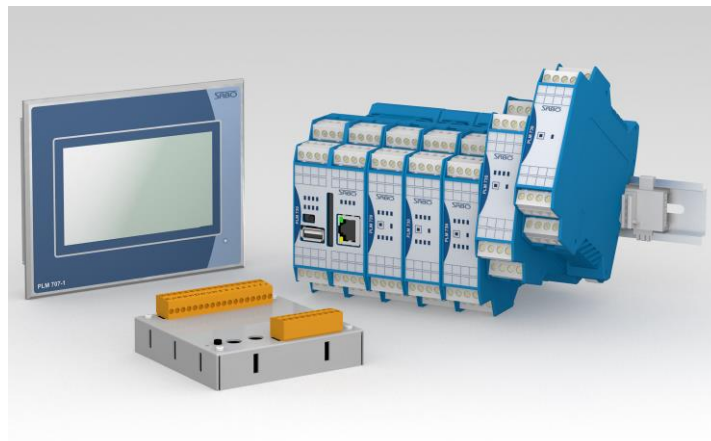


Systemfamilie PLM 700

Modulares CAN I/O-System



System-Handbuch

Teil 2

Anwendungsbeispiele und Bibliotheken für CoDeSys v2

SABO Elektronik GmbH
Lohbachstr. 14
58239 Schwerte
Tel. 02304 / 97102 - 0
Fax 02304 / 97102 - 22
E-Mail info@sabo.de
Internet www.sabo.de

Copyright © SABO Elektronik GmbH

Weitergabe oder Vervielfältigung dieses Dokuments ohne ausdrückliche schriftliche Genehmigung der SABO Elektronik GmbH nicht gestattet. Alle Rechte vorbehalten.

Haftungsausschluss

Der Inhalt des Dokuments wurde auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft; notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Verbesserungsvorschläge sind jederzeit willkommen.

SABO Elektronik GmbH
Lohbachstr. 14
58239 Schwerte
Tel. 02304 / 97102 - 0
Fax 02304 / 97102 - 22

E-Mail info@sabo.de
Internet www.sabo.de

Letzte Aktualisierung: 06. Jun. 2023

Inhaltsverzeichnis

Seite

1. WICHTIGE HINWEISE	9
1.1. BESTIMMUNGSGEMÄÙE VERWENDUNG.....	9
1.2. URHEBERSCHUTZ.....	9
1.3. PERSONALQUALIFIKATION.....	9
1.4. SICHERHEITSHINWEISE.....	9
2. SERIELLE SCHNITTSTELLEN (RS232/RS485)	10
2.1. ALLGEMEINES.....	10
2.2. INITIALISIERUNG DER EINGEBAUTEN SCHNITTSTELLEN (COM 0 ... 3).....	10
2.2.1. COMMSETPARAM (PLM_STD.LIB).....	10
2.2.2. PROGRAMMBEISPIEL ZUR INITIALISIERUNG DER COM-PORTS 0 ... 3.....	11
2.3. INITIALISIERUNG DER VIRTUELLEN USB-SCHNITTSTELLE (COM 8).....	12
2.4. INITIALISIERUNG DER VIRT. SCHNITTST. FÜR CAN-ERWEITERUNGSMODULE (COM 10 ... 25).....	13
2.4.1. SIM_730_COM (SIM_COM.LIB).....	13
2.5. VERWENDUNG DER SERIELLEN SCHNITTSTELLEN.....	14
2.5.1. COMMOUT (PLM_STD.LIB).....	14
2.5.2. COMMREAD (PLM_STD.LIB).....	14
2.6. DAUER DER DATENÜBERTRAGUNG.....	15
2.7. PROGRAMMBEISPIELE (ST).....	16
2.7.1. BEISPIEL SENDBYTES (ST).....	16
2.7.2. BEISPIEL SENDSTRING (ST).....	17
2.7.3. BEISPIEL RECEIVEBYTES (ST).....	17
2.7.4. BEISPIEL RECEIVESTRING (ST).....	18
2.8. PROGRAMMBEISPIELE (FUP).....	19
2.8.1. BEISPIEL SENDBYTES (FUP).....	19
2.8.2. BEISPIEL RECEIVEBYTES (FUP).....	20
3. CAN-SYSTEMBUS	22
4. DATEIEN SCHREIBEN UND LESEN	23
4.1. ALLGEMEINES.....	23
4.2. BENÖTIGTE BIBLIOTHEKEN.....	23
4.3. LAUFWERKE DER STEUERUNG.....	23
4.4. FTP-SERVER.....	24
4.5. VORGEHENSWEISE ZUM SCHREIBEN EINER DATEI.....	24
4.6. VORGEHENSWEISE ZUM LESEN EINER DATEI.....	25
4.7. BEARBEITUNGSGESCHWINDIGKEIT.....	26
4.8. PARAMETER-DATEIEN LESEN UND SCHREIBEN.....	26
4.8.1. CSVREAD (PLM_CSV.LIB).....	26
4.8.2. CSVWRITE (PLM_CSV.LIB).....	27
4.9. PROGRAMMBEISPIEL PARAMETERDATEI LESEN/SCHREIBEN (ST).....	28
5. ETHERNET EINSTELLUNGEN ABFRAGEN UND ÄNDERN	34
5.1. ALLGEMEINES.....	34
5.2. BENÖTIGTE BIBLIOTHEKEN.....	34

5.3.	VORGEHENSWEISE	34
5.4.	AUSFÜHRUNGSDAUER DER SET-BAUSTEINE.....	35
5.5.	VERWENDUNG DER VISU-BIBLIOTHEK	36
5.5.1.	BLÖCKE DER BIBLIOTHEK PLM_NETSETTINGSVISU.LIB.....	36
5.5.2.	PROGRAMMBEISPIEL ZUR BIBLIOTHEK PLM_NETSETTINGSVISU.LIB (ST).....	36
6.	VARIABLENAUSTAUSCH MITTELS UDP	38
6.1.	ALLGEMEINES	38
6.2.	CODESYS-NETZWERKVARIABLEN.....	38
6.2.1.	CODESYS-NETZWERKVARIABLEN, VORTEILE UND NACHTEILE	38
6.2.2.	CODESYS-NETZWERKVARIABLEN, PRINZIP.....	38
6.2.3.	CODESYS-NETZWERKVARIABLEN, EINRICHTUNG	39
6.2.4.	CODESYS-NETZWERKVARIABLEN, ALLGEMEINE HINWEISE	41
6.2.5.	CODESYS-NETZWERKVARIABLEN, ÜBERWACHEN DER VERBINDUNG.....	42
6.2.6.	CODESYS-NETZWERKVARIABLEN, FEHLERSUCHE.....	43
6.3.	PLM-BIBLIOTHEK UDP-COM.....	43
6.3.1.	PLM-BIBLIOTHEK UDP-COM, VORTEILE UND NACHTEILE	43
6.3.2.	PLM-BIBLIOTHEK UDP-COM, PRINZIP.....	43
6.3.3.	PLM_NETVARSEND (PLM_UDP.COM.LIB).....	44
6.3.4.	PLM_NETVARRECV (PLM_UDP.COM.LIB).....	46
6.3.5.	PLM-BIBLIOTHEK UDP-COM, ALLGEMEINE HINWEISE.....	47
6.3.6.	PLM-BIBLIOTHEK UDP-COM, DATENKONSISTENZ UND MTU	48
6.3.7.	PLM-BIBLIOTHEK UDP-COM, FEHLERSUCHE	48
7.	UNIVERSELLER TCP-SERVER	50
7.1.	ALLGEMEINES	50
7.2.	SPEZIFIKATION	50
7.3.	BENÖTIGTE BIBLIOTHEKEN.....	50
7.3.1.	TCP_COM9_WORK (PLM7XX_02.LIB)	50
7.4.	VERWENDUNG DES TCP-SERVERS.....	51
7.5.	FEHLERBEHANDLUNG	51
7.6.	BEISPIEL FÜR TCP-SERVER COM 9 (ST)	52
8.	UNIVERSELLER TCP-CLIENT.....	53
8.1.	ALLGEMEINES	53
8.2.	SPEZIFIKATION	53
8.3.	BENÖTIGTE BIBLIOTHEKEN.....	53
8.3.1.	TCPCLIENT2 (PLM_TcpCLIENT2.LIB)	53
8.3.2.	TCPCLIENT2TRANSMITDATA (PLM_TcpCLIENT2.LIB).....	55
8.3.3.	TCPCLIENT2RECEIVEDATA (PLM_TcpCLIENT2.LIB).....	56
8.4.	BEISPIEL FÜR TCP-CLIENT (ST).....	57
9.	MODBUS RTU (SLAVE).....	58
9.1.	ALLGEMEINES	58
9.2.	SPEZIFIKATION	58
9.3.	BENÖTIGTE BIBLIOTHEKEN.....	58
9.3.1.	PLM_MODBUSRTUSLAVE (PLM_MODBUSRTUSLAVE.LIB)	58
9.3.2.	PLM_MODBUSRTUSLAVE2 (PLM_MODBUSRTUSLAVE.LIB)	59
9.4.	PROGRAMMBEISPIEL (FUP)	61
9.5.	PROGRAMMBEISPIEL MIT DATENSEGMENTIERUNG (ST).....	62
10.	MODBUS RTU (MASTER).....	64

10.1. ALLGEMEINES	64
10.2. SPEZIFIKATION	64
10.3. BENÖTIGTE BIBLIOTHEKEN	64
10.3.1. PLM_MODBUSMASTERINIT (PLM_MODBUSRTU.LIB)	65
10.3.2. PLM_MODBUSMASTERTRANSFER (PLM_MODBUSRTU.LIB)	65
10.4. ABLAUF IM NORMALBETRIEB UND IM FEHLERFALL	67
10.5. PROGRAMMBEISPIEL (ST)	68
10.6. PROGRAMMBEISPIEL (FUP)	69
11. MODBUS TCP (SERVER).....	71
11.1. ALLGEMEINES	71
11.2. SPEZIFIKATION	71
11.3. BENÖTIGTE BIBLIOTHEKEN	71
11.3.1. MODBUSSERVER_TCPIP (PLM_MODBUSTCP.SRV.LIB)	72
11.4. PROGRAMMBEISPIEL (ST UND FUP)	74
12. MODBUS TCP (CLIENT)	75
12.1. ALLGEMEINES	75
12.2. SPEZIFIKATION	75
12.3. BENÖTIGTE BIBLIOTHEKEN	75
12.3.1. PLM_MODBUSTCPCLIENT (PLM_MODBUSTCPCLIENT.LIB)	76
12.3.2. PLM_MODBUSTCPCLIENTTRANSFER (PLM_MODBUSTCPCLIENT.LIB)	77
12.4. PROGRAMMBEISPIEL (FUP)	80
13. E-MAILS VERSENDEN.....	82
13.1. ALLGEMEINES	82
13.2. SPEZIFIKATION	82
13.3. BENÖTIGTE BIBLIOTHEKEN	82
13.3.1. PLM_SMTPINIT (PLM_MAIL_01.LIB)	82
13.3.2. PLM_SMTPSENDMAIL (PLM_MAIL_01.LIB)	83
13.4. DAUER DER DATENÜBERTRAGUNG	87
13.5. PROGRAMMBEISPIEL (FUP)	88
13.6. VERBINDUNGSPARAMETER EINIGER E-MAIL-ANBIETER	89
13.6.1. WEB.DE	89
13.6.2. T-ONLINE.....	90
13.6.3. GMX	90
13.6.4. OUTLOOK.....	90
13.7. FEHLERSUCHE	90
14. SMS VERSENDEN.....	92
14.1. ALLGEMEINES	92
14.2. SPEZIFIKATION	92
14.3. BENÖTIGTE BIBLIOTHEKEN	92
14.3.1. PLM_SMS_SEND.....	92
14.4. INITIALISIERUNG DER SCHNITTSTELLE	94
14.4.1. SERIELLE SCHNITTSTELLE RS232 (Z.B. FÜR SIM.730.34):	94
14.4.2. SIM.730.34 ÜBER VIRTUELLE SCHNITTSTELLE (CAN-BUS):	94
14.4.3. USB-MODEM ÜBER VIRTUELLE SCHNITTSTELLE (Z.B. FÜR SIM.730.36):	94
14.5. ZUSÄTZLICHE AT-KOMMANDOS	94
14.6. ZEICHENSATZ.....	95
14.7. FEHLERSUCHE	95

15. UHRZEIT UND DATUM	98
15.1. ALLGEMEINES	98
15.2. LOKALE ZEIT UND SOMMER-/WINTERZEITUMSCHALTUNG	98
15.3. BENÖTIGTE BIBLIOTHEKEN	99
15.3.1. PLM_ENABLESYSTIME (PLM_TIME.LIB)	99
15.3.2. PLM_GETRTC (PLM_TIME.LIB)	99
15.3.3. PLM_SETRTC (PLM_TIME.LIB)	100
15.3.4. PLM_SETRTCDST (PLM_TIME.LIB)	101
15.3.5. PLM_GETSNTPTIME (PLM_TIME.LIB)	102
15.3.6. PLM_CALCDST (PLM_TIME.LIB)	103
15.3.7. PLM_FORMATTIME (PLM_TIME.LIB)	104
15.3.8. PLM_FORMATTIMEEXT (PLM_TIME.LIB)	104
15.3.9. PLM_SETTIMEZONE (PLM_TIME.LIB)	105
15.3.10. PLM_GETTIMEZONE (PLM_TIME.LIB)	108
15.3.11. PLM_GETUTCTIME (PLM_TIME.LIB)	108
15.4. PROGRAMMBEISPIELE (FUP UND ST)	109
15.5. VERWENDUNG DER VISU-BIBLIOTHEK	111
15.5.1. BLÖCKE DER BIBLIOTHEK PLM_TIMEVISU.LIB	112
15.5.2. PROGRAMMBEISPIEL ZUR BIBLIOTHEK PLM_TIMEVISU.LIB (ST)	112
16. M-BUS.....	114
16.1. ALLGEMEINES	114
16.2. SPEZIFIKATION	114
16.3. VORGEHENSWEISE	114
16.4. PLMMBUSVALUE	116
16.5. BENÖTIGTE BIBLIOTHEKEN	116
16.5.1. PLMMBUS_ELECTRICITY, ..._GAS, ..._HEAT, ..._WATER, ..._DIGITALIN (PLM_MBUS.LIB)	116
16.5.2. PLMMBUS_UNIVERSALSLAVE (PLM_MBUS.LIB)	119
16.5.3. PLMMBUSVALUECONVERT (PLM_MBUS.LIB)	121
16.5.4. PLMMBUSVALUETODWORD (PLM_MBUS.LIB)	123
16.6. PROGRAMMBEISPIEL (FUP)	123
16.7. FEHLERSUCHE	126
17. KNX/EIB.....	128
17.1. ALLGEMEINES	128
17.2. SPEZIFIKATION	128
17.3. KNX-ADRESSEN UND -TELEGRAMME	128
17.3.1. PHYSIKALISCHE ADRESSEN.....	128
17.3.2. GRUPPENADRESSEN.....	129
17.3.3. TELEGRAMMINHALT	129
17.3.4. ÜBERTRAGUNG	129
17.4. BIBLIOTHEK KNX_SIM73027.LIB	129
17.4.1. PLMKNX_SIM73027 (KNX_SIM73027.LIB)	130
17.4.2. PLMKNX_OBJBOOL (KNX_SIM73027.LIB).....	132
17.5. PROGRAMMBEISPIEL (FUP)	133
18. FTP-CLIENT.....	135
18.1. ALLGEMEINES	135
18.2. VORGEHENSWEISE	135
18.3. SPEZIFIKATION	136
18.4. BENÖTIGTE BIBLIOTHEKEN	136
18.4.1. PLM_FTPCLIENT (PLM_FTPCLIENT.LIB)	136
18.4.2. PLM_FTPCLIENTVISUTASK (PLM_FTPCLIENT.LIB).....	139
18.4.3. PLM_FTPCHDIR (PLM_FTPCLIENT.LIB).....	139

18.4.4. PLM_FTPMKDIR (PLM_FTPCLIENT.LIB).....	140
18.4.5. PLM_FTPRMDIR (PLM_FTPCLIENT.LIB)	141
18.4.6. PLM_FTPDELETE (PLM_FTPCLIENT.LIB)	141
18.4.7. PLM_FTPRENAME (PLM_FTPCLIENT.LIB).....	142
18.4.8. PLM_FTPGET (PLM_FTPCLIENT.LIB).....	143
18.4.9. PLM_FTPLIST (PLM_FTPCLIENT.LIB).....	144
18.4.10. PLM_FTPPUT (PLM_FTPCLIENT.LIB).....	145
18.4.11. PLM_FTPQUIT (PLM_FTPCLIENT.LIB).....	146
18.4.12. PLM_FTPCONFIG (PLM_FTPCLIENT.LIB)	146
18.5. PROGRAMMBEISPIEL (FUP)	148
18.6. FEHLERSUCHE	148

19. MYSQL DATENBANK-CLIENT 150

19.1. ALLGEMEINES	150
19.2. SPEZIFIKATION	150
19.3. BENÖTIGTE BIBLIOTHEKEN.....	150
19.3.1. PLM_MYSQL_SIMPLE_QUERY (PLM_MYSQL_I.LIB).....	151
19.3.2. PLM_MYSQL_DB_OPEN (PLM_MYSQL_I.LIB)	153
19.3.3. PLM_MYSQL_DB_CLOSE (PLM_MYSQL_I.LIB).....	154
19.3.4. PLM_MYSQL_DB_QUERY (PLM_MYSQL_I.LIB)	154
19.4. ERGEBNIS-ARRAY	155
19.5. HILFSFUNKTIONEN ZUM ERSTELLEN DES QUERY-STRINGS	157
19.5.1. PLM_MYSQL_APPEND_DINT().....	158
19.5.2. PLM_MYSQL_APPEND_DWORD()	158
19.5.3. PLM_MYSQL_APPEND_REAL().....	158
19.5.4. PLM_MYSQL_APPEND_STRING().....	158
19.5.5. PLM_MYSQL_APPEND_STRINGP().....	158
19.5.6. PLM_MYSQL_APPEND_STRING_QUOTE()	158
19.5.7. PLM_MYSQL_APPEND_STRINGP_QUOTE()	158
19.6. PROGRAMMBEISPIEL (ST)	158
19.7. FEHLERSUCHE UND MYSQL-KONSOLE	159

20. GATEWAY-MODUL SIM.730.99 AUTOMOTIVE-CAN..... 163

20.1. ALLGEMEINES	163
20.2. SPEZIFIKATION	163
20.3. FUNKTIONSWEISE.....	163
20.3.1. SENDEN VON CAN-MESSAGES AUF DEN AUTOMOTIVE-CAN	163
20.3.2. EMPFANG VON CAN-MESSAGES VOM AUTOMOTIVE-CAN	164
20.4. BENÖTIGTE BIBLIOTHEKEN.....	164
20.4.1. ÜBERSICHT ÜBER DIE BIBLIOTHEKSBAUSTEINE	165
20.4.2. PLM_SIM73099	165
20.4.3. PLM_SIM73099_RxMsgID.....	167
20.4.4. PLM_SIM73099_RxMsgIDMASK.....	168
20.4.5. PLM_SIM73099_RxMsgAll	169
20.4.6. PLM_SIM73099_RxCfgFilter.....	170
20.4.7. PLM_SIM73099_RxCfgFilterMask.....	170
20.4.8. PLM_SIM73099_TxMsgSingle	171
20.4.9. PLM_SIM73099_TxMsgCyclic.....	172
20.5. PROGRAMMBEISPIEL.....	172
20.6. FEHLERSUCHE	172

21. SNMP 174

21.1. ALLGEMEINES	174
21.2. SPEZIFIKATION	174
21.3. INSTALLATION AUF DER STEUERUNG.....	175
21.4. KONFIGURATION DES SNMP-AGENTS.....	175

21.4.1. PLM_SNMP_INIT()	175
21.5. ZUGRIFFSRECHTE	176
21.5.1. SNMP V2C COMMUNITIES	176
21.5.2. SNMP V3 USER BASED SECURITY MODEL	177
21.6. SNMP-DATENOBJEKTE	177
21.6.1. PLM_SNMP_BASEOID().....	178
21.6.2. PLM_SNMP_DATALINE()	178
21.6.1. PLM_SNMP_DATALINESTRING().....	179
21.6.2. PLM_SNMP_DATALINEREAL().....	180
21.7. PROGRAMMBEISPIEL.....	181
22. OPC-UA SERVER.....	186
22.1. ALLGEMEINES	186
22.2. SPEZIFIKATION	186
22.3. BENÖTIGTE BIBLIOTHEKEN	187
22.4. INSTALLATION AUF DER STEUERUNG.....	187
22.5. KONFIGURATION DES OPC-UA-SERVERS	187
22.5.1. PLM_OPCUA_CONFIG_OPEN()	188
22.5.1. PLM_OPCUA_CONFIG_CLOSE().....	188
22.5.2. PLM_OPCUA_CONFIG_PORT().....	189
22.5.1. PLM_OPCUA_CONFIG_USER().....	189
22.5.2. PLM_OPCUA_CONFIG_FOLDER()	189
22.5.3. PLM_OPCUA_CONFIG_INT() ETC.....	190
22.5.1. PLM_OPCUA_CONFIG_INT_MINMAX() ETC.	191
22.5.2. PLM_OPCUA_CONFIG_STRING()	192
22.5.3. PLM_OPCUA_SERVER_MESSAGE().....	192
22.6. PROGRAMMBEISPIEL.....	193
23. FEHLERTAGEBUCH.....	195
24. KURVENDARSTELLUNG (TRENDS).....	196
25. TARGET-VISUALISIERUNG	197
25.1. ALLGEMEINES	197
25.2. LINIENZÜGE UND POLYGONE	197
25.2.1. GEFÜLLTE FLÄCHEN	197
25.3. BITMAPS	197
25.3.1. DATEINAME	197
25.3.2. FARBEN UND FARBPALETTE	197
25.3.3. SKALIERUNG UND ABSCHNEIDEN	198
25.3.4. BITMAPS MITTELS STRING-VARIABLE UMSCHALTEN	198
25.4. ZEICHENSATZGRÖßEN	198
25.5. SYSTEMZEICHENSATZ UND EXTERNE FONTS	200
25.5.1. SYSTEMZEICHENSATZ	200
25.5.2. EXTERNER FONT	201
25.6. SPRACHUMSCHALTUNG	202
25.6.1. STATISCHE SPRACHUMSCHALTUNG (SPRACHDATEI, *.VIS).....	202
25.6.2. DYNAMISCHE TEXTE (*.XML)	204

1. Wichtige Hinweise

1.1. Bestimmungsgemäße Verwendung

Die Anwendungsgebiete der SABO PLM Baureihe erstrecken sich von der Regelungs- und Steuerungstechnik über die Gebäudeautomation bis zur industriellen Nutzung in der Automatisierung. In allen Anwendungsbereichen ist darauf zu achten, dass die maximalen Spannungen, die auf dem technischen Datenblatt genannten Höchstgrenzen nicht überschritten werden. Änderungen sind nur im Rahmen des Handbuchs genannten Möglichkeiten zulässig.

Insbesondere ist die Verwendung von SABO-Produkten nicht zulässig für: Überwachung von Kernreaktionen in Kernkraftwerken, Flugleitsysteme, Flugsicherung, Steuerung von Massentransportmitteln, medizinische Lebenserhaltungssysteme, Steuerung von Waffensystemen.

1.2. Urheberrecht

Dieses Handbuch sowie alle dazugehörigen Bilder sind Eigentum der SABO Elektronik GmbH und sind urheberrechtlich geschützt. Eine Vervielfältigung, Veränderung oder Veräußerung an Dritte ist nicht gestattet, es sei denn es liegt eine schriftliche Einverständniserklärung der SABO Elektronik GmbH vor. Zuwiderhandlungen ziehen rechtliche Gegenmaßnahmen nach sich.

Die in dieser Dokumentation beschriebenen Produkte und Funktionen können jederzeit den neusten technologischen Entwicklungen angepasst werden. Die gegebenen Informationen können somit nicht als Vertragsgegenstand angesehen werden.

1.3. Personalqualifikation

SABO Produkte dürfen ausschließlich von Fachkräften mit einer Ausbildung in der SPS-Programmierung oder Elektrofachkräften mit einer Unterweisung in den dafür geltenden spezifischen Normen angeschlossen und gewartet werden.

Für Fehler und Schäden, die an SABO Produkten und/oder Fremdprodukten entstehen, die auf die Missachtung der Handhabung zurückzuführen sind, übernimmt die SABO Elektronik GmbH keine Haftung.

1.4. Sicherheitshinweise

Die in dieser Dokumentation gemachten Sicherheitshinweise erheben keinen Anspruch auf Vollständigkeit. Bei Unklarheiten und der Möglichkeit einer potenziellen Gefährdung von Mensch und Maschine ist im Zweifelsfall der zuständige Distributor der SABO Elektronik GmbH hinzuzuziehen. Die in dieser Dokumentation gemachten Hinweise sind Vorschläge und müssen bei der Übertragung auf die jeweilige Anwendung auf deren Machbarkeit und Funktionsfähigkeit überprüft werden.

Im Allgemeinen dienen die Vorschriften nach VDE beim Umgang mit elektrischen Anlagen als Richtlinie.

2. Serielle Schnittstellen (RS232/RS485)

2.1. Allgemeines

Die Steuerungen der Serie PLM 700 verfügen über eine oder mehrere eingebaute serielle Schnittstellen vom Typ RS232 oder RS485. Bei Bedarf können weitere serielle Schnittstellen in Form von Erweiterungsmodulen über den CAN-Systembus angeschlossen werden. Eine serielle Schnittstelle wird auch als COM-Port bezeichnet.

Die seriellen Schnittstellen sind gemäß folgendem Schema durchnummeriert:

COM 0 ... 3	Eingebaute serielle Schnittstellen (falls vorhanden)
COM 8	Virtuelle serielle Schnittstelle USB-Adapter
COM 9	Virtuelle serielle Schnittstelle TCP-Server (siehe Abschnitt 7)
COM 10 ... 25	Virtuelle serielle Schnittstellen CAN-Erweiterungsmodule

Erläuterungen zu den Schnittstellenstandards RS232 und RS485 finden sich z.B. im Internet unter

<http://de.wikipedia.org/wiki/Rs232> und

<http://de.wikipedia.org/wiki/RS485> .

Eine serielle Schnittstelle muss vor der Verwendung initialisiert werden. Hierfür stehen spezielle Bibliotheksfunktionen zur Verfügung, mit denen die Baudrate und andere Übertragungsparameter eingestellt werden. Die verfügbaren Parameter sind der jeweiligen Gerätedokumentation zu entnehmen.

Die erste serielle Schnittstelle COM 0 ist außerdem zunächst auf die serielle Kommunikation mit CoDeSys oder für den Betrieb mit einem Modem voreingestellt. Falls COM 0 für eigene Programmwzwecke verwendet werden soll, müssen die CoDeSys-Kommunikation und der Modembetrieb explizit abgeschaltet werden (siehe Abschnitt 2.2).

Nach erfolgter Initialisierung erfolgt der Programmzugriff auf alle seriellen Schnittstellen in gleicher Weise (siehe Abschnitt 2.5).

2.2. Initialisierung der eingebauten Schnittstellen (COM 0 ... 3)

Benötigt werden die folgenden Bibliotheken:

Plm_Std.lib
Modem.lib*

* bei Verwendung von COM 0.

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

2.2.1. COMMSETPARAM (PLM_Std.lib)

COMMSETPARAM	
comm_nr	: BYTE
mode	: BYTE
dat_len	: BYTE
stop_len	: BYTE
parity	: BYTE
baud	: DWORD
nc	: DWORD

Abb. 2-1: Funktion COMMSETPARAM (PLM_Std.lib)

Input-Parameter:		
comm_nr	BYTE	Nummer des COM-Ports (z.B. 0 für erste

		eingebaute serielle Schnittstelle)
mode	BYTE	Betriebsart des COM-Ports: 0 = Verwendung durch Anwenderprogramm 1 = Verwendung als CoDeSys-Kommunikationsschnittstelle
dat_len	BYTE	Anzahl der Datenbits pro Zeichen, 7 oder 8
stop_len	BYTE	Anzahl der Stopbits, 1 oder 2
parity	BYTE	0 = keine Parität (none) 1 = gerade Parität (even) 2 = ungerade Parität (odd)
baud	DWORD	Baudrate: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200
nc	DWORD	Div. Steuerbits: 0 = RS232-Standardbetrieb 16#80 = Sende-/Empfangsumschaltung für RS485 aktiv
Return-Wert:		
CommSetParam	BOOL	FALSE = Input-Parameter fehlerhaft

Bei Verwendung von COM 0 müssen die dort voreingestellte CoDeSys-Kommunikation und der automatische Modembetrieb explizit abgeschaltet werden. Dies geschieht, indem beide Funktionen einem anderen COM-Port zugewiesen werden; falls keine CoDeSys-Kommunikation über eine serielle Schnittstelle benötigt wird (z.B. bei Verwendung von Ethernet als Programmierschnittstelle), kann die Zuweisung auch an einen nicht vorhandenen COM-Port erfolgen (z.B. COM 255).

2.2.2. Programmbeispiel zur Initialisierung der COM-Ports 0 ... 3

Der in Abb. 2-2 und Abb. 2-3 dargestellte Funktionsblock ComInit wird vor der Verwendung von COM 0...3 im Programm aufgerufen. Bei zyklischem Aufruf sorgt ComInit selbst dafür, dass die Schnittstellen-Initialisierung nur einmal durchgeführt wird.

Die Reihenfolge der einzelnen Initialisierungsschritte sollte genau so gewählt werden, wie dargestellt.

```

ComInit (PRG-ST)
0001 PROGRAM ComInit
0002 VAR
0003     initDone: BOOL;
0004 END_VAR
0005
0001 IF NOT initDone THEN
0002     initDone := TRUE;
0003
0004     (* nur COM 0: Modem und CoDeSys-Kommunikation abschalten *)
0005     ModemIEC_online( Status := FALSE );
0006     ModemSetComm( Enable := TRUE, ComPort:=255 );
0007     CommSetParam( comm_nr := 255,
0008         mode := 1, dat_len := 8, stop_len := 1, parity := 0, baud := 9600, nc := 0
0009     );
0010
0011     (* COM-Port 0 für RS232 initialisieren *)
0012     CommSetParam( comm_nr := 0,
0013         mode := 0, dat_len := 8, stop_len := 1, parity := 0, baud := 9600, nc := 0
0014     );
0015
0016     (* COM-Port 1 für RS485 initialisieren *)
0017     CommSetParam( comm_nr := 1,
0018         mode := 0, dat_len := 8, stop_len := 1, parity := 0, baud := 9600, nc := 16#80
0019     );
0020 END_IF
    
```

Abb. 2-2: Programmbeispiel zur Initialisierung der COM-Ports 0 ... 8 (ST)

```

0001 PROGRAM ComInit
0002 VAR
0003     initDone: BOOL;
0004 END_VAR

0001
initDone ← Return

0002
TRUE ← initDone

0003
nur COM 0: Modem und CoDeSys-Kommunikation abschalten
FALSE ← Status
      MODEMIEC_ONLINE

0004
TRUE ← Enable
      255 ← ComPort
      MODEMSETCOMM

0005
255 ← comm_nr
1 ← mode
8 ← dat_len
1 ← stop_len
0 ← parity
9600 ← baud
0 ← nc
      COMMSETPARAM

0006
COM-Port 0 für RS232 initialisieren
0 ← comm_nr
0 ← mode
8 ← dat_len
1 ← stop_len
0 ← parity
9600 ← baud
0 ← nc
      COMMSETPARAM

0007
COM-Port 1 für RS485 initialisieren
1 ← comm_nr
0 ← mode
8 ← dat_len
1 ← stop_len
0 ← parity
9600 ← baud
16#80 ← nc
      COMMSETPARAM
  
```

Abb. 2-3: Programmbeispiel zur Initialisierung der COM-Ports 0 ... 8 (FUP)

2.3. Initialisierung der virtuellen USB-Schnittstelle (COM 8)

Die Steuerung erlaubt den Anschluss von Geräten, die als USB-Serial-Adapter arbeiten. Falls ein solches Gerät an den USB-Port angeschlossen ist, kann es über die virtuelle serielle Schnittstelle COM 8 angesprochen werden.

Die Initialisierung erfolgt genau wie bei den eingebauten seriellen Schnittstellen COM 0...3 mittels der Funktion `CommSetParam()` (siehe Abschnitt 2.2).

Entsprechende Geräte erscheinen intern auf der Steuerung meistens unter dem Gerätenamen `ttUSB0`, bevor sie intern der virtuellen Schnittstelle COM 8 zugewiesen werden können. Falls in Ausnahmefällen der interne Geräte name anders

lautet, kann dies vor dem Aufruf der Funktion `CommSetParam()` durch Aufruf der Funktion

```
SystemSetParameter( 10370, ADR(deviceName) );
```

konfiguriert werden. `deviceName` muss dabei ein String sein, der den gewünschten Namen enthält, z.B.

```
deviceName: STRING := 'ttyACM0';
```

Grundsätzlich gibt es Einschränkungen bezüglich der anschließbaren USB-Geräte. Im Zweifelsfall halten Sie bitte Rücksprache mit der SABO Elektronik GmbH.

2.4. Initialisierung der virt. Schnittst. für CAN-Erweiterungmodule (COM 10 ... 25)

Benötigt wird die folgende Bibliothek:

SIM_COM.lib

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

Bei Verwendung von seriellen Schnittstellen, die durch PLM700 CAN-Bus-Module (z.B. SIM.730.10) bereitgestellt werden, erfolgt die Initialisierung durch den Funktionsblock `SIM_730_COM`.

2.4.1. SIM_730_COM (SIM_COM.lib)

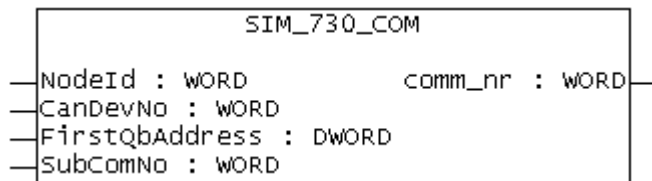


Abb. 2-4: Funktionsblock `SIM_730_COM` (SIM_COM.lib)

Input-Parameter:		
NodeId	WORD	CAN-NodeID des Moduls, wie unter <i>Ressourcen</i> → <i>Steuerungskonfiguration</i> → <i>SIM.730.10_v2</i> → <i>CAN Parameter</i> eingetragen, z.B. 2
CanDevNo	WORD	Nummer des zuständigen CAN-Masters, z.B. 0
FirstQbAddress	DWORD	Adresse des ersten %QBs des Moduls aus der Steuerungskonfiguration, z.B. %QB1.0.0
SubComNo	WORD	Schnittstellenauswahl, falls das Modul mehrere serielle Schnittstellen bietet, z.B. 0
Output-Parameter:		
comm_nr	WORD	Zugewiesene COM-Nummer der Schnittstelle, z.B. 10

Der Ausgabe-Parameter `comm_nr` erhält die zugewiesene COM-Nummer der Schnittstelle und muss bei folgenden Aufrufen von `CommOut()` und `CommRead()` verwendet werden.

Der Eingabe-Parameter `FirstQbAddress` muss aus der Steuerungskonfiguration abgelesen werden. Beispiel:

```

┌-----Can 0 Master [VAR]
├-----SIM.730.10_v2 (EDS) [VAR]
└-----%QB1.0 Can-Output

```

In diesem Fall ist `FirstQbAddress := ADR(%QB1.0)` anzugeben.

2.5. Verwendung der seriellen Schnittellen

Benötigt wird die folgende Bibliothek:

PIm_Std.lib

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

Für das Senden und das Empfangen auf einer seriellen Schnittstelle stehen u.a. die Funktionen `CommOut` und `CommRead` zur Verfügung.

2.5.1. CommOut (PIm_Std.lib)

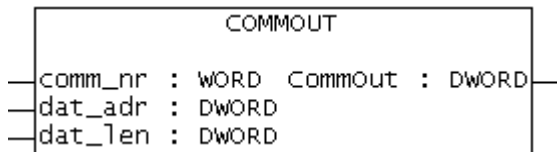


Abb. 2-5: Funktion `CommOut` (PIm_Std.lib)

Input-Parameter:		
comm_nr	WORD	Nummer des Com-Ports, z.B. 0 für die erste serielle Schnittstelle
dat_adr	DWORD	Adresse einer Variablen oder eines Arrays mit den zu sendenden Zeichen
dat_len	DWORD	Anzahl der zu sendenden Zeichen. Wenn <code>dat_len = 0</code> : Sende Inhalt eines Null-terminierten Strings
Return-Wert:		
CommOut	DWORD	Anzahl der von der Funktion tatsächlich entgegengenommenen Zeichen

Die Funktion `CommOut` übernimmt das Versenden von Zeichen über eine serielle Schnittstelle. Tatsächlich werden die Zeichen in einen internen Ringbuffer eingetragen, dessen Inhalt im Hintergrund durch die Steuerung versendet wird (vgl. Abschnitt 2.6, *Dauer der Datenübertragung*).

Der am Eingang `dat_adr` angegebene Datenspeicher (z.B. ein BYTE-Array) muss die durch `dat_len` bestimmte Anzahl an Zeichen bereitstellen. Die benötigte Speicheradresse wird üblicherweise mit der `ADR()`-Funktion ermittelt.

Wenn `dat_len` den Wert 0 hat, wird an `dat_adr` die Adresse einer String-Variablen erwartet. In diesem Fall werden soviele Zeichen versendet, wie der String momentan enthält. Das Versenden endet vor dem ersten Null-Zeichen im String, dies entspricht dem internen Speicherformat von CoDeSys.

2.5.2. CommRead (PIm_Std.lib)

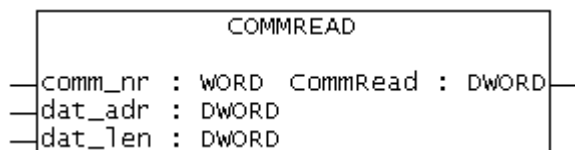


Abb. 2-6: Funktion `CommRead` (PIm_Std.lib)

Input-Parameter:		
comm_nr	WORD	Nummer des Com-Ports, z.B. 0 für die erste serielle

		Schnittstelle
dat_adr	DWORD	Adresse einer Variablen oder eines Arrays, in dem die Empfangszeichen gespeichert werden sollen
dat_len	DWORD	Maximale Anzahl der zu empfangenen Zeichen
Return-Wert:		
CommRead	DWORD	Anzahl der tatsächlich bei dat_adr gespeicherten Zeichen

Die Funktion `CommRead` liefert die von einer seriellen Schnittstelle empfangenen Zeichen.

Der am Eingang `dat_adr` angegebene Datenspeicher (z.B. ein BYTE-Array) muss die durch `dat_len` bestimmte Anzahl an Zeichen aufnehmen können. Die benötigte Speicheradresse wird üblicherweise mit der `ADR()`-Funktion ermittelt.

Bedingt durch die Dauer der Datenübertragung (vgl. Abschnitt 2.6) muss das Anwenderprogramm damit rechnen, die erwartete Anzahl von Zeichen nicht innerhalb eines einzigen Zyklus' von `CommRead` zu erhalten. Die übliche Vorgehensweise ist daher, die von `CommRead` erhaltenen Zeichen zunächst solange zu sammeln, bis die Datenübertragung vollständig ist und erst dann die Daten auszuwerten (siehe auch Programmbeispiele in Abschnitt 2.7 und 2.8).

