.B. PC/SPS) Kommandos auf die ggf. eine Antwort zurückgesendet wird. Der Datenaustausch erfolgt blockorientiert, wobei zur Erkennung von Übertragungsfehlern ein Prüfsummentest verwendet wird. Die Prüfsumme wird bei einem Empfang ohne Übertragungsfehler als Bestätigung zurückgesendet. Die Dateneinrichtung erkennt somit, daß der Kommando-/Datenblock korrekt übertragen wurde, und quittiert ihrerseits dies mit einem Rücksenden der Prüfsumme (immer fest 0AAH). Da die Größe eines Blocks auf max. 240 Bytes Nutzdatenlänge beschränkt ist, werden zu übertragene Nutzdaten mit größerer Datenlänge auf entsprechend viele Blöcke verteilt.

1.1 Beispiel Prüfsummenberechnung in C

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

buf[0] = data_len;      // Anzahl der Nutzdatenbytes
buf[1] = dst;           // Ziel
buf[2] = src;           // Quelle
buf[3] = cmd1;          // Kommando 1
buf[4] = cmd2;          // Kommando 2
memcpy(&buf[5], data, data_len); // Nutzdaten in Sendepuffer kopieren

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

// Pruefsumme in Sendepuffer schreiben
buf[i] = (unsigned char)(0xAA - sum);
```

2. Kommandoblockformat

Im folgenden ist das grundlegende Format eines Kommandoblockes beschrieben, den die Dateneinrichtung an die Steuerung sendet.

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Kommando-/Nutzdatalängenbyte (Länge der Blocknutzdaten) [0..240]
DST	1	dd	Zieladresse (Bus-Adresse des Targets) [1...x]
SRC	2	ss	Quelladresse (Bus-Adresse des Targets, bei PC = 0)
CMD1	3	##	Befehlsbyte 1
CMD2	4	##	Befehlsbyte 2
DATA1	5	##	Nutzdatenbyte 1
...
DATAnn	5+(nn-1)	##	Nutzdatenbyte nn
CHKSUM	5+nn	##	Prüfsummenbyte

- Das Blocklängenbyte enthält die Länge des Blockes (ohne Kommandolängenbyte, ohne Adressbytes, ohne Befehlsbytes, ohne Prüfsummenbyte).
- Das Prüfsummenbyte ergänzt die Bytesumme des gesamten Blockes (also alle Bytes) auf 0AAh.
- Die Quell- und Zieladresse hat bei Kommunikation mit nur einem Endgerät keine Funktion (Inhalt wird nicht verwendet). Bei busfähigen Geräte im Netzwerk (z.B. RS485) entspricht dies der Target-Busadresse.
- Die Kommando-/Nutzdatalänge ist niemals größer als 240 Bytes. Falls größere Datenmenge zu übertragen sind, müssen diese in entsprechend mehrere Telegramme/Blöcke (einen Erstblock und n Folgeblöcke) aufgeteilt werden.
- Die Übertragung von numerischen Werten (z.B. Daten, Größen- oder Offsetangabe), die größer als ein Byte sind (16 ode 32 Bit), erfolgt immer mit dem niederwertigen Byte zuerst (big endian).

Erläuterungen:

Sendungen vom PC/Host zum Steuergerät/Target werdem wie folgt gekennzeichnet:

Host → Target (Kommando)

Rücksendungen vom Steuergerät/Target zum PC/Host werden wei folgt gekennzeichnet:

Host ← Target (Antwort)

3. Befehle

In den folgenden Tabellen stellt eine Zeile, falls nicht anders angegeben, jeweils ein Byte dar. Falls in der Zeile "..." vermerkt ist, so steht dies für eine unbestimmte Anzahl von Bytes (kontextabhängig).

3.1 Versionsdaten anfordern

