Modbus-Grundlagen

Camille Bauer AG
CH-5610 Wohlen

CAMILLE BAUER
Auf uns ist Verlass.

Einführung

Das Modbus-Protokoll wurde ursprünglich von der Firma Modicon (heute Schneider Electric) für den Datenverkehr mit ihren Controllern entwickelt. Daten wurden in Form von 16-Bit-Registern (Integer-Format) oder als Status-Informationen in Form von Datenbytes übertragen. Im Laufe der Zeit wurde das Protokoll erweitert und auch von anderen Herstellern für ihre Geräte übernommen. Neue Datentypen wurden hinzugefügt, insbesondere um mehr Auflösung für die übertragenen Werte zu erhalten. Das Protokoll wurde für neue Übertragungsmedien adaptiert, Dialekte wie Modbus Plus oder Modbus/TCP entstanden.

Der grundsätzliche Aufbau des Datenbereichs und die Adressierungs-Mechanismen wurden dabei aber aus Kompatibilitätsgründen immer beibehalten.

Das Modbus-Protokoll ist ein Single-Master Protokoll. Dieser Master steuert die gesamte Übertragung und überwacht eventuell auftretende Timeouts (keine Antwort vom adressierten Gerät). Die angeschlossenen Geräte dürfen nur nach Anforderung durch den Master Telegramme versenden.

Die nachfolgenden Grundlagen beschränken sich auf die Protokolle Modbus/RTU und Modbus/TCP. Beschrieben werden zudem nur die Funktionen, welche von Modbus-Geräten der Firma **Camille Bauer** unterstützt werden.

Inhaltsverzeichnis

1. Modbus/RTU-Protokoll

1.1 Übertragungs-Modus

Zeichenformat: Normalerweise programmierbar
1 Start-, 8 Daten-, 1 Stopbit, even parity
1 Start-, 8 Daten-, 1 Stopbit, odd parity
1 Start-, 8 Daten-, 2 Stopbit, no parity
1 Start-, 8 Daten-, 1 Stopbit, no parity (gebräuchlich, aber nicht gemäss MODBUS-Spezifikation)

Baudrate: Normalerweise programmierbar, gebräuchliche Werte sind
1200, 2400, 4800, 9600 und 19200 Bd

1.2 Allgemeine Form der Telegramme

Geräte-Adresse	Funktion	Daten	CRC-Check
8 Bits	8 Bits	n * 8 Bit	16 Bits

Gemäss MODBUS®-Spezifikation muss zwischen zwei Telegrammen eine Pause von mind. 3.5 Zeichen eingehalten werden. Innerhalb eines Telegramms dürfen die einzelnen Zeichen nicht mehr als 1.5 Zeichen Abstand aufweisen. Eine typische Übertragung sieht z.B. so aus:

usw.

Anmerkung: Die Überwachung der vorgegebenen Intervallzeiten durch den Master ist extrem schwierig, da insbesondere Windows-Betriebssysteme nicht für solche Rahmenbedingungen ausgelegt sind. In der Praxis werden deshalb sehr oft viel grössere Zeichenabstände toleriert. Dies kann aber zu Problemen bei der Geräteadressierung führen, da das Protokoll-Framing verloren gehen kann. Daten können so vom Empfänger irrtümlich als Telegrammanfang interpretiert werden.

Geräte-Adresse

Gibt an, welches Gerät angesprochen werden soll (Master→Slave) bzw. welches Gerät Antwort gibt (Slave→Master). Erlaubt sind bei Modbus die Adressen 1..247. Die Adresse 0 kann für Mitteilungen an alle Geräte (broadcast) verwendet werden, sofern die gewählte Funktion dies unterstützt.