Wenn die empfangenen Daten als STRING verarbeitet werden sollen, ist zu beachten, dass Strings in CoDeSys durch ein unsichtbares Null-Zeichen terminiert sind. Vor der Verarbeitung als String muss das Anwenderprogramm in jedem Fall sicherstellen, dass an die empfangenen Zeichen ein Null-Zeichen angehängt wird, da sonst die String-Länge undefiniert ist. Die Verarbeitung eines Strings mit undefinierter Länge kann zum Fehlverhalten des IEC-Programms oder sogar zum Absturz der Steuerung führen.

2.6. Dauer der Datenübertragung

Die Datenübertragung über RS232 oder RS485 erfolgt unsynchronisiert in Bezug auf die Zyklus- und Verarbeitungszeiten der Steuerung. Daher arbeiten sowohl Senden als auch Empfangen *gepuffert*, d.h. die Daten werden zunächst in Ringbuffer geschrieben, aus denen sie dann zur Verarbeitung ausgelesen werden.

Die Größe der Ringbuffer beträgt jeweils 5000 Zeichen.

Die Dauer der Datenübertragung muss vom Anwenderprogramm berücksichtigt werden:

- Beim Senden kann das Anwenderprogramm eine "große" Datenmenge (z.B. 500 Bytes) innerhalb eines Zyklus' zum Versenden bereitstellen. Die Sendefunktion `CommOut` schreibt diese Daten zunächst in einen Senderingbuffer. Die Steuerung sorgt dann im Hintergrund dafür, dass die einzelnen Zeichen mit größtmöglicher Geschwindigkeit über die serielle Schnittstelle übertragen werden. Diese Übertragung kann jedoch mehrere Zykluszeiten in Anspruch nehmen.
- Empfangsdaten stehen häufig nicht sofort oder zunächst unvollständig zur Verfügung. Das Anwenderprogramm muss dies entsprechend berücksichtigen und darf erst mit der Verarbeitung der Daten beginnen, wenn alle erwarteten Zeichen eingetroffen sind. Das Eintreffen aller Zeichen im Empfangsringbuffer kann dabei ebenfalls mehrere Zykluszeiten in Anspruch nehmen.
- Beim Empfang, insbesondere mit hohen Baudraten, werden pro Zyklus mehrere Zeichen empfangen. Bei großer Datenmengen müssen diese pro Zyklus vollständig abgearbeitet werden um einen Überlauf des Empfangsringbuffers zu vermeiden.

Die für eine Datenübertragung minimal benötigte Zeit (T_{min}) lässt sich aus der Baudrate ($Baudrate$), der Anzahl der Bits pro Zeichen ($AnzBits$) und der Anzahl der übertragenen Zeichen (Num) berechnen:

$$T_{min} = Num \times AnzBits / Baudrate$$

Die Anzahl der Bits pro Zeichen (*AnzBits*) berechnet sich aus 1 Startbit + Anzahl Datenbits + Anzahl Paritätbits + Anzahl Stopbits.

Beispiel 1: 200 Zeichen werden seriell übertragen mit 9600 Baud, 8 Datenbits, keine Parität, 1 Stopbit (9600, 8N1). Die minimale Übertragungszeit ist dann

$$T_{min} = 200 \times 10 / 9600 = 208,3 \text{ ms.}$$

Dies entspricht bei einer Zykluszeit von 20 ms ca. 11 Zyklen. Wenn die Steuerung der Empfänger ist, müssen pro Zyklus 200 Zeichen / 11 Zyklen \approx 19 Zeichen/Zyklus verarbeitet werden.

Beispiel 2: 5000 Zeichen werden seriell übertragen mit 115200 Baud, 8 Datenbits, keine Parität, 1 Stopbit (115200, 8N1). Die minimale Übertragungszeit ist dann

$$T_{min} = 5000 \times 10 / 115200 = 434,0 \text{ ms.}$$

Dies entspricht bei einer Zykluszeit von 20 ms ca. 22 Zyklen. Wenn die Steuerung der Empfänger ist, müssen pro Zyklus 5000 Zeichen / 22 Zyklen \approx 228 Zeichen/Zyklus verarbeitet werden.

2.7. Programmbeispiele (ST)

Die folgenden Programmbeispiele setzen voraus, dass die verwendete serielle Schnittstelle geeignet initialisiert ist (siehe Abschnitte 2.2, 7.1 und 2.3).

In den Beispielen werden die Sende- und Empfangsfunktionen `CommOut` und `CommRead` (siehe Abschnitte 2.5.1 und 2.5.2) verwendet.

2.7.1. Beispiel SendBytes (ST)

```

0001 PROGRAM sendBytes
0002 VAR
0003   txData: ARRAY[0..100] OF BYTE;
0004   txLen: DWORD;
0005   do_send: BOOL;
0006 END_VAR
0007
0001 (* Bytes senden *)
0002
0003 IF do_send THEN
0004   do_send := FALSE;
0005   txData[0] := 65; (* A *)
0006   txData[1] := 66; (* B *)
0007   txData[2] := 67; (* C *)
0008   txLen := 3;
0009   CommOut( comm_nr := 0, dat_adr := ADR(txData), dat_len := txLen );
0010 END_IF
0011
0012

```

Abb. 2-7: Senden von BYTE-Daten über COM 0 (ST)

Sobald `do_send` auf TRUE gesetzt ist, wird das BYTE-Array `txData[]` mit den drei Zeichen A – B – C (65 – 66 – 67) gefüllt und diese drei Zeichen über COM 0 übertragen.

2.7.2. Beispiel SendString (ST)

```

0001 PROGRAM sendString
0002 VAR
0003     txLine: STRING(100) := 'ABC';
0004     cr_lf: STRING(2) := '$R$N';
0005     do_send: BOOL;
0006 END_VAR
0007
0008
0009
0010
0011
0001 (* string senden mit Zeilenende = CR LF (ASCII 13, ASCII 10) *)
0002
0003 IF do_send THEN
0004     do_send := FALSE;
0005     CommOut( comm_nr := 0, dat_adr := ADR(txLine), dat_len := LEN(txLine) );
0006     CommOut( comm_nr := 0, dat_adr := ADR(cr_lf), dat_len := 2 );
0007 END_IF
0008
0009
0010
0011
  
```

Abb. 2-8: Senden eines Strings mit Zeilenendekennung über COM 0 (ST)

Sobald `do_send` auf `TRUE` gesetzt ist, wird der `STRING txLine` über `COM 0` übertragen. Anschließend wird im Beispiel noch eine Zeilenendekennung mit `CR LF` (ASCII 13 10) übertragen, anhand derer der Empfänger das Ende des Strings erkennen kann. Die Kennung muss jedoch zwischen Sender und Empfänger vereinbart werden.

2.7.3. Beispiel ReceiveBytes (ST)

```

0001 PROGRAM ReceiveBytes
0002 VAR
0003     rxData: ARRAY[0..100] OF BYTE;
0004     rxLen: DWORD;
0005     num: DWORD;
0006     wantedLen: DWORD := 10;
0007 END_VAR
0008
0009
0010
0011
0012
0013
0014
0015
0001 (* Bytes empfangen, Verarbeitung nach best. Anzahl *)
0002
0003 REPEAT
0004     num := CommRead( comm_nr := 0, dat_adr := ADR(rxData)+rxLen, dat_len := 1 );
0005     rxLen := rxLen + num;
0006 UNTIL num = 0 OR rxLen = wantedLen END_REPEAT
0007
0008 IF rxLen = wantedLen THEN
0009     (* ... Verarbeitung von rxLen Bytes in rxData[0..rxLen-1] ... *)
0010     rxLen := 0;
0011 END_IF
0012
0013
0014
0015
  
```

Abb. 2-9: Empfangen von BYTE-Daten über COM 0 (ST)

Im Beispiel wird eine feste Anzahl von Zeichen (hier: `wantedLen = 10`) von `COM 0` gelesen und in einem `BYTE-Array rxData[]` gespeichert. `rxLen` enthält stets die Anzahl der gültigen Zeichen in `rxData[]`.

Sobald 10 Zeichen vorhanden sind, erfolgt die Verarbeitung der Array-Daten innerhalb des `IF-Zweiges` (Zeile 9). Nach der Verarbeitung muss `rxLen` wieder auf Null gesetzt werden (Zeile 10).

Falls insgesamt mehr als 10 Zeichen empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. nach der Verarbeitung im IF-Zweig erneut Empfangsdaten abgerufen werden.

2.7.4. Beispiel ReceiveString (ST)

```

0001 PROGRAM ReceiveString
0002 VAR
0003     rxLine: STRING(100);
0004     lineLen: DWORD;
0005     rxPtr: POINTER TO BYTE;
0006     num: DWORD;
0007     rxChr: BYTE;
0008 END_VAR
0009
0010 (* string empfangen, verarbeitung nach zeilenende = CR LF (ASCII 13, ASCII 10) *)
0011 rxPtr := ADR(rxLine) + lineLen;
0012 WHILE TRUE DO
0013     num := CommRead( comm_nr := 0, dat_adr := ADR(rxChr), dat_len := 1 );
0014     IF num = 0 THEN
0015         rxChr := 0;
0016         EXIT;
0017     END_IF
0018     IF rxChr = 10 OR lineLen >= 100 THEN
0019         EXIT;
0020     END_IF
0021     IF rxChr <> 13 THEN
0022         rxPtr^ := rxChr;
0023         rxPtr := rxPtr + num;
0024         lineLen := lineLen + num;
0025     END_IF
0026 END_WHILE
0027 rxPtr^ := 0;
0028
0029 IF rxChr = 10 THEN
0030     (* ... verarbeitung der string-variablen "rxLine" ... *)
0031     lineLen := 0;
0032 END_IF

```

Abb. 2-10: Empfangen eines String über COM 0 (ST)

Das Beispiel implementiert vollständig den Empfang eines Strings, der mit einer Zeilenendekennung CR LF (ASCII 13 10) abgeschlossen ist. Der empfangene String wird ohne die Zeilenendekennung in der STRING-Variablen `rxLine` gespeichert.

Nach dem Empfang des letzten Zeichens (LF) erfolgt die String-Verarbeitung innerhalb des IF-Zweiges. Nach der Verarbeitung muss `lineLen` auf Null gesetzt werden.

Die String-Variable `rxLine` wurde mit `STRING(100)` deklariert und kann somit 100 Zeichen aufnehmen. Falls der empfangene String länger als 100 Zeichen ist, wird der Rest abgeschnitten. Die Verarbeitung erfolgt jedoch erst nach Empfang des LF-Zeichens.

Strings werden in CoDeSys intern durch ein unsichtbares Null-Zeichen terminiert. Eine mit `STRING(100)` deklarierte Variable bietet daher intern Platz für 101 Zeichen, um in jedem Fall das abschließende Null-Zeichen aufnehmen zu können. Wenn Empfangsdaten von der seriellen Schnittstelle als String verarbeitet werden sollen, ist vor der Verarbeitung das Null-Zeichen explizit anzuhängen. Dies geschieht im Beispiel in Zeile 19 durch `rxPtr^:=0`.

Falls mehrere Strings empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. nach der Verarbeitung im IF-Zweig erneut Empfangsdaten abgerufen werden.

2.8. Programmbeispiele (FUP)

Die folgenden Programmbeispiele setzen voraus, dass die verwendete serielle Schnittstelle geeignet initialisiert ist (siehe Abschnitte 2.2, 7.1 und 2.3).

Es werden die Sende- und Empfangsfunktionen `CommOut` und `CommRead` (siehe Abschnitte 2.5.1 und 2.5.2) verwendet.

Eine komplexere Datenverarbeitung in FUP wird nicht empfohlen, da dies wesentlich übersichtlicher in ST gelöst werden kann.

2.8.1. Beispiel SendBytes (FUP)

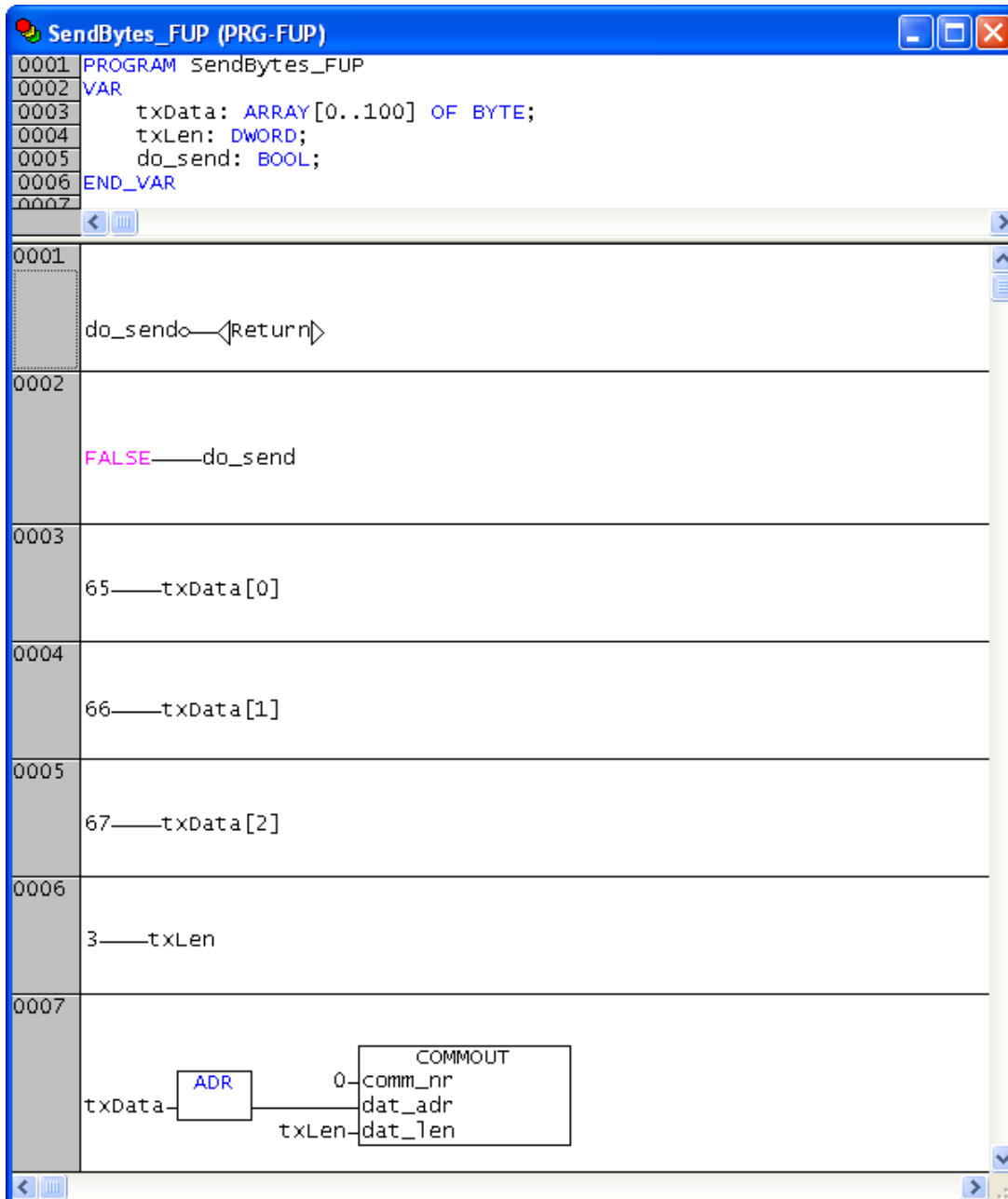


Abb. 2-11: Senden von BYTE-Daten über COM 0 (FUP)

Das Beispiel entspricht dem aus Abschnitt 2.7.1.

Sobald `do_send` auf `TRUE` gesetzt ist, wird das `BYTE`-Array `txData[]` mit den drei Zeichen A – B – C (65 – 66 – 67) gefüllt und diese drei Zeichen über `COM 0` übertragen.

2.8.2. Beispiel ReceiveBytes (FUP)

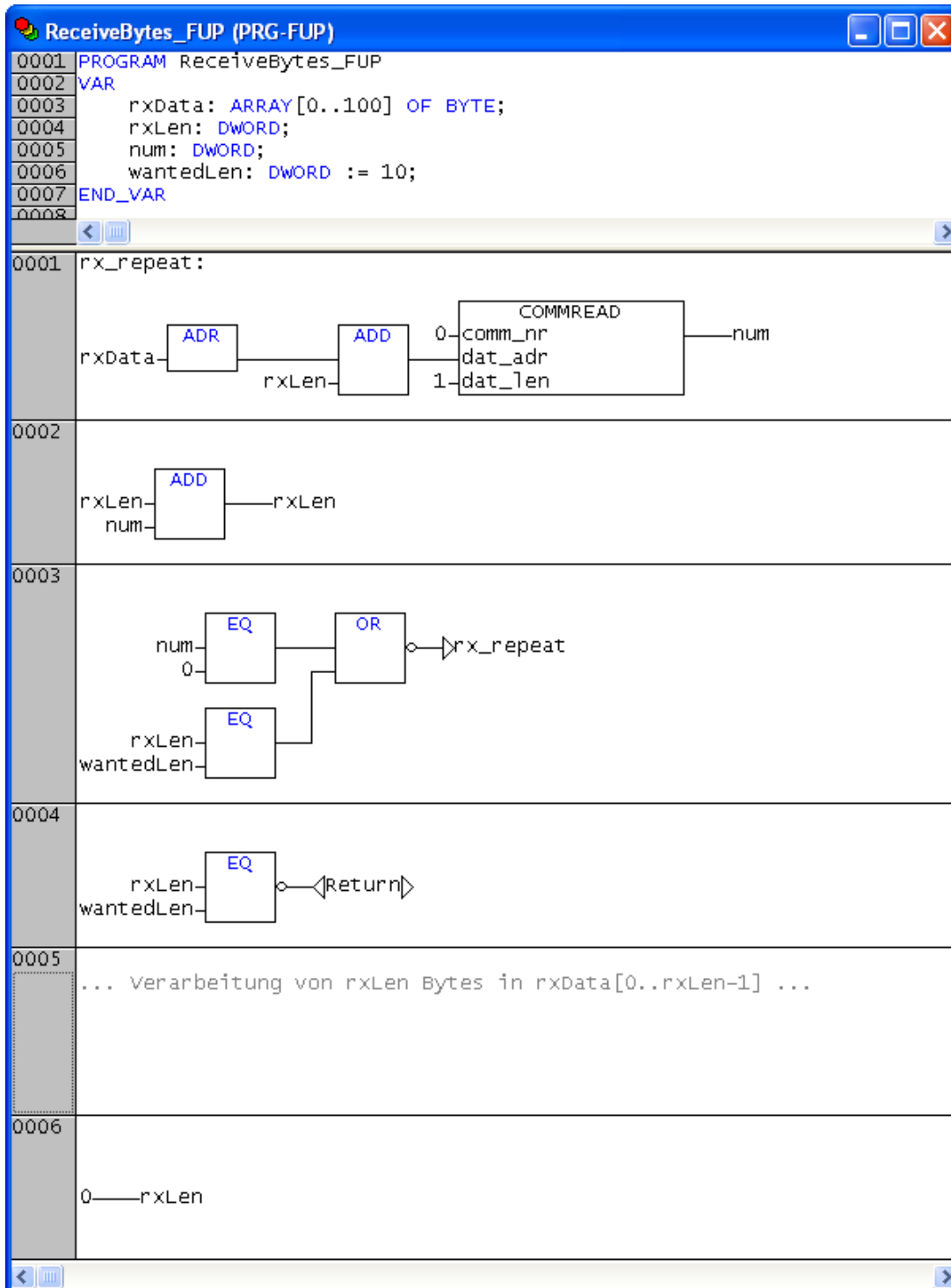


Abb. 2-12: Empfangen von BYTE-Daten über COM 0 (FUP)

Das Beispiel entspricht dem aus Abschnitt 2.7.3.

Es wird eine feste Anzahl von Zeichen (hier: wantedLen = 10) von COM 0 gelesen und in einem BYTE-Array rxData[] gespeichert. rxLen enthält stets die Anzahl der gültigen Zeichen in rxData[].

Sobald 10 Zeichen vorhanden sind, erfolgt die Verarbeitung der Array-Daten (Netzwerk 5). Nach der Verarbeitung muss rxLen wieder auf Null gesetzt werden (Netzwerk 6).

Falls insgesamt mehr als 10 Zeichen empfangen wurden, verbleiben die restlichen im Empfangs-Ringbuffer. Das Programm muss ggf. sicherstellen, dass durch einen Ringbuffer-Überlauf keine Zeichen verlorengehen, indem z.B. der Programmbaustein

mehrfach aufgerufen wird, bis keine Empfangsdaten in diesem Zyklus mehr vorliegen.

3. CAN-Systembus

[Dokumentation in Vorbereitung]

4. Dateien schreiben und lesen

4.1. Allgemeines

Häufig besteht der Wunsch, Daten eines Programms in eine Datei zu schreiben oder Daten aus einer Datei zu lesen. Eine Anwendung ist, Parameter eines Programms in einer Datei zu speichern oder aus einer Datei zurückzulesen, z.B. zur Archivierung der Parameter oder des Einlesens von "Rezepten".

Eine Datei kann auf einem internen Laufwerk der Steuerung oder auf einem USB-Stick gespeichert sein.

Das Dateiformat, in dem Daten gespeichert werden, kann grundsätzlich frei gewählt werden. Häufig bietet sich die Speicherung in einem zeilenorientierten ASCII-Format an, welches zu Kontrollzwecken direkt mit einem Texteditor gelesen oder bearbeitet werden kann.

4.2. Benötigte Bibliotheken

Benötigt werden die folgenden Bibliotheken:

PlmFile2_E.lib (oder spätere Version)
Plm_Std.lib
UPD_E_005.lib (oder spätere Version)
PLM_Csv.lib (nur für CSV-Dateien)
PLM_EasyFile.lib (alternativ)
PLM_IniFile.lib (alternativ)

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

4.3. Laufwerke der Steuerung

Steuerungen der Serie PLM700 verfügen über die folgenden Laufwerke:

Laufwerk	Größe	Beschreibung
a/	ca. 1 MB	RAM-Disk, Dateien werden bei Neustart gelöscht
b/	ca. 800 kB	Batteriegepufferte RAM-Disk, Dateien bleiben bei Neustart erhalten
c/	(abhängig von ext. Medium)	USB-Stick oder USB-Festplatte
p/	(ca. 1 MB)	(ab LZS v21305155) Dateien werden persistent im internen Flash-Speicher gespeichert, darf nur selten beschrieben werden (s.u.)

Laufwerk c/ steht erst nach dem Einstecken eines USB-Sticks bzw. einer USB-Festplatte zur Verfügung.

Laufwerk p/ kann erst ab LZS v21305155 genutzt werden.

Wichtig: Laufwerk p/ ist im internen Flash-Speicher der Steuerung realisiert. Die Anzahl der Schreibzyklen von Flash-Speicherbausteinen ist generell begrenzt und liegt bei einigen zehntausend. Danach sind Schäden am Flash-Speicher zu erwarten, dabei wird die Steuerung irreparabel beschädigt. Laufwerk p/ darf daher auf keinen Fall für Logger-Daten o.ä. verwendet werden und es darf auf keinen Fall zyklisch in das Laufwerk geschrieben werden. Das Laufwerk darf nur für die Ablage von selten

geänderten Parameter-Dateien verwendet werden, bei denen sichergestellt ist, dass der Schreibvorgang einzeln, z.B. durch eine manuelle Eingabe, ausgelöst wird.

Zum Einrichten des Laufwerks ist ein einmaliger Aufruf des Befehls

```
SystemSetParameter( 4500, nnn );
```

im IEC-Programm notwendig, z.B. im Baustein `InitOnce()`. Der Parameter `nnn` bestimmt die FTP-Zugriffsrechte:

```
SystemSetParameter( 4500, 3 ) → Laufwerk p/ einrichten,  
kein Zugriff über FTP möglich.
```

```
SystemSetParameter( 4500, 7 ) → Laufwerk p/ einrichten,  
Dateien über FTP zugreifbar.
```

4.4. FTP-Server

Auf die Laufwerke kann mittels des internen FTP-Servers der Steuerung zugegriffen werden. Dazu ist eine Ethernet-Verbindung notwendig und die IP-Adresse der Steuerung muss bekannt sein.

Der Zugriff auf den FTP-Server erfolgt z.B. durch einen PC mit einem FTP-Client-Programm. Unter Windows kann der standardmäßig vorhandene Windows-Explorer (Filemanager) als FTP-Client verwendet werden. Der Zugriff erfolgt über die folgende URL:

```
ftp://<ip-adresse>/
```

Anstelle von `<ip-adresse>` ist die vierstellige IP-Adresse der Steuerung einzusetzen, z.B. `ftp://10.1.1.231/`.

Bei der Verwendung von FTP-Client-Programmen ist darauf zu achten, dass die Datenübertragung im Binärmodus erfolgt, nicht im ASCII-Modus. Dies kann meistens im FTP-Client-Programm konfiguriert werden. Andernfalls besteht die Gefahr, dass Binärdateien fehlerhaft übertragen werden.

4.5. Vorgehensweise zum Schreiben einer Datei

Das Schreiben einer Datei im IEC-Programm geschieht grundsätzlich mit folgenden Befehlen aus der Bibliothek `PlmFile2_E.lib` (oder spätere Version):

```
VAR
    fid: DWORD;
    num: DINT;
    data: STRING := 'Hallo Welt$R$N';
END_VAR

fid := PLM_fileopen( FileName:='a/test.txt', FileFlag:='w+' );
if fid <> 0 then
    num := PLM_filewrite( file_id:=fid, Buffer:=ADR(data), Size:=LEN(data) );
    PLM_fileclose( file_id:=fid );
end_if
```

Der Befehl `PLM_fileopen()` öffnet die Datei `test.txt` auf Laufwerk `a/`. Die Angabe von `FileFlag:='w+'` erstellt eine leere Datei, egal ob diese neu oder bereits vorhanden ist. Falls stattdessen an eine vorhandene Datei angehängt werden soll, ist `FileFlag:='a+'` zu verwenden.

Im Erfolgsfall dient `fid` den folgenden Dateifunktionen als Referenz auf die geöffnete Datei. Falls die Datei nicht erstellt werden kann, hat der Rückgabewert `fid` den Wert 0; in diesem Fall dürfen keine weiteren Dateifunktionen mit `fid` ausgeführt werden.

Die vorgegebene Stringvariable `data` enthält eine ASCII-Zeichenkette, die mit '\$R\$N' (Zeilenumbruch CR-LF) abgeschlossen ist. Alternativ ist jeder andere Variablentyp denkbar, der auch Binärdaten enthalten kann.

Der Befehl `PLM_filewrite()` liefert in `num` die Anzahl der tatsächlich geschriebenen Zeichen zurück. Die Anzahl entspricht normalerweise dem bei `Size` angegebenen Wert, kann jedoch im Fehlerfall kleiner oder sogar Null sein. Dies wird in obigem Beispiel nicht überprüft.

Nach dem Schreiben muss die Datei mit `PLM_fileclose()` geschlossen werden; dadurch wird `fid` ungültig und darf nicht weiter verwendet werden.

In vielen Fällen können die komfortablen Funktionen der Bibliotheken `Plm_EasyFile.lib` oder `Plm_IniFile.lib` verwendet werden, die hier allerdings nicht beschrieben werden.

4.6. Vorgehensweise zum Lesen einer Datei

Das Lesen einer Datei im IEC-Programm geschieht grundsätzlich mit folgenden Befehlen aus der Bibliothek `PlmFile2_E.lib` (oder spätere Version):

```
VAR
    fid: DWORD;
    num: DINT;
    data: STRING;
    pBy: POINTER TO BYTE;
END_VAR

fid := PLM_fileopen( FileName:='a/test.txt', FileFlag:='r' );
if fid <> 0 then
    num := PLM_fileread( file_id:=fid, Buffer:=ADR(data), Size:=SIZEOF(data) );
    pBy := ADR(data) + num; (* nur für String-Data *)
    pBy^ := 0; (* nur für String-Data *)
    PLM_fileclose( file_id:=fid );
end_if
```

Der Befehl `PLM_fileopen()` öffnet die Datei `test.txt` auf Laufwerk `a/`. Die Angabe von `FileFlag:='r'` öffnet die Datei zum Lesen; dies setzt voraus, dass die Datei bereits vorhanden ist.

Im Erfolgsfall dient `fid` den folgenden Dateifunktionen als Referenz auf die geöffnete Datei. Falls die Datei nicht gelesen werden kann, hat der Rückgabewert `fid` den Wert 0; in diesem Fall dürfen keine weiteren Dateifunktionen mit `fid` ausgeführt werden.

Der Befehl `PLM_fileread()` erhält in `Buffer` die Adresse einer Variablen, in die die gelesenen Daten gespeichert werden. `Size` enthält in obigem Beispiel die Größe dieser Variablen, um einen Überlauf zu vermeiden. Es werden max. `Size` Bytes aus der Datei gelesen, jedoch weniger, falls die Datei kürzer ist. Der Rückgabewert in `num` enthält die Anzahl der tatsächlich gelesenen Zeichen; er ist beim Lesen normalerweise unbedingt auszuwerten.

Beim Einlesen von Strings ist zu beachten, dass ein ASCII-String aus einer Datei noch nicht dem internen Speicherformat von CoDeSys entspricht und daher zunächst nicht direkt weiterverarbeitet oder in der Visu angezeigt werden darf. Die Umwandlung erfolgt hier durch die beiden Zeilen mit der Variablen `pBy`. Das Einlesen von mehreren Strings mit unbekannter Länge aus einer Datei kann auf diese Weise *nicht* erfolgen; dies ist jedoch nicht Thema dieses Abschnitts.

Nach dem Lesen muss die Datei mit `PLM_fileclose()` geschlossen werden; dadurch wird `fid` ungültig und darf nicht weiter verwendet werden.

In vielen Fällen können die komfortablen Funktionen der Bibliotheken `Plm_EasyFile.lib` oder `Plm_IniFile.lib` verwendet werden, die hier allerdings nicht beschrieben werden.

4.7. Bearbeitungsgeschwindigkeit

Die zum Schreiben und Lesen notwendigen Dateioperationen benötigen in einigen Fällen beträchtliche Bearbeitungszeiten durch das Betriebssystem der Steuerung, z.B. kann das Speichern einer großen Datei auf einem USB-Stick mehrere Sekunden dauern, bedingt durch den Flash-Speicher auf dem USB-Stick.

Eine Dateioperation, die länger als ein Steuerungszyklus (20 ms) dauert, hält den Zyklus auf, so dass auch alle anderen Bearbeitungen des Zyklus' aufgehalten werden. Längere Dateioperationen sollten daher außerhalb des Steuerungszyklus' erfolgen.

Wir empfehlen, die entsprechenden Programmteile an den Visu-Task anzuhängen. In diesem Fall läuft die Visualisierung kurzzeitig langsamer ab, was meist nicht wahrgenommen wird, ohne den Zyklus zu stören.

Die Dateioperationen werden dann durch globale Variablen synchronisiert, die vom Zyklusprogramm gesetzt werden und im Visu-Task-Programm den gewünschten Schreib- oder Lese-Vorgang auslösen.

Diese Technik wird im Beispiel in Abschnitt 4.9 illustriert.

4.8. Parameter-Dateien lesen und schreiben

In diesem Abschnitt wird die Speicherung von Parametern als Binärdatei und als CSV-Datei behandelt.

Alternativ können in vielen Fällen die komfortablen Funktionen der Bibliotheken `Plm_EasyFile.lib` oder `Plm_IniFile.lib` verwendet werden, die hier allerdings nicht beschrieben werden.

Binärdateien bieten die kompakteste und einfachste Möglichkeit der Datenspeicherung, allerdings mit den Nachteilen, dass erstens die Parameter nicht im Klartext lesbar sind und zweitens eine nachträgliche Erweiterung der zu speichernden Daten Probleme bereiten kann.

CSV-Dateien (CSV = Comma Separated Value) sind ebenfalls einfach zu erzeugen, können im Klartext gelesen und sogar nachträglich mit einem Text-Editor bearbeitet werden. Die Dateien sind meist etwas größer als entsprechende Binärdateien. Die Werte werden hier im Klartext (ASCII-Format) mit einem definierten Trennzeichen gespeichert. Als Trennzeichen wird häufig ein Komma, ein Semikolon oder ein Tabulatorzeichen verwendet. Eine CSV-Datei könnte z.B. so aussehen:

```
Wert1 <TAB> Wert2 <TAB> Wert3 <CRLF>
Wert4 <TAB> Wert5 <TAB> Wert6 <CRLF>
Wert7 <TAB> Wert8 <TAB> Wert9 <CRLF>
```

Das aufwändigere XML-Format mit seinen Vor- und Nachteilen wird hier nicht behandelt.

Die Bibliothek `PLM_Csv.lib` bietet Funktionen zum Schreiben und Lesen von CSV-Dateien.

4.8.1. CsvRead (PLM_Csv.lib)

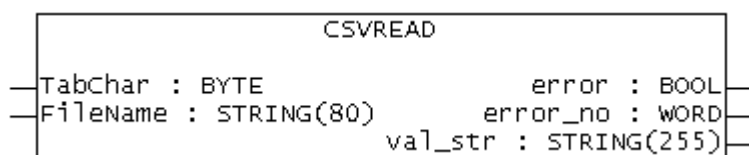


Abb. 4-1: Funktionsblock `CsvRead` (`PLM_Csv.lib`)

Input-Parameter:

TabChar	BYTE	ASCII-Code des Trennzeichens, z.B. 9 = Tabulator 44 = Komma 59 = Semikolon
FileName	STRING	Dateiname, z.B. 'a/test.dat' (RAM-Disk) oder 'c/test.dat' (USB-Stick)
Output-Parameter:		
error	BOOL	TRUE, wenn während des letzten Aufrufs ein Fehler aufgetreten ist
error_no	WORD	Nummer des letzten aufgetretenen Fehlers: 1 = Read Error 3 = Zeilenende anstelle Trennzeichen gefunden 4 = Trennzeichen anstelle Zeilenende gefunden 10 = Fehler beim Öffnen der Datei (z.B. Datei existiert nicht)
val_str	STRING	Zuletzt gelesener Wert als String

Der Funktionsblock `CsvRead` wird über seine Aktionen verwendet:

- `open_file` Öffnet die Datei `FileName` zum Lesen
- `close_file` Schließt die Datei
- `set_tabchar` Legt den ASCII-Code des Trennzeichens fest
- `read_val_tab` Liest einen Wert gefolgt von einem Trennzeichen, speichert den Wert als String in `val_str`
- `read_val_crlf` Liest einen Wert gefolgt von einem Zeilenende, speichert den Wert als String in `val_str`

Die gelesenen Werte im String-Format müssen ggf. über explizite Typumwandlungen in die gewünschte Werte-Formate umgewandelt werden, z.B. mittels `STRING_TO_BOOL()`, `STRING_TO_WORD()`, `STRING_TO_REAL()` etc.

4.8.2. CsvWrite (PLM_Csv.lib)

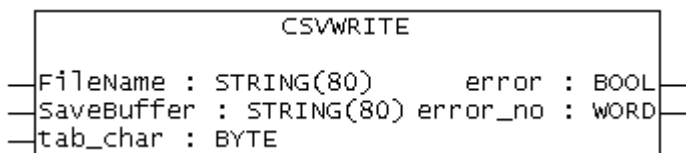


Abb. 4-2: Funktionsblock `CsvWrite` (PLM_Csv.lib)

Input-Parameter:		
FileName	STRING	Dateiname, z.B. 'a/test.dat' (RAM-Disk) oder 'c/test.dat' (USB-Stick)
SaveBuffer	STRING	Zu schreibender Wert als String
tab_char	BYTE	ASCII-Code des

		Trennzeichens, z.B. 9 = Tabulator 44 = Komma 59 = Semikolon
Output-Parameter:		
error	BOOL	TRUE, wenn während des letzten Aufrufs ein Fehler aufgetreten ist
error_no	WORD	Nummer des letzten aufgetretenen Fehlers: 10 = Fehler beim Schreiben der Datei (z.B. Laufwerk existiert nicht)

Der Funktionsblock `CsvWrite` wird über seine Aktionen verwendet:

- `open_file` Öffnet die Datei `FileName` zum Schreiben
- `close_file` Schließt die Datei
- `set_tabchar` Legt den ASCII-Code des Trennzeichens fest
- `write_val_tab` Schreibt den String aus `SaveBuffer`, gefolgt vom Trennzeichen
- `write_val_crlf` Schreibt den String aus `SaveBuffer`, gefolgt vom Zeilenende

Die zu schreibenden Werte müssen ggf. über explizite Typumwandlungen in das String-Format umgewandelt werden, z.B. mittels `BOOL_TO_STRING()`, `WORD_TO_STRING()`, `REAL_TO_STRING()` etc.

4.9. Programmbeispiel Parameterdatei Lesen/Schreiben (ST)

Zum Speichern eines Parametersatzes als Binärdatei bietet es sich an, alle Parameter zu einer `STRUCT` zusammenzufassen. Eine `STRUCT` ist eine fest definierte Zusammenfassung von verschiedenen Variablen, die in dieser Form zusammen als eine einzige Variable angesprochen werden können. `STRUCTS` werden unter `CoDeSys` im Reiter *Datentypen* angelegt und verwaltet.

Abb. 4-6 zeigt als Beispiel eine `STRUCT`, die einen `BOOL`-Wert, ein Array von vier `REAL`-Werten und ein `WORD` zu einem neuen Variablentyp `MACHINE_PARAM_STRUCT` zusammenfasst. Die globale Retain-Variablen `MachineParam` ist von diesem Typ und enthält den Parametersatz des Beispiels.

Der Wert `min_speed` innerhalb von `MachineParam` wird z.B. mit der Syntax `MachineParam.min_speed` angesprochen, der vierte `REAL`-Wert aus dem Array `heizung` mit `MachineParam.heizung[3]`.

Das Beispiel demonstriert sowohl den Umgang mit Binärdateien als auch den mit CSV-Dateien. In der Praxis wird meistens nur eine der beiden Möglichkeiten genutzt, so dass der dann überflüssige Programmcode entfallen kann.

Der Programmblock `FileVisuTask` wird in der in Abb. 4-4 gezeigten Form an den `VISU_TASK` von `CoDeSys` angehängt und übernimmt das Ausführen des gewünschten Lese- und Schreib-Vorgangs.

Die Globalen Variablen `Do_Read_Csv`, `Do_Write_Csv`, `Do_Read_Bin` und `Do_Write_Bin` lösen die jeweiligen Dateioperationen aus. Sie müssen vom Zyklusprogramm gesetzt werden und werden von `FileVisuTask` zurückgesetzt.