Host → Target (Versions-Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	00	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	94	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
CHKSUM	5	15	Prüfsummenbyte (15h = AAh – [94h+01h])

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde).

Host ← Target (Versionsantwort)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	0C	Nutzdatenlängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	94	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
DEV-ID	5	iiii	Geräte-Software-ID
VERS-SOFT	7	ssss	Softwareversion (SHORT)
VERS-KALI	9	aaaa	Kalibrierdatenversion (SHORT)
VERS-SYS	11	bbbb	Systemdatenversion (SHORT)
VERS-PAR	13	cccc	Parameterdatenversion (SHORT)
VERS-PROC	15	dddd	Prozessdatenversion (SHORT)
CHKSUM	17	##	Prüfsummenbyte

3.2 Statusblock anfordern

Die Statusdaten beinhalten eine Größenübersicht aller Datenbereiche sowie dies sich betriebsbedingt ändernden Daten (EA-Daten, Flag-Daten, Prozessdaten). Die Kommunikation läuft wie folgt ab:

Host → Target (Status-Erstblock anfordern)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	00	Kommando-/Nutzdatalängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	12	Befehlsbyte 1
CMD2	4	0	Befehlsbyte 2, Blocknummer 0, je nach Anzahl der Daten müssen mehrere Blöcke (je max. 240 Bytes) angefordert werden
CHKSUM	5	97	Prüfsummenbyte (97h = AAh – [12h+01h])

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde).

Host ← Target (Status-Erstblockantwort)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Nutzdatalängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	12	Befehlsbyte 1
CMD2	4	0	Befehlsbyte 2, Nummer des angeforderten Blocks
C.-VERS0	5	00	Kommunikationsversion Byte 0
C.-VERS1	6	01	Kommunikationsversion Byte 1
IO-LEN	7	aaaa	Anzahl der E/A-Daten in Bytes (SHORT)
FLAG-LEN	9	bbbb	Anzahl der Flag-Daten in Bytes (SHORT)
PAR-LEN	11	0000	Anzahl der Parameter-Daten in Bytes (SHORT, hier = 0)
PROC-LEN	13	cccc	Anzahl der Prozessdaten in Bytes (SHORT)
HMI-LEN	15	0000	HMI-Daten (SHORT, hier = 0)
IO-DATA	ab 17	##	E/A-Daten
...
FLAG-DATA	##	##	Flag-Daten
...
PROC-DATA	##	##	Prozess-Daten
...
CHKSUM	5+nn	##	Prüfsummenbyte

Wenn die Summe der E/A-, Flag- und Prozessdaten größer als 240 Bytes ist, muß entsprechend oft eine Folge-Statusblock, wie im unten beschrieben, angefordert werden. Sollte dabei ein Fehler auftreten, dann muß der gesamte Vorgang komplett wieder mit Blocknummer 0 (d.h. Erstblock, s.o.) gestartet werden!

Host → Target (Status-Folgeblock anfordern)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	00	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	12	Befehlsbyte 1
CMD2	4	blknr	Befehlsbyte 2, Blocknummer blknr=1..nn
CHKSUM	5	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

Host ← Target (Status-Folgeblockantwort)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Nutzdatenlängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	12	Befehlsbyte 1
CMD2	4	blknr	Befehlsbyte 2, Nummer des angeforderten Blocks
IO-DATA	##	##	E/A-Daten
...
FLAG-DATA	##	##	Flag-Daten
...
PROC-DATA	##	##	Prozess-Daten
...
CHKSUM	5+nn	##	Prüfsummenbyte

Erläuterung der Datenbereiche:

IO-DATA:

Der Aufbau der Daten ist in der Datei „\DEVELOP\ memio.dok“ dokumentiert.

FLAG-DATA:

Der Aufbau dieser Daten entspricht der Struktur „FLAG“ aus der Datei „\APPL\source2.h“. Die Anzahl der davon übertragenen Informationen ist über „NUM_FLAGCOM_BYTES“ ersichtlich. Diese Information steht ganz vorne in der Datei.

Außerdem sind die Daten auch in der Datei „\DEVELOP\ memio.dok“ aufgelistet.

PROC-DATA:

Die Anordnung der Daten entspricht der Struktur „RD_PROC“ aus der Datei „\APPL\source2.h“. Die Daten sind auch in der Datei „\DEVELOP\ memio.dok“ aufgelistet.

3.3 Statusdaten partiell anfordern

Dieser Befehl ermöglicht die gezielte Rückübertragung (Anforderung) von ausgewählten Teilen der IO-, Flag-, Parameter-, System- und Prozeßdaten. Über den Offset und die Länge kann der Startpunkt sowie die Anzahl der Bytes vorgegeben werden. Diese Angaben beziehen sich auf die jeweils in der „memio.dok“ angegebenen Daten.

Host → Target (Statusdaten-partiell Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	3	Kommando-/Nutzdatalängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	13	Befehlsbyte 1
CMD2	4	tt	Befehlsbyte 2 tt = 01h: von IO-Daten tt = 02h: von Flag-Daten tt = 03h: von Parameter-Daten tt = 04h: von System-Daten tt = 05h: von Prozeß-Daten
OFFSET_L	5	ol	Offset, niederwertiges Byte
OFFSET_H	6	oh	Offset, höherwertiges Byte
LENGTH	7	len	Anzahl der rückzugebenden Bytes (< 240)
CHKSUM	8	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

Host ← Target (Statusdaten-Rückgabe)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Nutzdatalängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	13	Befehlsbyte 1
CMD2	4	tt	Befehlsbyte 2
REQ-DATA	ab 5	##	angeforderte Datenbytes
CHKSUM	nn+5	##	Prüfsummenbyte

3.4 Parameterblock anfordern (Param- und , Sysparamdaten)

Kommunikationsablauf:

Host → Target (Parameter-Firstblock anfordern)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	00	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	99	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2, Blocknummer 0, je nach Anzahl der Daten müssen mehrere Blöcke (je max. 240 Bytes) angefordert werden
CHKSUM	5	10	Prüfsummenbyte (10h = AAh – [99h+01h])

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

Host ← Target (Parameter-Firstblockantwort)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Nutzdatenlängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	99	Befehlsbyte 1
CMD2	4	0	Befehlsbyte 2, Nummer des angeforderten Blocks
C.-VERS0	5	00	Kommunikationsversion Byte 0
C.-VERS1	6	01	Kommunikationsversion Byte 1
IO-LEN	7	0000	Anzahl der E/A-Daten in Bytes (SHORT)
FLAG-LEN	9	0000	Anzahl der Flag-Daten in Bytes (SHORT)
PAR-LEN	11	aaaa	Anzahl der Parameter-Daten in Bytes (SHORT)
SYS-LEN	13	bbbb	Anzahl der Systemdaten in Bytes (SHORT)
HMI-LEN	15	0000	HMI-Daten (SHORT, hier = 0)
PAR-DATA	ab 17	##	Parameter-Daten
...
SYS-DATA	##	##	Systemparameter-Daten
...
CHKSUM	nn+5	##	Prüfsummenbyte