Funktion

Gibt den Zweck der Datenübertragung an. Folgende Standard-Funktionen werden von CB-Geräten verwendet:

Code	MODBUS-Funktion	Register	Anwendungs-Beispiele
01_H	READ COIL STATUS	0xxxx	- Auslesen von Digitalausgangs-Zuständen
02_H	READ INPUT STATUS	1xxxx	- Auslesen von Digitaleingangs-Zuständen
03_H	READ HOLDING REGISTERS	4xxxx	- Auslesen von Messwerten, Zählerständen, Mittelwerten, - Auslesen der Geräte-Konfiguration
08_H	DIAGNOSTIC		- Geräte-Verbindungstest (Subfunktion 0)
$0F_H$	FORCE MULTIPLE COILS	0xxxx	- Setzen / Simulieren von Digitalausgangs-Zuständen
10_H	PRESET MULTIPLE REGISTERS	4xxxx	- Geräte-Programmierung

Daten

Enthält die zu übertragende Information. Dieses Feld wird unterteilt in Register, Anzahl zu übertragende Register und gegebenenfalls in ausgelesene oder abzuspeichernde Information. Daten werden normalerweise als Vielfaches von 16-Bit-Registern übertragen.

CRC-Check (Prüfwort)

Die CRC16-Checksumme wird über alle Bytes eines Telegramms berechnet. Sie wird vom Empfänger ebenfalls berechnet, um Übertragungsfehler feststellen zu können. Die Berechnung des CRC ist im Kapitel 1.5 beschrieben.

1.3 Datentypen

- Standardisierte Datentypen: **Byte** (8-Bit) und **Register** (16-Bit). Gemäss Modbus-Spezifikation wird bei einem Register immer zuerst das High-Byte, gefolgt vom Low-Byte übertragen.

- Erweiterte Datentypen: **32-Bit-Integer** und **32-Bit-Float** werden als 2 aufeinander folgende 16-Bit-Register übertragen. **64-Bit-Integer** und **64-Bit-Float** werden als 4 aufeinander folgende 16-Bit-Register übertragen. Das Format der Float-Zahl entspricht dem IEEE Standard 754. Nicht festgelegt ist die Übertragungs-Reihenfolge der Register. In den meisten Anwendungen ist sie jedoch wie folgt:

32-Bit-Zahlen

Reg_L (Bit 15..0)		*Reg_H (Bit 31..16)*	
HByte	LByte	HByte	LByte

64-Bit-Zahlen

Reg_L (15..0)		*Reg_H (31..16)*		*Reg_L (47..32)*		*Reg_H (63..48)*	
HByte	LByte	HByte	LByte	HByte	LByte	HByte	LByte
1.	2.	3.	4.	5.	6.	7.	8.

Reihenfolge

1.4 Adressierung der Daten

Modbus gruppiert verschiedenartige Datentypen nach Referenzen. Die Telegrammfunktionen 03_H und 10_H verwenden z.B. Register-Adressen ab 40001. Die Referenz 4xxxx ist dabei implizit, d.h. durch die verwendete Telegrammfunktion gegeben. Im Telegramm wird deshalb die 4 weggelassen und die Referenz in den Modbus-Beschreibungen zumeist nicht angegeben.

Speziell beim Modbus-Telegramm ist auch, dass die Nummerierung der Register bei 1, die Adressierung jedoch bei 0 beginnt. So wird also z.B. beim Lesen des Registers 40001 im Telegramm die Adresse 0 verwendet. Dies ist im Detail auch aus den Telegramm-Beispielen ersichtlich.

1.5 Berechnung des Prüfwortes (CRC16) *(Beispiel in 'C')*

Die Berechnung erfolgt über alle Zeichen des Telegramms mit Ausnahme des Prüfwortes. Das niederwertige Byte (Crc_LByte) wird an zweitletzter, das höherwertige Byte (Crc_HByte) an letzter Stelle im Telegramm eingesetzt. **ACHTUNG**: Dies ist verglichen mit der Übertragung von Datenregistern eine umgekehrte Reihenfolge.