Eine Schreiboperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt zunächst `Write_Ok` und `Write_Err` auf `FALSE`
2. Das Zyklusprogramm setzt `Do_Write_Bin` bzw. `Do_Write_Csv` auf `TRUE`

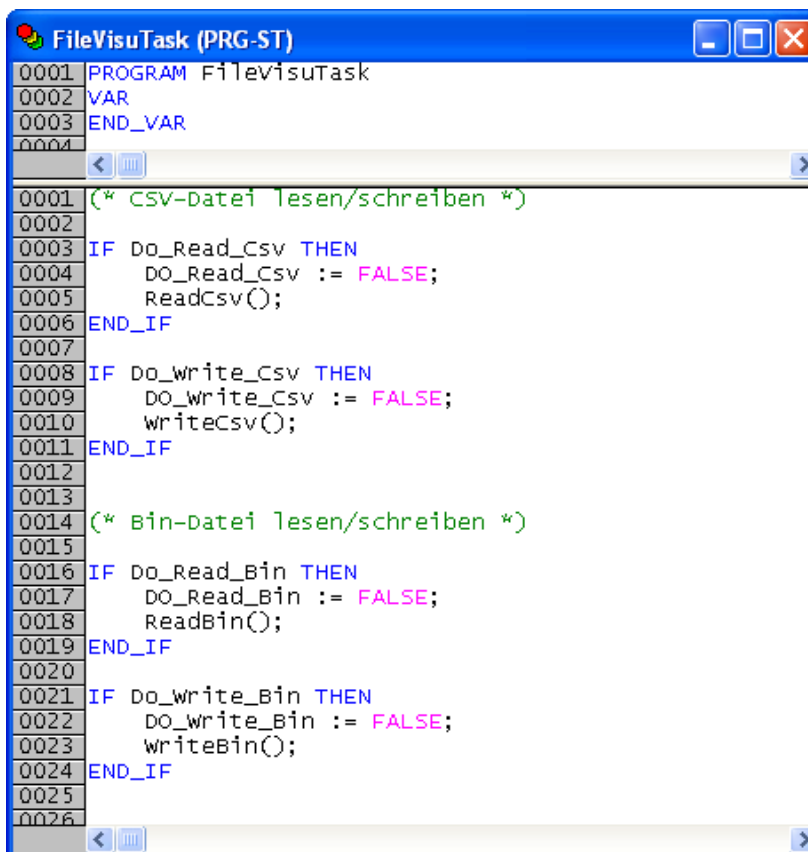
3. Der Programmblock `FileVisuTask` führt die gewünschte Schreiboperation aus und bestätigt das Ende durch Setzen von `Write_Ok` oder `Write_Err`.
4. Das Zyklusprogramm wartet, bis entweder `Write_Ok` oder `Write_Err` `TRUE` ist. Bei `Write_Ok = TRUE` wurde der Parametersatz erfolgreich in die Datei geschrieben.

Eine Leseoperation läuft wie folgt ab:

1. Das Zyklusprogramm setzt zunächst `Read_Ok` und `Read_Err` auf `FALSE`
2. Das Zyklusprogramm setzt `Do_Read_Bin` bzw. `Do_Read_Csv` auf `TRUE`
3. Der Programmblock `FileVisuTask` führt die gewünschte Leseoperation aus und bestätigt das Ende durch Setzen von `Read_Ok` oder `Read_Err`.
4. Das Zyklusprogramm wartet, bis entweder `Read_Ok` oder `Read_Err` `TRUE` ist. Bei `Read_Ok = TRUE` wurde der Parametersatz erfolgreich aus der Datei gelesen.

Während eine Lese- oder Schreiboperation läuft (d.h. gestartet und noch nicht mit `Ok` oder `Err` beendet wurde), dürfen die Daten in der `STRUCT`-Variablen `MachineParam` nicht durch das Zyklusprogramm verändert werden.

Die Schreib- und Leseprogrammblöcke prüfen bei Aufruf zunächst, ob die Datei auf dem USB-Stick (`c:/`) angelegt bzw. von diesem gelesen werden soll, und wenn ja, ob überhaupt ein USB-Stick vorhanden ist. Andernfalls kehren sie sofort mit Fehler zurück.



```

0001 PROGRAM FilevisuTask
0002 VAR
0003 END_VAR
0004
0001 (* CSV-Datei lesen/schreiben *)
0002
0003 IF Do_Read_Csv THEN
0004     DO_Read_Csv := FALSE;
0005     ReadCsv();
0006 END_IF
0007
0008 IF Do_write_Csv THEN
0009     DO_write_Csv := FALSE;
0010     writeCsv();
0011 END_IF
0012
0013
0014 (* Bin-Datei lesen/schreiben *)
0015
0016 IF Do_Read_Bin THEN
0017     DO_Read_Bin := FALSE;
0018     ReadBin();
0019 END_IF
0020
0021 IF Do_write_Bin THEN
0022     DO_write_Bin := FALSE;
0023     writeBin();
0024 END_IF
0025
0026

```

Abb. 4-3: Programmblock `FileVisuTask`, welcher an den `VISU_TASK` von CoDeSys angehängt wird

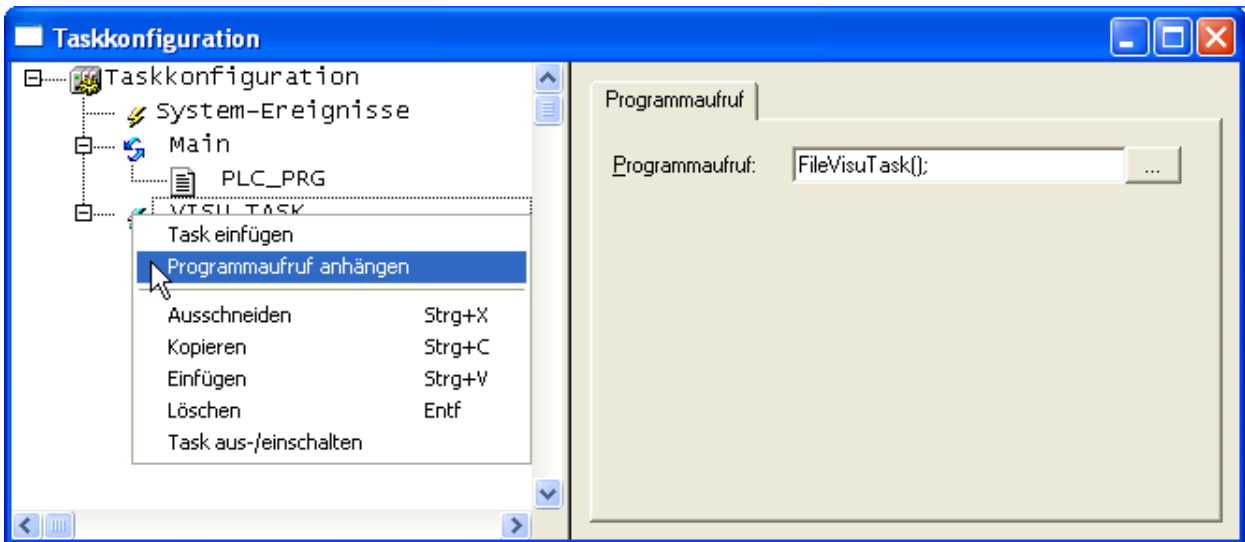


Abb. 4-4: Anhängen von FileVisuTask an den VISU_TASK von CoDeSys in der Taskkonfiguration

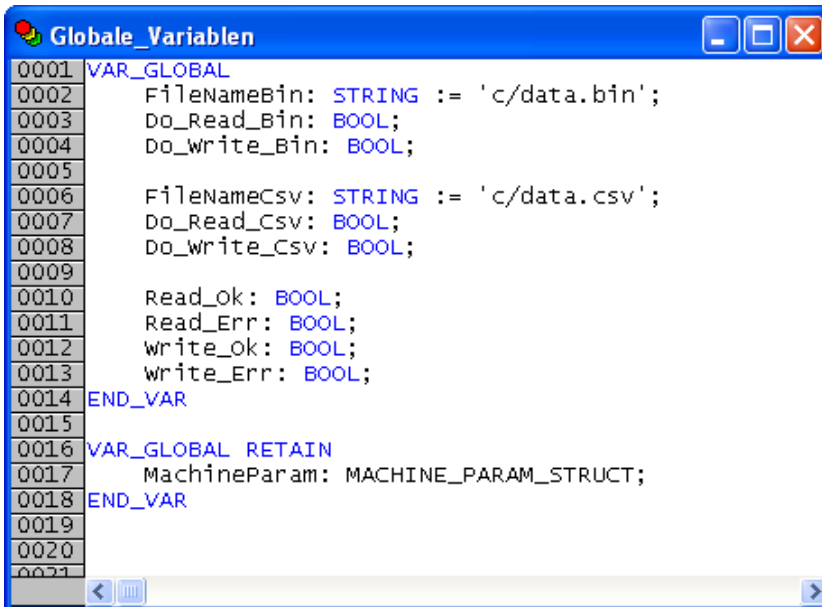


Abb. 4-5: Liste der Globalen Variablen; MachineParam enthält den Parametersatz des Beispiels.

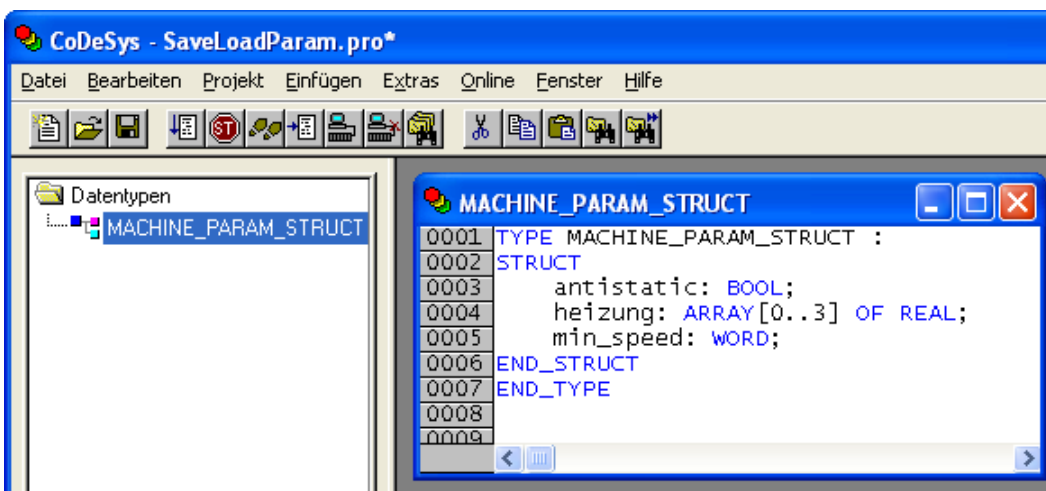


Abb. 4-6: Definition MACHINE_PARAM_STRUCT

```

0001 PROGRAM writeBin
0002 VAR
0003     fid: DWORD;
0004 END_VAR
0005
0006
0001 write_ok := FALSE;
0002 write_err := FALSE;
0003
0004 IF MID(FileNameBin,1,1) = 'c' AND NOT PLM_file_exist( FileName:='c/' ) THEN
0005     write_err := TRUE;
0006     RETURN; (* Error, USB-Stick nicht eingesteckt *)
0007 END_IF
0008
0009 fid := PLM_fileopen( FileName:=FileNameBin, FileFlag:='w+' ); (* File öffnen *)
0010 IF fid = 0 THEN
0011     write_err := TRUE;
0012 ELSE
0013     PLM_filewrite( (* MachineParam-struct als Block schreiben *)
0014         file_id:=fid,
0015         Buffer:=ADR(MachineParam),
0016         size:=SIZEOF(MachineParam)
0017     );
0018     PLM_fileclose( file_id:=fid );
0019     write_ok := TRUE;
0020 END_IF
0021

```

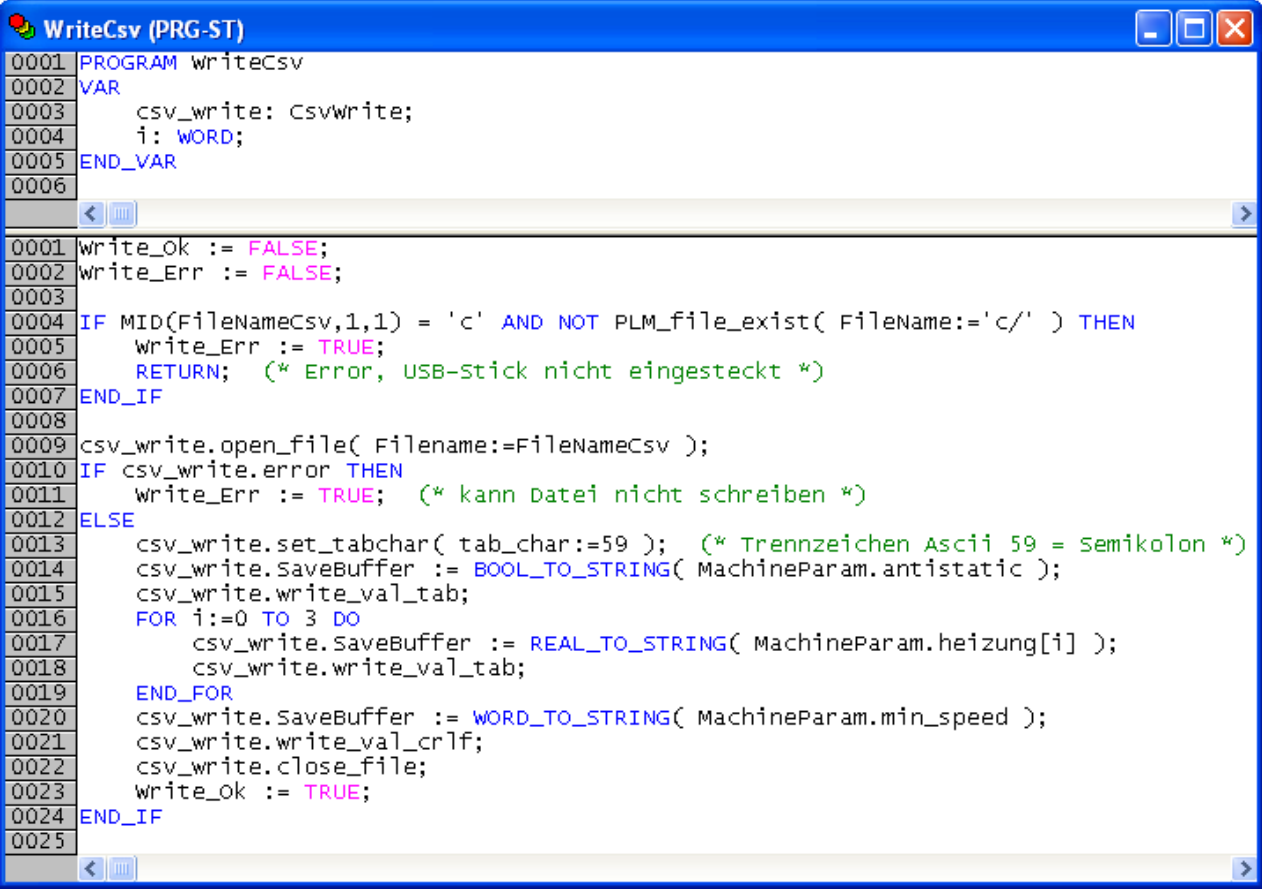
Abb. 4-7: Programmblock WriteBin zum Speichern des Parametersatzes in einer Binärdatei

```

0001 PROGRAM ReadBin
0002 VAR
0003     fid: DWORD;
0004 END_VAR
0005
0006
0001 Read_ok := FALSE;
0002 Read_err := FALSE;
0003
0004 IF MID(FileNameBin,1,1) = 'c' AND NOT PLM_file_exist( FileName:='c/' ) THEN
0005     Read_err := TRUE;
0006     RETURN; (* Error, USB-Stick nicht eingesteckt *)
0007 END_IF
0008
0009 fid := PLM_fileopen( FileName:=FileNameBin, FileFlag:='r' ); (* File öffnen *)
0010 IF fid = 0 THEN
0011     Read_err := TRUE;
0012 ELSE
0013     PLM_fileread( (* MachineParam-struct als Block lesen *)
0014         file_id:=fid,
0015         Buffer:=ADR(MachineParam),
0016         SIZEOF(MachineParam)
0017     );
0018     PLM_fileclose( file_id:=fid );
0019     Read_ok := TRUE;
0020 END_IF
0021

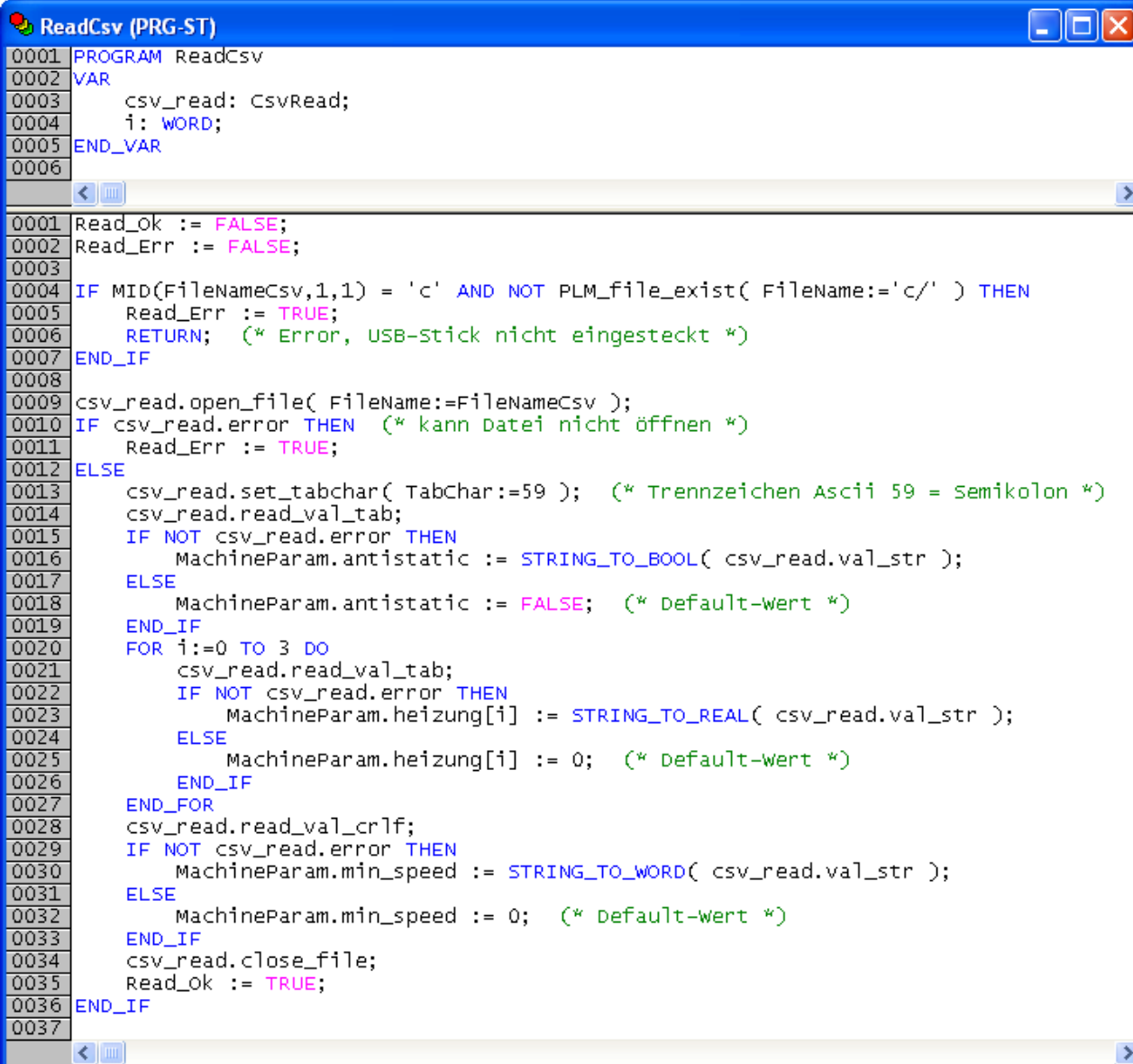
```

Abb. 4-8: Programmblock ReadBin zum Laden des Parametersatzes aus einer Binärdatei



```
0001 PROGRAM writeCsv
0002 VAR
0003     csv_write: Csvwrite;
0004     i: WORD;
0005 END_VAR
0006
0007
0008 write_ok := FALSE;
0009 write_Err := FALSE;
0010
0011 IF MID(FileNameCsv,1,1) = 'c' AND NOT PLM_file_exist( FileName:='c/' ) THEN
0012     write_Err := TRUE;
0013     RETURN; (* Error, USB-stick nicht eingesteckt *)
0014 END_IF
0015
0016 csv_write.open_file( Filename:=FileNameCsv );
0017 IF csv_write.error THEN
0018     write_Err := TRUE; (* kann Datei nicht schreiben *)
0019 ELSE
0020     csv_write.set_tabchar( tab_char:=59 ); (* Trennzeichen Ascii 59 = semikolon *)
0021     csv_write.SaveBuffer := BOOL_TO_STRING( MachineParam.antistatic );
0022     csv_write.write_val_tab;
0023     FOR i:=0 TO 3 DO
0024         csv_write.SaveBuffer := REAL_TO_STRING( MachineParam.heizung[i] );
0025         csv_write.write_val_tab;
0026     END_FOR
0027     csv_write.SaveBuffer := WORD_TO_STRING( MachineParam.min_speed );
0028     csv_write.write_val_crlf;
0029     csv_write.close_file;
0030     write_ok := TRUE;
0031 END_IF
0032
```

Abb. 4-9: Programmblock WriteCsv zum Speichern des Parametersatzes in einer CSV-Datei



```
0001 PROGRAM ReadCsv
0002 VAR
0003     csv_read: CsvRead;
0004     i: WORD;
0005 END_VAR
0006
0001 Read_Ok := FALSE;
0002 Read_Err := FALSE;
0003
0004 IF MID(FileNameCsv,1,1) = 'c' AND NOT PLM_file_exist( FileName:='c/' ) THEN
0005     Read_Err := TRUE;
0006     RETURN;  (* Error, USB-Stick nicht eingesteckt *)
0007 END_IF
0008
0009 csv_read.open_file( FileName:=FileNameCsv );
0010 IF csv_read.error THEN  (* kann Datei nicht öffnen *)
0011     Read_Err := TRUE;
0012 ELSE
0013     csv_read.set_tabchar( Tabchar:=59 );  (* Trennzeichen Ascii 59 = semikolon *)
0014     csv_read.read_val_tab;
0015     IF NOT csv_read.error THEN
0016         MachineParam.antistatic := STRING_TO_BOOL( csv_read.val_str );
0017     ELSE
0018         MachineParam.antistatic := FALSE;  (* Default-wert *)
0019     END_IF
0020     FOR i:=0 TO 3 DO
0021         csv_read.read_val_tab;
0022         IF NOT csv_read.error THEN
0023             MachineParam.heizung[i] := STRING_TO_REAL( csv_read.val_str );
0024         ELSE
0025             MachineParam.heizung[i] := 0;  (* Default-wert *)
0026         END_IF
0027     END_FOR
0028     csv_read.read_val_crlf;
0029     IF NOT csv_read.error THEN
0030         MachineParam.min_speed := STRING_TO_WORD( csv_read.val_str );
0031     ELSE
0032         MachineParam.min_speed := 0;  (* Default-wert *)
0033     END_IF
0034     csv_read.close_file;
0035     Read_Ok := TRUE;
0036 END_IF
0037
```

Abb. 4-10: Programmblock ReadCsv zum Laden des Parametersatzes aus einer CSV-Datei

5. Ethernet Einstellungen abfragen und ändern

5.1. Allgemeines

Bei Verwendung der Ethernet-Schnittstelle der PLM-Steuerungen ist diese vorher geeignet zu parametrieren. Dies kann über eine serielle Verbindung im Bootloader der Steuerung geschehen.

Häufig ist es jedoch wünschenswert, dass die Einstellungen auch aus dem IEC-Programm heraus geändert werden und ggf. sogar vom Anwender eingestellt können. Hierfür steht eine Bibliothek *Plm_NetSettings.lib* zur Verfügung.

Zusätzlich steht eine Visu-Bibliothek *Plm_NetSettingsVisu.lib* zur Verfügung, mit der auf einfachste Weise ein Dialog zum Einstellen der Netzwerkparameter in eigene Projekte eingebunden werden kann.

5.2. Benötigte Bibliotheken

Benötigt werden die folgenden Bibliotheken:

Plm_NetSettings.lib
Plm_Std.lib
UPD_E_005.lib (oder spätere Version)

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

5.3. Vorgehensweise

Die Bibliothek enthält Funktionen zum Setzen (Set) und Auslesen (Get) der Netzwerkeinstellungen (siehe Abb. 5-1).

Folgende Netzwerkparameter stehen zur Verfügung:

IP-Adresse
Netzmaske
Default-Gateway
DNS-Serveradresse
MAC-Adresse

Sowohl die Get- als auch die Set-Bausteine stehen in zwei Varianten zur Verfügung:

- Übergabe der Parameter als Einzelwerte, z.B. bei IP-Adresse als vier einzelne Bytes
- Übergabe der Parameter als String

Die Bausteine mit String-Parameter führen das Wort *String* im Namen. Im Fall der Übergabe als String ist das Format:

- IP-Adresse, Netzmaske, Default-Gateway, DNS-Serveradresse:
vier Dezimalzahlen 0...255, durch Punkte getrennt,
z.B. '10.1.1.231'
- MAC-Adresse:
sechs Hex-Zahlen 00...FF, durch Doppelpunkte getrennt,
z.B. '01:38:FE:30:23:A0'

Achtung: Das Ändern der Netzwerkparameter kann dazu führen, dass die Steuerung nicht mehr über Ethernet erreichbar ist.

Achtung: Änderungen der MAC-Adresse und/oder der IP-Adresse können dazu führen, dass die ARP-Tabellen von Netzwerkteilnehmern, die Kontakt mit der Steuerung aufnehmen wollen, aktualisiert werden müssen. Unter Windows ist z.B. der Konsolenbefehl `arp -d` auszuführen.

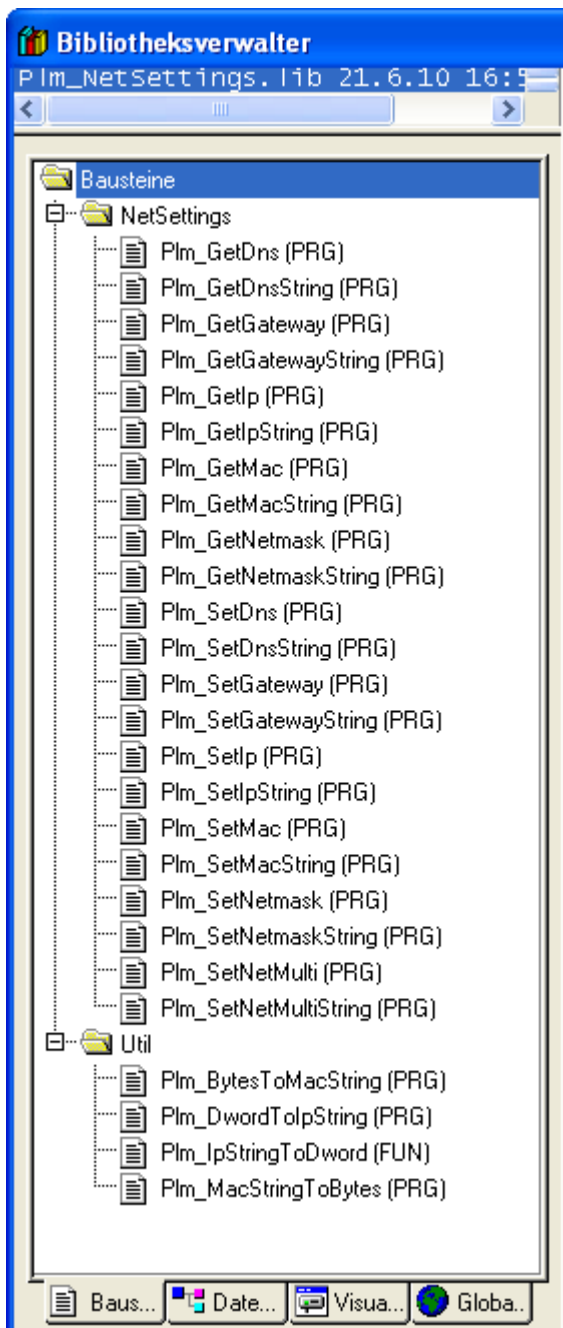


Abb. 5-1: Bausteine der Bibliothek `Plm_NetSettings.lib`

5.4. Ausführungsdauer der Set-Bausteine

Beim Aufruf der Set-Bausteine werden die neuen Werte im Flash-Speicher der Steuerung gespeichert, damit sie nach dem Einschalten wieder zur Verfügung stehen.

Jeder Flash-Vorgang nimmt mehrere Sekunden in Anspruch. Während dieser Zeit ist der IEC-Zyklus blockiert und auch der Hardware-Watchdog der Steuerung wird nicht neu getriggert.

Um die Dauer mehrerer aufeinander folgender Set-Befehle zu reduzieren und ein Ansprechen des Hardware-Watchdogs auszuschließen, stehen zwei Bausteine `Plm_SetNetMulti` und `Plm_SetNetMultiString` zur Verfügung. Diese erlauben die gleichzeitige Einstellung mehrerer Netzwerkparameter mit einem einzigen Flash-Vorgang.

5.5. Verwendung der Visu-Bibliothek

Die Bibliothek *Plm_NetSettingsVisu.lib* ergänzt die oben beschriebene Bibliothek *Plm_NetSettings.lib* und enthält folgende vorgefertigte Blöcke:

- Baustein `Plm_NetSettingsPrg()`
- SubVisu `PLM_NETSETTINGSVISU`

Mit diesen beiden Blöcken kann auf einfache Weise ein Einstelldialog für die Netzwerkparameter in eigene Projekte eingebaut werden.

5.5.1. Blöcke der Bibliothek *Plm_NetSettingsVisu.lib*

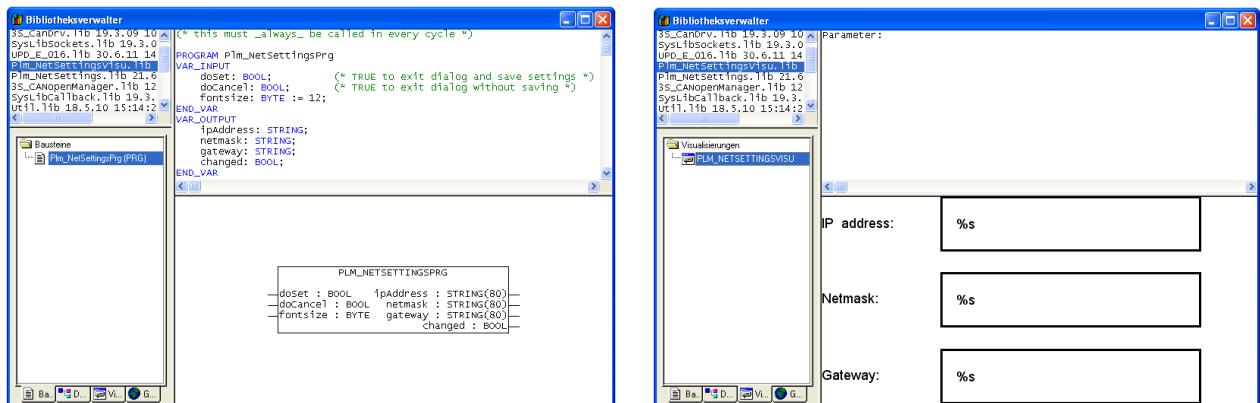


Abb. 5-2: Blöcke der Bibliothek *Plm_NetSettingsVisu.lib*

Der Baustein `Plm_NetSettingsPrg()` muss zyklisch aufgerufen werden. Er übernimmt die Auswertung der Eingaben in die SubVisu und stellt die Netzwerkparameter in Variablen zur Verfügung.

Die Input-Parameter `doSet` und `doCancel` müssen durch entsprechende Buttons OK und Cancel der Visu-Seite getastet werden, in die die SubVisu eingebunden ist. Bei `doSet = TRUE` werden Änderungen in den SubVisu-Eingabefeldern übernommen, bei `doCancel = TRUE` werden Änderungen verworfen.

Als Ausgabewerte werden IP-Adresse, Netzmaske und Gateway-Adresse geliefert. Der zusätzliche Parameter `changed` hat den Wert TRUE, wenn Einstellungen geändert und die Änderungen mit `doSet` übernommen wurden. In diesem Fall muss die Steuerung evtl. neu gestartet werden.

Die Subvisu `PLM_NETSETTINGSVISU` stellt drei Eingabefelder mit Beschriftung zur Verfügung, in denen zunächst die aktuellen Netzwerkparameter angezeigt werden.

Neben dem Einbinden der Subvisu muss der Anwender zwei Buttons OK und Cancel erstellen, die die Variablen `doSet` und `doCancel` tasten.

Durch Eingabe eines Strings der Form xxx.xxx.xxx.xxx in eines der Felder kann der jeweilige Wert geändert werden. Änderungen werden erst übernommen, wenn `doSet` für mindestens einen Zyklus auf TRUE gesetzt wird. Andernfalls ist `doCancel` für mindestens einen Zyklus auf TRUE zu setzen, dadurch werden die Änderungen verworfen.

Falls die Beschriftungen nicht gewünscht sind, können diese nach dem Einbinden der SubVisu mit weißen Flächen überdeckt werden.

5.5.2. Programmbeispiel zur Bibliothek *Plm_NetSettingsVisu.lib* (ST)

Das folgende Beispiel verwendet die Bibliothek *Plm_NetSettingsVisu.lib*.

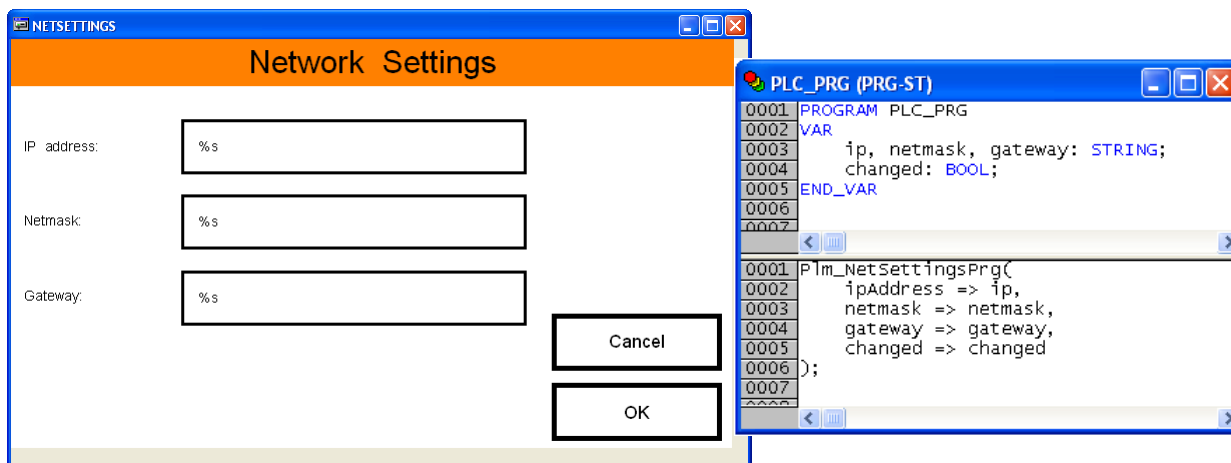


Abb. 5-3: Verwendung der Bibliothek *Plm_NetSettingsVisu.lib* (ST)

Die SubVisu `PLM_NETSETTINGSVISU` wurde in eine eigene Visu-Seite eingefügt und mit einer Überschrift und zwei Buttons (Cancel/OK) zum Verlassen der Visu-Seite ergänzt. Sowohl der Cancel-Button als auch der OK-Button führen eine Zoom-Funktion zu einer anderen Visu-Seite aus (*Eingabe → Zoomen nach Visu*).

Außerdem tastet der Cancel-Button die Variable `Plm_NetSettingsPrg.doCancel`, der Ok-Button die Variable `Plm_NetSettingsPrg.doSet` (*Eingabe → Variable tasten*).

Der Baustein `Plm_NetSettingsPrg()` wird zyklisch aufgerufen und stellt für den Rest des Programms die aktuellen Netzwerkparameter zur Verfügung.

6. Variablen austausch mittels UDP

6.1. Allgemeines

In einigen Anwendungsfällen sind mehrere Steuerungen in ein Anlagenkonzept eingebunden, die untereinander Variablenwerte austauschen sollen.

PLM-Steuerungen bieten hier die Möglichkeit, automatisch eine oder mehrere Variablen oder Variablenblöcke zwischen Steuerungen auszutauschen und diese so quasi auf dem gleichen Stand zu halten.

Dieses Konzept wird als *Netzwerkvariablen* bezeichnet. Der Datenaustausch erfolgt mittels UDP-Paketen über Ethernet. Voraussetzung ist, dass die beteiligten Steuerungen über ein funktionsfähiges und korrekt konfiguriertes Netzwerk miteinander verbunden sind.

UDP-Datenpakete können grundsätzlich an einen einzigen Empfänger geschickt werden, oder per Broadcast an alle Empfänger im lokalen Netzwerk. Dies ermöglicht eine sehr effizientes Verteilen von Variablenwerten an mehrere Steuerungen.

Auf PLM-Steuerungen stehen Netzwerkvariablen in zwei Varianten zur Verfügung:

1. CoDeSys-Netzwerkvariablen UDP (siehe Abschnitt 6.2)
2. PLM-Bibliothek Plm_UdpCom.lib (siehe Abschnitt 6.3)

Die beiden Varianten werden im folgenden behandelt.

6.2. CoDeSys-Netzwerkvariablen

6.2.1. CoDeSys-Netzwerkvariablen, Vorteile und Nachteile

Vorteile:

- CoDeSys-Netzwerkvariablen erlauben den Variablen austausch zwischen CoDeSys-Steuerungen verschiedener Hersteller
- Grafische Dialoge zur Einrichtung sind in der CoDeSys-Entwicklungsumgebung vorhanden

Nachteile:

- Die Implementierung benötigt relativ viel Speicher, Zykluszeit und Netzwerkbandbreite
- Keine Datenkonsistenz innerhalb einer Variablenliste
- Für Sender und Empfänger sind grundsätzlich verschiedene Programme notwendig, auch wenn die Programmfunktion eigentlich identisch ist; dies ist bedingt durch das Konzept der Listenkennung

6.2.2. CoDeSys-Netzwerkvariablen, Prinzip

Die auszutauschenden Variablen werden in globalen Variablenlisten zusammengefasst. Für jede Variablenliste wird einzeln festgelegt, ob Sie gesendet oder empfangen werden soll.

Zum Unterscheiden der Variablenlisten erhält jede eine eindeutige Kennung (COB-ID), die beim Erstellen der Liste festgelegt wird. Die COB-ID kann beliebig gewählt werden; es empfehlen sich kleine Zahlen im Bereich 1...100.

Eine Sendevariablenliste kann entweder an genau eine bestimmte Empfängersteuerung geschickt werden, oder per Broadcast an alle Steuerungen im Subnetz.

Ein Variablen austausch ist grundsätzlich auch über den CAN-Systembus möglich, diese Variante wird jedoch aus vielen Gründen selten verwendet und hier deshalb nicht näher beschrieben. Falls der CAN-Systembus zum Datenaustausch zwischen Steuerungen verwendet werden soll, ist der Betrieb einer Steuerung als CAN-Slave (Device) eine Alternative.

Das Einrichten von UDP-Netzwerkvariablen erfolgt in der CoDeSys-Entwicklungsumgebung und wird im Folgenden beschrieben. Zusätzliche Hinweise finden sich in der CoDeSys-Hilfe unter dem Stichwort *Netzwerkvariablen*.

Voraussetzung für die Verwendung von CoDeSys-Netzwerkvariablen:

- es existiert eine Ethernet-Verbindung zwischen den beteiligten Steuerungen,
- IP-Adresse, Netzmaske und ggf. Default-Gateway sind korrekt konfiguriert,
- die Library NetVarUdp_LIB.lib muss geladen sein,
- es wurden eine oder mehrere globale Variablenlisten für Senden und Empfangen getrennt eingerichtet,
- die entsprechenden Variablenlisten müssen auf allen beteiligten Steuerungen identisch sein, d.h. sie müssen die gleichen Variablen in der gleichen Reihenfolge enthalten.