Wenn die Summe der Parameter- und Systemparameterdaten größer als 240 Bytes ist, dann muß entsprechend oft ein Folgeblock, wie im folgenden beschrieben, angefordert werden. Sollte dabei ein Fehler auftreten, dann muß der Vorgang komplett wieder mit Blocknummer 0 gestartet werden!

Host → Target (Parameter-Folgeblock anfordern)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	00	Kommando-/Nutzdatalängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	99	Befehlsbyte 1
CMD2	4	blknr	Befehlsbyte 2, Blocknummer blknr=1..nn
CHKSUM	5	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

Host ← Target (Parameter-Folgeblockantwort)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	nn	Nutzdatalängenbyte (dieses Telegramms)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	99	Befehlsbyte 1
CMD2	4	blknr	Befehlsbyte 2, Nummer des angeforderten Blocks
PAR-DATA	ab 5	##	Parameter-Daten
...
SYS-DATA	##	##	Systemparameter-Daten
...
CHKSUM	nn+5	##	Prüfsummenbyte

Erläuterung der Datenbereiche:

PAR-DATA:

Der Aufbau dieser Daten entspricht der Struktur „PARAM“ aus der Datei „\APPL\source2.h“. Die Daten sind auch in der Datei „\DEVELOP\ memio.dok“ dokumentiert.

SYS-DATA:

Die Angaben entsprechen der Struktur „SYSTEM“ aus der Datei „\APPL\source2.h“. Außerdem sind die Werte auch in der Datei „\DEVELOP\ memio.dok“ aufgelistet.

3.5 Parameterdaten setzen

Dieser Befehl ermöglicht das gezielte Manipulieren von ausgewählten Parameter-Daten auf dem Target. Über den Offset und der Länge kann der Startpunkt sowie die Anzahl der zu manipulierenden Bytes vorgegeben werden. Diese Angaben beziehen sich auf die jeweils in der „memio.dok“ angegebenen Daten. (Es kann immer nur der Wert eines Elements gesetzt werden.)

Host → Target (Parameter –setzen Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	10+nn	Kommando-/Nutzdatalängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	8F	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
Offset (long)	5-8	## ## ## ##	32Bit-Zahl, die den Offset innerhalb der Parameterblocks bestimmt
Wert (long)	9-12	## ## ## ##	32Bit-Zahl, die den zu setzenden Wert enthält
Länge (long)	13-16	## ## ## ##	32Bit-Zahl, die die Größe (Länge) des Wertes bestimmt (Byte = 1, Short = 2, Long = 4)
Länge der erweiterten Informationen (long)	17-20	nn nn nn nn (00 00 00 00)	Anzahl Bytes der erweiterten Informationen (im Normalfall = 0)
CHKSUM	nn+21	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

(keine Rückantwort)

3.6 Systemparameterdaten setzen

Dieser Befehl ermöglicht das gezielte Manipulieren eines ausgewählten Systemparameters auf dem Target. Über den Offset und der Länge kann der Startpunkt sowie die Byte-Anzahl des zu manipulierenden Wertes vorgegeben werden. Diese Angaben beziehen sich auf die jeweils in der „memio.dok“ angegebenen Einträge/Zeilen.

Host → Target (Systemparameter –setzen Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	10+nn	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	95	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
Offset (long)	5-8	## ## ## ##	32Bit-Zahl, die den Offset innerhalb der Systemparameterblocks bestimmt
Wert (long)	9-12	## ## ## ##	32Bit-Zahl, die den zu setzenden Wert enthält