Der Empfänger des Telegramms berechnet das Prüfwort erneut und vergleicht es mit dem empfangenen.

```
void main()
{
   unsigned char data[NUMDATA+2];                  // Telegrammbuffer
   unsigned char Crc_HByte,LByte;                  //
   unsigned int Crc;
   ....
   Crc=0xFFFF;
   for (i=0; i<NUMDATA; i++) {
     Crc = CRC16 (Crc, data[i] );
   }
   Crc_LByte = (Crc & 0x00FF);                     // Low-Byte bestimmen
   Crc_HByte = (Crc & 0xFF00) / 256;               // High-Byte bestimmen
}
// Berechnung CRC16
// ----------------
unsigned int CRC16(unsigned int crc, unsigned int data)
{
  const unsigned int Poly16=0xA001;
  unsigned int LSB, i;

  crc = ((crc^data) | 0xFF00) & (crc | 0x00FF);
  for (i=0; i<8; i++)
  {
    LSB=(crc & 0x0001);
    crc=crc/2;
    if (LSB)
      crc=crc^Poly16;
  }
  return(crc);
```

}

1.6 Fehlerbehandlung

Bei einem Übertragungsfehler, wenn also das vom Empfänger berechnete CRC16 nicht mit dem empfangenen übereinstimmt, so wird keine Quittierung an den Master gesendet und somit ein Timeout provoziert. Dasselbe geschieht, wenn ein nicht vorhandenes (oder ausgeschaltetes) Gerät adressiert wird.

Falls der Empfänger der Nachricht einen anderen Fehler feststellt, so sendet er eine entsprechende Fehlermeldung an den Master zurück.

Geräte-Antwort:

Adresse	Code	Daten	Checksumme	
			LByte	HByte
11_H	$Code+80_H$	***Fehlercode***	CRC16	

Der vom Gerät empfangene Funktions-Code wird zurückgeschickt. Es wird jedoch das höchstwertige Bit (MSB) gesetzt, um einen Fehler anzuzeigen. Folgende Fehlercodes können auftreten:

Fehlercode	Bedeutung
01_H	Verwendung eines nicht unterstützten Funktionscodes
02_H	Verwendung eines unerlaubten Speicherregisters: Ungültige Registeradresse verwendet oder Versuch auf eine schreibgeschützte Registeradresse zu schreiben.
03_H	Verwendung unerlaubter Datenwerte, z.B. eine falsche Anzahl Register.
06_H	Gerät kann Anfrage momentan nicht bearbeiten. Anfrage später wiederholen.

1.7 Telegramm-Beispiele

Funktion 01_H : READ COIL STATUS

Beispiel: Lesen der (Digital)-Ausgangszustände 2 bis 11 von Gerät 17. Das sind 10 Zustände, welche mit 2 Datenbytes abgebildet werden können.

Aufforderung Master->Slave	Adresse	Funktion	Daten				CRC-Check
			Startadresse		Anzahl Zustände		
	addr	01_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Antwort Slave->Master	Adresse	Funktion	Daten			CRC-Check
			Anz. Datenbytes	Zustand 9..2	Zustand 11..10	
	addr	01_H	8 Bit	8 Bit	8 Bit	crc16

```
Beispiel (Hex): >>>> 11 01 00 01 00 0A crc_l crc_h
                <<<< 11 01 02 11 01 crc_l crc_h
                11H=00010001B: Output 6,2 EIN; Output 9,8,7,5,4,3 AUS
                01H=00000001B: Output 10 EIN; Output 11 AUS
```

Anmerkung: Die Startadresse 2 wird gemäss MODBUS-Spezifikation als Register 1 adressiert

Funktion 02_H : READ INPUT STATUS

Beispiel: Lesen der (Digital)-Eingangszustände 4 bis 17 von Gerät 17. Das sind 14 Zustände, welche mit 2 Datenbytes abgebildet werden können.

Aufforderung Master->Slave	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Startadresse		**Anzahl Zustände**		
	addr	02_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Antwort Slave->Master	**Adresse**	**Funktion**	**Daten**			**CRC-Check**
			Anz. Datenbytes	**Zustand 11..4**	**Zustand 17..12**	
	addr	02_H	8 Bit	8 Bit	8 Bit	crc16

```
Beispiel (Hex): >>>> 11 02 00 03 00 0D crc_l crc_h
                <<<< 11 02 02 2D 3C crc_l crc_h
                2D_H=00101110_B: Input 9,7,6,5 EIN; Input 11,10,8,4 AUS
                3C_H=00111100_B: Input 17,16,15,14 EIN; Input 13,12 AUS
```

Anmerkung: Die Startadresse 4 wird gemäss MODBUS-Spezifikation als Register 3 adressiert