6.2.3. CoDeSys-Netzwerkvariablen, Einrichtung

- Unter *Zielsystem Einstellungen* → *Netzfunktionen* ist das UDP-Protokoll einzutragen (Abb. 6-1).

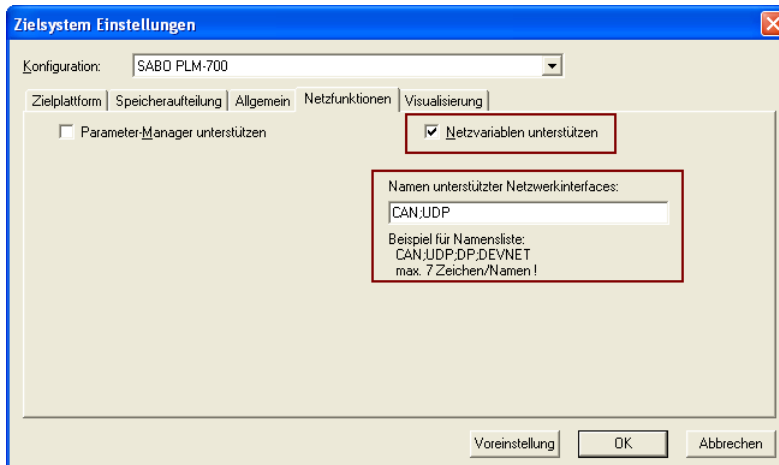


Abb. 6-1: Netzwerkvariablen, Zielsystem-Einstellungen

- Wählen Sie unter der linken Spalte von CoDeSys den Reiter *Ressourcen*, öffnen Sie *Globale Variablen* und klicken Sie mit der rechten Maustaste auf eine globale Variablenliste. Im erscheinenden Kontextmenü klicken Sie auf *Objekt Eigenschaften* (Abb. 6-2).

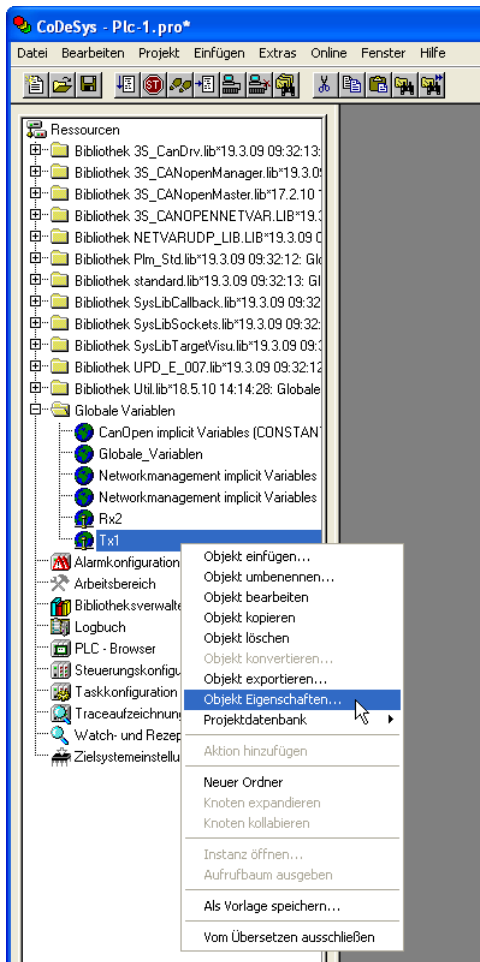


Abb. 6-2: Netzwerkvariablen, Konfigurieren einer Variablenliste

- Klicken Sie auf den Button *Netzwerkverbindung hinzufügen* und wählen Sie als Netzwerktyp *UDP*. Klicken Sie dann auf den Button *Einstellungen*, um die UDP-Eigenschaften zu konfigurieren. Im erscheinenden Dialog tragen Sie die IP-Adresse und die UDP-Portnummer für diese Liste ein (Abb. 6-3).

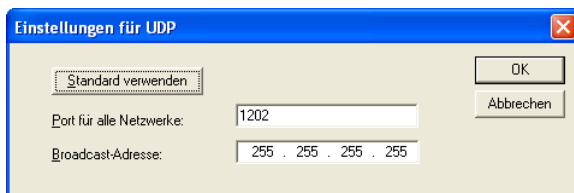


Abb. 6-3: Netzwerkvariablen, UDP-Einstellungen

- Falls es sich um eine Sendeliste handelt, aktivieren Sie das Kästchen *Schreiben* und vergeben Sie eine eindeutige Variablenlistenkennung. Nehmen Sie die weiteren Einstellungen gemäß Abb. 6-4 vor. Stellen Sie außerdem sicher, dass die entsprechende Liste auf allen Empfängern die gleiche Kennung und die gleichen Variablen in der gleichen Reihenfolge enthält.

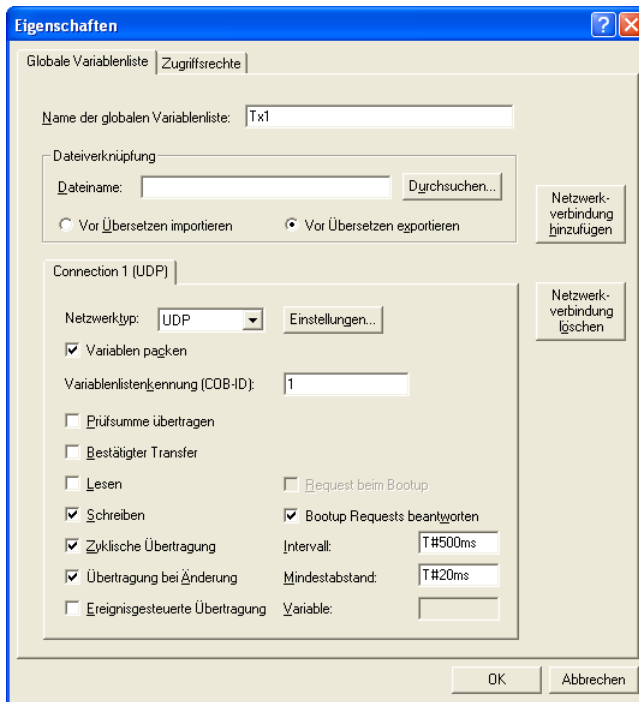


Abb. 6-4: Netzwerkvariablen, Sendeliste

- Falls es sich um eine Empfangsliste handelt, aktivieren Sie das Kästchen *Lesen* und tragen Sie die eindeutige Variablenlistenkennung der zugehörigen Sendeliste ein. Nehmen Sie die weiteren Einstellungen gemäß Abb. 6-5 vor. Stellen Sie außerdem sicher, dass die Liste des Senders die gleiche Kennung und die gleichen Variablen in der gleichen Reihenfolge enthält.

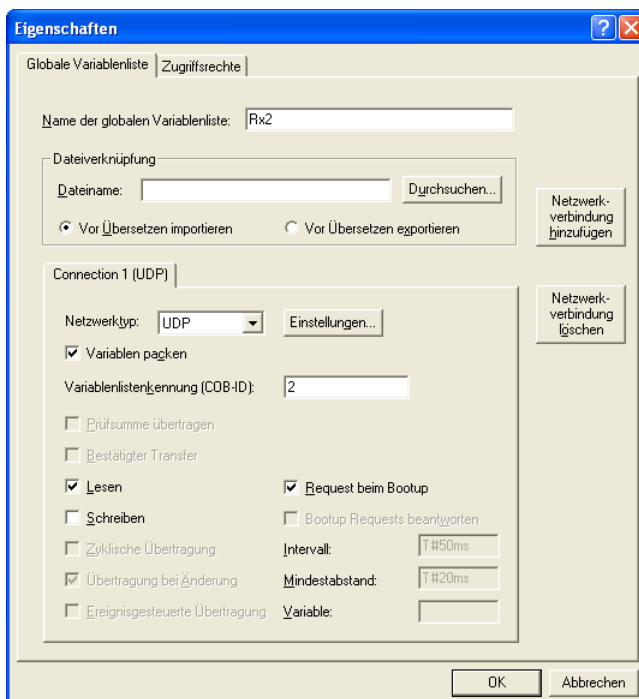


Abb. 6-5: Netzwerkvariablen, Empfangsliste

6.2.4. CoDeSys-Netzwerkvariablen, Allgemeine Hinweise

Änderungen einer Variablenliste werden mit einer gewissen Verzögerung von der Sendesteuerung auf die beteiligten Empfangssteuern übertragen. Diese Verzögerung beträgt mindestens zwei Zykluszeiten ($2 \times 20 \text{ ms}$) zuzüglich der Übertragungsdauer der UDP-Pakete. Diese Verzögerung ist von den IEC-Programmen auf Sender und Empfänger zu berücksichtigen.

Die Variablen innerhalb einer Variablenliste werden von CoDeSys einzeln per UDP-Message versendet, daher ist grundsätzlich keine Datenkonsistenz innerhalb der Liste gewährleistet.

Verbindungen mittels UDP funktionieren üblicherweise sehr stabil in einem lokalen Netzwerk (LAN). Bei der Verwendung von UDP über das Internet kann es jedoch zu Zeitverschiebungen von Datenpaketen und zu Paketverlusten kommen. Solche Fehler werden nicht automatisch vom UDP-Layer der Steuerung oder von CoDeSys festgestellt.

Der Abgleich der Variablenlisten zwischen Sender- und Empfängerprogramm kann einfach über die Windows-Zwischenablage erfolgen, indem das IEC-Projekt des Senders mit dem entsprechenden Listenfenster geöffnet und der gesamte Text vollständig mit der Maus markiert wird. Dann auf *Bearbeiten* → *Kopieren* klicken. Anschließend wird das IEC-Projekt des Empfängers mit dem entsprechenden Listenfenster geöffnet und der gesamte Text vollständig mit der Maus markiert. Dann auf *Bearbeiten* → *Einfügen* klicken; der vorhandene Programmtext wird dabei ersetzt.

Alternativ bietet CoDeSys die Möglichkeit, einen automatischen Abgleich zwischen mehreren Projekten zu erzielen. Dazu geben Sie bei *Dateiverknüpfung* den Namen einer Exportdatei an, die von Sende- und Empfangs-Projekt gemeinsam genutzt wird. Aktivieren Sie bei der Sendeliste *Vor Übersetzen exportieren* und bei der Empfangsliste *Vor Übersetzen importieren*. Änderungen an der Variablenliste dürfen jetzt nur noch im Sende-Projekt vorgenommen werden. Bei Änderungen der Variablenliste ist zunächst das Sende-Projekt zu übersetzen und anschließend das Empfangsprojekt. Danach sind die Listen in beiden Projekten im gleichen Zustand.

6.2.5. CoDeSys-Netzwerkvariablen, Überwachen der Verbindung

CoDeSys bietet keine direkte Überwachung, ob der Netzwerkvariablen austausch arbeitet oder nicht. Eine solche Überwachung kann jedoch einfach implementiert werden:

- Legen Sie in der Sender- und der zugehörigen Empfängerliste eine Zählervariable an, z.B. `TxCount`. Auf der Empfängerseite wird zusätzlich an anderer Stelle ein Vergleichswert `TxCountOld` und ein Timeout-Zähler `TxTimeout` benötigt.
- Im Sender wird die Variable `TxCount` regelmäßig inkrementiert. Ein Überlauf von `TxCount` spielt keine Rolle.
- Der Empfänger prüft, ob die Variable `TxCount` sich geändert hat. Wenn ja, wird `TxTimeout` auf 0 gesetzt, andernfalls wird `TxTimeout` inkrementiert. Wenn `TxTimeout` einen gewissen Wert überschreitet, ist die Verbindung unterbrochen.

Beispiel Sender:

```
VAR_GLOBAL
    TxCount: WORD;
END_VAR

TxCount := TxCount + 1;
```

Beispiel Empfänger:

```
VAR_GLOBAL
    TxCount: WORD
END_VAR

VAR
    TxCountOld: WORD;
    TxTimeout: WORD;
END_VAR

IF TxCount <> TxCountOld THEN
    TxTimeout := 0;
    TxCountOld := TxCount;
ELSE
    TxTimeout := TxTimeout + 1;
```

```

END_IF
IF TxTimeout > 50 THEN
    TxTimeout := 999; (* Überlauf verhindern *)
    (* Verbindung unterbrochen *)
END_IF

```

Im Beispiel wird `TxCount` in jedem Zyklus inkrementiert. Dadurch muss die Variablenliste in jedem Zyklus per UDP verschickt werden, was zu einer relativ hohen CPU- und Ethernet-Auslastung führt. Es empfiehlt sich daher, auf der Sender-Seite `TxCount` nur seltener zu inkrementieren, z.B. einmal pro Sekunde. Entsprechend muss die Timeout-Schwelle am Empfänger angepasst werden.

6.2.6. CoDeSys-Netzwerkvariablen, Fehlersuche

Fehlermeldung beim Übersetzen *Es ist keine zyklische oder freilaufende Task zum Netzwerkvariablen austausch vorhanden*: Sie haben eine Netzwerkvariablenliste angelegt, verwenden jedoch keine Variable daraus in Ihrem IEC-Programm. Entfernen Sie entweder die Netzwerkverbindung aus der entsprechenden Variablenliste oder verwenden Sie eine Variable aus der Liste in Ihrem IEC-Programm.

Die Listenkennungen müssen zwischen Sender und Empfänger genau übereinstimmen.

Die Variablen der Variablenlisten müssen zwischen Sender und Empfänger genau übereinstimmen. Verwenden Sie ggf. die Funktion *Dateiverknüpfung*, um einen automatischen Abgleich zwischen Projekten zu erzielen.

6.3. PLM-Bibliothek UDP-Com

6.3.1. PLM-Bibliothek UDP-Com, Vorteile und Nachteile

Vorteile:

- Schlanke Implementierung, ressourcensparend
- Einfache Bausteine zum Senden und Empfangen
- Datenkonsistenz sichergestellt

Nachteile:

- Variablen austausch nur zwischen PLM-Steuerungen möglich

6.3.2. PLM-Bibliothek UDP-Com, Prinzip

Die Bibliothek `Plm_UdpCom.lib` stellt zwei Bausteine `Plm_NetVarSend` und `Plm_NetVarRecv` zur Verfügung.

Ein Baustein vom Typ `Plm_NetVarSend` ist in der Lage, einen beliebigen Speicherbereich in UDP-Messages zu zerteilen und zu versenden. Das Versenden erfolgt entweder an eine bestimmte IP-Adresse oder an die Broadcast-Adresse 255.255.255.255. Im ersten Fall können die UDP-Messages nur von der bestimmten Steuerung empfangen werden, im zweiten Fall von allen Steuerungen im Subnetz.

Ein Baustein vom Typ `Plm_NetVarRecv` ist in der Lage, die UDP-Messages eines Sendebausteins vom Typ `Plm_NetVarSend` zu empfangen und wieder zu einem Speicherbereich zusammensetzen. Dabei wird geprüft, ob alle notwendigen Messages empfangen wurden und ggf. erst dann der Zielspeicherbereich aktualisiert (konsistenter Empfang).

Zum Unterscheiden mehrerer gesendeter Speicherbereiche wird bei Sender und Empfänger eine sog. UDP-Portnummer angegeben. Dies ist eine Zahl im Bereich 1024...65535, die zur Zuordnung zwischen Sender und Empfänger dient. Die Portnummer kann im angegebenen Bereich beliebig gewählt werden.

Voraussetzung für die Verwendung der PLM-Bibliothek UDP-Com:

- es existiert eine Ethernet-Verbindung zwischen den beteiligten Steuerungen,
- IP-Adresse, Netzmaske und ggf. Default-Gateway sind korrekt konfiguriert,

- die Library Plm_UdpCom.lib muss geladen sein,
- die Bausteine Plm_NetVarSend bzw. Plm_NetVarRecv werden zyklisch aufgerufen,
- die an einen Sendebaustein angelegte Datenstruktur ist mit der am zugehörigen Empfangsbaustein identisch.

6.3.3. Plm_NetVarSend (Plm_UdpCom.lib)

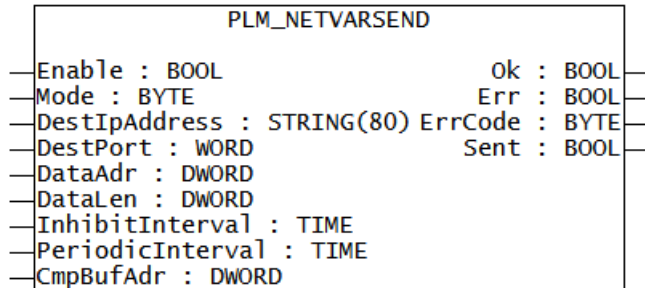


Abb. 6-6: Funktionsblock Plm_NetVarSend (Plm_UdpCom.lib)

Input-Parameter:		
Enable	BOOL	TRUE = Baustein aktiv, FALSE = Baustein inaktiv
Mode	BYTE	0 = Senden nur bei Mode +128 oder durch Periodic Interval, 1 = Zusätzlich automatisches Senden bei Änderung der Daten, +128 = Senden direkt auslösen
DestIpAddress	STRING	IP-Adresse der Zielsteuerung, z.B. '10.1.1.231', oder '255.255.255.255' für Broadcast
DestPort	WORD	UDP-Port der Zielsteuerung, z.B. 3456
DataAdr	DWORD	Startadresse der zu übertragenden Daten (s.u.)
DataLen	DWORD	Länge (Anzahl Bytes) der zu übertragenden Daten (s.u.)
InhibitInterval	TIME	Sperrzeit nach Senden, z.B. t#100ms, Abschalten der Funktion (keine Sperrzeit) mit t#0s
PeriodicInterval	TIME	Periodisches Senden der Daten nach Ablauf der angegebenen Zeit, z.B. t#2s, Abschalten der Funktion mit t#0s
CmpBufAdr	DWORD	Nur bei Mode = 1: Startadresse eines Vergleichsbuffers zum Feststellen von Datenänderungen, Länge (Anzahl Bytes) muss DataLen entsprechen
Output-Parameter:		
Ok	BOOL	TRUE, wenn Baustein sendebereit ist oder erfolgreich UDP-Messages gesendet hat
Err	BOOL	TRUE, wenn während des letzten Aufrufs ein Fehler aufgetreten ist
ErrCode	BYTE	Nummer des letzten aufgetretenen Fehlers. Die Fehlercodes sind außerdem als globale Variablen in der Bibliothek

		gespeichert. 200 = UDPCOM_ERR_ILLEGALDEST = DestIpAddress oder DestPort ungültig 201 = UDPCOM_ERR_SOCKET = Fehler beim Erstellen des UDP-Sockets 202 = UDPCOM_ERR_DATAADR = DataAdr hat ungültigen Wert 203 = UDPCOM_ERR_DATALEN = DataLen ist Null oder größer als 65535 206 = UDPCOM_ERR_BROADCAST = Fehler beim Einstellen des Broadcast- Modus' 207 = UDPCOM_ERR_SENDFAILED = Fehler beim Ausführen des Systembefehls zum Senden einer UDP- Message 208 = UDPCOM_ERR_CMPBUFINVALID = CmpBufAdr hat ungültigen Wert und Mode = 1 210 = UDPCOM_ERR_NUMSEGMENTS = Der Datenbereich muss in mehr als 32 UDP-Messages zerlegt werden 211 = UDPCOM_ERR_MTUSIZE = Der Wert der globalen Bibliotheksvariablen Plm_UdpCom_MTU muss zwischen 128 und 1500 liegen
Sent	BOOL	TRUE, wenn beim letzten Aufruf mind. eine UDP-Message verschickt wurde (zu Debugging-Zwecken)

Bei `Mode = 0` wird das Versenden der Variablen entweder ausgelöst, indem Bit 7 von `Mode` gesetzt wird (`Mode +128`), oder durch Ablauf der Zeit in `PeriodicInterval`.

Bei `Mode = 1` wird zusätzlich `CmpBufAdr` benötigt. Diese Adresse muss auf einen leeren Speicherbereich zeigen, der die gleiche Größe (in Bytes) hat wie `DataLen`. Der angegebene Buffer dient dazu, Änderungen an den Daten festzustellen und dadurch das Senden der Variablen automatisch auszulösen.

`DataAdr` und `DataLen` spezifizieren den Datenbereich, der übertragen werden soll. Je nachdem, auf welche Art dieser Speicher vorliegt, können z.B. folgende Angaben verwendet werden:

- Einzelne Variable vom Typ BYTE, WORD, INT, DWORD, REAL etc.:

```
DataAdr := ADR( var );
DataLen := SIZEOF( var );
```

- Variablenstruktur (*Datentyp*):

```
DataAdr := ADR( structvar );
DataLen := SIZEOF( structvar );
```

- Array von BYTE, WORD, INT, DWORD, REAL etc.:

```
DataAdr := ADR( arrayvar );
DataLen := SIZEOF( arrayvar );
```

Empfohlen wird, eine Variablenstruktur (*Datentyp*) anzulegen, in der alle benötigten Werte gespeichert werden. Ein *Datentyp* kann in CoDeSys relativ einfach zwischen Sender- und Empfänger-Projekt ausgetauscht werden (Menü *Projekt* → *Kopieren*).

Beispiel: Definition eines Datentyps `ExchgData` (CoDeSys linke Spalte, zweiter Reiter *Datentypen*, Rechte Maustaste, *Objekt einfügen*):

```

TYPE ExchgData :
STRUCT
a: ARRAY[0..7] OF BYTE;
b: WORD;
c: STRING(40);
d: REAL;
e: REAL;
END_STRUCT
END_TYPE
    
```

Anlegen einer Variablen `mydata` vom Typ `ExchgData` im Programm:

```

VAR
mydata: ExchgData;
cmpBuf: ExchgData;
END_VAR
    
```

Auf die Elemente in `mydata` kann mit der Schreibweise z.B. `mydata.a` zugegriffen werden (Strukturvariable—Punkt—Strukturelement). Die Variable `mydata` kann dann komplett mit `Plm_NetVarSend` versendet bzw. mit `Plm_NetVarRecv` empfangen werden:

```

DataAdr := ADR( mydata );
DataLen := SIZEOF( mydata );
CmpBufAdr := ADR( cmpBuf );
    
```

Durch Anlegen einer zweiten Variablen `cmpBuf` vom gleichen Datentyp steht auf einfache Weise der für `Mode = 1` benötigte Vergleichsbuffer mit passender Größe zur Verfügung.

6.3.4. Plm_NetVarRecv (Plm_UdpCom.lib)

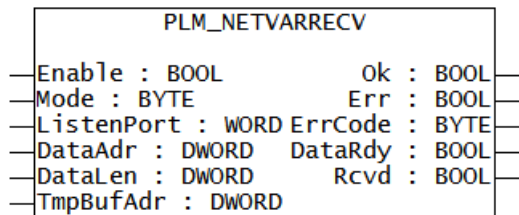


Abb. 6-7: Funktionsblock `Plm_NetVarRecv` (`Plm_UdpCom.lib`)

Input-Parameter:		
Enable	BOOL	TRUE = Baustein aktiv, FALSE = Baustein inaktiv
Mode	BYTE	0 = Inkonsistenter Datenempfang, Daten sind nur gültig wenn <code>DataRdy = TRUE</code> , 1 = Konsistenter Datenempfang, benötigt <code>TmpBufAdr</code> +128 = UDP-Messages von der eigenen IP-Adresse ignorieren (Broadcast)
ListenPort	WORD	UDP-Empfangs-Port, z.B. 3456, muss mit <code>DestPort</code> des Senders übereinstimmen
DataAdr	DWORD	Startadresse der zu empfangenen Daten (s. Baustein <code>Plm_NetVarSend</code>)
DataLen	DWORD	Länge (Anzahl Bytes) der zu empfangenen Daten (s. Baustein <code>Plm_NetVarRecv</code>)
TmpBufAdr	DWORD	Nur bei <code>Mode = 1</code> : Startadresse eines Buffers zum Zusammensetzen der Daten aus einzelnen UDP-Messages, Länge (Anzahl Bytes) muss <code>DataLen</code>

		entsprechen
Output-Parameter:		
Ok	BOOL	TRUE, wenn Baustein sendebereit ist oder erfolgreich UDP-Messages gesendet hat
Err	BOOL	TRUE, wenn während des letzten Aufrufs ein Fehler aufgetreten ist
ErrCode	BYTE	<p>Nummer des letzten aufgetretenen Fehlers. Die Fehlercodes sind außerdem als globale Variablen in der Bibliothek gespeichert.</p> <p>200 = UDPCOM_ERR_ILLEGALDEST = DestIpAddress oder DestPort ungültig</p> <p>201 = UDPCOM_ERR_SOCKET = Fehler beim Erstellen des UDP-Sockets</p> <p>202 = UDPCOM_ERR_DATAADR = DataAdr hat ungültigen Wert</p> <p>203 = UDPCOM_ERR_DATALEN = DataLen ist Null oder größer als 65535</p> <p>204 = UDPCOM_ERR_ILLEGALPORT = ListenPort ungültig</p> <p>205 = UDPCOM_ERR_RECVFAILED = Fehler beim Ausführen des Systembefehls zum Empfangen einer UDP-Message</p> <p>211 = UDPCOM_ERR_MTUSIZE = Der Wert der globalen Bibliotheksvariablen Plm_UdpCom_MTU muss zwischen 128 und 1500 liegen</p> <p>212 = UDPCOM_ERR_TMPBUFINVALID = TmpBufAdr hat ungültigen Wert und Mode = 1</p> <p>213 = UDPCOM_ERR_RXDATALEN-MISMATCH = Der Wert von DataLen am zugehörigen Sendebaustein ist ungleich DataLen</p>
DataRdy	BOOL	<p>Mode = 0: TRUE für einen Zyklus, wenn beim Daten konsistent und gültig sind,</p> <p>Mode = 1: TRUE, wenn Daten konsistent aktualisiert wurden</p>
Rcvd	BOOL	TRUE, wenn beim letzten Aufruf mind. eine UDP-Message empfangen wurde (zu Debugging-Zwecken)

DataAdr und DataLen spezifizieren den Datenbereich, der empfangen werden soll. Die Angaben entsprechen denen in Abschnitt 6.3.3.

Wenn das Bit 7 in Mode gesetzt ist (Mode + 128) ignoriert Plm_NetVarRecv alle UDP-Messages, die von einem eigenen Sendebaustein Plm_NetVarRecv stammen. Dies ist erforderlich, wenn der Sendebaustein Broadcast-Messages erzeugt, da diese ansonsten (bei passender Port-Nummer) von den eigenen Empfangsbausteinen empfangen werden.

6.3.5. PLM-Bibliothek UDP-Com, Allgemeine Hinweise

Die Datenübertragung erfolgt mit einer gewissen Verzögerung von der Sendesteuerung auf die beteiligten Empfangssteuern. Bis die Daten auf der Empfangssteuerung zur Verfügung stehen, vergehen mindestens zwei Zykluszeiten (2×20 ms) zuzüglich der Übertragungsdauer der UDP-Pakete. Diese Verzögerung ist von den IEC-Programmen auf Sender und Empfänger zu berücksichtigen.

Eine Steuerung kann gleichzeitig an einen UDP-Port senden und unabhängig davon auf demselben UDP-Port empfangen, z.B.

Steuerung A, DestPort 3456 → Steuerung B, ListenPort 3456,
Steuerung B, DestPort 3456 → Steuerung A, ListenPort 3456.

Falls eine Steuerung Broadcasts (DestIpAddress = '255.255.255.255') an einen bestimmten DestPort sendet, werden die Daten von allen Steuerungen im lokalen Subnetz empfangen, die einen Baustein Plm_NetVarRecv mit diesem ListenPort aktiviert haben. Das gilt auch für die sendende Steuerung selbst, wenn sie einen Empfangsbaustein auf dem passenden ListenPort betreibt, d.h. sie empfängt ggf. ihre eigenen Daten. Um dies zu verhindern, kann im Baustein Plm_NetVarRecv der Wert 128 zu Mode addiert werden. In diesem Fall werden UDP-Messages, die von der eigenen IP-Adresse stammen, verworfen.

6.3.6. PLM-Bibliothek UDP-Com, Datenkonsistenz und MTU

Die max. Größe der übertragbaren Datenblöcke auf dem Ethernet ist durch den Parameter MTU (Maximum Transfer Unit) begrenzt. Dieser ist bei fast allen PCs und ebenso auf PLM-Steuerungen auf den Wert 1500 Bytes fest eingestellt.

Der MTU-Wert muss der Bibliothek Plm_UdpCom.lib bekannt gemacht werden und zwischen Sender und Empfänger gleich eingestellt sein. Dazu existiert eine globale Bibliotheksvariable Plm_UdpCom_MTU. Diese kann bei Bedarf in der Initialisierungsphase des IEC-Programms verändert werden. Der voreingestellte Wert ist 1500 (Bytes).

Der MTU-Wert bezieht sich auf die Paketgröße auf dem Netzwerk-Layer. Die reine Nutzdatengröße (DataLen) für Bausteine vom Typ Plm_NetVarSend und Plm_NetVarRecv ist (Plm_UdpCom_MTU - 34). Bei Plm_UdpCom_MTU = 1500 sind dies 1466 Bytes.

Sofern DataLen kleiner oder gleich (Plm_UdpCom_MTU - 34) ist, erfolgt keine Segmentierung, d.h. die Daten werden in einer einzigen UDP-Message und damit in jedem Fall konsistent übertragen. Bei größeren Werten von DataLen muss die Konsistenz explizit sichergestellt werden, indem dem Empfangsbaustein ein Buffer TmpBufAdr zur Verfügung gestellt und Mode = 1 aktiviert wird. In diesem Buffer werden die empfangenen UDP-Messages zunächst gesammelt und einsortiert. Erst wenn alle zusammengehörenden Messages eingetroffen sind, wird der Inhalt des Buffers innerhalb eines Aufrufs von Plm_NetVarRecv an DataAdr kopiert.

DataLen kann in max. 32 UDP-Pakete der Datengröße (Plm_UdpCom_MTU - 34) zerteilt werden. Bei Plm_UdpCom_MTU = 1500 entspricht dies einem Wert von DataLen = 32 × 1466 = 46912 Bytes pro Sendebaustein.

6.3.7. PLM-Bibliothek UDP-Com, Fehlersuche

Die UDP-Portnummern DestPort und ListenPort müssen zwischen Sender und Empfänger übereinstimmen.

Die Größe des übertragenen Speicherbereichs muss zwischen Sender und Empfänger genau übereinstimmen. Dies wird anhand von DataLen beim Empfang kontrolliert.

Der Wert der globalen Bibliotheksvariablen Plm_UdpCom_MTU muss zwischen Sender und Empfänger übereinstimmen.

IP-Adresse, Netzmaske und ggf. Default-Gateway müssen auf allen beteiligten Steuerungen korrekt konfiguriert sein.

Anhand des Ausgangs Sent am Baustein Plm_NetVarSend kann beobachtet werden, ob in einem Zyklus mind. eine UDP-Message verschickt wurde. Zum einfacheren Beobachten empfiehlt es sich, einen Zähler zu inkrementieren sobald Sent = TRUE ist und dann diesen Zähler zu betrachten.

Anhand des Ausgangs Rcvd am Baustein Plm_NetVarRecv kann beobachtet werden, ob überhaupt UDP-Messages empfangen werden, auch wenn diese ggf. nicht ausgewertet werden können. Auch hier empfiehlt es sich, zum einfacheren

Beobachten einen Zähler zu inkrementieren sobald `Rcvd = TRUE` ist und dann diesen Zähler zu betrachten.

Bei korrekter Funktion und ungestörter Übertragung muss die Anzahl der empfangenen UDP-Messages mit der der gesendeten übereinstimmen, d.h. die Zählerstände aus `Plm_NetVarRecv.Rcvd` und `Plm_NetVarSend.sent` müssen übereinstimmen.

7. Universeller TCP-Server

7.1. Allgemeines

Ein TCP-Server stellt einem anfragenden TCP-Client eine Datenverbindung über Ethernet zur Verfügung. Der Server wartet darauf, dass die Verbindung durch einen Client auf dem vom Server belegten Port initiiert wird.

Die Port-Nummer ist eine Zahl im Bereich 0...65535, die zur Identifikation eines bestimmten TCP-Servers auf der Steuerung dient. Zahlreiche Portnummern im Bereich unterhalb von 1023 sind für Standarddienste reserviert, z.B. Port 21 für den FTP-Server der Steuerung.

Alle PLM 700-Steuerungen stellen einen universellen TCP-Server als virtuelle serielle Schnittstelle COM 9 zur Verfügung. Dadurch wird auf einfachste Weise ein Datenaustausch über Ethernet ermöglicht.

Das zu verwendende Protokoll muss bei Verwendung des universellen TCP-Servers im IEC-Programm implementiert werden. Für zahlreiche Standardprotokolle (z.B. Modbus TCP) stehen allerdings vorgefertigte Bibliotheken für PLM-Steuerungen zur Verfügung, so dass der universelle TCP-Server eher selten zum Einsatz kommt.

Zur Verwendung des TCP-Servers muss dieser lediglich mit Angabe einer Port-Nummer initialisiert werden. Der Server wird anschließend im IEC-Programm wie eine serielle Schnittstelle mit der Nummer COM 9 behandelt und tauscht Daten mittels der Funktionen `Comm_Read()` und `CommOut()` aus (siehe Abschnitt 2.5).

7.2. Spezifikation

- Universeller TCP-Server
- Server-Port wählbar
- max. 1 aktiver Client
- Datenaustausch wie bei serieller Schnittstelle

7.3. Benötigte Bibliotheken

Benötigt werden die folgenden Bibliotheken:

PLM7xx_02.lib
UPD_E_007.lib (oder spätere Version)

Die angegebenen Bibliothek müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

7.3.1. TCP_COM9_Work (PLM7xx_02.lib)

```

TCP_COM9_WORK
PortID : WORD TCP_COM9_work : BOOL
    
```

Abb. 7-1: Funktion `TCP_COM9_Work` (PLM7xx_02.lib)

Input-Parameter:		
PortID	WORD	TCP-Port, unter dem der Server angesprochen werden kann.
Return-Wert:		
TCP_COM9_Work	BOOL	TRUE = Server arbeitet

Die Funktion `TCP_COM9_Work` muss vom Anwenderprogramm zyklisch aufgerufen werden.

7.4. Verwendung des TCP-Servers

Durch zyklischen Aufruf der Funktion `TCP_COM9_Work()` steht der TCP-Server auf dem angegebenen Port zur Verfügung.

Der universelle TCP-Server kann genau eine Client-Verbindung gleichzeitig bedienen.

Zur Datenübertragung werden die Standardfunktionen `CommOut()` und `CommRead()` verwendet (siehe Abschnitt 2.5). Als Nummer der seriellen Schnittstelle (`comm_nr`) ist dabei der Wert 9 anzugeben.

Im Gegensatz zu den Initialisierungen der anderen COM-Ports muss der Baustein `TCP_COM9_WORK` nicht nur einmal, sondern zyklisch aufgerufen werden, da hierdurch der interne Server-Ablauf gesteuert wird.

Abschnitt 7.6 zeigt ein Beispiel für einen einfachen TCP-Server, der eine Echo-Funktion auf dem angegebenen Port realisiert.

Der Verbindungsstatus kann mit dem Befehl

```
state := SystemGetParameterDW( 10313 )
```

abgefragt werden. Das zurückgelieferte DWORD kann folgende Werte enthalten:

Wert	Bedeutung
0	keine Client-Verbindung
5	Client-Anfrage erhalten
10	Client verbunden und Daten übertragen

Standardmäßig bleibt die Verbindung solange bestehen, bis sie vom Client ordnungsgemäß beendet wird. Erst danach steht der TCP-Server für einen weiteren Client zur Verfügung.

7.5. Fehlerbehandlung

Eine abgerissene Client-Verbindung, die nicht ordnungsgemäß beendet wurde (z.B. bei Absturz des Client-Rechners), kann den TCP-Server blockieren, da dieser keine Möglichkeit hat, das Ende der Verbindung zu erkennen und er somit keine neue Verbindung zulässt.

Dies kann verhindert werden, indem mit dem Befehl

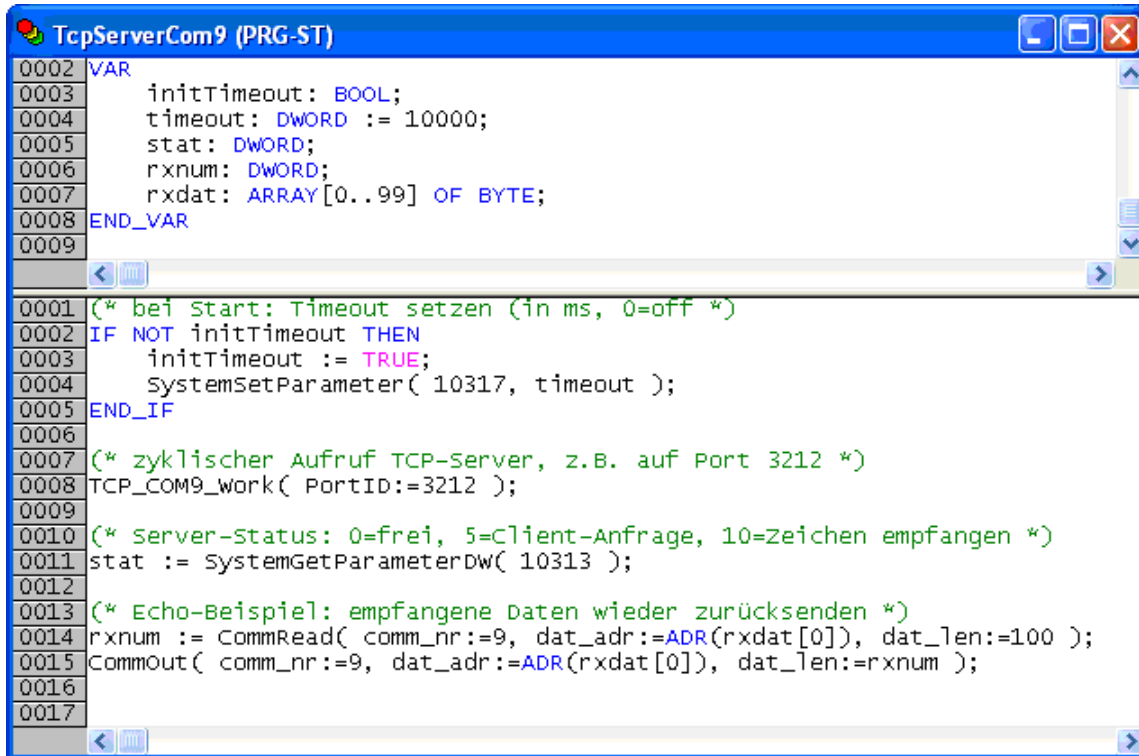
```
SystemSetParameter( 10317, timeout )
```

eine Timeout-Zeit als DWORD in Millisekunden vorgegeben wird. Der Timeout kann abgeschaltet werden (Voreinstellung), indem die Timeout-Zeit auf 0 gesetzt wird.

Wenn der verbundene Client innerhalb der angegebenen Timeout-Zeit keine Daten überträgt, beendet der Server von sich aus ordnungsgemäß die Verbindung, woraufhin sich ein Client erneut verbinden kann.