Länge (long)	13-16	## ## ## ##	32Bit-Zahl, die die Größe (Länge) des Wertes bestimmt (Byte = 1, Short = 2, Long = 4)
Länge der erweiterten Informationen (long)	17-20	nn nn nn nn (00 00 00 00)	Anzahl Bytes der erweiterten Informationen (im Normalfall = 0)
CHKSUM	nn+21	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

(keine Rückantwort)

3.7 Prozeßdaten setzen

Dieser Befehl ermöglicht das gezielte Manipulieren von ausgewählten Prozeßdaten auf dem Target. Über den Offset und der Länge kann der Startpunkt sowie die Anzahl der zu manipulierenden Bytes vorgegeben werden. Diese Angaben beziehen sich auf die jeweils in der „memio.dok“ angegebenen Daten. (je Kommunikationszyklus kann immer nur der Wert **eines** Elements gesetzt werden.)

Host → Target (Prozeßdaten –setzen Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	0C	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	91	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
Offset (long)	5-8	## ## ## ##	32Bit-Zahl, die den Offset innerhalb des Prozeßdatenblocks bestimmt
Wert (long)	9-12	## ## ## ##	32Bit-Zahl, die den zu setzenden Wert enthält
Länge (long)	13-16	## ## ## ##	32Bit-Zahl, die die Größe (Länge) des Wertes bestimmt (Byte = 1, Short = 2, Long = 4)
CHKSUM	17	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

(keine Rückantwort)

3.8 Flagdaten setzen

Dieser Befehl ermöglicht das gezielte Manipulieren von ausgewählten Flagdaten auf dem Target (**VORSICHT - hiermit kann der Ablauf der Software extrem gestört werden**). Über den Offset und der Länge kann der Startpunkt sowie die Anzahl der zu manipulierenden Bytes vorgegeben werden. Diese Angaben beziehen sich auf die jeweils in der „memio.dok“ angegebenen Daten. (je Kommunikationszyklus kann immer nur der Wert **eines** Elements gesetzt werden.)

Host → Target (Prozeßdaten –setzen Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	0C	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	89	Befehlsbyte 1
CMD2	4	00	Befehlsbyte 2
Offset (long)	5-8	## ## ## ##	32Bit-Zahl, die den Offset innerhalb des Flagdatenblocks bestimmt
Wert (long)	9-12	## ## ## ##	32Bit-Zahl, die den zu setzenden Wert enthält
Länge (long)	13-16	## ## ## ##	32Bit-Zahl, die die Größe (Länge) des Wertes bestimmt (Byte = 1, Short = 2, Long = 4)
CHKSUM	17	##	Prüfsummenbyte

Host ← Target

0AAh (innerhalb von 0.2s, falls korrekte Prüfsumme ermittelt wurde)

(keine Rückantwort)

4. radCASE-Informationen nutzen

Zu jedem radCASE-Projekt wird eine Datei „memio.dok“ generiert, die Informationen über Messdaten (IOs), Arbeitsvariablen/Merker (Flags), feste Einstelldaten (Parameter), feste Daten (Processdaten) und feste Systemeinstelldaten (Systemparameter) enthält.

Die **E/A-, Flag- und Prozess-Daten** können komplett mit dem Kommando Statusblock anfordern (3.2) von der Steuerung geholt werden.

Die **Parameter- und Systemparameter-Daten** werden komplett mit Parameterblock anfordern (3.4) gelesen.

Wenn **nur bestimmte Werte oder Bereiche** von Interesse sind, dann besteht die Möglichkeit mit dem Kommando Statusdaten partiell anfordern (3.3) diese zu ermitteln.