Funktion 03_H : READ HOLDING REGISTERS

Beispiel: Auslesen einer Float-Zahl (32-Bit) auf den Registeradressen 108 und 109 von Gerät 17

Aufforderung Master->Slave	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Startadresse		**Anzahl Register**		
	addr	03_H	High-Byte	Low-Byte	High-Byte	Low-Byte	crc16

Antwort Slave->Master	**Adresse**	**Funktion**	**Daten**		**CRC-Check**
			Anzahl Datenbytes	**Information**	
	addr	03_H	n (8 Bit)	n/2 Register	crc16

```
Beispiel (Hex):   >>>> 11 03 00 6B 00 02 crc_l crc_h
                  <<<< 11 03 04 CC CD 42 8D crc_l crc_h
```

Anmerkung: Die Startadresse 108 wird gemäss MODBUS-Spezifikation als Register 107 adressiert

Funktion 08_H : DIAGNOSTICS

Beispiel: Mit der Subfunktion 00 (Diagnose) wird getestet, ob das Gerät 17 angeschlossen ist. Das gesendete Telegramm wird 1:1 zurückgesendet.

Aufforderung Master->Slave	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Subfunktion		**Daten**		
	addr	08_H	0	0	High-Byte	Low-Byte	crc16

Antwort Slave->Master	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Subfunktion		**Information**		
	addr	08_H	0	0	High-Byte	Low-Byte	crc16

```
Beispiel (Hex):  >>>> 11 08 00 00 AA 55 crc_l crc_h
```

```
<<<< 11 08 00 00 AA 55 crc_l crc_h
```

Funktion $0F_H$: FORCE MULTIPLE COILS

Beispiel: Setzen der (Digital)-Ausgangszustände 30..46 von Gerät 17. Das sind 17 Zustände, welche mit 3 Datenbytes abgebildet werden können.

Aufforderung Master->Slave	**Adresse**	**Funktion**	**Daten**						**CRC-Check**
			Startadresse		**Anz. Zustände**		**Anz. Bytes**	**Information**	
	addr	$0F_H$	High	Low	High	Low	n	n Bytes	crc16

Antwort Slave->Master	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Startadresse		**Anzahl Zustände**		
	addr	$0F_H$	High	Low	High	Low	crc16

```
Beispiel (Hex):  >>>> 11 0F 00 1D 00 11 03 AC 38 01 crc_l crc_h
                 <<<< 11 0F 00 1D 00 11 crc_l crc_h
```

AC_H=10101100_B: Output 37,35,33,32 EIN; Output 36,34,31,30 AUS

38_H=00111000_B: Output 43,42,41 EIN; Output 45,44,40,39,38 AUS

01_H=00000001_B: Output 46 EIN;

Anmerkung: Die Startadresse 30 wird gemäss MODBUS-Spezifikation als Register 29 adressiert.

Funktion 10_H : PRESET MULTIPLE REGISTERS

Unterstützt Broadcast. Über Adresse 0 kann für alle Geräte gleichzeitig eine Aktion ausgeführt werden. Diese Art von Telegrammen wird nicht quittiert. Typische Anwendung: Setzen der Anzeigehelligkeit aller Geräte.

Beispiel: Setzen eines Long-Integers (32-Bit) auf den Registeradressen 302 und 303 von Gerät 17

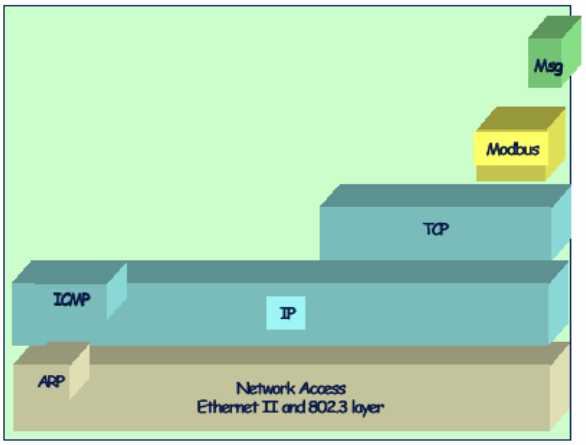
Aufforderung Master->Slave	**Adresse**	**Funktion**	**Daten**						**CRC-Check**
			Startadresse		**Anz. Register**		**Anz. Bytes**	**Information**	
	addr	10_H	High	Low	High	Low	n	n Bytes	crc16