Ein eingeschalteter Timeout setzt voraus, dass der Client im Normalbetrieb regelmäßig Daten überträgt. Ist dies nicht der Fall, muss der Client zyklisch Dummy-Daten übertragen, die auf Seiten des TCP-Servers zwar nicht auszuwerten sind, aber die serverseitige Verbindung geöffnet halten. Es ist Sache des Anwendungsprogrammierers, ein geeignetes Protokoll zwischen Client und Server zu vereinbaren.

7.6. Beispiel für TCP-Server COM 9 (ST)



```
0002 VAR
0003     initTimeout: BOOL;
0004     timeout: DWORD := 10000;
0005     stat: DWORD;
0006     rxnum: DWORD;
0007     rxdat: ARRAY[0..99] OF BYTE;
0008 END_VAR
0009
0001 (* bei Start: Timeout setzen (in ms, 0=off *)
0002 IF NOT initTimeout THEN
0003     initTimeout := TRUE;
0004     SystemSetParameter( 10317, timeout );
0005 END_IF
0006
0007 (* zyklischer Aufruf TCP-Server, z.B. auf Port 3212 *)
0008 TCP_COM9_work( PortID:=3212 );
0009
0010 (* Server-Status: 0=frei, 5=Client-Anfrage, 10=Zeichen empfangen *)
0011 stat := SystemGetParameterDW( 10313 );
0012
0013 (* Echo-Beispiel: empfangene Daten wieder zurücksenden *)
0014 rxnum := CommRead( comm_nr:=9, dat_adr:=ADR(rxdat[0]), dat_len:=100 );
0015 CommOut( comm_nr:=9, dat_adr:=ADR(rxdat[0]), dat_len:=rxnum );
0016
0017
```

Abb. 7-2: TCP-Server COM 9 mit Echo-Funktion auf Port 3212 (ST)

Der TCP-Server im Beispiel sendet die erhaltenen Empfangsdaten unverändert wieder zurück (Echo).

Der Timeout wurde auf 10.000 Millisekunden (10 Sekunden) gesetzt. Falls innerhalb dieser Zeit keine Zeichen vom Client empfangen werden, beendet der Server die Verbindung.

Der TCP-Server des Beispiels erwartet Client-Anfragen auf Port 3212. Andere Port-Nummern können verwendet werden, sofern sie nicht anderweitig vom System belegt sind (z.B. durch CoDeSys, Webserver, FTP-Server etc.).

8. Universeller TCP-Client

8.1. Allgemeines

Ein TCP-Client stellt eine Datenverbindung über Ethernet zu einem TCP-Server her. Anschließend kann ein Datenaustausch erfolgen.

Für das Herstellen der Verbindung sind IP-Adresse und Port-Nummer des TCP-Servers erforderlich.

Die Port-Nummer ist eine Zahl im Bereich 0...65535, die zur Identifikation eines bestimmten TCP-Servers auf dem angesprochenen Rechner dient. Die Portnummer wird vom Serverbetreiber festgelegt.

Das für den Datenaustausch zu verwendende Protokoll muss bei Verwendung des universellen TCP-Clients im IEC-Programm implementiert werden. Für zahlreiche Standardprotokolle (z.B. Modbus TCP) stehen allerdings vorgefertigte Bibliotheken für PLM-Steuerungen zur Verfügung, so dass der universelle TCP-Client eher selten zum Einsatz kommt.

8.2. Spezifikation

- Universeller TCP-Client
- Konfigurierbare Timeout-Überwachung
- Automatisches Neuverbinden (Reconnect)

8.3. Benötigte Bibliotheken

Benötigt werden die folgenden Bibliotheken:

Plm_TcpClient2.lib
SysLibSockets.lib*
SysLibCallback.lib*
UPD_E_007.lib* (oder spätere Version)
Plm_Std.lib*

* wird von Plm_TcpClient2.lib benötigt

Die angegebenen Bibliothek müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

8.3.1. TcpClient2 (Plm_TcpClient2.lib)

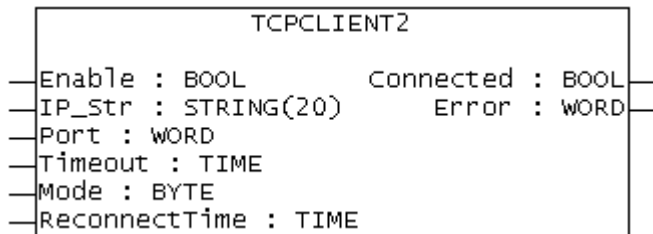


Abb. 8-1: Funktionsblock TcpClient2 (Plm_TcpClient2.lib)

Input-Parameter:		
Enable	BOOL	TRUE: Verbindung wird aufgebaut FALSE: Verbindung wird beendet
IP_Str	STRING(20)	IP-Adresse des Servers (z.B. '10.1.1.231')
Port	WORD	TCP-Portnummer des Servers (z.B. 1234)
Timeout	TIME	Timeout für Verbindungsaufbau,

		z.B. t#10s
Mode	BYTE	Bit 0: 0 = einmaliger Verbindungsaufbau bei Enable 1 = bei Verbindungsabbruch wird nach Ablauf von <code>ReconnectTime</code> automatisch ein neuer Verbindungsversuch gestartet Bit 1: 0 = kein Timeout für Datenübertragung 1 = Timeout gilt auch für Datenübertragung
ReconnectTime	TIME	Wartezeit nach Verbindungsabbruch, bis neuer Verbindungsaufbau gestartet wird, z.B. t#10s (nur wenn <code>Mode.0 = TRUE</code>)
Return-Wert:		
Connected	BOOL	TRUE = Verbindung zu Server hergestellt
Error	WORD	Fehlercode: 0 = kein Fehler 236 = interner Socket-Fehler 239 = Server nicht erreichbar, Verbindungsaufbau gescheitert 241 = Fehler beim Ausführen von <code>TcpClient2ReceiveData()</code> 242 = Fehler beim Ausführen von <code>TcpClient2TransmitData()</code> 243 = Timeout

Zur Realisierung eines TCP-Clients muss eine Instanz des Funktionsblocks `TcpClient2` gebildet und vom Anwenderprogramm zyklisch aufgerufen werden.

Die Version des Laufzeitsystems der PLM-Steuerung sollte mindestens v21012012 sein.

Der Client baut die Verbindung auf, wenn der Eingang `Enable` auf TRUE gesetzt wird. Bei FALSE wird die Verbindung beendet. Sowohl Aufbau als auch Abbau einer TCP-Verbindung benötigen mehrere IEC-Zyklen auf der Steuerung.

`Timeout` ist die Zeit, die ein Verbindungsaufbau maximal dauern darf, z.B. t#10s. Bei TCP-Verbindungen in lokalen Netzen ist, je nach Server, mit Zeiten unter 500 ms zu rechnen.

Eine erfolgreich eingerichtete Verbindung wird mit `Connected = TRUE` signalisiert. Im Fehlerfall ist `Connected = FALSE` und `Error` enthält einen Fehlercode. Die Fehlercodes sind auch als globale Konstanten in der Bibliothek definiert. Ein Datenaustausch (Senden oder Empfangen) kann nur bei `Connected = TRUE` erfolgen.

Die Variable `Mode` legt die Betriebsart des TCP-Clients fest:

Bei `Mode.0 = 0` wird nach `Enable = TRUE` einmalig versucht, die Verbindung aufzubauen. Nach dem Ende der Verbindung muss das IEC-Programm `Enable` für einige Zyklen wieder auf FALSE setzen, bevor ein neuer Verbindungsaufbau möglich ist.

Bei `Mode.0 = 1` wird die Verbindung nach einem Verbindungsabbruch automatisch immer wieder aufgebaut, solange `Enable = TRUE` ist. Die Zeit zwischen zwei Versuchen wird in `ReconnectTime` eingestellt, z.B. t#10s.

Bei `Mode.1 = 0` wird lediglich der Verbindungsaufbau mit `Timeout` überwacht, nicht jedoch der spätere Datenaustausch.

Bei `Mode.1 = 1` erfolgt die Timeout-Überwachung sowohl für den Verbindungsaufbau

als auch für den Datenaustausch. In diesem Fall muss innerhalb der Timeout-Zeit mind. ein Zeichen entweder gesendet oder empfangen werden, andernfalls erfolgt ein Abbruch der Verbindung mit Fehlercode 243.

Das Einschalten der Timeout-Überwachung für den Datenaustausch erfordert ein geeignetes Protokoll, welches regelmäßig Daten austauscht.

8.3.2. TcpClient2TransmitData (Plm_TcpClient2.lib)

```

TCPCLIENT2TRANSMITDATA
- pTcpClient : POINTER TO TcpClient2 TcpClient2TransmitData : DINT
- pBuffer : DWORD
- diBufferSize : DINT
    
```

Abb. 8-2: Funktion TcpClient2TransmitData (Plm_TcpClient2.lib)

Input-Parameter:		
pTcpClient	POINTER TO TcpClient2	Adresse des zugehörigen Client-Bausteins, ADR(client)
pBuffer	DWORD	Adresse eines Buffers mit den Sendedaten, ADR(buffer)
diBufferSize	DINT	Anzahl der zu sendenden Bytes
Return-Wert:		
TcpClient2TransmitData	DINT	Anzahl der tatsächlich gesendeten Bytes

Die Funktion TcpClient2TransmitData () muss vom IEC-Programm aufgerufen werden, um Daten an den Server zu senden.

Die Verknüpfung mit dem zuständigen TCP-Client erfolgt, indem in pTcpClient die Adresse des zuständigen Client-Bausteins vom Typ TcpClient2 angegeben wird.

Ein Senden von Daten ist nur möglich, wenn der Client-Baustein eine Serververbindung erfolgreich aufgebaut hat. Andernfalls liefert TcpClient2TransmitData den Wert 0 zurück. Falls beim Senden ein Verbindungsfehler festgestellt wird, wird die Verbindung des Client-Bausteins beendet.

Der Rückgabewert der Funktion ist die tatsächlich gesendete Anzahl Bytes aus dem angegebenen Buffer.

Normalerweise ist die Anzahl der tatsächlich gesendeten Bytes identisch mit der Angabe in diBufferSize. In extremen Lastsituationen (sehr schnelles Senden vieler langer Messages, langsame Verbindung zum Server über das Internet) kann es jedoch sein, dass nicht alle gewünschten Zeichen mit einem Aufruf von TcpClient2TransmitData () gesendet werden können. In diesem Fall ist das IEC-Programm dafür verantwortlich, die ungesendeten Zeichen aus dem Buffer mit weiteren Aufrufen von TcpClient2TransmitData () zu senden, bis der Buffer vollständig abgearbeitet ist.

Als Buffer dient häufig ein ARRAY[] OF BYTE mit geeigneter Länge, in welches vor dem Versenden die gewünschten Daten eingetragen werden. Bei Verwendung eines Strings als Sendebuffer kann der Wert von diBufferSize mit LEN(string) festgestellt werden. Strings können in CoDeSys eine maximale Länge von 255 Zeichen haben, wenn sie mit STRING(255) deklariert wurden.

8.3.3. TcpClient2ReceiveData (Plm_TcpClient2.lib)

TCPCLIENT2RECEIVEDATA	
pTcpClient	POINTER TO TcpClient2 TcpClient2ReceiveData : DINT
pBuffer	DWORD
diBufferSize	DINT

Abb. 8-3: Funktion TcpClient2ReceiveData (Plm_TcpClient2.lib)

Input-Parameter:		
pTcpClient	POINTER TO TcpClient2	Adresse des zugehörigen Client-Bausteins, $ADR(client)$
pBuffer	DWORD	Adresse eines Buffers für die Empfangsdaten, $ADR(buffer)$
diBufferSize	DINT	max. Größe des Buffers in Bytes
Return-Wert:		
TcpClient2ReceiveData	DINT	Anzahl der tatsächlich empfangenen Bytes

Die Funktion `TcpClient2ReceiveData()` muss vom IEC-Programm aufgerufen werden, um Daten vom Server zu empfangen.

Die Verknüpfung mit dem zuständigen TCP-Client erfolgt, indem in `pTcpClient` die Adresse des zuständigen Client-Bausteins vom Typ `TcpClient2` angegeben wird.

Ein Empfang von Daten ist nur möglich, wenn der Client-Baustein eine Serververbindung erfolgreich aufgebaut hat. Andernfalls liefert `TcpClient2ReceiveData` den Wert 0 zurück. Falls beim Empfang ein Verbindungsfehler festgestellt wird, wird die Verbindung des Client-Bausteins beendet.

Der Rückgabewert der Funktion ist die tatsächlich empfangene Anzahl Bytes, die im angegebenen Buffer abgelegt wurden. Wenn keine Daten empfangen wurden, ist der Rückgabewert 0.

Die Variable `diBufferSize` spezifiziert die Größe des Buffers, um einen Überlauf zu vermeiden. Hierdurch wird gleichzeitig die maximale Anzahl der in einem Aufruf empfangenen Bytes begrenzt. Falls die empfangene TCP-Message mehr Daten als `diBufferSize` enthält, werden diese bei folgenden Aufrufen von `TcpClient2ReceiveData()` geliefert.

Als Buffer dient häufig ein `ARRAY[] OF BYTE` mit geeigneter Länge, welches nach dem Aufruf die ggf. empfangenen Daten enthält. Bei Verwendung eines Strings als Buffer ist unbedingt die maximale Länge des Strings zu berücksichtigen. Strings können in CoDeSys eine maximale Länge von 255 Zeichen haben, wenn sie mit `STRING(255)` deklariert wurden. Ohne explizite Längenangabe haben Strings eine maximale Länge von 80 Zeichen. Falls die empfangenen Daten als String ausgewertet werden sollen, muss das Stringende (Null-Zeichen) anhand des Rückgabewertes durch das IEC-Programm explizit eintragen werden.

Beim Empfang größerer Datenmengen unter Lastbedingungen (langsame Verbindung über das Internet, stark ausgelasteter Server, großes Datenformat) kann im Allgemeinen *nicht* davon ausgegangen werden, dass alle Daten in einer einzigen TCP-Message eintreffen und entsprechend durch einen einzigen Aufruf von `TcpClient2ReceiveData()` empfangen werden. In diesem Fall müssen die empfangenen Bytes aus mehreren Aufrufen von `TcpClient2ReceiveData()` in einem zweiten Buffer gesammelt werden, bis die Empfangsmessage komplett ist.

Vor dem Auswerten der Empfangsmessage sollte daher in jedem Fall eine Längenüberprüfung stattfinden.

8.4. Beispiel für TCP-Client (ST)

```

0001 PROGRAM TpcClientTest2
0002 VAR
0003     client: TpcClient2;
0004     address: STRING := '127.0.0.1';
0005     port: WORD := 21;
0006     enable, ok: BOOL;
0007     error: WORD;
0008     txBuf, rxBuf: ARRAY[0..1023] OF BYTE;
0009     txLen, rxLen: DINT;
0010     doReceive: BOOL := TRUE;
0011     doTransmit: BOOL;
0012 END_VAR
0013
0001 client(
0002     Enable := enable,          (* connect/disconnect *)
0003     IP_Str := address,        (* IP address of server *)
0004     Port := port,            (* TCP port of server *)
0005     Mode := 1,                (* 1 = auto reconnect *)
0006     Timeout := t#10s,        (* timeout for connect *)
0007     ReconnectTime := t#10s,  (* delay for reconnect *)
0008     Connected => ok,        (* connect status *)
0009     Error => error           (* error code *)
0010 );
0011
0012 IF doTransmit THEN
0013     doTransmit := FALSE;
0014     txLen := TpcClient2TransmitData( pTcpClient:=ADR(client), pBuffer:=ADR(txBuf), diBufferSize:=txLen );
0015 END_IF
0016
0017 IF doReceive THEN
0018     rxLen := TpcClient2ReceiveData( pTcpClient:=ADR(client), pBuffer:=ADR(rxBuf), diBufferSize:=1024 );
0019     IF rxLen > 0 THEN
0020         ; (* process data in rxBuf[] *)
0021     END_IF
0022 END_IF
0023

```

Abb. 8-4: Beispiel für TCP-Client (ST)

Wenn die Variable `enable` auf TRUE gesetzt wird, verbindet sich der TCP-Client im Beispiel mit der eigenen Steuerung (localhost, IP-Adresse 127.0.0.1) auf Port 21. An dieser Stelle befindet sich auf PLM 700-Steuerungen der eingebaute FTP-Server, der hier zu Testzwecken verwendet wird.

Solange `doReceive` den Wert TRUE hat, werden Daten vom FTP-Server empfangen, z.B. direkt nach dem Verbinden die sog. Welcome-Message des Servers. Die Empfangsdaten werden in das Byte-Array `rxBuf[]` kopiert.

Wenn `doTransmit` auf TRUE gesetzt wird, werden einmalig `txLen` Bytes aus dem Byte-Array `txBuf[]` an den Server gesendet. Im Falle des FTP-Servers muss es sich hierbei um einen gültigen FTP-Befehl handeln.

9. Modbus RTU (Slave)

9.1. Allgemeines

Modbus RTU erlaubt einen einfachen Datenaustausch zwischen einem Master und einem oder mehreren Slaves. Der Master spricht dabei die Slaves über eine serielle Schnittstelle an, üblicherweise RS485. Für Modbus-Verbindungen über Ethernet steht Modbus TCP zur Verfügung (siehe Abschnitte 11 und 12).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus RTU-Slave.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus-Nutzerorganisation unter <http://modbus.org/> zur Verfügung.

Ein Modbus RTU-Slave stellt dem Master ein Werte-Array zur Verfügung. Die Werte des Arrays können vom Master abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Das Anwenderprogramm stellt ein WORD-Array beliebiger Größe für Word-Zugriffe bereit, sowie ein BYTE-Array beliebiger Größe für Bit-Zugriffe. Mit diesen wird der Baustein `PLM_MODBUSRTUSLAVE` zyklisch aufrufen. Anschließend steht der Slave-Dienst dem Master über die festgelegte serielle Schnittstelle zur Verfügung.

Der alternative Baustein `PLM_MODBUSRTUSLAVE2` bietet die Möglichkeit, mehrere kleinere Datenbereiche in nicht zusammenhängenden Index-Bereichen anzulegen.

Die verwendete serielle Schnittstelle muss vor der Verwendung geeignet initialisiert werden (siehe Abschnitt 2).

9.2. Spezifikation

- Modbus RTU Slave (Slave-ID 1...247)
- Verwendung einer beliebigen seriellen Schnittstelle, z.B. RS485
- Daten-Arrays: max. 65536 Words für Register-Zugriffe, max. 65536 Bytes für Bit-Zugriffe
- Unterstützte Function-Codes (FC): 1 (Read Coils), 2 (Read Discrete Inputs), 3 (Read Holding Registers), 4 (Read Input Registers), 5 (Write Single Coil), 6 (Write Single Register), 15 (Write Multiple Coils), 16 (Write Multiple Registers)
- Realisierung verschiedener Datenmodelle

9.3. Benötigte Bibliotheken

PLM_ModbusRtuSlave_2141219.lib
Plm_Std.lib
UPD_E_005.lib (oder spätere Version)

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

9.3.1. PLM_MODBUSRTUSLAVE (PLM_ModbusRtuSlave.lib)

PLM_MODBUSRTUSLAVE	
ComPort : WORD	Requestok : BOOL
SlaveId : BYTE	
BitData : POINTER TO ARRAY [0..65535] OF BYTE	
NumBitData : WORD	
WordData : POINTER TO ARRAY [0..65535] OF WORD	
NumwordData : WORD	
Enable : BOOL	

Abb. 9-1: Funktionsblock PLM_ModbusRtuSlave (PLM_ModbusRtuSlave.lib)

Input-Parameter:		
ComPort	WORD	Nummer des ComPorts, z.B. 0 für die erste serielle Schnittstelle
SlaveId	BYTE	Slave-Adresse (1...247)
BitData	POINTER TO ARRAY [0..65535] OF BYTE;	Adresse des Byte-Arrays für Function-Codes 1, 2, 5 und 15; kann mit Adresse des Word-Arrays identisch sein
NumBitData	WORD	Größe des Byte-Arrays (Anzahl Bytes in BitData)
WordData	POINTER TO ARRAY [0..65535] OF WORD	Adresse des Word-Arrays für Function-Codes 3, 4, 6 und 16
NumWordData	WORD	Größe des Word-Arrays (Anzahl Words in WordData)
Enable	BOOL	Enable-Flag, bei TRUE arbeitet der Slave
Output-Parameter:		
RequestOk	BOOL	TRUE, wenn während des letzten Aufrufs eine Anfrage erfolgreich bearbeitet wurde

Der Funktionsblock PLM_ModbusRtuSlave muss zyklisch aufgerufen werden.

Bei der Auswahl einer seriellen Schnittstelle (ComPort) ist sicherzustellen, dass diese nicht von anderen Diensten verwendet wird (z.B. CoDeSys, Modem). Ggf. sind andere Dienste zuvor abzuschalten, siehe dazu Abschnitt 2.

9.3.2. PLM_MODBUSRTUSLAVE2 (PLM_ModbusRtuSlave.lib)

PLM_MODBUSRTUSLAVE2	
ComPort : WORD	Requestok : BOOL
SlaveId : BYTE	
BitDataStructAdr : POINTER TO ARRAY [0..99] OF Plm_ModbusRtuSlaveBitStruct	
NumBitDataStruct : WORD	
WordDataStructAdr : POINTER TO ARRAY [0..99] OF Plm_ModbusRtuSlaveWordStruct	
NumWordDataStruct : WORD	
Enable : BOOL	

Abb. 9-2: Funktionsblock PLM_ModbusRtuSlave2 (PLM_ModbusRtuSlave.lib)

Input-Parameter:		
ComPort	WORD	Nummer des ComPorts, z.B. 0 für die erste serielle Schnittstelle
SlaveId	BYTE	Slave-Adresse (1...247)
BitDataStructAdr	POINTER TO ARRAY [] OF Plm_ModbusRtuSlaveBitStruct	Adresse eines Arrays von Bit-Structs für die Function-Codes 1, 2, 5 und 15 (0 wenn nicht benötigt)

NumBitDataStruct	WORD	Größe des Struct-Arrays (0 wenn nicht benötigt)
WordDataStructAdr	POINTER TO ARRAY[] OF Plm_ModbusRtuSlaveWordStruct	Adresse eines Arrays von Word-Structs für die Function-Codes 3, 4, 6 und 16 (0 wenn nicht benötigt)
NumWordDataStruct	WORD	Größe des Struct-Arrays (0 wenn nicht benötigt)
Enable	BOOL	Enable-Flag, bei TRUE arbeitet der Slave
Output-Parameter:		
RequestOk	BOOL	TRUE, wenn während des letzten Aufrufs eine Anfrage erfolgreich bearbeitet wurde

Der Funktionsblock `PLM_ModbusRtuSlave2()` muss zyklisch aufgerufen werden.

Pro Schnittstelle kann nur einer der beiden Bausteine `PLM_ModbusRtuSlave()` oder `PLM_ModbusRtuSlave2()` verwendet werden.

Bei der Auswahl einer seriellen Schnittstelle (`ComPort`) ist sicherzustellen, dass diese nicht von anderen Diensten verwendet wird (z.B. CoDeSys, Modem). Ggf. sind andere Dienste zuvor abzuschalten, siehe dazu Abschnitt 2.

Der Funktionsblock `PLM_ModbusRtuSlave2()` ermöglicht eine Segmentierung der Daten und eine Unterscheidung der Daten nach Function-Code.

Dazu werden an den `PLM_ModbusRtuSlave2()` Arrays vom Typ `Plm_ModbusRtuSlaveBitStruct` (für die Bit-Daten) und `Plm_ModbusRtuSlaveWordStruct` (für die Register-Daten) angelegt. Diese beiden Datentypen sind ebenfalls in der Bibliothek `PLM_ModbusRtuSlave.lib` definiert.

Jedes Array-Element beschreibt einen Teildatenbereich (Datensegment) mit folgenden Informationen:

- `StartIdx` (Adressindex im Modbus-Telegramm),
- `NumBitData/NumWordData` (Anzahl der Bytes bzw. Register, die ab dieser Adresse zur Verfügung gestellt werden),
- `BitDataAdr/WordDataAdr` (Zeiger auf die Datenwerte, üblicherweise die mit dem IEC-Operator `ADR()` ermittelte Speicheradresse eines Daten-Arrays),
- `Fc` (Function-Code, für den dieses Segment gilt).

Aus allen Segmenten zusammen entsteht der über den Modbus in Abhängigkeit vom Function-Code zugreifbare Datenbereich.

Mit dem Funktionsblock `PLM_ModbusRtuSlave2()` können verschiedene Varianten von Modbus-Datenmodellen realisiert werden, z.B. die (historische) Zuweisung von vier getrennten Datenbereichen zu 8 Function-Codes:

Datenbereich	Zugriff durch Function-Code	Virtuelle Bit- bzw. Register-Nummern
Bit-Ausgänge ("Coils")	1 Read Coils 5 Write Single Coil 15 Write Multiple Coils	00000...09999
Bit-Eingänge	2 Read Discrete Inputs	10000...19999
Analog-Eingänge	4 Read Input Registers	30000...39999

16 Bit			
Universal-Register	3	Read Holding Registers	40000...49999
Input/Output	6	Write Single Register	
16 Bit	16	Write Multiple Registers	

Die virtuellen Bit- bzw. Register-Nummern (die gelegentlich auch bei 1 anstelle von 0 beginnen) werden üblicherweise lediglich zur *Benennung* der Bits bzw. Register verwendet, erscheinen jedoch nicht im Modbus-Telegramm, dort beginnt die Adressierung immer mit 0.

Das Programmbeispiel in Abschnitt 9.5 erzeugt einen Slave, der die Zugriffe mit den Function-Codes 3, 4, 6 und 16 gemäß obiger Tabelle realisiert.

9.4. Programmbeispiel (FUP)

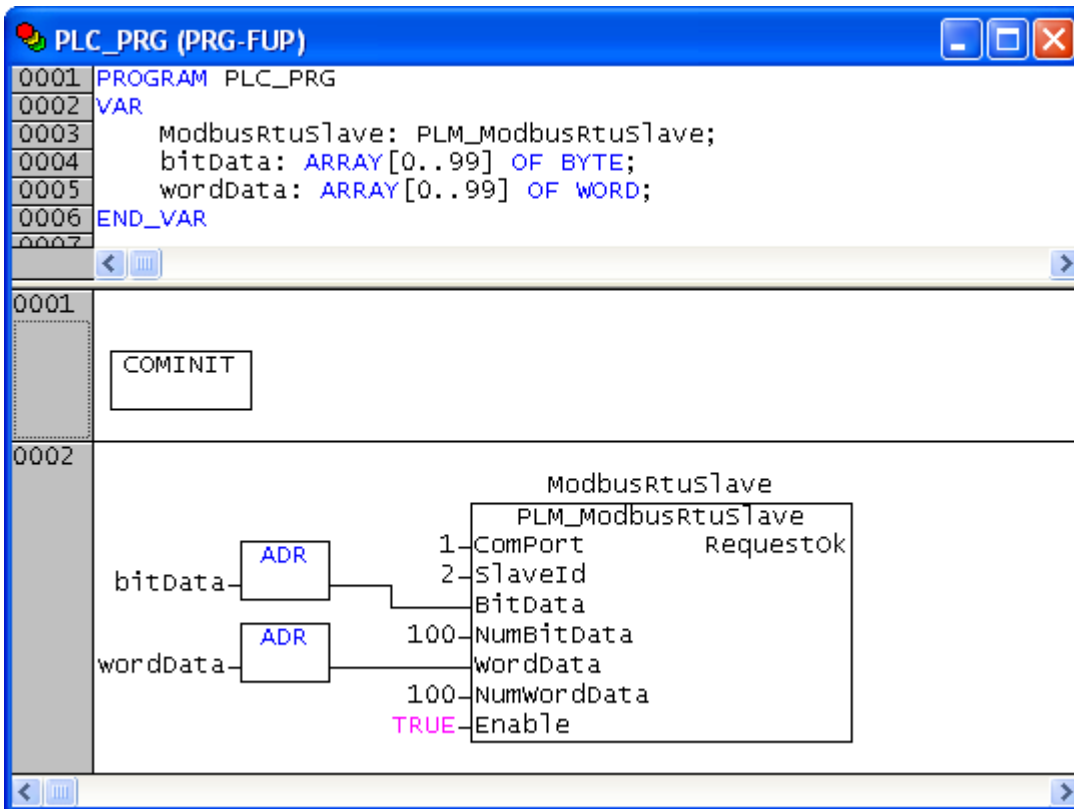


Abb. 9-3: Realisierung eines Modbus RTU-Slaves (FUP, Teil 1/2)

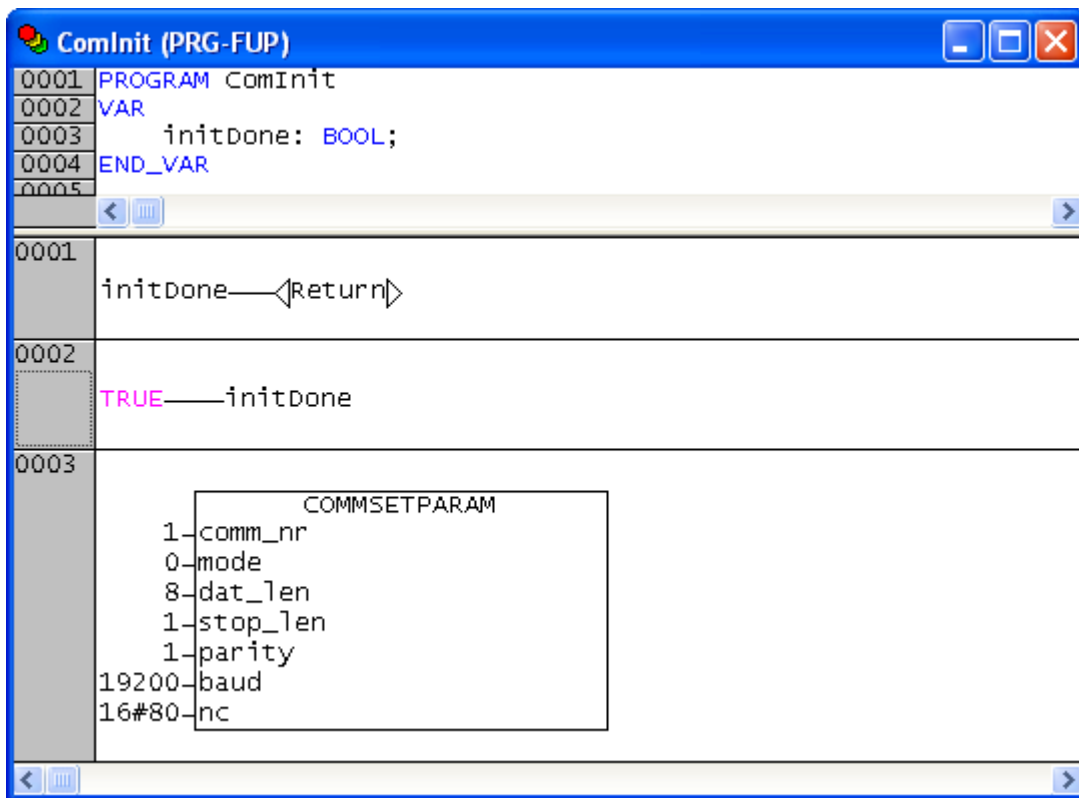


Abb. 9-4: Realisierung eines Modbus RTU-Slaves (FUP, Teil 2/2)

Das in Abb. 11-2 und Abb. 11-3 gezeigte Programmbeispiel realisiert einen Modbus RTU-Slave mit dem Baustein `PLM_MODBUSRTUSLAVE`, der die Arrays `bitData[]` mit 100 Bytes (800 Bits) und `wordData[]` mit 100 Words auf COM-Port 1 unter der Modbus-Slave-Adresse 2 bereitstellt.

Nach dem Start wird zunächst die serielle Schnittstelle COM 1 (RS485 an einer PLM 707) mit den Einstellungen 19200 Baud, 8 Datenbits, 1 Stopbit und Even Parity initialisiert. Die Übertragung mit Even Parity ist häufig die Voreinstellung für Modbus RTU.

Je nach Anwendung muss das Anwenderprogramm dafür sorgen, dass die aktuellen Prozesswerte zyklisch in den Arrays `bitData[]` und `wordData[]` abgelegt bzw. aus diesen gelesen werden.

9.5. Programmbeispiel mit Datensegmentierung (ST)

Das folgende Beispiel erzeugt einen Slave, der die Zugriffe mit den Function-Codes 3, 4, 6 und 16 gemäß der Tabelle am Ende von Abschnitt 9.3.2 realisiert.

Es wird die Bibliothek `PLM_ModbusRtuSlave_2141219.lib` oder eine spätere Version benötigt.

```

VAR
    modbusRtuSlave2: PLM_ModbusRtuSlave2;
    initDone: BOOL;
    (* Segment-Tabelle *)
    segmentTab: ARRAY[0..3] OF Plm_ModbusRtuSlaveWordStruct;
    (* Daten-Arrays *)
    txData_30000: ARRAY[0..15] OF WORD; (* 16 analog inputs; FC 4 *)
    rxData_40000: ARRAY[0..15] OF WORD; (* 16 output registers; FC 3, 6, 16 *)
END_VAR

IF NOT initDone THEN
    initDone := TRUE;

    segmentTab[0].StartIdx := 0;
    segmentTab[0].NumWordData := 16;

```

```
segmentTab[0].WordDataAdr := ADR( rxData_40000 );
segmentTab[0].Fc := 3; (* Read Holding Registers *)

segmentTab[1].StartIdx := 0;
segmentTab[1].NumWordData := 16;
segmentTab[1].WordDataAdr := ADR( txData_30000 );
segmentTab[1].Fc := 4; (* Read Input Registers *)

segmentTab[2].StartIdx := 0;
segmentTab[2].NumWordData := 16;
segmentTab[2].WordDataAdr := ADR( rxData_40000 );
segmentTab[2].Fc := 6; (* Write Single Register *)

segmentTab[3].StartIdx := 0;
segmentTab[3].NumWordData := 16;
segmentTab[3].WordDataAdr := ADR( rxData_40000 );
segmentTab[3].Fc := 16; (* Write Multiple Registers *)
END_IF

modbusRtuSlave2 (
  ComPort := 1,
  SlaveId := 2,
  WordDataStructAdr := ADR(segmentTab),
  NumWordDataStruct := 4,
  Enable := TRUE
);
```

In diesem Beispiel werden vier Datensegmente definiert. Jedes der Datensegmente bedient den speziellen angegebenen Function-Code. Die Segmente 0, 2 und 3 zeigen auf denselben Datenbereich `rxData_40000[]`, während Segment 1 auf `txData_30000[]` zeigt.

Über den Modbus greifen anschließend die Function-Codes 3, 6, und 16 auf das Daten-Array `rxData_40000[]` zu (16-Bit-Daten, die vom Slave empfangen werden), der Function-Code 4 auf das Daten-Array `txData_30000[]` (16-Bit-Daten, die vom Slave gesendet werden).

Die Initialisierung der seriellen Schnittstelle und die Verarbeitung der Datenwerte der beiden Arrays ist hier nicht dargestellt.

10. Modbus RTU (Master)

10.1. Allgemeines

Modbus RTU erlaubt einen einfachen Datenaustausch zwischen einem Master (PLM-Steuerung) und einem oder mehreren Slaves. Der Master spricht dabei die Slaves über eine serielle Schnittstelle an, üblicherweise RS485. Für Modbus-Verbindungen über Ethernet steht Modbus TCP zur Verfügung (siehe Abschnitte 11 und 12).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus RTU-Master.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus-Nutzerorganisation unter <http://modbus.org/> zur Verfügung.

Ein Modbus RTU-Master greift auf eine Werte-Tabelle zu, welcher jeder Slave zur Verfügung stellt. Die Werte können physikalischen Ein- oder Ausgängen des Slaves entsprechen. Werte können gelesen (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Register (16-Bit-Words) oder Coils (Bits). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Die PLM-Bibliothek für den Modbus RTU-Master stellt einen Funktionsblock `PLM_MODBUSMASTERTRANSFER` zur Verfügung, der ein Array von Werten aus einem bestimmten Slave liest (Read) oder in diesen schreibt (Write). Es kann eine beliebige Anzahl von Instanzen des Funktionsblocks verwendet werden, um z.B. mit verschiedenen Slaves zu kommunizieren oder um verschiedene Zugriffe auf einen Slave durchzuführen.

Der Zugriff auf die Slaves erfolgt normalerweise zyklisch, d.h. die Werte in den Slaves werden regelmäßig gelesen oder geschrieben. Ebenso ist jedoch auch ein programmgesteuerter einmaliger Zugriff möglich, z.B. Schreibzugriffe in der Initialisierungsphase.

Bei Verwendung des Funktionsblocks `PLM_MODBUSMASTERTRANSFER` braucht der Anwender sich nicht um die zeitliche Abfolge der verschiedenen Slave-Zugriffe zu kümmern; der Ablauf wird komplett von der Bibliothek koordiniert.

Die verwendete serielle Schnittstelle muss vor der Verwendung geeignet initialisiert werden (siehe Abschnitt 2).

10.2. Spezifikation

- Modbus RTU Master
- Verwendung einer beliebigen seriellen Schnittstelle, z.B. RS485
- Lese- und Schreibzugriffe auf beliebig viele Slaves (max. 246 gemäß Modbus-Spezifikation)
- Automatische Steuerung des zeitlichen Ablaufs der Slave-Zugriffe
- Unterstützte Function-Codes (FC): 1 (Read Coils), 2 (Read Discrete Inputs), 3 (Read Registers), 4 (Read Input Registers), 5 (Write Single Coil), 6 (Write Single Register), 15 (Write Multiple Coils), 16 (Write Multiple Registers)

10.3. Benötigte Bibliotheken

Plm_ModbusRtu.lib
Plm_Std.lib*

* wird zur Initialisierung der seriellen Schnittstelle benötigt

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

10.3.1. PLM_ModbusMasterInit (Plm_ModbusRtu.lib)

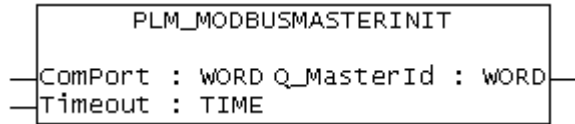


Abb. 10-1: Funktionsblock PLM_ModbusMasterInit (Plm_ModbusRtu.lib)

Input-Parameter:		
ComPort	WORD	Nummer des ComPorts, z.B. 0 für die erste serielle Schnittstelle
Timeout	TIME	Timeout-Zeit für Slave-Antwort, z.B. t#100ms
Output-Parameter:		
Q_MasterId	WORD	ID dieser Master-Instanz; wird am Funktionsblock PLM_MODBUSMASTERTRANSFER benötigt

Der Funktionsblock PLM_ModbusMasterInit darf nur einmalig beim Start des Programms aufgerufen werden und initialisiert die internen Datenstrukturen. Das gelieferte Handle Q_MasterId wird anschließend für die Instanzen von PLM_ModbusMasterTransfer benötigt. Bei Bedarf können mehrere Master auf verschiedenen Schnittstellen instanziiert werden. Die eindeutige Zuordnung zu diesen Mastern erfolgt mittels des jeweils gelieferten Handles Q_MasterId.

Bei der Auswahl einer seriellen Schnittstelle (ComPort) ist sicherzustellen, dass diese nicht von anderen Diensten verwendet wird (z.B. CoDeSys, Modem). Ggf. sind andere Dienste zuvor abzuschalten, siehe dazu Abschnitt 2.