In der „memio.dok“ sind folgende Abschnitte und Informationen enthalten:

- IOs:	Element	Offset	Length/Bit	Description	Unit/Selections	Format
- Flags:	Element	Offset	Length/Bit	Description	Unit/Selections	Format
- Parameter:	Element	Offset	Length/Bit	Description	Unit/Selections	Format
- Processdaten:	Element	Offset	Length/Bit	Description	Unit/Selections	Format
- Systempar.:	Element	Offset	Length/Bit	Description	Unit/Selections	Format

Element:

Der (radCASE-)Name für den Datenwert

Offset:

Die Lage des Datenwerts innerhalb seines spezifischen Blocks (IO/Flag/Process/Parameter/System)

Length/Bit:

Hier steht entweder die Länge/Größe des Datenwerts in Bytes oder die Bitnummer („Bit n“), falls es sich um einen binären Datenwert handelt.

Description:

Kurze Beschreibung des Elements

Unit/Selections:

Hat der Datenwert eine Einheit („Unit“, z.B. „°C“), dann ist der Text hier zu finden. Wenn der Zahlenwert eine bestimmte Bedeutung hat, dann gibt „Selections“ die dazugehörigen Auswahlen an → Bsp.: „0 = UNDEF, 1 = INAKTIV, 2 = AKTIV“.

Format:

Wenn es sich um einen Zahlenwert handelt, dann gibt „Format“ die Art der Festkommazahl (Digits + Dezimalen) an → Bsp.: 123.45 entspricht 6 Digits und 2 Dezimale, 9876 entspricht 4 Digits und 0 Dezimale. (*Festkommazahl: 123.45 wird als 12345 verwaltet.*)

4.1 Bestimmten Datenwert anfordern

Wenn man z.B. einen E/A-Wert wissen will, dann kann das folgendermaßen aussehen:

Beispiel: Projekt enthält einen analogen Eingang mit folgendem Eintrag in der „memio.dok“ im Abschnitt „IOs“:

Wasserkrei_Wassertemp_AI_TVorlaufRohEx (Element)
 32 (OFFSET→0020hex)
 2 (LENGTH/Bit)
 Vorlauf-Roh (Description)
 °C (Unit/Selections)
 6 Digits, 2 Dezimalen (Format)

Host → Target: (Statusdaten partiell anfordern-Kommando)

Name	Offset dez	Wert hex	Bedeutung
LEN	0	03	Kommando-/Nutzdatenlängenbyte
DST	1	01	Zieladresse
SRC	2	00	Quelladresse
CMD1	3	13	Befehlsbyte 1
CMD2	4	01	Befehlsbyte 2 tt = 01h: von IO-Daten tt = 02h: von Flag-Daten tt = 03h: von Parameter-Daten tt = 04h: von System-Daten tt = 05h: von Prozeß-Daten
OFFSET_L	5	20	Offset, niederwertiges Byte
OFFSET_H	6	00	Offset, höherwertiges Byte
LENGTH	7	02	Anzahl der rückzugebenden Bytes (< 240)
CHKSUM	8	70	Prüfsummenbyte

Host ← Target

Als Antwort bekommt man zuerst die von der Steuerung ermittelte Prüfsumme (also im Normalfall 0xAA) und danach dann die angeforderten Daten zurück.

Name	Offset dez	Wert hex	Bedeutung
LEN	0	02	Nutzdatenlängenbyte (diese wurden ja angefordert)
DST	1	00	Zieladresse
SRC	2	01	Quelladresse
CMD1	3	13	Befehlsbyte 1
CMD2	4	01	Befehlsbyte 2 (IO-Daten wurd angefordert)
REQ-DATA0	5	30	Low, angeforderte Datenbytes (Wasserkrei_Wassertemp_AI_TVorlaufRohEx)
REQ-DATA1	6	06	High, angeforderte Datenbytes (Wasserkrei_Wassertemp_AI_TVorlaufRohEx)
CHKSUM	2+5	5D	Prüfsummenbyte

Der Wert für Wasserkrei_Wassertemp_AI_TVorlaufRohEx ist also 0630hex (= 1584dez) bzw. im richtigen Format + Einheit = „015.84 °C“.

5. Implementierungsbeispiel in C

5.1 Kommando senden