Antwort Slave->Master	**Adresse**	**Funktion**	**Daten**				**CRC-Check**
			Startadresse		**Anzahl Register**		
	addr	10_H	High	Low	High	Low	crc16

```
Beispiel (Hex):  >>>> 11 10 01 2D 00 02 04 00 0A 01 02 crc_l crc_h
                 <<<< 11 10 01 2D 00 02 crc_l crc_h
```

Anmerkung: Die Startadresse 302 wird gemäss MODBUS-Spezifikation als Register 301 adressiert

2. Modbus/TCP-Protokoll



2.1 Allgemeine Form der Telegramme

Die ADU (Application Data Unit) des Modbus/TCP Protokolls setzt sich aus folgenden Blöcken zusammen

MBAP Header	Funktions-Code	Daten
7 Bytes	1 Byte	n Bytes

MPAP Header (Modbus Application Protocol Header)

Byte 0,1: transaction identifier - Identifikations-Nr. bei mehreren gleichzeitig aktiven Anfragen.

Byte 2,3: protocol identifier - immer 0 (Modbus Protokoll)

Byte 4: Anzahl nachfolgende Datenbytes (high byte) - immer 0 (da alle Mitteilungen kürzer als 256 Bytes sind)

Byte 5: Anzahl nachfolgende Datenbytes (low byte)

Byte 6: unit identifier (vorher 'Geräteadresse'). Da die Geräte direkt über die IP-Adresse angesprochen werden, hat dieser Parameter keine Funktion und sollte auf 0xFF gesetzt werden. Ausnahme: Bei einer Kommunikation via Gateway wird die Geräteadresse wie bisher gesetzt.

Funktions-Code

Byte 7: Funktions-Code des Standard MODBUS-Protokolls. Siehe Kapitel 1.2

Daten

Byte 8..n: Der Datenbereich entspricht demjenigen des Standard MODBUS-Protokolls (vgl. Kapitel 1). Die CRC-Prüfsumme entfällt jedoch, da sie auf TCP/IP-Protokollstufe implementiert ist.

2.2 Kommunikations-Management

Die Modbus-Kommunikation erfordert den Aufbau einer TCP-Verbindung zwischen einem Client (z.B. PC) und dem Server (Gerät). Für die Kommunikation wird normalerweise das für Modbus reservierte **TCP-Port 502** verwendet. Der Anwender kann jedoch auch eine andere Port-Nr. konfigurieren. Normalerweise sind Server dafür ausgelegt, dass nebst dem konfigurierten Port jederzeit auch eine zusätzliche Verbindung über Port 502 möglich ist.

Falls zwischen Server und Client eine Firewall angeordnet ist, muss sichergestellt werden, dass das konfigurierte TCP-Port freigeschalten ist.

Als Server kann auch ein Modbus RTU/TCP Gateway verwendet werden, an welchem bis zu 32 Geräte seriell angeschlossen werden können. Dies erlaubt ohne Änderung der Firmware jedes Modbus RTU-Gerät direkt ans Ethernet anzuschliessen. Bei dieser kostengünstigen Lösung wird aber die Übertragungsgeschwindigkeit auf die Baudrate des seriellen Buses reduziert.

2.3 Fehlerbehandlung

Bei einem Übertragungsfehler oder falls ein nicht vorhandenes (oder ausgeschaltetes) Gerät adressiert wird, sendet der Server keine Quittierung an den Client. Dies führt zu einem Timeout. Erfolgt die Kommunikation jedoch über ein Modbus RTU/TCP Gateway, erhält man von diesem eine Fehlermeldung, dass das angesprochene Gerät nicht antwortet.