10.3.2. PLM_ModbusMasterTransfer (Plm_ModbusRtu.lib)

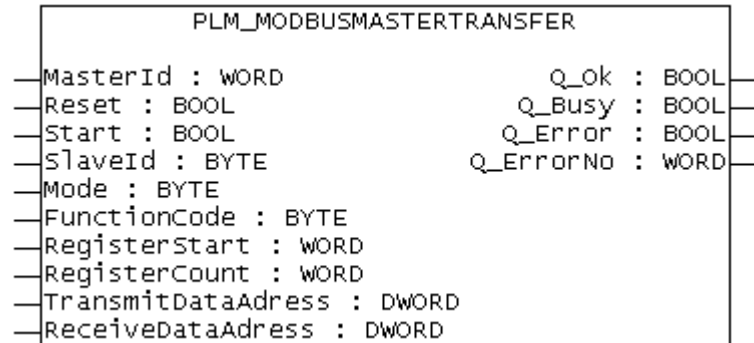


Abb. 10-2: Funktionsblock PLM_ModbusMasterTransfer (Plm_ModbusRtu.lib)

Input-Parameter:		
MasterId	WORD	Handle der Master-Instanz (PLM_MODBUSMASTERINIT), der dieser Zugriff zugeordnet werden soll
Reset	BOOL	TRUE bewirkt Rücksetzen des Bausteins
Start	BOOL	TRUE, damit Baustein abgearbeitet wird
SlaveId	BYTE	Adresse des angesprochenen Slaves (1...247) oder 0 für Broadcast
Mode	BYTE	Betriebsart dieses Bausteins: 0 = einzelner Slave-Zugriff, wird erst wiederholt, wenn Start FALSE und wieder TRUE wird 1 = Slave-Zugriff wird zyklisch durchgeführt
FunctionCode	BYTE	Art des Slave-Zugriffs. Mögliche Function

		<p>Codes: 1 = Read Coils (1...2000 Bits lesen), 2 = Read Discrete Inputs (1...2000 Bits lesen), 3 = Read Registers (1...125 Words lesen), 4 = Read Input Registers (1...125 Words lesen), 5 = Write Single Coil (1 Bit schreiben), 6 = Write Single Register (1 Word schreiben), 15 = Write Multiple Coils (1...1968 Bits schreiben), 16 = Write Multiple Registers (1...123 Words schreiben)</p>
RegisterStart	WORD	Adresse des ersten zu lesenden/schreibenden Bits/Registers (0...65535)
RegisterCount	WORD	Anzahl der zu lesenden/schreibenden Bits/Register, FC 1 oder FC 2: 1...2000 FC 3 oder FC 4: 1...125 FC 5 oder FC 6: (keine Bedeutung) FC 15: 1...1968 FC 16: 1...123
TransmitDataAdress	DWORD	Adresse der Variablen/des Arrays mit den Sendedaten
ReceiveDataAdress	DWORD	Adresse der Variablen/des Arrays für die Empfangsdaten
Output-Parameter:		
Q_Ok	BOOL	TRUE = Slave-Zugriff erfolgreich beendet, Empfangsdaten gültig
Q_Busy	BOOL	TRUE = Slave-Zugriff läuft noch
Q_Error	BOOL	TRUE = Fehler bei Slave-Zugriff aufgetreten, Fehlercode in Q_ErrorNo
Q_ErrorNo	WORD	Nummer des aufgetretenen Fehlers, Slave-Meldungen: 16#xx01 = Unzulässiger Functioncode 16#xx02 = Unzulässige Datenadresse 16#xx03 = Unzulässiger Datenwert Master-Meldungen: 16#81xx = Keine Antwort vom Slave nach Timeoutzeit, Retries fehlgeschlagen 16#82xx = Prüfsummen-Error in Slave-Antwort 16#83xx = Unzulässiger Function Code am Eingang FunctionCode 16#84xx = Function Code der Slave-Antwort falsch 16#85xx = unzulässige Anzahl Register/Coils am Eingang RegisterCount 16#86xx = interner Fehlercode 16#87xx = interner Fehlercode, keine Antwort vom Slave nach Timeoutzeit, versuche Retry 16#88xx = Falsche Adresse an Eingang

		TransmitDataAdress
--	--	--------------------

Üblicherweise verwendet ein Programm mehrere Instanzen von `PLM_ModbusMasterTransfer`, z.B. um mehrere Slaves anzusprechen oder um bei einem Slave sowohl Werte zu schreiben als auch zu lesen. Die Bibliothek sorgt dafür, dass die Slave-Zugriffe automatisch in Reihenfolge der Bausteine im Programm durchgeführt werden. Wenn alle vorhandenen Bausteine abgearbeitet sind, d.h. die Slave-Zugriffe stattgefunden haben, wird wieder mit dem ersten Baustein begonnen.

Bei Bausteinen mit `Mode = 0` wird der Zugriff nur einmalig ausgeführt; dies kann z.B. zur Initialisierung von Slaves nach dem Programmstart verwendet werden. Bausteine mit `Mode = 1` werden ständig zyklisch abgearbeitet; dies dient üblicherweise dem Übertragen von Prozessdaten im Betrieb.

Die am Eingang `TransmitDataAdress` und `ReceiveDataAdress` angegebenen Datenspeicher müssen in jedem Fall die durch `FunctionCode` und `RegisterCount` bestimmte Anzahl an Datenwerten aufnehmen können. Die benötigten Speicheradressen werden üblicherweise mit der `ADR()`-Funktion ermittelt.

Die von einem bestimmten Slave zugelassenen Function Codes sowie die verfügbaren Register sind der Geräte-Dokumentation des jeweiligen Slaves zu entnehmen.

10.4. Ablauf im Normalbetrieb und im Fehlerfall

Der Modbus-RTU-Master arbeitet alle Bausteine vom Typ `PLM_ModbusMasterTransfer` nach einem internen Schema nacheinander zyklisch ab.

Der Master sendet dabei für jeden Transferbaustein zunächst eine Anfrage (Request) auf die serielle Schnittstelle und erwartet darauf eine Antwort (Response) des angesprochenen Slaves. Dabei kann es sich um eine Ok-Response oder um eine Error-Response handeln. Die Antwort wird dann dem entsprechenden Transfer-Baustein zugeordnet.

Falls im Fehlerfall innerhalb der Timeout-Zeit (Parameter `Timeout`) keine Antwort vom Slave empfangen wird, wiederholt der Master die Anfrage. Dies geschieht normalerweise bis zu dreimal (Parameter `NumTries`).

Wenn dann noch immer keine Antwort empfangen wurde, wird der entsprechende Transferbaustein zurückgestellt und bei den nächsten Abfragen übersprungen. Dadurch wird verhindert, dass der Modbus jedesmal durch den mehrfachen Ablauf der Timeout-Zeit blockiert wird.

Nach einigen Abfragezyklen (Parameter `SkipCount`) wird der zurückgestellte Transferbaustein erneut abgefragt. Dadurch wird ein Slave, der wegen einer kurzzeitigen Störung keine Anfragen beantwortet, nach einiger Zeit automatisch wieder in den Abfrage-Zyklus integriert.

Der Parameter `Timeout` wird bei der Initialisierung im Baustein `PLM_ModbusMasterInit` angegeben (siehe Abschnitt 10.3.1).

Die Parameter `NumTries` und `SkipCount` können ab Laufzeitsystem v21012012 wie folgt eingestellt werden:

```
SystemSetParameter(
    ParameterID := 4200 + Q_MasterId,
    Value := NumTries
);

SystemSetParameter(
    ParameterID := 4220 + Q_MasterId,
    Value := SkipCount
);
```

Der Wert von `Q_MasterId` entspricht dem Ausgang des entsprechenden Bausteins `PLM_ModbusMasterInit`. Die Voreinstellung der Parameter ab Laufzeitsystem v21012012 ist `NumTries = 3` und `SkipCount = 25`.

10.5. Programmbeispiel (ST)

```

PLC_PRG (PRG-ST)
0001 PROGRAM PLC_PRG
0002 VAR
0003     ModbusMasterInit: PLM_ModbusMasterInit;
0004     ModbusMasterTransfer1: PLM_ModbusMasterTransfer;
0005     ModbusMasterTransfer2: PLM_ModbusMasterTransfer;
0006     initDone: BOOL;
0007     mb_fh: WORD;
0008     dat1: ARRAY[0..1] OF WORD;
0009     dat2: ARRAY[0..1] OF WORD;
0010 END_VAR
0011
0001 IF NOT initDone THEN
0002     initDone := TRUE;
0003     (* Com 1 RS485 Init für Modbus (19200 Baud, 8 Databits, Even Parity, 1 Stopbit) *)
0004     CommSetParam( comm_nr:=1, mode:=0, dat_len:=8, stop_len:=1, parity:=1, baud:=19200, nc:=16#80 );
0005     ModbusMasterInit( ComPort := 1, Timeout := t#100ms, Q_MasterId => mb_fh );
0006 END_IF
0007
0008 ModbusMasterTransfer1(
0009     MasterId := mb_fh,
0010     Reset := FALSE,
0011     Start := TRUE,
0012     SlaveId := 2, (* slave-Adresse 2 *)
0013     Mode := 1, (* zyklisch ausführen *)
0014     FunctionCode := 3, (* FC 3 = Read Holding Registers *)
0015     RegisterStart := 300, (* Start-Register = 300 *)
0016     RegisterCount := 2, (* 2 werte abfragen *)
0017     TransmitDataAddress := 0, (* FC 3 braucht keine Sendedaten *)
0018     ReceiveDataAddress := ADR( dat1 ), (* Empfangsdaten in dat1[] ablegen *)
0019 );
0020
0021 ModbusMasterTransfer2(
0022     MasterId := mb_fh,
0023     Reset := FALSE,
0024     Start := TRUE,
0025     SlaveId := 3, (* slave-Adresse 3 *)
0026     Mode := 1, (* zyklisch ausführen *)
0027     FunctionCode := 3, (* FC 3 = Read Holding Registers *)
0028     RegisterStart := 300, (* Start-Register = 300 *)
0029     RegisterCount := 2, (* 2 werte abfragen *)
0030     TransmitDataAddress := 0, (* FC 3 braucht keine Sendedaten *)
0031     ReceiveDataAddress := ADR( dat2 ), (* Empfangsdaten in dat2[] ablegen *)
0032 );
0033

```

Abb. 10-3: Realisierung eines Modbus RTU Masters (ST)

Nach dem Start wird zunächst die serielle Schnittstelle COM 1 (RS485 an einer PLM 707) mit den Einstellungen 19200 Baud, 8 Databits, Even Parity und 1 Stopbit initialisiert. Die Übertragung mit Even Parity ist häufig die Voreinstellung für Modbus RTU. Anschließend wird eine Instanz von `PLM_ModbusMasterInit` auf COM-Port 1 gebildet mit einer Timeout-Zeit von 100 ms. Das Handle `mb_fh` wird anschließend am Eingang `MasterId` von `PLM_ModbusMasterTransfer` angegeben.

Im Beispiel erfolgt ein zyklischer Zugriff (`Mode = 1`) auf zwei Slaves mit den Adressen 2 und 3. Aus beiden Slaves werden zwei Datenworte ab der Registeradresse 300 ausgelesen (`FunctionCode = 3`, `RegisterStart = 300`, `RegisterCount = 2`). Die gelesenen Datenworte werden in den Arrays `dat1[]` und `dat2[]` abgelegt.

Die Transferbausteine werden von der Bibliothek automatisch so koordiniert, dass immer eine Slave-Anfrage vollständig abgeschlossen wird bevor die nächste beginnt.

Der Function Code 3 (Read Holding Registers) benötigt keine Sendedaten (vgl. Abschnitt 10.3.2), daher braucht am Eingang `TransmitDataAddress` keine Adresse angegeben zu werden. Dem Eingang `ReceiveDataAddress` wird die Adresse des jeweiligen Empfangsdaten-Arrays zugewiesen. Das Array muss groß genug sein, um die angeforderte Anzahl von Werten aufzunehmen. In diesem Fall (`FunctionCode = 3`, d.h. Word-Zugriff und `RegisterCount = 2`) muss das Array vom Typ WORD sein und mindestens zwei Werte aufnehmen können.

Die Werte in `dat1[]` und `dat2[]` sind erst gültig, nachdem der jeweilige Zugriff mindestens einmal erfolgreich abgeschlossen wurde, was durch `Ok = TRUE` angezeigt wird. Dies wird im Beispiel nicht ausgewertet.

10.6. Programmbeispiel (FUP)

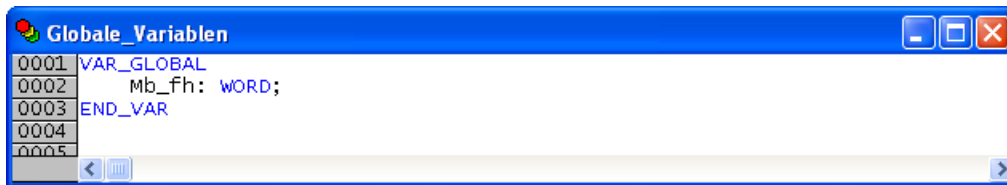


Abb. 10-4: Realisierung eines Modbus RTU Masters (FUP, Teil 1/3)

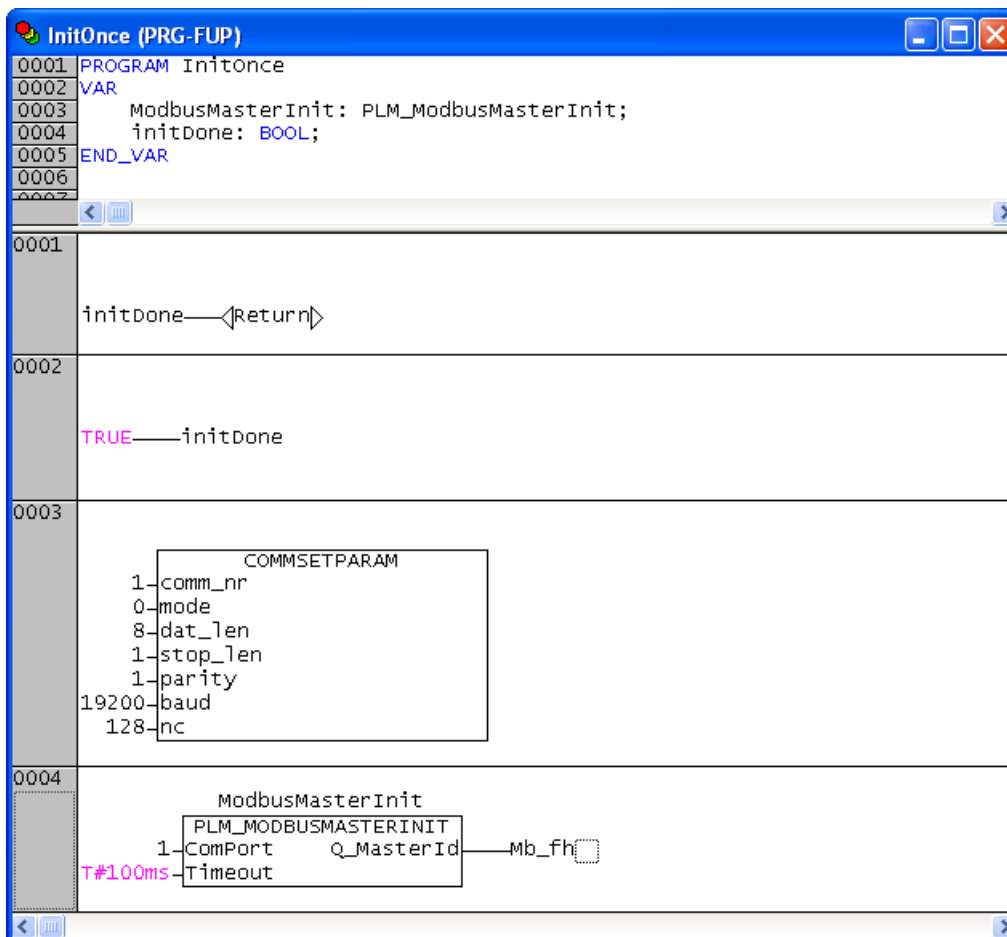


Abb. 10-5: Realisierung eines Modbus RTU Masters (FUP, Teil 2/3)

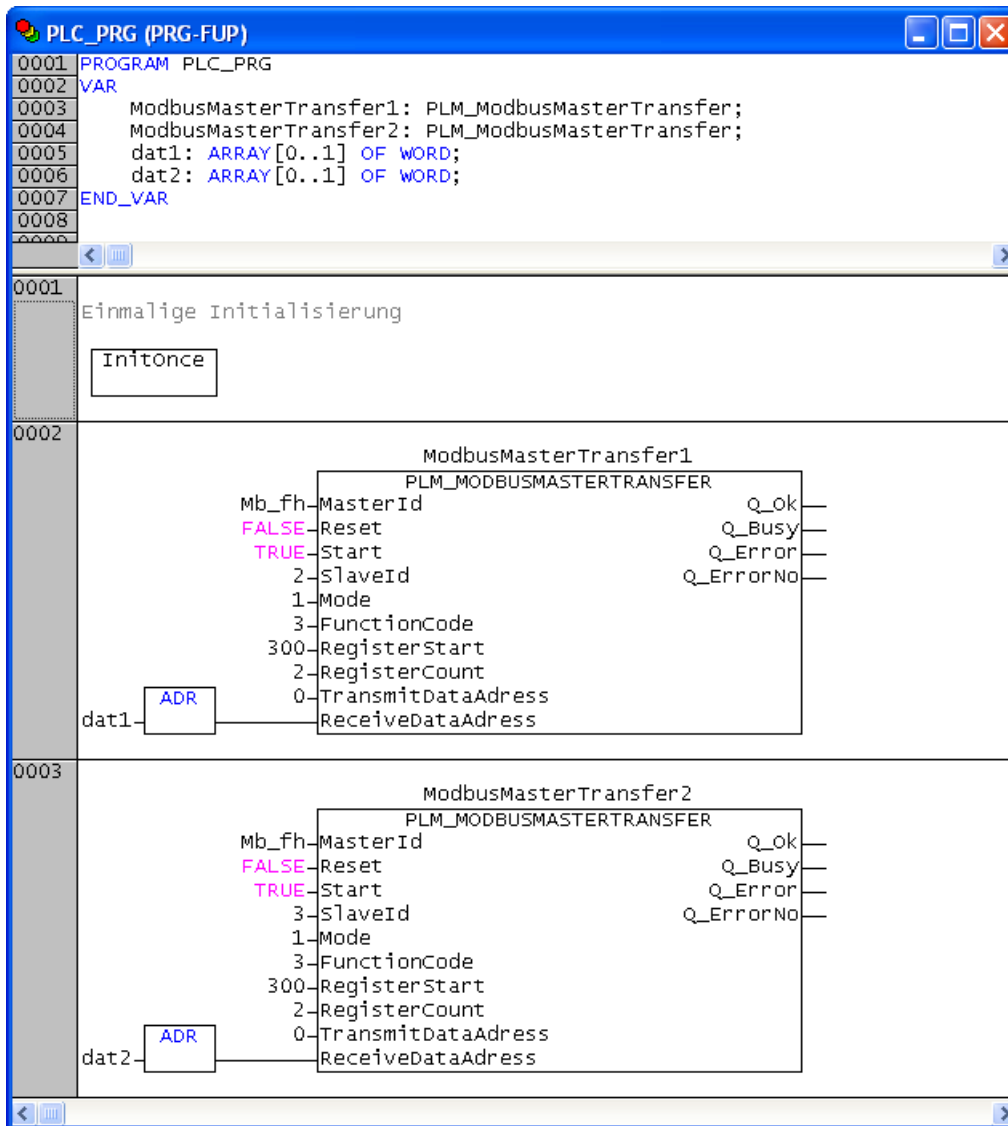


Abb. 10-6: Realisierung eines Modbus RTU Masters (FUP, Teil 3/3)

Die Arbeitsweise ist analog zum Beispiel in ST (siehe Abschnitt 10.5). Allerdings ist die einmalige Initialisierung nach dem Start in einen eigenen Funktionsblock `InitOnce` ausgelagert. Da das Handle `Mb_fh` sowohl in `InitOnce` als auch in `PLC_PRG` verwendet wird, ist es hier als Globale Variable angelegt.

Auch hier werden die Transferbausteine von der Bibliothek automatisch so koordiniert, dass immer eine Slave-Anfrage vollständig abgeschlossen wird bevor die nächste beginnt.

11. Modbus TCP (Server)

11.1. Allgemeines

Modbus TCP erlaubt einen einfachen Datenaustausch zwischen einem oder mehreren Clients und einem Server. Die Clients sprechen den Server dabei über eine Ethernet-Verbindung (TCP) an. Für Modbus-Verbindungen über die serielle Schnittstelle (RS232/RS485) steht Modbus RTU zur Verfügung (siehe Abschnitt 7 und 9.1).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus TCP-Server.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus-Nutzerorganisation unter <http://modbus.org/> zur Verfügung.

Für den Zugriff über Ethernet muss die IP-Adresse der Steuerung bekannt sein, auf der der Modbus TCP-Server läuft. Außerdem müssen sowohl der Server als auch die Clients so konfiguriert sein, dass sie miteinander über Ethernet kommunizieren können. Allgemeine Hinweise zu Netzwerkverbindungen über Ethernet finden sich im System-Handbuch Teil 1.

Ein Modbus-Server stellt den Clients ein Werte-Array zur Verfügung. Die Werte des Arrays können von den Clients abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Das Anwenderprogramm stellt ein WORD-Array beliebiger Größe für Word-Zugriffe bereit, sowie ein BYTE-Array beliebiger Größe für Bit-Zugriffe. Mit diesen wird der Baustein `MODBUSSERVER_TCPIP` zyklisch aufgerufen. Anschließend steht der Server-Dienst den Clients über Ethernet zur Verfügung (standardmäßig auf Port 502).

11.2. Spezifikation

- Modbus TCP Server, Port wählbar (standardmäßig 502)
- Max. Anzahl gleichzeitiger Client-Verbindungen: 64
- Daten-Arrays: max. 65536 Words und max. 65536 Bytes für Bit-Zugriffe
- Unterstützte Function-Codes (FC): 1 (Read Coils), 2 (Read Discrete Inputs), 3 (Read Holding Registers), 4 (Read Input Registers), 5 (Write Single Coil), 6 (Write Single Register), 15 (Write Multiple Coils), 16 (Write Multiple Registers), 23 (Read/Write Multiple Registers)

11.3. Benötigte Bibliotheken

Plm_ModbusTcpSrv.lib
SysLibSockets.lib*
SysLibCallback.lib*
SysLibMem.lib*
UPD_E_005.lib* (oder spätere Version)
Plm_Std.lib*

* wird von Plm_ModbusTcpSrv.lib benötigt

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

11.3.1. MODBUSSERVER_TCPIP (Plm_ModbusTcpSrv.lib)

MODBUSSERVER_TCPIP	
wPort : WORD	dwNumReadBits : DWORD
pBitData : POINTER TO ARRAY [0..65535] OF BYTE	dwNumReadRegisters : DWORD
wMaxBitData : WORD	dwNumWriteBit : DWORD
pWorkingData : POINTER TO ARRAY [0..65535] OF WORD	dwNumWriteRegister : DWORD
wMaxWorkingData : WORD	dwNumWriteBits : DWORD
bEnableModSrv : BOOL	dwNumWriteRegisters : DWORD
iModBusState : MODBUS_SERVER_STATE	dwNumReadwriteRegs : DWORD
	dwNumExceptions : DWORD
	bReady : BOOL
	iClientNumber : INT

Abb. 11-1: Funktionsblock MODBUSSERVER_TCPIP (Plm_ModbusTcpSrv.lib)

Input-Parameter:		
wPort	WORD	TCP-Port, auf dem der Dienst bereitgestellt wird; Voreinstellung: 502
pBitData	POINTER TO ARRAY[0..65535] OF BYTE	Adresse des Byte-Arrays für Coil-Zugriffe (Function-Codes 1, 2, 5 und 15)
wMaxBitData	WORD	Anzahl Bytes in pBitData
pWorkingData	POINTER TO ARRAY[0..65535] OF WORD	Adresse des Word-Arrays für Register-Zugriffe (Function-Codes 3, 4, 6, 16 und 23)
wMaxWorkingData	WORD	Anzahl Words in pWorkingData
bEnableModSrv	BOOL	TRUE = Server starten, FALSE = Server anhalten, bestehende Client-Verbindungen werden dabei getrennt
iModBusState	BOOL	Interner Zustand des Servers (darf nur gelesen werden). Nur im Zustand MODBUSSERVER_LISTEN können Client-Verbindungen aufgebaut werden.
Output-Parameter:		
dwNumReadBits, dwNumReadRegisters, dwNumWriteBit, dwNumWriteRegister, dwNumWriteBits, dwNumWriteRegisters, dwNumReadwriteRegs, dwNumExceptions	DWORD	Event-Counter für Modbus-Client-Zugriffe
bReady	BOOL	TRUE = Server ist bereit für Client-Verbindungen
iClientNumber	INT	Anzahl der verbundenen Clients

Die Bibliothek Plm_ModbusTcpSrv.lib stellt einen Funktionsblock als globale Variable ModbusTcpServer[0] zur Verfügung. Diese Variable muss für den zyklischen Aufruf verwendet werden (siehe Programmbeispiel).

Coil-Werte werden auf dem Modbus immer in Achtergruppen zu einem Byte zusammengefasst, wobei immer acht Coil-Werte in einem Byte von `pBitData` gespeichert werden. Unabhängig davon können aber mit den entsprechenden Modbus-Function-Codes auch einzelne Coil-Werte beschrieben oder abgefragt werden.

Der Parameter `wMaxBitData` enthält die Array-Größe von `pBitData` in Bytes, nicht die Anzahl der Coils.

Coil-Adressen können auf dem Modbus zwischen 0 und 65535 liegen, somit ist eine maximale Array-Größe von `pBitData` von 8192 Bytes (=65536 Coils) sinnvoll.

In der momentanen Version der Bibliothek sind Coil-Zugriffe mit den Function-Codes 1, 2 und 15 (Read/Write Multiple Coils) nur auf Byte-Grenzen möglich, d.h. die Coil-Startadresse im Request muss durch 8 teilbar sein (0, 8, 16, 24 etc.)

Wenn der Server durch einen CoDeSys-Event (z.B. *Reset Kalt*) gestoppt und anschließend wieder gestartet wird, kann es vorkommen, dass der angegebene Server-Port (üblicherweise 502) vom Betriebssystem der Steuerung ca. 1 Minute lang blockiert ist. Der Server wartet dann bei `bEnableModSrv = TRUE` solange mit `bReady = FALSE` im Zustand `MODBUS_SERVER_BINDSOCKET`, bis der Port wieder zur Verfügung steht. Anschließend wird der Port wieder automatisch belegt und Client-Verbindungen werden zugelassen (`bReady = TRUE`).

Falls alle 64 möglichen Client-Verbindungen belegt sind und eine weitere Verbindungsanfrage von einem Client eintrifft, wird die älteste Verbindung (auf der am längsten keine Modbus-Anfrage mehr bearbeitet wurde) geschlossen und dem neuen Client zugewiesen.

Der Server verwendet intern ein Array `Client[1..64]`. In diesem Array werden die bis zu 64 möglichen Client-Verbindungen gespeichert. Die Elemente des Arrays sind vom Typ `MODBUS_CLIENT_ACCEPT`, der ebenfalls in der Bibliothek `Plm_ModbusTcpSrv.lib` deklariert ist:

```

TYPE MODBUS_CLIENT_ACCEPT:
STRUCT
    diSocket:    DINT;
    stIPAddress: STRING(20);
    activity:    BOOL;
    accessTime:  DWORD;
END_STRUCT
END_TYPE

```

Die Elemente von `MODBUS_CLIENT_ACCEPT` haben folgende Bedeutung:

MODBUS_CLIENT_ACCEPT:		
<code>diSocket</code>	DINT	Internes Socket-Handle der Client-Verbindung. Wenn kein Client verbunden ist: <code>SOCKET_INVALID</code>
<code>stIPAddress</code>	STRING(20)	IP-Adresse des verbundenen Clients
<code>activity</code>	BOOL	Bei Client-Aktivität (Modbus Read oder Write): TRUE für einen Zyklus
<code>accessTime</code>	DWORD	interner Zähler

Auf die Elemente des Arrays kann zu Diagnosezwecken in der Form `ModbusTcpServer[0].Client[n]` mit `n = 1..64` zugegriffen werden, dabei ist nur lesender Zugriff möglich.

11.4. Programmbeispiel (ST und FUP)

```

Server (PRG-ST)
0001 PROGRAM Server
0002 VAR
0003     CoilData: ARRAY[0..15] OF BYTE;
0004     RegisterData: ARRAY[0..15] OF WORD;
0005 END_VAR
0006
0001
0002 ModbusTcpServer [0](
0003     wPort := 502,
0004     pBitData := ADR( CoilData ),
0005     wMaxBitData := 16,
0006     pworkingData := ADR( RegisterData ),
0007     wMaxworkingData := 16,
0008     bEnableModSrv := TRUE
0009 );
0010
    
```

Abb. 11-2: Realisierung eines Modbus TCP-Servers (ST)

```

Server (PRG-FUP)
0001 PROGRAM Server2
0002 VAR
0003     CoilData: ARRAY[0..15] OF BYTE;
0004     RegisterData: ARRAY[0..15] OF WORD;
0005 END_VAR
0006
0007
0001
    ModbusTcpServer [0]
    MODBUSSERVER_TCPIP
    502-wPort dwNumReadBits
    ADR(CoilData)-pBitData dwNumReadRegisters
    16-wMaxBitData dwNumWriteBit
    ADR(RegisterData)-pworkingData dwNumWriteRegister
    16-wMaxworkingData dwNumWriteBits
    TRUE-bEnableModSrv dwNumWriteRegisters
    -iModbusState dwNumReadwriteRegs
    dwNumExceptions
    bReady
    iclientNumber
    
```

Abb. 11-3: Realisierung eines Modbus TCP-Servers (FUP)

Das in Abb. 11-2 und Abb. 11-3 gezeigte Programmbeispiel realisiert einen Modbus TCP-Server, der das Array RegisterData[] mit 16 Words und das Array CoilData[] mit 16 Bytes (= 128 Coils) auf Port 502 bereitstellt.

Der Funktionsbaustein ModbusTcpServer [0] wird durch die Bibliothek Plm_ModbusTcpSrv.lib als globale Variable bereitgestellt und muss zyklisch aufgerufen werden.

Je nach Anwendung muss das Anwenderprogramm dafür sorgen, dass die aktuellen Prozesswerte zyklisch in den Arrays RegisterData[] und CoilData[] abgelegt bzw. aus diesen gelesen werden.

12. Modbus TCP (Client)

12.1. Allgemeines

Modbus TCP erlaubt einen einfachen Datenaustausch zwischen einem oder mehreren Clients und einem Server. Die Clients sprechen den Server dabei über eine Ethernet-Verbindung (TCP) an. Für Modbus-Verbindungen über die serielle Schnittstelle (RS232/RS485) steht Modbus RTU zur Verfügung (siehe Abschnitt 7 und 9.1).

Dieser Abschnitt beschreibt die Arbeitsweise einer PLM-Steuerung als Modbus TCP-Client.

Weitere Informationen zum Modbus-Protokoll stehen auf der Website der Modbus-Nutzerorganisation unter <http://modbus.org/> zur Verfügung.

Für den Zugriff über Ethernet muss die IP-Adresse der Steuerung bekannt sein, auf der der Modbus TCP-Server läuft. Außerdem müssen sowohl der Server als auch die Clients so konfiguriert sein, dass sie miteinander über Ethernet kommunizieren können. Allgemeine Hinweise zu Netzwerkverbindungen über Ethernet finden sich im System-Handbuch Teil 1.

Ein Modbus-Server stellt den Clients ein Werte-Array zur Verfügung. Die Werte des Arrays können von den Clients abgefragt (Read) oder beschrieben werden (Write). Das Modbus-Protokoll erlaubt den Zugriff auf Words (Register) oder Bits (Coils). Die Art des jeweiligen Zugriffs wird über einen Function Code (FC) beim Zugriff festgelegt.

Der PLM Modbus-TCP-Client wird durch zyklischen Aufruf des eigentlichen Client-Bausteins `Plm_ModbusTcpClient` und der daran angehängten Datentransferbausteine `Plm_ModbusTcpClientTransfer` realisiert. Der Client-Baustein stellt die Netzwerk-Verbindung zum Modbus TCP Server her, während die Transferbausteine darüber ihren Datenaustausch abwickeln.

12.2. Spezifikation

- Modbus TCP Client
- Unterstützte Function-Codes (FC): 1 (Read Coils), 2 (Read Discrete Inputs), 3 (Read Holding Registers), 4 (Read Input Registers), 5 (Write Single Coil), 6 (Write Single Register), 15 (Write Multiple Coils), 16 (Write Multiple Registers)

12.3. Benötigte Bibliotheken

<code>Plm_ModbusTcpClient.lib</code>
<code>SysLibSockets.lib*</code>
<code>SysLibCallback.lib*</code>
<code>SysLibMem.lib*</code>
<code>UPD_E_005.lib*</code> (oder spätere Version)
<code>Plm_Std.lib*</code>

* wird von `Plm_ModbusTcpClient.lib` benötigt

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

12.3.1. PLM_MODBUSTCPCLIENT (Plm_ModbusTcpClient.lib)

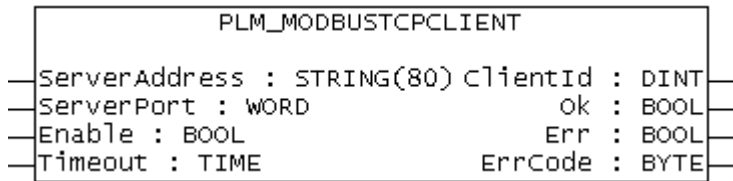


Abb. 12-1: Funktionsblock PLM_MODBUSTCPCLIENT (Plm_ModbusTcpClient.lib)

Input-Parameter:		
ServerAddress	STRING	IP-Adresse des Modbus-TCP-Servers, zu dem eine Verbindung hergestellt werden soll, z.B. '10.1.1.156'
ServerPort	WORD	TCP-Port, auf dem der Server den Dienst bereitstellt; normalerweise 502
Enable	BOOL	Enable-Flag, bei TRUE wird eine Verbindung zum Server aufgebaut, bei FALSE wird diese wieder beendet
Timeout	TIME	Timeout-Zeit für alle Server-Reaktionen, z.B. t#2s
Output-Parameter:		
ClientId	DINT	Referenz (Handle) für das Anhängen der Transferbausteine
Ok	BOOL	TRUE = Verbindung zum Server hergestellt
Err	BOOL	TRUE = Fehler beim Herstellen der Verbindung zum Server, ErrCode enthält Fehlernummer
ErrCode	BYTE	0 = kein Fehler 1 = socket create error 2 = kann Verbindung zum Server nicht herstellen (z.B. falsche Server-Adresse, keine Netzwerkverbindung) 3 = kein Transferbaustein angehängt 4 = Verbindung wurde durch Server beendet

Der Baustein PLM_MODBUSTCPCLIENT muss zyklisch aufgerufen werden.

Das Aufbauen der Verbindung zum Modbus-TCP-Server geschieht, indem der Eingang Enable auf TRUE gesetzt wird. Ist der anschließende Verbindungsaufbau innerhalb der vorgegebenen Timeout-Zeit erfolgreich, wird der Ausgang Ok auf TRUE gesetzt. Wenn ein Fehler beim Verbindungsaufbau auftritt, wird Err auf TRUE gesetzt und ein entsprechender Fehlercode in ErrCode ausgegeben.

Solange Enable auf TRUE ist, bleibt die Verbindung zum Server bestehen. Sollte der Server seinerseits die Verbindung beenden, so wird Err auf TRUE gesetzt und ein entsprechender Fehlercode in ErrCode ausgegeben. Ein solcher Verbindungsabbruch kann allerdings erst beim nächsten Datentransfer erkannt werden. Solange Enable den Wert TRUE hat, versucht der Client automatisch, die Verbindung wieder aufzubauen.

12.3.2. PLM_MODBUSTCPCLIENTTRANSFER (Plm_ModbusTcpClient.lib)

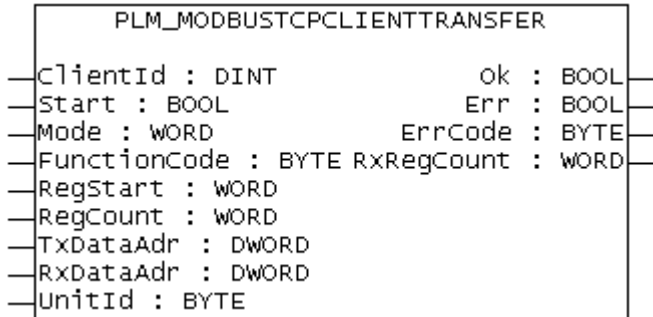


Abb. 12-2: Funktionsblock PLM_MODBUSTCPCLIENTTRANSFER (Plm_ModbusTcpClient.lib)

Input-Parameter:		
ClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Start	BOOL	Mode = 0: Bei steigender Flanke (FALSE → TRUE) wird ein Einzeltransfer ausgelöst Mode = 1: Bei TRUE ist der zyklische Transfer aktiv
Mode	WORD	0 = einzelner Transfer bei Start 1 = zyklischer Transfer
FunctionCode	BYTE	Art des Transfers. Mögliche Function Codes: 1 = Read Coils (1...2000 Bits lesen), 2 = Read Discrete Inputs (1...2000 Bits lesen), 3 = Read Registers (1...125 Words lesen), 4 = Read Input Registers (1...125 Words lesen), 5 = Write Single Coil (1 Bit schreiben), 6 = Write Single Register (1 Word schreiben), 15 = Write Multiple Coils (1...1968 Bits schreiben), 16 = Write Multiple Registers (1...123 Words schreiben)
RegStart	WORD	Modbus-Index des ersten zu lesenden/schreibenden Registers bzw. Coils
RegCount	WORD	Anzahl der zu lesenden/schreibenden Register bzw. Coils FC 1 oder FC 2: 1...2000 FC 3 oder FC 4: 1...125 FC 5 oder FC 6: (keine Bedeutung) FC 15: 1...1968 FC 16: 1...123
TxDataAdr	DWORD	Adresse der Variablen/des Arrays mit den Sendedaten für FC 5, 6, 15 und 16
RxDataAdr	DWORD	Adresse der Variablen/des Arrays für die Empfangsdaten für FC 1, 2,

		3 und 4
UnitId	BYTE	Optionaler Sub-Unit-Identifizier für den Modbus-TCP-Server
Output-Parameter:		
Ok	BOOL	TRUE = Transfer war erfolgreich
Err	BOOL	TRUE = Fehler beim Transfer, ErrCode enthält Fehlernummer
ErrCode	BYTE	0 = kein Fehler 1 = Server exception response, illegal request FunctionCode 2 = Server exception response, illegal request Data Address 3 = Server exception response, illegal request Data Value 81 = Request Timeout, no server response 82 = Request Timeout, incomplete server response 83 = Unzulässiger FunctionCode 84 = Function Code der Response falsch 85 = RegCount unzulässig 86 = ungültige TxDataAdr bzw. RxDataAdr 87 = Transaction Identifier der Response falsch 88 = Unit Identifier der Response falsch 89 = RxRegCount der Response falsch
RxRegCount	WORD	Anzahl der tatsächlich empfangenen Register/Coils bei FC 1, 2, 3 und 4

Alle Bausteine `PLM_MODBUSTCPCLIENTTRANSFER` müssen mindestens einmal beim Programmstart aufgerufen werden, oder zyklisch.