```

int SendCommand(unsigned char src, unsigned char dst, unsigned char cmd1, unsigned 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; // Anzahl der Nutzdatenbytes
    buf[1] = dst;      // Ziel
    buf[2] = src;      // Quelle
    buf[3] = cmd1;     // Kommando 1
    buf[4] = cmd2;     // Kommando 2
    memcpy(&buf[5], data, data_len); // Nutzdaten in Puffer kopieren

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

    // Pruefsumme schreiben
    buf[i] = (unsigned char)(0xAA - sum);

    // Maximal 5mal versuchen das Kommando loszuwerden
    count = 0;
    do
    {
        int anz;

        // serielle Lese- und Schreibepuffer loeschen
        g_serial.Clear();

        // Gesamten Datenblock senden
        g_serial.Write(buf, data_len + 6, 100);

        // Empfangsantwort der Steuerung lesen (AAh = korrekt Empfangen), max. Response-Zeit abwarten
        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);
        }

        // Keine Antwort?
        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;
        }

        // Falsche Antwort erhalten?
        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) );

    // Nach x Versuchen Kommandoblock loszuwerden immer noch keine richtige Antwort?
    if ( rec_byte != 0xAA )
    {
        // Ja, also Fehler melden
        char szText[160];
        sprintf(szText, "SendCommand-Abbruch: Fehler bei Uebertragung: %d\n", (int)rec_byte);
        OutputDebugString(szText);
        return -1;
    }

    return 0;
}

```

5.2 Antwort empfangen

```

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

    sum = 0;
    int ret;

    // Anzahl der Nutzdatenbytes einlesen
    ret = g_serial.Read(&anz, sizeof(anz), 1000);
    if ( ret <= 0 )
    {
        OutputDebugString("ReceiveAnswer: Nichts empfangen.\n");
        return -1;
    }
    sum = (unsigned char)(sum + anz);

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

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

    // Kommando 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);

    // Kommando 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);

    // Eventuell extended Infos einlesen
    if ( g_cmd2 & 0x80 )
    {
        // Da Bit 7 in cmd2 gesetzt war, wird zusätzlich ein Byte mit der Art der nachfolgenden Info(s) gelesen
        if ( !g_serial.Read(&ext_info_types, sizeof(ext_info_types), TIMEOUT_PER_BYTE) )
        {
            OutputDebugString("ReceiveAnswer: Kein ext_info_types empfangen\n");
            return -1;
        }
        sum = (unsigned char)(sum + ext_info_types);

        // Entsprechend dem Typ/Bit die dazugehörigen erweiterten Infos einlesen
        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 Steuerung gesendet

    // Nutzdatenbytes einlesen
    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]);
    }

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

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

    return (int)anz;
}

```

5.3 Statusblock anfordern und empfangen

```

#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
// Konnte Anforderung abgesetzt werden?
if (ret >= 0)
{
    // Ja, also eventuelle Geraeteantwortverzoeigerung abwarten
    Sleep(15);

    int anz = cclReceiveAnswer(RecBuf);
    if (anz > 0)
    {
        // Kommunikationsversion 1 / neueste Kommunikation?
        if ( (RecBuf[0] == 0) && (RecBuf[1] == 1) )
        {
            if ( RecvMultiBlock(RecBuf) < 0 )
            {
                // Error
            }
        }
    }
}

int RecvMultiBlock(unsigned char* RecBuf)
{
    unsigned short IoToRecv, FlagToRecv, ParamToRecv, ProcToRecv, SystemToRecv, ToRecv;
    unsigned short IoRecvd=0, FlagRecvd=0, ParamRecvd=0, ProcRecvd=0, SystemRecvd=0;
    unsigned short* usBuf;
    unsigned char blk nr=1;
    unsigned char recvd=12; // CommVersion, IoToRecv, FlagToRecv, ParamToRecv, ProcToRecv, HMIToRecv ueberspringen ...
    unsigned long par_align_corr, proc_align_corr;
    /*** in paramLenSum werden die Längen der einzelnen parameterblöcke die aus den verschiedenen
    /*** knoten kommen summiert
    static short paramLenSum = 0;
    bool bParams = false;

    long IoOffset = 0, FlagOffset=0, ParamOffset=0, ProcOffset=0, SystemOffset=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];

    // Wenn Param-Daten kommen, dann steht in usBuf[4] die System-Datenlaenge (falls mitgesendet)
    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 )
    {
        // E/A-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[FlagRecvd+FlagOffset], &RecBuf[recvd], n);
    FlagRecvd += n;
    recvd += n;
}

// Parameter-Daten
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[ParamRecvd+ParamOffset], &RecBuf[recvd+corrOff], n - corrLen);
    ParamRecvd += n - corrLen;
    recvd += n;
}

// System-Daten
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[SystemRecvd+SystemOffset], &RecBuf[recvd+corrOff], n - corrLen);
    SystemRecvd += n - corrLen;
    recvd += n;
}

// Process-Daten
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;

// Status-Block anfordern und holen
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;
}

```