Fehler werden vom Empfänger mit einer entsprechenden Fehlermeldung an den Master zurückgeschickt:

Geräte-Antwort:

MBAP Header	Funktions-Code	Daten
Kopie der Anforderung	Code+80_H	Fehlercode

Der empfangene Funktions-Code wird kopiert und das höchstwertige Bit (MSB) gesetzt. Der Fehlercode zeigt einen Bedienungs- bzw. Programmierfehler an. Folgende Fehlercodes sind unterstützt:

Fehlercode	Bedeutung
01_H	Verwendung eines nicht unterstützten Funktionscodes
02_H	Verwendung einer ungültigen Speicher-Adresse: Ungültige Registeradresse verwendet oder Versuch auf eine schreibgeschützte Registeradresse zu schreiben.
03_H	Verwendung unerlaubter Datenwerte, z.B. eine unerlaubte Anzahl Register.
06_H	Server busy (max. Anzahl gleichzeitiger Transaktionen erreicht).
$0B_H$	Fehlermeldung des Gateways: Keine Anwort vom adressierten Gerät.

2.4 Telegramm-Beispiele

Funktion 03_H : READ HOLDING REGISTERS

Beispiel: Auslesen einer Float-Zahl (32-Bit) auf den Registeradressen 108 und 109 von Gerät 17

Anforderung Client->Server

Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funktion	Daten			
								Startadresse		Anzahl Register	
0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	03_H	High-Byte	Low-Byte	High-Byte	Low-Byte

Antwort Server->Client

Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funktion	Daten	
								Anzahl Datenbytes	Information
0x00	***tno***	0x00	0x00	0x00	n+3	0xFF	03_H	n	n/2 Register

```
Beispiel (Hex)  >>>    00 00 00 00 00 06 FF 03 00 6B 00 02
                <<<    00 00 00 00 00 07 FF 03 04 CC CD 42 8D
```

Anmerkung: Die Register-Adresse 108 wird gemäss MODBUS-Spezifikation als Register 107 adressiert. Bei einer Kommunikation via Gateway muss der unit identifier auf die Geräteadresse (17) gesetzt werden.

tno = Identifikations-Nr. bei mehreren gleichzeitig aktiven Anfragen

Funktion 08_H : DIAGNOSTICS

Beispiel: Mit der Subfunktion 00 (Diagnose) wird getestet, ob das Gerät 17 angeschlossen ist. Das gesendete Telegramm wird 1:1 zurückgesendet.

Anforderung Client->Server	Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funktion	Daten			
									Sub-Funktion		Daten	
	0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	08_H	0x00	0x00	High-Byte	Low-Byte

Antwort Server->Client	Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funktion	Daten		
									Sub-Funktion	Daten	
	0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	03_H	n	High-Byte	Low-Byte

```
Beispiel (Hex) >>> 00 00 00 00 00 06 FF 08 00 00 AA 55
               <<< 00 00 00 00 00 06 FF 08 00 00 AA 55
```

Anmerkung: Bei einer Kommunikation via Gateway muss der unit identifier auf die Geräteadresse (17) gesetzt werden.

Funktion 10_H : PRESET MULTIPLE REGISTERS

Beispiel: Setzen eines Long-Integers (32-Bit) auf den Registeradressen 400 und 401 von Gerät 17

Anforderung Client->Server	Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funkt.	Daten					
									Startadr.		#Reg.		#Bytes	Info
	0x00	***tno***	0x00	0x00	0x00	n+7	0xFF	10_H	High	Low	High	Low	n	n Bytes

Antwort Server->Client	Transact. identifier		Protocol identifier		Anzahl Datenbytes		unit identifier	Funkt.	Daten			
									Startadresse		Anzahl Register	
	0x00	***tno***	0x00	0x00	0x00	0x06	0xFF	10_H	High-Byte	Low-Byte	High-Byte	Low-Byte

```
Beispiel (Hex) >>> 00 00 00 00 00 0B FF 10 01 8F 00 02 04 d2 d1 d4 d3
               <<< 00 00 00 00 00 06 FF 10 01 8F 00 02
```

Anmerkung: Die Register-Adresse 400 wird gemäss MODBUS-Spezifikation als Register 399 adressiert. Bei einer Kommunikation via Gateway muss der unit identifier auf die Geräteadresse (17) gesetzt werden.

tno = Identifikations-Nr. bei mehreren gleichzeitig aktiven Anfragen