Jeder Transfer-Baustein muss an einen Client-Baustein vom Typ `PLM_MODBUSTCPCLIENT` angehängt werden, indem der Eingang `ClientId` mit dem entsprechenden Ausgang des Client-Bausteins verbunden wird.

Falls mehrere Transferbausteine mit einem Client-Baustein verbunden sind, erfolgt die Abarbeitung automatisch nacheinander in der Aufrufreihenfolge der Transferbausteine.

Die Eingänge `TxDataAdr` und `RxDataAdr` erwarten die Angabe einer Variablenadresse, die als Ergebnis der Adressfunktion `ADR()` geliefert wird (siehe Beispiel in Abschnitt 12.4). Der Typ der Variablen hängt vom Function-Code ab.

Bei den Function-Codes 3 und 4 (Read Registers) muss am Eingang `RxDataAdr` die Adresse eines WORD-Arrays mit mindestens `RegCount` WORDs angegeben werden.

Bei den Function-Codes 6 und 16 (Write Single/Multiple Registers) muss am Eingang `TxDataAdr` die Adresse eines WORD-Arrays mit mindestens `RegCount` WORDs angegeben werden.

Coil-Werte werden auf dem Modbus immer in Achtergruppen zu einem Byte zusammengefasst. Entsprechend erwarten die Transferbausteine bei den Coil-Funktionen BYTE-Daten, wobei immer acht Coil-Werte in einem Byte gespeichert werden. Unabhängig davon können aber auch weniger als acht Coil-Werte beeinflusst werden (entsprechender Wert von `RegCount`), oder Coil-Werte, die nicht

auf einer Byte-Grenze beginnen, so dass sich nur die angegebenen Bits in den entsprechenden Bytes ändern.

Bei den Function-Codes 1 und 2 (Read Coils) muss am Eingang `RxDataAdr` die Adresse eines BYTE-Arrays mit mindestens $(\text{RegCount}+7) / 8$ Bytes angegeben werden. `RegCount` enthält die gewünschte Anzahl Coils, nicht Bytes. Die Verwendung von BOOL-Arrays ist nicht möglich.

Bei Function-Code 5 (Write Single Coil) muss am Eingang `TxDataAdr` die Adresse einer BYTE- oder BOOL-Variablen angegeben werden. Ist der Wert 0 (oder FALSE), so wird das Coil ausgeschaltet (auf 0 gesetzt), ist der Wert ungleich 0 (oder TRUE), so wird das Coil eingeschaltet (auf 1 gesetzt). Der Eingang `RegCount` wird hierbei ignoriert.

Bei Function-Code 15 (Write Multiple Coils) muss am Eingang `TxDataAdr` die Adresse eines BYTE-Arrays mit mindestens $(\text{RegCount}+7) / 8$ Bytes angegeben werden. `RegCount` enthält die gewünschte Anzahl Coils, nicht Bytes. Die Verwendung von BOOL-Arrays ist nicht möglich.

12.4. Programmbeispiel (FUP)

```

0001 PROGRAM ModbusTcpClient
0002 VAR
0003     client: Plm_ModbusTcpClient;
0004     transf1, transf2: Plm_ModbusTcpClientTransfer;
0005     ok, err: BOOL;
0006     errCode1, errCode2: BYTE;
0007     txData: ARRAY[0..15] OF WORD;
0008     rxData: ARRAY[0..15] OF WORD;
0009     do_connect, do_read, do_write: BOOL;
0010 END_VAR
0011
0001 Server connection
    client
    '10.1.1.156' - ServerAddress
    502 - ServerPort
    do_connect - Enable
    t#2s - Timeout
    ClientId - ClientId
    ok - ok
    Err - err
    ErrCode - errCode

0002 Read registers
    transf1
    client.ClientId - ClientId
    do_read - Start
    0 - Mode
    3 - FunctionCode
    0 - RegStart
    16 - RegCount
    ADR(rxData) - RxDataAdr
    - TxDataAdr
    - UnitId
    Ok - ok
    Err - Err
    ErrCode - errCode1
    RxRegCount - RxRegCount

0003 write multiple registers
    transf2
    client.ClientId - ClientId
    do_write - Start
    0 - Mode
    16 - FunctionCode
    0 - RegStart
    16 - RegCount
    ADR(txData) - TxDataAdr
    - RxDataAdr
    - UnitId
    Ok - ok
    Err - Err
    ErrCode - errCode2
    RxRegCount - RxRegCount
    
```

Abb. 12-3: Realisierung eines Modbus-TCP-Clients (FUP)

Das in Abb. 12-3 dargestellte Beispiel realisiert einen Modbus-TCP-Client. Zum Aufbau einer Verbindung muss zunächst `do_connect` auf TRUE gesetzt werden. Der Client versucht dann, eine Verbindung zu einem Modbus-TCP-Server mit der IP-Adresse 10.1.1.156 auf Port 502 aufzubauen. Gelingt dies, wird `ok` auf TRUE gesetzt

und die angehängten Transferbausteine können mit `do_read` und `do_write` aktiviert werden. Der angegebene Timeout-Wert muss bei sehr langsamen Verbindungen und stark belasteten Servern evtl. erhöht werden.

Die Eingänge `ClientId` beider Transferbausteine sind mit dem entsprechenden Ausgang `ClientId` des Client-Bausteins verbunden. Dies kann alternativ auch durch Zuweisung zu einer entsprechenden Variablen geschehen, die dann am Ausgang des Clients und an den Eingängen aller Transferbausteine angeschaltet wird.

Im Beispiel hat `Mode` bei beiden Transferbausteinen den Wert 0, so dass bei einem Wechsel von `do_read` oder `do_write` von FALSE auf TRUE jeweils ein einziger Transfer (Modbus-Request) ausgeführt wird.

Der erste Transferbaustein `transf1` führt im Beispiel einen Lesevorgang von 16 Registerwerten ab Registerindex 0 (also Register 0...15) auf den Server aus. Anschließend werden die gelesenen Registerwerte im WORD-Array `rxData[]` gespeichert.

Der zweite Transferbaustein `transf2` führt im Beispiel einen Schreibvorgang von 16 Registerwerten ab Registerindex 0 (also Register 0...15) auf den Server aus. Dabei werden die Registerwerte aus dem WORD-Array `txData[]` an den Server übertragen.

13. E-Mails versenden

13.1. Allgemeines

Alle PLM700-Steuerungen mit Ethernet-Anschluss bieten die Möglichkeit, E-Mails durch das IEC-Programm zu versenden.

Voraussetzung ist, dass die Steuerung einen Internet-Zugang besitzt. Hierfür sind u.a. einzustellen:

- IP-Adresse
- Netzmaske
- Default-Gateway
- ggf. DNS

Default-Gateway ist die IP-Adresse des Routers, über den die Steuerung ins Internet gelangt, sofern sie nicht direkt mit dem Internet verbunden ist und eine öffentliche IP-Adresse besitzt.

Der DNS (Domain Name Service) löst Namen in IP-Adressen auf. Da einige Mail-Provider keine feste IP-Adresse verwenden, sollte der Zugriff auf den SMTP-Server über einen Namen erfolgen (z.B. `smtp.web.de`). Hierzu muss ein DNS-Server auf der Steuerung bekannt sein.

Desweiteren muss der Anwender eine Zugangsberechtigung auf einem SMTP-Server (Mail-Server) haben, der die Mail der Steuerung weiterleitet. Dabei kann es sich um einen kommerziellen Account handeln oder um einen kostenlosen Mail-Dienst, wie er von vielen Providern angeboten wird.

13.2. Spezifikation

- E-Mail-Client für SMTP-Server
- Unterstützte Authentifizierungsverfahren: NONE, PLAIN, STARTTLS und SSL (erfordert mind. LZS v21905200 und Servicepack 2019-05-20 auf PLM700A)
- Versand von Attachments (ab LZS v21004013)

13.3. Benötigte Bibliotheken

PLM_Mail_01.lib

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

13.3.1. PLM_SmtpInit (PLM_Mail_01.lib)

PLM_SMTPINIT

Abb. 13-1: Programmblock `Plm_SmtpInit` (PLM_Mail_01.lib)

Dieser Programmblock muss einmalig beim Start des IEC-Programms aufgerufen werden und initialisiert die internen Datenstrukturen für nachfolgende Aufrufe von `PLM_SmtpSendMail`.

13.3.2. PLM_SmtpSendMail (Plm_Mail_01.lib)

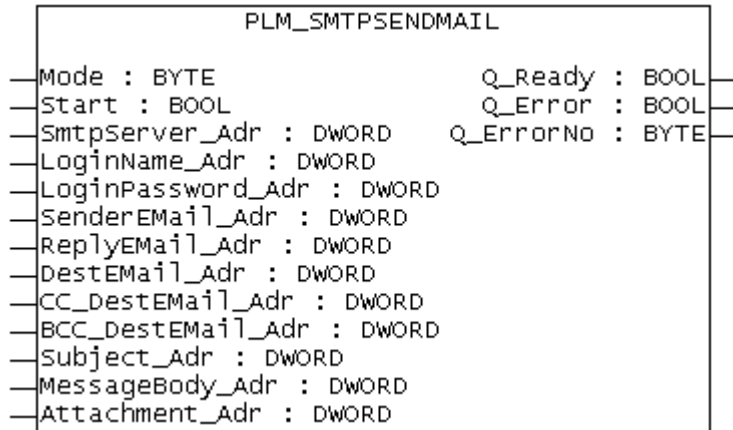


Abb. 13-2: Funktionsblock PLM_SmtpSendMail (Plm_Mail_01.lib)

Input-Parameter:		
Mode	BYTE	0 = AUTH_NONE (keine Authentifizierung) 1 = AUTH_PLAIN (Authentifizierung mit Name und Password) 2 = STARTTLS (verschlüsselte Authentifizierung mit Name und Password, ab LZS v21905200) 3 = SSL (verschlüsselte Authentifizierung mit Name und Password, ab LZS v21905200) +32 = Content-Transfer-Encoding <i>8bit</i> erzwingen (sonst: <i>quoted-printable</i>) +64 = Content-Type <i>multipart</i> erzwingen, auch wenn keine Attachments angegeben +128 = Debug-Mode einschalten (s. Abschnitt 13.6 Fehlersuche)
Start	BOOL	Steigende Flanke FALSE → TRUE löst Versand der E-Mail aus. Anschließend kann Start sofort auf FALSE zurückfallen.
SmtServer_Adr	DWORD	ADR () von IP-Adresse und TCP-Port des SMTP-Servers, z.B. '217.72.192.157:25' oder 'smtp.web.de:25'
LoginName_Adr	DWORD	ADR () des Login-Namens für den SMTP-Server (nicht erforderlich bei AUTH_NONE)
LoginPassword_Adr	DWORD	ADR () des Passwords für den SMTP-Server (nicht erforderlich bei AUTH_NONE)
SenderEMail_Adr	DWORD	ADR () der E-Mail-Adresse des Absenders
ReplyEMail_Adr	DWORD	ADR () der E-Mail-Adresse, an die Antworten geschickt werden sollen
DestEMail_Adr	DWORD	ADR () der E-Mail-Adresse des Empfängers (oder der Empfängerliste)
CC_DestEMail_Adr	DWORD	ADR () der E-Mail-Adresse des CC-Empfängers (oder der Empfängerliste)
BCC_DestEMail_Adr	DWORD	ADR () der E-Mail-Adresse des BCC-Empfängers (oder der Empfängerliste)
Subject_Adr	DWORD	ADR () der Betreffzeile der E-Mail

MessageBody_Adr	DWORD	ADR () des Textinhalts der E-Mail
Attachment_Adr	DWORD	ADR () des Dateinamens eines Attachments (oder der Attachment-Liste, ab LZS v21004013)
Output-Parameter:		
Q_Ready	BOOL	TRUE = Versand erfolgreich
Q_Error	BOOL	TRUE = Fehler beim Versand
Q_ErrorNo	BYTE	<p>0 = Versand ok</p> <p>102 = SSL/TLS Write-Fehler</p> <p>103 = SSL/TLS Read-Fehler</p> <p>104 = SSL/TLS Verbindungsfehler</p> <p>105 = SSL/TLS smtpServer Name kann durch DNS nicht aufgelöst werden</p> <p>106 = SSL/TLS Verbindung kann nicht hergestellt werden</p> <p>107 = SSL/TLS Interner Fehler</p> <p>108 = SSL/TLS Interner Fehler</p> <p>109 = SSL/TLS Interner Fehler</p> <p>112 = SSL/TLS Fehler in SenderEMail</p> <p>113 = SSL/TLS Fehler in SenderEMail</p> <p>115 = SSL/TLS Fehler in DestEMail, CC_DestEMail oder BCC_DestEMail</p> <p>116 = SSL/TLS Fehler in DestEMail, CC_DestEMail oder BCC_DestEMail</p> <p>117 = SSL/TLS Fehler in LoginName</p> <p>118 = SSL/TLS Fehler in LoginPassword</p> <p>119 = SSL/TLS Fehler in LoginPassword</p> <p>120 = SSL/TLS Digest-Response-Fehler</p> <p>121 = SSL/TLS Interner Fehler</p> <p>122 = SSL/TLS Fehler in DestEMail, CC_DestEMail oder BCC_DestEMail</p> <p>123 = SSL/TLS Fehlercode in Server-Antwort</p> <p>124 = SSL/TLS Fehlercode in Server-Antwort nach EHLO</p> <p>125 = SSL/TLS Fehlercode in Server-Antwort nach AUTH PLAIN</p> <p>126 = SSL/TLS Fehlercode in Server-Antwort nach AUTH LOGIN</p> <p>127 = SSL/TLS Fehlercode in Server-Antwort nach AUTH CRAM-MD5</p> <p>128 = SSL/TLS Fehlercode in Server-Antwort nach AUTH DIGEST-MD5</p> <p>129 = SSL/TLS Fehlercode in Server-Antwort</p> <p>130 = SSL/TLS Fehlercode in Server-Antwort</p> <p>131 = SSL/TLS Fehlercode in Server-Antwort nach QUIT</p> <p>132 = SSL/TLS Fehlercode in Server-Antwort</p> <p>133 = SSL/TLS Fehlercode in Server-Antwort</p> <p>134 = SSL/TLS Unerwarteter Verbindungsabbruch zum Server</p> <p>136 = SSL/TLS Timeout bei Server-Verbindung</p> <p>137 = SSL/TLS Timeout bei Server-Verbindung</p> <p>138 = SSL/TLS Attachment-Datei kann nicht geöffnet oder gelesen werden</p> <p>139 = SSL/TLS Nachricht inkl. Attachments</p>

		<p>zu groß</p> <p>140 = SSL/TLS Fehler in LoginName oder LoginPassword</p> <p>141 = SSL/TLS Fehlercode in Server-Antwort</p> <p>142 = SSL/TLS nicht genügend Speicher vorhanden</p> <p>143 = SSL/TLS Interner Fehler</p> <p>144 = SSL/TLS Interner Fehler</p> <p>145 = SSL/TLS Interner Fehler</p> <p>146 = SSL/TLS Interner Fehler</p> <p>147 = SSL/TLS Fehlercode in Serverantwort nach STARTTLS</p> <p>148 = SSL/TLS Read-Fehler</p> <p>149 = SSL/TLS Write-Fehler</p> <p>150 = SSL/TLS Init-Fehler</p> <p>151 = SSL/TLS Connect-Fehler</p> <p>152 = SSL/TLS Connect-Fehler</p> <p>153 = SSL/TLS Connect-Fehler</p> <p>154 = SSL/TLS Fehlercode in Server-Antwort nach Datenblock</p> <p>155 = SSL/TLS STARTTLS wird vom SMTP-Server nicht unterstützt</p> <p>156 = SSL/TLS AUTH LOGIN wird vom SMTP-Server nicht unterstützt</p> <p>157 = SSL/TLS Fehlercode in Server-Antwort</p> <p>170 = SSL/TLS Interner Fehler</p> <p>171 = SSL/TLS Interner Fehler</p> <p>225 = Interner Fehler</p> <p>226 = Fehler in DestEMail, CC_DestEMail oder BCC_DestEMail</p> <p>227 = Fehler in LoginName oder LoginPassword</p> <p>228 = Fehler in SenderEMail</p> <p>229 = Fehler in SmtServer</p> <p>230 = Interner Fehler</p> <p>231 = Message-Daten vom SMTP-Server zurückgewiesen</p> <p>232 = Eine oder mehrere Empfängeradressen vom SMTP-Server zurückgewiesen</p> <p>233 = Absenderadresse vom SMTP-Server zurückgewiesen</p> <p>234 = zu wenig Speicher</p> <p>235 = Befehl vom SMTP-Server zurückgewiesen</p> <p>236 = Verbindung zum SMTP-Server konnte nicht hergestellt werden</p> <p>237 = Read-Fehler</p> <p>238 = Write-Fehler</p> <p>239 = Socket-Fehler</p> <p>240 = Close-Fehler</p> <p>241 = Übertragung abgebrochen</p> <p>242 = Fehlender Parameter SmtServer_Adr oder SenderEMail_Adr, bei AUTH_PLAIN auch LoginName_Adr oder LoginPassword_Adr</p> <p>243 = Timeout bei Kommunikation mit SMTP-Server (20 Sek.)</p> <p>244 = Interner Fehler</p> <p>245 = Authentifizierung fehlgeschlagen (LoginName/LoginPassword falsch)</p>
--	--	--

		246 = Attachments nicht unterstützt 247 = Leerer Attachment-Dateiname 248 = Attachment-Datei kann nicht geöffnet oder gelesen werden 249 = SmtPserver Name kann durch DNS nicht aufgelöst werden 250 = fehlende Lizenz 251 = Nachricht inkl. Attachments zu groß 252 = SSL/TLS-Mailer nicht installiert 253 = SSL/TLS nicht verfügbar 254 = Interner Fehler
--	--	---

Dieser Funktionsblock muss aus einem SLOW_TASK heraus zum Versenden einer E-Mail mit `Start = TRUE` solange aufgerufen werden, bis entweder `Q_Ready` oder `Q_Error` den Wert `TRUE` hat. Vor dem Versand einer weiteren E-Mail muss der Funktionsblock mindestens einmal mit `Start = FALSE` aufgerufen werden.

Alle String-Eingänge, deren Namen mit `..._Adr` enden, erwarten die Adresse einer String-Variablen; hierzu bietet CoDeSys den Operator `ADR()` (siehe Beispiel in Abschnitt 13.4). An unbenutzte String-Eingänge muss als Adresse der Wert 0 gelegt werden.

Solange der Versand läuft, werden die Werte an den String-Eingängen intern verwendet und dürfen deshalb nicht geändert werden.

Das in `Mode` eingestellte Authentifizierungsverfahren für die Anmeldung am SMTP-Server wird vom Server vorgegeben und ist ggf. beim Provider nachzufragen. `AUTH_NONE` (keine Authentifizierung) wird aus Sicherheitsgründen nur von sehr wenigen Mailservern unterstützt.

Wenn kein TCP-Port hinter dem Server-Namen angegeben wird, verwendet der Baustein als Standard den SMTP-Port 25. Für STARTTLS ist es üblich, entweder den Port 25 oder 578 zu verwenden, für SLL den Port 465.

Der Versand der E-Mail beginnt, wenn der Eingang `Start` von `FALSE` auf `TRUE` wechselt und kann mehrere Sekunden dauern, abhängig von der Menge der zu übertragenden Daten und der Reaktionsgeschwindigkeit des SMTP-Servers. Nach Beginn des Versands kann `Start` sofort wieder auf `FALSE` zurückwechseln, ohne dadurch den Versand zu beeinträchtigen.

Die Erzeugung eines Busy-Signals kann durch Abfrage einer internen Variablen des Funktionsblocks erfolgen; dies ist im Beispiel in Abschnitt 13.5 gezeigt.

Bei Erfolg (d.h. der SMTP-Server hat die Mail entgegengenommen) geht `Q_Ready` für mind. einen Zyklus auf `TRUE`. Andernfalls geht `Q_Error` für mind. einen Zyklus auf `TRUE` und `Q_ErrorNo` enthält einen entsprechenden Fehlercode. `Q_Ready` und `Q_Error` bleiben anschließend `TRUE`, solange `Start` noch `TRUE` ist.

Zu Hinweisen zur Fehlersuche siehe Abschnitt 13.6.

In `DestEMail_Adr`, `CC_DestEMail_Adr` und `BCC_DestEMail_Adr` können jeweils eine oder mehrere E-Mail-Adressen angegeben werden. Bei mehreren E-Mail-Adressen müssen diese in einem String durch Komma oder Semikolon getrennt werden.

`SenderEMail_Adr` wird meist von SMTP-Servern überprüft und kann dann nicht beliebig gewählt werden.

Falls der Empfänger eine Antwort auf die Mail verfasst, soll diese normalerweise an eine bestimmte Person gehen und nicht zurück an die Steuerung, daher ist eine entsprechende Angabe in `ReplyEMail_Adr` möglich. Diese Angabe wird ggf. vom Mail-Empfänger ausgewertet.

Der String `MessageBody_Adr` enthält den eigentlichen Text der E-Mail. Zeilenümbrüche im Text können durch Einfügen der String-Konstanten `'$N'` erzeugt werden. Der Text kann maximal ca. 1024 Zeichen lang sein. Längere Texte können als Attachment geschickt werden.

Sollen ein oder mehrere Attachments (Anhangsdateien) mitgeschickt werden, so sind diese in einem String aufzulisten, dessen Adresse an `Attachment_Adr` angelegt wird. Jeder Dateiname muss einen Laufwerksbuchstaben enthalten, z.B. `'a/test.dat'`. Bei mehreren Dateien müssen die Dateinamen durch Komma oder Semikolon getrennt werden, z.B. `'a/test.dat;a/xy.log'`. Optional kann hinter jedem Dateinamen ein Doppelpunkt und dahinter der MIME-Typ der Datei angegeben werden. Ohne explizite Angabe ist die Voreinstellung `application/octet-stream`. Diese Angabe wird ggf. vom Mail-Empfänger ausgewertet.

13.4. Dauer der Datenübertragung

Die Dauer des E-Mail-Versands hängt davon ab, mit welchen Parametern `PLM_SmtpSendMail` aufgerufen wird, ob Attachments mitgeschickt werden sollen, wie groß diese sind, sowie von der Reaktionsgeschwindigkeit des SMTP-Servers.

Da die Übertragung auf keinen Fall in einem einzigen IEC-Zyklus (Standardwert 20 ms) abgeschlossen werden kann, sind mehrere Aufrufe des Bausteins `PLM_SmtpSendMail` notwendig, bis der Baustein an den Ausgängen `Q_Ready` oder `Q_Error` den Abschluss der Datenübertragung meldet.

Der IEC-Zyklus kann durch Aufrufe von `PLM_SmtpSendMail()` blockiert werden, daher muss für die Aufrufe ein `SLOW_TASK` eingerichtet werden.

Die minimale Übertragungsdauer bei Versand ohne STARTTLS oder SSL beträgt 35 Zykluszeiten, d.h. eine Mail mit kurzem Message Body und ohne Attachments, die an einen einzigen Empfänger adressiert ist, benötigt mind. 35 Aufrufe von `PLM_SmtpSendMail()`. Bei einem Aufrufintervall von 20 ms entspricht dies ca. 0,7 Sekunden, vorausgesetzt, dass der SMTP-Server innerhalb von 20 ms die jeweiligen Anfragen beantwortet.

Bei langsamer Reaktion des SMTP-Servers werden automatisch Wartezyklen ausgeführt, die die Dauer der Datenübertragung verlängern. Antwortet der SMTP-Server auf eine Anfrage nicht innerhalb von 20 Sekunden, so wird der Versand mit Timeout abgebrochen.

Die Übertragung von Attachments erfolgt bei Versand ohne STARTTLS oder SSL in Blöcken mit je 702 Bytes pro Zyklus. Ein Attachment von 10 kByte benötigt daher mind. 15 Zyklen für die Übertragung bei ausreichend schneller Reaktion des SMTP-Servers.

13.5. Programmbeispiel (FUP)

The screenshot shows a ladder logic program for sending emails. The program is titled "Mailer (PRG-FUP)".

0001 PROGRAM Mailer
0002 VAR
 0003 start, initDone, busy, ready, error: **BOOL**;
 0004 errorNo: **BYTE**;
 0005 sendmail: **PLM_SmtpSendMail**;
 0006 server: **STRING** := '217.72.192.157';
 0007 user: **STRING** := 'user@web.de';
 0008 pass: **STRING** := 'mypass';
 0009 from: **STRING** := 'user@web.de';
 0010 sendto: **STRING** := 'personal@leitwarte.de';
 0011 subject: **STRING** := 'Mail von PLM';
 0012 message: **STRING** := 'Zeile 1\$NZeile 2\$NZeile3';
0013 END_VAR

0001 initDone → do_send

0002 PLM_SmtpInit

0003 do_send:
 TRUE → initDone

0004 sendmail
 PLM_SmtpSendMail

1-Mode	Q_Ready	→ ready
start-Start	Q_Error	→ error
ADR(server)-SmtPserver_Adr	Q_ErrorNo	→ errorNo
ADR(user)-LoginName_Adr		
ADR(pass)-LoginPassword_Adr		
ADR(from)-SenderEMail_Adr		
0-ReplyEMail_Adr		
ADR(sendto)-DestEMail_Adr		
0-CC_DestEMail_Adr		
0-BCC_DestEMail_Adr		
ADR(subject)-Subject_Adr		
ADR(message)-MessageBody_Adr		
0-Attachment_Adr		

0005 sendmail.a[0] > 0 AND sendmail.a[0] < 500 → busy

Abb. 13-3: Versand von E-Mails (FUP)

Das in Abb. 13-3 gezeigte Beispiel demonstriert den Versand von E-Mails. Der Aufruf von Mailer() muss aus einem SLOW_TASK heraus erfolgen.

Vor dem Aufruf des Sendebausteins wird einmalig die Initialisierungsroutine `PLM_SmtpInit()` aufgerufen.

Zum Starten muss die Variable `start` auf `TRUE` gesetzt werden. Die Variable `busy` wird in Zeile 0005 aus einer internen Variable der Bibliothek abgeleitet und ist `TRUE`, solange der Versand läuft.

Nach Ende des Versands hat entweder `ready` oder `error` den Wert `TRUE`. Wenn `start` bereits wieder auf `FALSE` zurückgesetzt wurde, liegen die Werte an `ready` bzw. `error` nur für einen einzigen Zyklus an, andernfalls solange, wie `start` noch auf `TRUE` steht.

Der SMTP-Server hat im Beispiel die IP-Adresse 217.72.192.157 (alte IP-Adresse von *smtp.web.de*) und erwartet eine unverschlüsselte Authentifizierung (`Mode=1`) mit Name (hier: *user@web.de*) und Password (hier: *mypass*). Mit den gleichen Anmeldedaten ist eine Anmeldung am Freemail-Webinterface von Web.de möglich. Bei Web.de müssen Loginname (`LoginName_Adr`) und Absender der E-Mail (`SenderEMail_Adr`) identisch sein.

Der Empfänger der E-Mail ist *personal@leitwarte.de*, die Betreffzeile lautet *Mail von PLM*. Der eigentliche Text der E-Mail besteht hier aus drei Zeilen, die durch Zeilenumbrüche (`$N`) getrennt sind.

An die nicht verwendeten Eingänge wird die Adresse 0 angelegt.

Im Erfolgsfall geht `Q_Ready` für einen Zyklus auf `TRUE`. Andernfalls geht `Q_Error` für einen Zyklus auf `TRUE` und `Q_ErrorNo` liefert einen Fehlercode, der Rückschlüsse auf die Art des Fehlers zulässt (siehe Abschnitt 13.3.2).

Der zeitliche Ablauf der Signale `start`, `busy`, `ready` und `error` ist in Abb. 13-4 dargestellt.

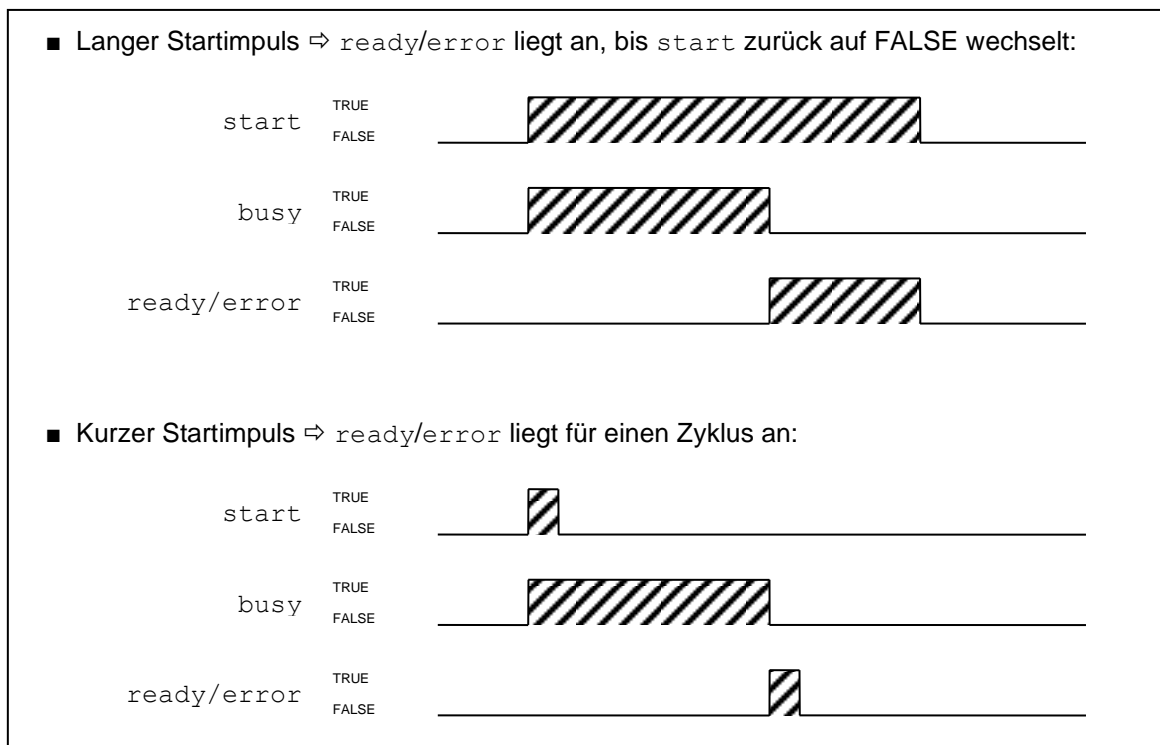


Abb. 13-4: Zeitlicher Ablauf der Signale `start`, `busy`, `ready` und `error`

13.6. Verbindungsparameter einiger E-Mail-Anbieter

Die folgenden Einstellungen wurden Stand Mai 2023 getestet.

13.6.1. Web.de

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu im Postfach im Menü links auf *Einstellungen*,

dann unter *POP3/IMAP-Abruf* das Häkchen *POP3 und IMAP Zugriff erlauben* setzen und *Speichern*.

Parameter am Baustein `PLM_SmtpSendMail`:

Mode	2 (STARTTLS)
SmtServer	smtp.web.de:587
LoginName	Web.de-User oder vollständige Web.de-E-Mail-Adresse
LoginPassword	Password des Web.de-Users

13.6.2. T-Online

Zunächst muss auf der Webseite des Anbieters *Ihr speziell für E-Mail-Programme angelegtes Passwort* angelegt werden. Dafür oben im Header links auf das Zahnrad und im Drop Down Menü ganz unten auf *Alle Einstellungen*. Dann in der linken Menüleiste auf *Passwörter* und *Passwort für E-Mail-Programme* auswählen, dort das Passwort definieren. IMAP und POP3 explizit zu aktivieren ist nicht notwendig, da standardmäßig aktiviert.

Parameter am Baustein `PLM_SmtpSendMail`:

Mode	3 (SSL)
SmtServer	securesmtp.t-online.de:465
LoginName	Vollständige T-Online-E-Mail-Adresse
LoginPassword	Das im Web-Interface speziell angelegte Passwort

13.6.3. GMX

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu im Postfach im Menü links auf *Einstellungen*, dann unter *POP3/IMAP-Abruf* das Häkchen *POP3 und IMAP Zugriff erlauben* setzen und *Speichern*.

Parameter am Baustein `PLM_SmtpSendMail`:

Mode	2 (STARTTLS) oder 3 (SSL)
SmtServer	securesmtp.t-online.de:587 oder securesmtp.t-online.de:465
LoginName	Vollständige GMX-E-Mail-Adresse
LoginPassword	Password des GMX-Users

13.6.4. Outlook

Zunächst muss auf der Webseite des Anbieters im Mail-Account der Zugriff über IMAP und POP3 aktiviert werden. Dazu auf das Zahnrad (*Einstellungen*) oben rechts im Postfach-Header, dort ganz unten auf *Alle Outlook-Einstellungen anzeigen*, dort *E-Mail* → *E-Mail-Einstellungen*, im Abschnitt *POP und IMAP* unter *POP-Optionen* *Geräten und Apps die Verwendung von POP gestatten* auf *Ja* einstellen.

Parameter am Baustein `PLM_SmtpSendMail`:

Mode	2 (STARTTLS)
SmtServer	smtp.office365.com:587
LoginName	Vollständige E-Mail-Adresse
LoginPassword	Password des Users

13.7. Fehlersuche

Der Baustein `PLM_SmtpSendMail()` liefert im Fehlerfall einen Fehlercode, der eine erste Fehlerdiagnose erlaubt.

Bei Fehler 236 (Verbindung zum SMTP-Server konnte nicht hergestellt werden) ist zu prüfen, ob die Internet-Verbindung der Steuerung richtig konfiguriert wurde und funktioniert (insbesondere Default-Gateway prüfen).

Bei Fehler 236 oder 243 (Timeout) ist zunächst die IP-Adresse des SMTP-Servers zu kontrollieren. Öffnen Sie dazu unter Windows eine Eingabekonsole (z.B. *Start* → *Ausführen* → *cmd*) und geben Sie den Befehl *nslookup servername* ein, z.B. *nslookup smtp.web.de*. Sie erhalten dann eine Antwort wie z.B. *Nicht autorisierte Antwort: Name: smtp.web.de, Address: 217.72.192.157*. Die *Address* ist die benötigte IP-Adresse des SMTP-Servers. Den Namen des SMTP-Servers (z.B. *smtp.web.de*) erfahren Sie von Ihrem Provider.

In manchen Fällen ist es hilfreich, die Kommunikation mit dem SMTP-Server im Detail zu beobachten. Hierzu kann am Eingang *Mode* der Wert 128 addiert werden, wodurch eine Log-Datei mit dem Namen *mail_debug.txt* auf FTP-Laufwerk *a/* angelegt wird. Diese Textdatei kann über FTP von der Steuerung heruntergeladen und mit einem Texteditor geöffnet werden. Die Textdatei enthält mitunter Klartextfehlermeldungen des SMTP-Servers, aus denen häufig die genaue Fehlerursache diagnostiziert werden kann. Die Datei wird bei jedem neuen Mail-Versand überschrieben.

Da die Log-Datei, insbesondere beim Versand von Attachments, sehr groß werden kann, sollte diese Funktion nach dem Debuggen unbedingt wieder abgeschaltet werden.

14. SMS versenden

14.1. Allgemeines

SMS (Short Message Service) bezeichnet den Versand von kurzen Textmeldungen über das GSM-Mobilfunknetz. Die Länge der Meldungen ist auf 160 Zeichen beschränkt.

Der Sender erhält keine Bestätigung, ob die Nachricht tatsächlich empfangen wurde; ein erfolgreicher SMS-Versand bestätigt lediglich, dass die Textmeldung an den Mobilfunk-Provider zur Auslieferung übergeben wurde. Falls der Empfänger nicht erreichbar ist oder die SMS aus anderem Grund eine gewisse Zeitlang nicht zugestellt werden kann, verfällt die Nachricht.

Zum Versand ist der Anschluss eines geeigneten GSM-Modems mit eingesetzter SIM-Karte erforderlich. Die SIM-Karte wird von einem Mobilfunkprovider (z.B. Vodafone, O2 etc.) auf Basis eines Nutzungsvertrags ausgegeben und beschränkt ggf. die nutzbaren Mobilfunkdienste des Modems. Die SIM-Karte muss hier den Versand von SMS erlauben, das ist bei SIM-Karten mit reinen Datentarifen häufig nicht der Fall.

Das Modem muss über Funk mit dem Mobilfunknetz verbunden sein, dazu ist normalerweise eine geeignete externe Antenne erforderlich.

14.2. Spezifikation

- Versand von SMS-Nachrichten über angeschlossenes GSM-Modem
- Nachrichtenlänge max. 160 Zeichen
- optionale Entsperrung der SIM-Karte mit PIN
- Fehlerdiagnose über Log-Datei
- Kompatibel zu den Modems SIM.730.34 und SIM.730.36

14.3. Benötigte Bibliotheken

```
PLM_SMS-v20161018.lib
```

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

Auf der Steuerung muss ein Laufzeitsystem mit mindestens der Version v21606204 installiert sein.

14.3.1. Plm_SMS_Send

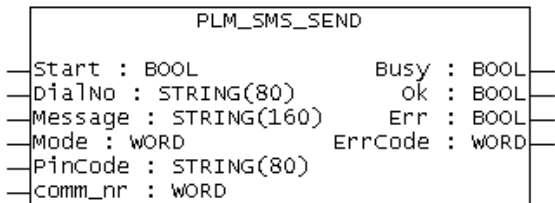


Abb. 14-1: Funktionsblock Plm_SMS_Send (PLM_SMS-v20161018.lib)

Input-Parameter:		
Start	BOOL	Steigende Flanke FALSE → TRUE löst Versand der SMS aus. Anschließend kann Start sofort auf FALSE zurückfallen.
DialNo	STRING	Telefonnummer des SMS-Empfängers. Die Nummer darf die Zeichen 0 1 2 3 4 5 6 7 8 9 A B C D + * # , W P T enthalten. Zeichen, die keine

		Ziffern sind, haben abhängig vom Modem und vom Mobilfunk-Provider eine Sonderfunktion.
Message	STRING (160)	Inhalt der Textnachricht (max. 160 Zeichen).
Mode	WORD	Konfiguriert verschiedene Eigenschaften des Bausteins für die Kommunikation mit dem angeschlossenen Modem. Die angegebenen Werte müssen addiert werden. + 16#0001: Modem-Reset ausführen + 16#0002: SIM-Karte testen + 16#0004: SIM-Karte mit PinCode entsperren + 16#0010: LF anstelle von CR für Modembefehle verwenden
PinCode	STRING	PIN zum Entsperren der SIM-Karte (üblicherweise eine vierstellige Zahl); wird nur verwendet, wenn 16#0004 in Mode enthalten ist
comm_nr	WORD	Nummer der seriellen Schnittstelle, über die die Kommunikation mit dem Modem erfolgt
Output-Parameter:		
Busy	BOOL	TRUE = Verarbeitung läuft
Ok	BOOL	TRUE = SMS erfolgreich versendet
Err	BOOL	TRUE = Fehler beim Versenden, siehe ErrCode
ErrCode	WORD	0 = Versand ok 100 = Falsche LZS-Version 101 = Keine gültigen Zeichen in DialNo Werte von ErrCode oberhalb von 1000 setzen sich zusammen aus dem Fehlercode und einer internen Zustandsnummer. Fehlercodes: 1000 = Modembefehl wurde innerhalb von Timeoutzeit nicht oder nicht wie erwartet beantwortet 1001 = Modembefehl wurde nicht wie erwartet beantwortet 1002 = Modembefehl wurde mit ERROR beantwortet Zustandsnummern: + 20 = Fehler bei Modem-Test, + 30 = Fehler bei Modem-Reset, + 40 = Fehler beim Testen (1) der SIM-Karte, + 50 = Fehler beim Testen (2) der SIM-Karte, + 60 = Fehler bei PIN-Entsperrung, + 100 = Fehler bei zusätzlichen User-Befehlen, + 200 = Fehler beim Aktivieren des Text-Modus, + 210 = Fehler beim Übertragen der DialNo, + 220 = Fehler beim Senden der SMS über GSM

Eine Instanz des Bausteins Plm_SMS_Send muss zyklisch aufgerufen werden. Sobald der Eingang Start von FALSE auf TRUE wechselt, werden die anliegenden Input-Parameter ausgewertet und die SMS versendet.

Die Ausgänge Ok bzw. Err, die den Erfolg oder Misserfolg des SMS-Versands signalisieren, bleiben solange anliegen, wie Start auf TRUE liegt. Falls Start vor Ablauf des Bausteins wieder auf FALSE gewechselt wurde, liegen Ok und Err nach dem Ablauf nur für einen einzigen Zyklus an.

Der interne Ablauf des Bausteins im Erfolgsfall von Start bis Ok dauert typischerweise zwischen 10 und 20 Sekunden. Während dieser Zeit findet eine Kommunikation mit dem Modem über die mit comm_nr angegebene Schnittstelle statt. Das Modem erhält dabei die Befehle als sogenannte AT-Kommandos. Die

Antworten des Modems müssen innerhalb einer kurzen Timeout-Zeit eintreffen und werden dann ausgewertet. Die Bibliothek verwendet Standard-AT-Kommandos, die von den meisten Modems verstanden werden.

14.4. Initialisierung der Schnittstelle

Die für die Modem-Kommunikation verwendete Schnittstelle `comm_nr` muss ggf. vor Aufruf des Bausteins initialisiert werden. Das Verfahren unterscheidet sich je nach Schnittstelle. Die Initialisierung erfolgt nur einmalig nach Programmstart, d.h. im Baustein `InitOnce()` o.ä.

14.4.1. Serielle Schnittstelle RS232 (z.B. für SIM.730.34):

Der Betrieb erfolgt über eine der eingebauten seriellen RS232-Schnittstellen der Steuerung (COM1...COM3). Die seriellen Schnittstellenparameter müssen mit den Einstellungen am Modem übereinstimmen.

```
comm_nr := 0;      (* 0..2 = RS232-Schnittstelle COM1..COM3 der Steuerung *)
CommSetParam(
  comm_nr := comm_nr,
  mode := 0,
  dat_len := 8,
  stop_len := 1,
  parity := 0,
  baud := 57600,
  nc := 16#00
);
```

14.4.2. SIM.730.34 über virtuelle Schnittstelle (CAN-Bus):

Das Modem im SIM.730.34 kann auch direkt über den CAN-Bus des Moduls betrieben werden (virtuelle Schnittstelle). Die Initialisierung erfolgt mit dem Baustein `SIM_730_COM` aus der Bibliothek `SIM_COM.lib` (siehe Abschnitt 2.4), der die benötigte `comm_nr` als Ausgang liefert. Die Parameter `NodeId`, `CanDevNo` und `FirstQbAddress` müssen mit den entsprechenden Modul-Parametern aus der Steuerungskonfiguration übereinstimmen.

```
VAR
  simCom: SIM_730_COM;
END_VAR
simCom(
  NodeId := 2,
  CanDevNo := 0,
  FirstQbAddress := ADR(%QB1.0.0)
);
comm_nr := WORD_TO_BYTE( simCom.comm_nr );
```

14.4.3. USB-Modem über virtuelle Schnittstelle (z.B. für SIM.730.36):

Dies erfordert ein LZS mit der Version v21606204 oder höher.

```
comm_nr := 7;      (* USB device *)
comName := 'ttyACM3';
SystemSetParameter( 10369, ADR(comName) );
```

14.5. Zusätzliche AT-Kommandos

Die Bibliothek erlaubt das Ausführen von zusätzlichen AT-Kommandos nach dem Initialisieren des Modems vor dem Senden der SMS.

Hierzu existiert das globale String-Array `Plm_SMS_UserCmd_1[]`, in das jeweils max. vier AT-Befehle eingetragen werden können. Die Ergebnisse erscheinen im String-Array `Plm_SMS_UserCmd_1_Result[]`.

Beispiel: Abfrage der Empfangsfeldstärke mit AT+CSQ

```
Plm_SMS_UserCmd_1[0] := 'AT+CSQ';
(* Ergebnis nach Ausführung in Plm_SMS_UserCmd_1_Result[0] = '$R$N+CSQ: 18,0$R$N$R$NOK$R$N' *)
```

Die Auswertung muss mit eigenem Programmcode erfolgen.

14.6. Zeichensatz

Beim Versenden von SMS können nicht alle Zeichen verwendet werden. Die möglichen Zeichen hängen vom speziellen Modem ab. Die Verwendung nicht vom Modem unterstützter SMS-Zeichen kann zu Problemen bei der Übertragung führen.

Die folgende Tabelle zeigt die vom SIM.730.36 unterstützten Zeichencodes:

		Most Significant Nibble							
		0x	1x	2x	3x	4x	5x	6x	7x
Least Significant Nibble	x0			SP ¹	0	@	P		p
	x1			!	1	A	Q	a	q
	x2			“	2	B	R	b	r
	x3			#	3	C	S	c	s
	x4			\$	4	D	T	d	t
	x5			%	5	E	U	e	u
	x6			&	6	F	V	f	v
	x7			‘	7	G	W	g	w
	x8			(8	H	X	h	x
	x9)	9	I	Y	i	y
	xA	LF ²		*	:	J	Z	j	z
	xB			+	;	K		k	
	xC			,	<	L		l	
	xD	CR ³		-	=	M		m	
	xE			.	>	N		n	
	xF			/	?	O	£	o	

¹ – SP stands for space character

² – LF stands for Line Feed character

³ – CR stands for Carriage Return character

14.7. Fehlersuche

Zur Fehlersuche bietet die Bibliothek außer der Auswertung des Ausgangs `ErrCode` am Baustein `Plm_SMS_Send()` verschiedene Diagnosemöglichkeiten.

Aktivieren Sie zunächst die Log-Datei. Dazu muss der Wert 1 in die globale Bibliotheksvariable `Plm_SMS_LogMode` geschrieben werden (entspricht der Voreinstellung). Anschließend wird bei jedem Start des SMS-Bausteins eine Textdatei mit dem Namen `'_plm_sms_log.txt'` auf Laufwerk `a/` der Steuerung erzeugt. Diese Textdatei kann mit einem FTP-Client heruntergeladen und in einem Texteditor betrachtet werden.

Stellen Sie fest, ob die Datenverbindung zum Modem funktioniert. Bei einem CAN-Bus-Teilnehmer (SIM.730.34) muss die CAN-Status-LED am Modul grün dauerleuchten. Außerdem muss die virtuelle serielle Verbindung korrekt initialisiert sein (Input-Parameter am Baustein `SIM_730_COM` überprüfen). Der Ausgang `comm_nr` am Baustein `SIM_730_COM` muss einen Wert größer oder gleich 10 liefern. Bei fehlerhafter Verbindung zum Modem werden die Fehlercodes 1020, 1021 oder 1022 geliefert.

Bei funktionierender Datenverbindung erscheint in der Log-Datei die Zeile `"testing modem ok"` (ca. Zeile 12). Prüfen Sie in diesem Zusammenhang auch den Parameter `Mode` am Baustein `Plm_SMS_Send()`: Bei einigen Modems muss `Mode` den Wert `16#0010` enthalten (Befehle müssen mit LF abgeschlossen werden), bei anderen Modems nicht (Befehle müssen mit CR abgeschlossen werden). Bei Verwendung des Modems SIM.730.34 braucht der Wert `16#0010` nicht gesetzt zu werden.

Muss die SIM-Karte mit einer PIN freigeschaltet werden? Wenn hier wiederholt die falsche PIN verwendet wird, ist die SIM-Karte evtl. automatisch gesperrt worden (üblicherweise nach drei Fehlversuchen). Es empfiehlt sich, dies durch Einlegen der SIM-Karte in ein Handy zu überprüfen. Falls die SIM-Karte es erlaubt, sollte die Abfrage der PIN generell deaktiviert werden, um diese Fehlerquelle auszuschließen.

Kann das Modem sich in das GSM-Mobilfunknetz einbuchen? Beim SIM.730.34 leuchtet im eingebuchten Zustand die LED "GSM" auf der Front. Falls diese LED nicht leuchtet, ist zunächst der Anschluss und die Lage der Antenne zu überprüfen. Ggf. reicht die Netzabdeckung am Aufstellort der Antenne nicht aus, in diesem Fall muss der Aufstellort verändert werden. Das Einbuchen des Modems in das GSM-Mobilfunknetz kann nach dem Einschalten mehrere Minuten dauern, vorher ist kein Versand von SMS möglich. Im Log erkennt man das an einem Fehlerabbruch im Abschnitt "Sending SMS header" mit den Fehlercodes 1210, 1211 und 1212.

Erlaubt die SIM-Karte den Versand von SMS' und ist noch ausreichend Guthaben vorhanden (z.B. bei Prepaid-Tarifen)? Testen Sie dies, indem Sie die Karte in ein Handy einlegen und eine SMS versenden. Bei reinen Datentarifen ist der SMS-Versand u.U. nicht möglich. In der Log-Datei ist dann ein Timeout-Fehler hinter "sending sms text" (kurz vor Dateiende) ersichtlich.

Wurden unerlaubte Zeichen in der Textnachricht (Eingang `Message` am Baustein `Plm_SMS_Send()`) verwendet? Dies kann zu einem Fehlerabbruch mit Fehlercode 1220, 1221 oder 1222 führen.

Falls lange Nachrichten abgeschnitten sind: Wird mind. die Bibliothek `PLM_SMS-v20180504` verwendet? Ist die Nachricht kürzer als 160 Zeichen? Sind die zum Speichern der Message vorgesehenen String-Variablen ausreichend groß, d.h. mind. vom Typ `STRING(160)`?

Wenn der Baustein `Plm_SMS_Send()` "Ok" liefert, aber dennoch keine SMS ankommt, wurde die Nachricht zwar erfolgreich an den Mobilfunkprovider übertragen, kann von diesem jedoch nicht ausgeliefert werden. Prüfen Sie, ob die `DialNo` korrekt ist und ob das Empfangs-Handy SMS' empfangen kann. Unter Umständen kann zwischen Senden und Empfangen einer SMS eine Zeit von einigen Minuten vergehen.

Beispiel für eine Log-Datei bei erfolgreichem SMS-Versand mit Modem-Modul SIM.730.34 (Kommunikation über CAN-Bus) und `Mode = 16#0003`:

```

-----
SMS send started
testing modem
  modem chat started
    tx: AT[0D]
    tx:
    rx: AT[0D][0D][0A]
    rx: OK[0D][0A]
    rx:
  modem chat finished: OK
testing modem ok
sending modem reset
  modem chat started
    tx: AT&F[0D]
    tx:
    rx: AT&F[0D][0D][0A]
    rx: OK[0D][0A]
    rx:
  modem chat finished: OK
modem reset ok
testing SIM device
  modem chat param started
    tx: AT+CPIN?[0D]
    tx:
    rx: AT+CPIN?[0D][0D][0A]
    rx: +CPIN:[20]READY[0D][0A]
    rx:
  modem chat param finished: timeout error
testing SIM device
  modem chat param started
    tx: AT+CPIN?[0D]
    tx:
    rx: AT+CPIN?[0D][0D][0A]
    rx: +CPIN:[20]READY[0D][0A]
    rx:
  modem chat param finished: OK
SIM card ok, no PIN needed
setting SMS textmode
  modem chat started
    tx: AT+CMGF=1[0D]
    tx:

```



```
rx: AT+CMGF=1[0D][0D][0A]
rx: OK[0D][0A]
rx:
modem chat finished: OK
setting textmode ok
sending SMS header
modem chat started
tx: AT+CMGS="01722397420"[0D]
tx:
rx: AT+CMGS="01722397420"[0D][0D][0A]
rx: >[20]
modem chat finished: OK
sending SMS header ok
sending SMS text
modem chat param started
tx: Hello[20]world![1A]
rx: Hello[20]world![1A][0D][0A]
rx: +CMGS:[20]34[0D][0A]
rx: [0D][0A]
rx: OK[0D][0A]
rx:
modem chat param finished: OK
sending SMS text ok
SMS send finished successfully
```

15. Uhrzeit und Datum

15.1. Allgemeines

Alle PLM-Steuerungen besitzen eine eingebaute Echtzeituhr (engl. RTC, Real TimeClock), die, einmal gestellt, die korrekte Uhrzeit und das Datum auch bei abgeschalteter Steuerung weiterführt.

Die Verwendung der Echtzeituhr im IEC-Programm und Darstellungen von Datum und Uhrzeit werden durch die Bibliothek `Plm_Time.lib` erleichtert. Die Bibliothek enthält Programmbausteine zum einfachen Abfragen und Setzen der Echtzeituhr, zur Berücksichtigung von Sommer-/Winterzeitumschaltung (engl. DST, Daylight Saving Time) und zum Erzeugen von Strings mit Uhrzeit und Datum in verschiedenen Formaten.

Die Bibliothek enthält auch einen Programmbaustein zur Abfrage von NTP/SNTP-Servern. Dabei handelt es sich um Zeit-Server im Internet, die die genaue Uhrzeit bereitstellen.

Zusätzlich zu den Funktionen steht eine Visu-Bibliothek zur Verfügung, mit der auf einfachste Weise ein Dialog zum Stellen der Uhrzeit in eigene Projekte eingebunden werden kann.

Die Bibliothek arbeitet auf Coldfire- und PLM700-A-Systemen in ähnlicher Weise, dennoch sind einige Unterschiede nicht zu vermeiden. Hierauf wird bei den einzelnen Bausteinen hingewiesen.

15.2. Lokale Zeit und Sommer-/Winterzeitumschaltung

Zeitangaben beziehen sich international immer auf die sogenannte "Koordinierte Weltzeit" (engl. UTC, Universal Time Coordinated), die der Greenwich Mean Time am Nullmeridian entspricht. Die UTC wird auch von Zeit-Servern im Internet geliefert.

Als Ortszeit oder "lokale Zeit" (engl. Local Time) wird die Zeit am Aufstellort der PLM-Steuerung bezeichnet. Diese lokale Zeit weicht, bedingt durch geografische Lage und Sommer-/Winterzeitumschaltung, von der UTC ab. Während die Abweichung durch die geografische Lage konstant ist, hängt die Abweichung durch Sommer-/Winterzeitumschaltung vom Datum und von der zugrundeliegenden Gesetzgebung ab (in Deutschland z.B. basierend auf dem Zeitgesetz vom 25. Juli 1978 und div. Zusatzverordnungen). Beispielsweise besteht zwischen der lokalen Zeit in Berlin/Deutschland und der UTC im Winter eine Zeitdifferenz von 1 Stunde (deutsche Winterzeit), im Sommer von 2 Stunden (deutsche Sommerzeit).

Es gibt mehrere Möglichkeiten, Zeitdifferenzen gegenüber der UTC und die Sommer-/Winterzeitumschaltung korrekt zu berücksichtigen. Bei Verwendung der Bibliothek `Plm_Time.lib` ist folgende Vorgehensweise vorgesehen:

- Die RTC wird auf lokale Winterzeit gestellt, z.B. in Berlin/Deutschland: $\text{RTC-Zeit} = \text{UTC} + 1 \text{ Stunde}$.
- Die Berücksichtigung von Sommer-/Winterzeit erfolgt durch den Abfragebaustein `Plm_GetRtc()`.
- Nur PLM700-A: Damit die Zeitdifferenz zwischen der lokalen Zeit und der UTC richtig berücksichtigt wird, muss auf der Steuerung die korrekte Zeitzone eingestellt werden, z.B. `Europe/Berlin` für Deutschland. Dies kann entweder mit dem Baustein `Plm_SetTimezone()` durch das IEC-Programm erfolgen (siehe Abschnitt 15.3.9), oder über die Konfigurations-Website der Steuerung.

Diese Vorgehensweise hat den Vorteil, dass die Echtzeituhr nicht manuell auf Sommer-/Winterzeit umgestellt werden muss und ein Uhrzeitabgleich durch einen NTP/SNTP-Server einfach möglich ist. Die entsprechend anzugebenden Parameter können den untenstehenden Beispielen entnommen werden.

Nur Coldfire: Ab LZS v2110624 kann die Bibliothek die Kontrolle über alle RTC-Zeitfunktionen der Steuerung übernehmen (z.B. `SysVar`, `Dateidatum`, `Fehlertagebuch` etc.). Hierzu ist der Baustein `Plm_EnableSysTime()` aufzurufen (siehe Abschnitt 15.3.1).

15.3. Benötigte Bibliotheken

PLM_Time_v20150413.lib
Plm_TimeVisu.lib (optional)
Plm_Std.lib
UPD_E_005.lib (oder spätere Version)
SysLibSockets.lib

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

15.3.1. Plm_EnableSysTime (PLM_Time.lib)

Dieser Baustein hat auf PLM700-A Systemen keine Funktion.

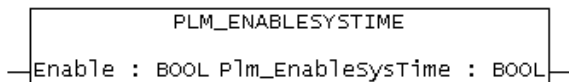


Abb. 15-1: Baustein Plm_EnableSysTime() (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	TRUE = Bibliothek übernimmt die Steuerung der Systemzeit (erfordert LZS ab v2110624) FALSE = Abschalten der Funktion
Rückgabewert:		
	BOOL	TRUE, wenn Steuerung der Systemzeit durch Bibliothek erfolgt

Die Funktion Plm_EnableSysTime() muss einmalig beim Programmstart mit dem Parameter TRUE aufgerufen werden. Die Bibliothek Plm_TimeLib übernimmt damit die Kontrolle über alle RTC-Zeitfunktionen der Steuerung (z.B. SysVar, Dateidatum, Fehlertagebuch etc.).

Nach Aufruf dieser Funktion muss die Funktion Plm_GetRtc() zyklisch im Programm aufgerufen werden.

Diese Funktion ist ab LZS v2110624 verfügbar.

15.3.2. Plm_GetRtc (PLM_Time.lib)

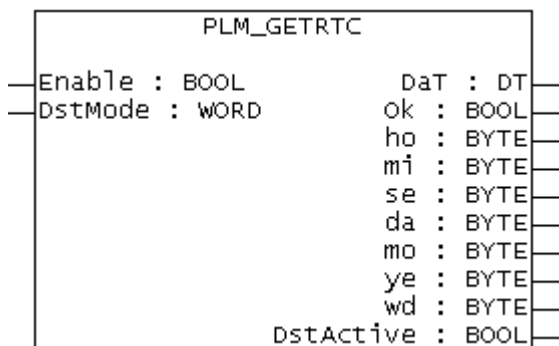


Abb. 15-2: Baustein Plm_GetRtc() (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	Auslesen der RTC erfolgt bei Enable = TRUE

DstMode	WORD	(keine Funktion bei PLM700-A) Berücksichtigung von Sommer- und Winterzeit (vgl. Plm_CalcDST), z.B. 0 = keine Sommerzeit 1 = deutsche Sommerzeit berücksichtigen
Output-Parameter:		
DaT	DATE_AND_TIME	Datum und Uhrzeit im DATE_AND_TIME-Format
Ok	BOOL	Ausgabewerte sind gültig
ho	BYTE	Stunden (0...23)
mi	BYTE	Minuten (0...59)
se	BYTE	Sekunden (0...59)
da	BYTE	Tag (1...31)
mo	BYTE	Monat (1...12)
ye	BYTE	Jahr (0...99 für die Jahre 2000...2099)
wd	BYTE	Wochentag (0=Montag, 1=Dienstag ... 6=Sonntag)
DstActive	BOOL	TRUE = Sommerzeit ist aktiv, FALSE = Winterzeit

Zur universellen Weiterverarbeitung wird die Zeit der RTC sowohl im kompakten Variablentyp DATE_AND_TIME als auch in einzelnen BYTE-Werten geliefert.

Wenn an mehreren Stellen im Programm die ausgegebene Uhrzeit benötigt wird, empfiehlt es sich aus Effizienzgründen, Plm_GetRtc() nur an einer einzigen Stelle im Programm aufzurufen und die benötigten Ausgabewerte in globalen Variablen zu speichern, die dann von allen Stellen zugreifbar sind.

Nur Coldfire: Bei Verwendung von Plm_EnableSysTime() (siehe Abschnitt 15.3.1) muss Plm_GetRtc() in jedem Zyklus aufgerufen werden.

Nur Coldfire: Der Algorithmus zur Berücksichtigung der deutschen Sommer- und Winterzeit ist fest in der Bibliothek implementiert und wird durch DstMode ungleich Null aktiviert. Die ausgegebene Zeit liefert dann datumsabhängig Sommerzeit oder Winterzeit. Dies setzt voraus, dass die RTC immer auf Winterzeit läuft. Der Ausgang DstActive signalisiert, ob gerade Sommerzeit oder Winterzeit ausgegeben wird.

Nur PLM700-A: Die ausgegebene Zeit berücksichtigt automatisch die Sommer-/Winterzeitschaltung auf Basis der eingestellten Zeitzone (siehe Abschnitt 15.3.9).

15.3.3. Plm_SetRtc (PLM_Time.lib)

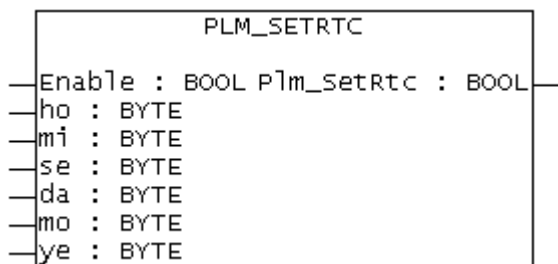


Abb. 15-3: Baustein Plm_SetRtc() (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	TRUE bewirkt die Übernahme der

		anliegenden Werte in die RTC
ho	BYTE	Stunden (0...23)
mi	BYTE	Minuten (0...59)
se	BYTE	Sekunden (0...59)
da	BYTE	Tag (1...31)
mo	BYTE	Monat (1...12)
ye	BYTE	Jahr (0...99 für die Jahre 2000...2099)
Rückgabewert:		
	BOOL	TRUE, wenn die RTC erfolgreich gesetzt wurde

Der Wochentag (Montag-Sonntag) kann nicht angegeben zu werden, da er intern aus dem angegebenen Datum eindeutig berechnet wird.

Diese Funktion schreibt direkt das angegebene Datum in die Echtzeituhr.

Nur Coldfire: Bei Verwendung mit einem Eingabedialog, in den je nach Datum Sommer- oder Winterzeit eingegeben werden, muss stattdessen die Funktion `Plm_SetRtcDst()` verwendet werden (siehe Abschnitt 15.3.4).

15.3.4. Plm_SetRtcDst (PLM_Time.lib)

Dieser Baustein hat auf PLM700-A Systemen keine Funktion.

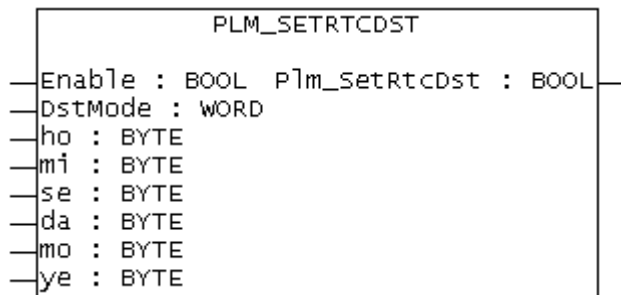


Abb. 15-4: Baustein `Plm_SetRtcDst()` (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	TRUE bewirkt die Übernahme der anliegenden Werte in die RTC
DstMode	WORD	Berücksichtigung von Sommer- und Winterzeit (vgl. <code>Plm_CalcDST</code>), z.B. 0 = keine Sommerzeit 1 = deutsche Sommerzeit berücksichtigen
ho	BYTE	Stunden (0...23)
mi	BYTE	Minuten (0...59)
se	BYTE	Sekunden (0...59)
da	BYTE	Tag (1...31)
mo	BYTE	Monat (1...12)
ye	BYTE	Jahr (0...99 für die Jahre 2000...2099)
Rückgabewert:		
	BOOL	TRUE, wenn die RTC erfolgreich gesetzt wurde

Der Wochentag (Montag-Sonntag) braucht nicht angegeben zu werden, sondern wird aus dem angegebenen Datum eindeutig berechnet.

Wenn `DstMode` ungleich 0 ist, wird die Echtzeituhr datumsabhängig immer auf Winterzeit eingestellt, auch wenn eine Zeit in Sommerzeit eingegeben wurde.

15.3.5. Plm_GetSntpTime (PLM_Time.lib)

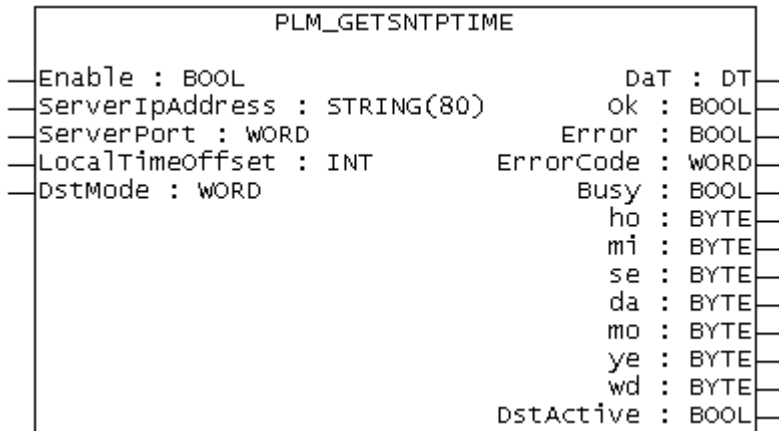


Abb. 15-5: Baustein `Plm_GetSntpTime()` (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	Änderung von FALSE auf TRUE an diesem Eingang bewirkt Start der Abfrage
ServerIpAddress	STRING	IP-Adresse eines NTP/SNTP-Servers, z.B. '192.53.103.108'
ServerPort	WORD	UDP-Port des Servers, üblicherweise 123
LocalTimeOffset	INT	(keine Funktion bei PLM700-A) Zeitdifferenz in Minuten zwischen lokaler Zeit und der UTC des Servers, z.B. +60 für Berlin/Deutschland
DstMode	WORD	(keine Funktion bei PLM700-A) Berücksichtigung von Sommer- und Winterzeit (vgl. <code>Plm_CalcDST</code>). Wenn das Ergebnis der Serverabfrage zum Stellen der RTC verwendet wird, sollte hier 0 angegeben werden.
Output-Parameter:		
Ok	BOOL	TRUE = Abfrage war erfolgreich, Ergebniswerte sind gültig
Error	BOOL	TRUE = Fehler bei Abfrage
ErrorCode	WORD	Fehlercode bei <code>Error = TRUE</code> : 1 = cannot create socket, 2 = sending request failed, 3 = receive failed, 4 = illegal server address or port, 5 = server timeout, 6 = illegal server response
Busy	BOOL	TRUE = Abfrage läuft
ho	BYTE	Stunden (0...23)

ho	BYTE	Stunden (0...23)
mi	BYTE	Minuten (0...59)
se	BYTE	Sekunden (0...59)
da	BYTE	Tag (1...31)
mo	BYTE	Monat (1...12)
ye	BYTE	Jahr (0...99 für die Jahre 2000...2099)
wd	BYTE	Wochentag (0=Montag, 1=Dienstag ... 5=Samstag, 6=Sonntag)
DstActive	BOOL	TRUE = Sommerzeit ist aktiv, FALSE = Winterzeit

Der Programmblock `Plm_GetSntpTime()` führt die Abfrage eines NTP/SNTP-Servers im Internet durch.

Die NTP/SNTP-Serverabfrage erfolgt über UDP/IP, daher ist eine funktionierende Netzwerk- bzw. Internetanbindung erforderlich. Insbesondere müssen die IP-Adresse der Steuerung, die Netzmaske und der Default Gateway richtig eingestellt sein.

Bei erfolgreicher Serverabfrage sind für genau einen IEC-Zyklus die Ausgänge `Busy` und `Ok` gleichzeitig auf TRUE. Durch eine AND-Operation lässt sich daraus ein kurzer Impuls zum Stellen der RTC mittels `Plm_SetRtc()` gewinnen.

Nur Coldfire: Die vom SNTP-Server gelieferte UTC wird durch Angabe eines `LocalTimeOffset` in die lokale Zeit (Ortszeit) umgerechnet, z.B. `LocalTime = 60` für Berlin/Deutschland. Wenn das Ergebnis der Serverabfrage zum Stellen der RTC verwendet wird, wird empfohlen, immer `DstMode = 0` anzugeben und die Berücksichtigung von Sommer-/Winterzeit bei der Abfrage der RTC in `Plm_GetRtc` durchzuführen. Die von `Plm_GetSntpTime` gelieferte Zeitangabe entspricht dann der deutschen Winterzeit.

Nur PLM700-A: Die vom SNTP-Server gelieferte UTC wird automatisch in die lokale Zeit (Ortszeit) umgerechnet. Dazu muss die Zeitzone richtig eingestellt sein (siehe Abschnitt 15.3.9).

15.3.6. Plm_CalcDST (PLM_Time.lib)

Dieser Baustein hat auf PLM700-A Systemen keine Funktion.

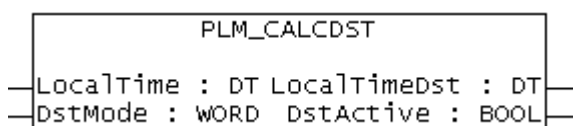


Abb. 15-6: Baustein `Plm_CalcDST()` (PLM_Time.lib)

Input-Parameter:		
LocalTime	DATE_AND_TIME	Momentane lokale Zeit
DstMode	WORD	Berücksichtigung von Sommer- und Winterzeit: 0 = keine Berücksichtigung 1 = deutsche Sommerzeit berücksichtigen
Output-Parameter:		
LocalTimeDst	DATE_AND_TIME	Lokale Zeit mit Berücksichtigung der Sommer- und Winterzeitumschaltung gemäß <code>DstMode</code>
DstActive	BOOL	TRUE = Sommerzeit ist aktiv, FALSE = Winterzeit

Der Baustein `Plm_CalcDST` wird intern verwendet und enthält den Algorithmus zur automatischen Sommer-/Winterzeitschaltung (DST = Daylight Saving Time). Die verfügbaren Länderalgorithmen sind dem Kommentar in der Bibliothek zu entnehmen.

15.3.7. Plm_FormatTime (PLM_Time.lib)

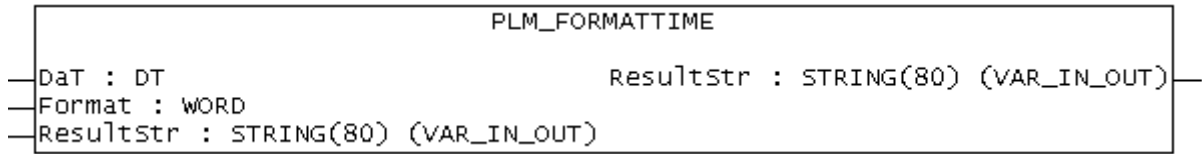


Abb. 15-7: Baustein `Plm_FormatTime()` (PLM_Time.lib)

Input-Parameter:		
DaT	DATE_AND_TIME	Zeitwert im Format DATE_AND_TIME
Format	WORD	Nummer des gewählten Formats, z.B. 0
Input-Output-Parameter:		
ResultStr	STRING	Formatierter Zeitwert, z.B. '2009-01-31 16:25:44'

Formatiert den angegebenen Zeitwert in `DaT` in den String `ResultStr`. Die Formatierung wird mit dem Parameter `Format` aus einer Liste von vorgefertigten Formatierungen ausgewählt.

Die verfügbaren Werte für `Format` sind dem Kommentar in der Bibliothek zu entnehmen. Falls eine dort nicht aufgeführte Formatierung benötigt wird, kann diese mit dem Programmblock `Plm_FormatTimeExt` erzeugt werden.

Der `ResultStr`, in den der formatierte Zeitwert eingetragen wird, muss eine ausreichende Länge haben (abhängig von `Format`) und wird aus Effizienzgründen als Input-Output-Parameter (`VAR_IN_OUT`) übergeben.

15.3.8. Plm_FormatTimeExt (PLM_Time.lib)

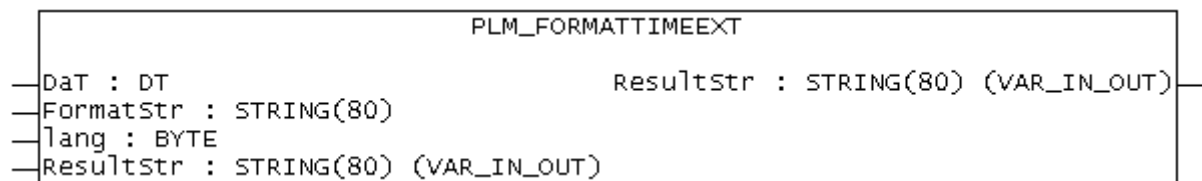


Abb. 15-8: Baustein `Plm_FormatTimeExt()` (PLM_Time.lib)

Input-Parameter:		
DaT	DATE_AND_TIME	Zeitwert im Format DATE_AND_TIME
FormatStr	STRING	Formatbeschreibung, z.B. '%Y-%m-%d %H:%M:%S'
lang	BYTE	Nummer der Sprache für Klartextangaben wie 'Montag', vgl. <code>Plm_WeekdayName</code> und <code>Plm_MonthName</code>
Input-Output-Parameter:		
ResultStr	STRING	Formatierter Zeitwert, z.B.


```

'2009-01-31 16:25:44'
    
```

Formatiert den angegebenen Zeitwert in DaT in den String ResultStr. Die Formatierung wird mit dem Parameter FormatStr in einer Syntax spezifiziert, die an das UNIX-Programm date angelehnt ist. Die genaue Syntax und die verfügbaren Platzhalter sind dem Kommentar in der Bibliothek zu entnehmen.

Für Standardformate steht der einfacher zu verwendende Programmblock Plm_FormatTime zur Verfügung.

Der Parameter lang (Language) wird nur ausgewertet, wenn durch einen der Platzhalter %A, %a, %B oder %b eine Klartextangabe wie z.B. 'Montag' oder 'September' angefordert wird. Die verfügbaren Werte für lang sind den Kommentaren zu Plm_WeekdayName und Plm_MonthName in der Bibliothek zu entnehmen.

Der ResultStr, in den der formatierte Zeitwert eingetragen wird, muss eine ausreichende Länge haben (abhängig von FormatStr) und wird aus Effizienzgründen als Input-Output-Parameter (VAR_IN_OUT) übergeben. Nach dem Aufruf von Plm_FormatTime enthält der übergebene ResultStr den angegebenen Zeitwert in der gewünschten Formatierung.

15.3.9. Plm_SetTimezone (PLM_Time.lib)

Dieser Baustein hat nur auf PLM700-A Systemen eine Funktion.

```

PLM_SETTIMEZONE
tzStrAdr : POINTER TO BYTE
    
```

Abb. 15-9: Baustein Plm_SetTimezone() (PLM_Time.lib)

Input-Parameter:		
tzStrAdr	POINTER TO BYTE	Adresse ADR () eines Strings mit dem Namen der Zeitzone

Der Baustein braucht nur einmalig aufgerufen zu werden und stellt die aktuelle Zeitzone der Steuerung auf den angegebenen Wert.

Alternativ kann die Zeitzone über die Konfigurations-Website der Steuerung eingestellt werden.

Mit der Zeitzone sind mehrere Informationen verknüpft, z.B. die Verschiebung der lokalen Zeit gegenüber UTC und die Zeitpunkte der Sommer-/Winterzeit-Umschaltung. Die Steuerung greift dabei auf eine interne Datenbank zu.

Für den Betrieb in Deutschland ist als Zeitzone Europe/Berlin einzustellen.

Der Name der Zeitzone muss aus folgender Tabelle gewählt werden; die Schreibweise muss dabei genau übereinstimmen:

Africa/Abidjan	America/St_Kitts	Etc/GMT-13
Africa/Accra	America/St_Lucia	Etc/GMT-14
Africa/Addis_Ababa	America/St_Thomas	Etc/GMT-2
Africa/Algiers	America/St_Vincent	Etc/GMT-3
Africa/Asmara	America/Swift_Current	Etc/GMT-4
Africa/Asmera	America/Tegucigalpa	Etc/GMT-5
Africa/Bamako	America/Thule	Etc/GMT-6
Africa/Bangui	America/Thunder_Bay	Etc/GMT-7
Africa/Banjul	America/Tijuana	Etc/GMT-8
Africa/Bissau	America/Toronto	Etc/GMT-9
Africa/Blantyre	America/Tortola	Etc/GMT0
Africa/Brazzaville	America/Vancouver	Etc/Greenwich
Africa/Bujumbura	America/Virgin	Etc/UCT
Africa/Cairo	America/Whitehorse	Etc/UTC
Africa/Casablanca	America/Winnipeg	Etc/Universal
Africa/Ceuta	America/Yakutat	Etc/Zulu
Africa/Conakry	America/Yellowknife	Europe/Amsterdam
Africa/Dakar	Antarctica/Casey	Europe/Andorra
Africa/Dar_es_Salaam	Antarctica/Davis	Europe/Athens

Africa/Djibouti	Antarctica/DumontDURville	Europe/Belfast
Africa/Douala	Antarctica/Macquarie	Europe/Belgrade
Africa/El_Aaiun	Antarctica/Mawson	Europe/Berlin
Africa/Freetown	Antarctica/McMurdo	Europe/Bratislava
Africa/Gaborone	Antarctica/Palmer	Europe/Brussels
Africa/Harare	Antarctica/Rothera	Europe/Bucharest
Africa/Johannesburg	Antarctica/South_Pole	Europe/Budapest
Africa/Kampala	Antarctica/Syowa	Europe/Chisinau
Africa/Khartoum	Antarctica/Vostok	Europe/Copenhagen
Africa/Kigali	Arctic/Longyearbyen	Europe/Dublin
Africa/Kinshasa	Asia/Aden	Europe/Gibraltar
Africa/Lagos	Asia/Almaty	Europe/Guernsey
Africa/Libreville	Asia/Amman	Europe/Helsinki
Africa/Lome	Asia/Anadyr	Europe/Isle_of_Man
Africa/Luanda	Asia/Aqtou	Europe/Istanbul
Africa/Lubumbashi	Asia/Aqtobe	Europe/Jersey
Africa/Lusaka	Asia/Ashgabat	Europe/Kaliningrad
Africa/Malabo	Asia/Ashkhabad	Europe/Kiev
Africa/Maputo	Asia/Baghdad	Europe/Lisbon
Africa/Maseru	Asia/Bahrain	Europe/Ljubljana
Africa/Mbabane	Asia/Baku	Europe/London
Africa/Mogadishu	Asia/Bangkok	Europe/Luxembourg
Africa/Monrovia	Asia/Beirut	Europe/Madrid
Africa/Nairobi	Asia/Bishkek	Europe/Malta
Africa/Ndjamena	Asia/Brunei	Europe/Mariehamn
Africa/Niamey	Asia/Calcutta	Europe/Minsk
Africa/Nouakchott	Asia/Choibalsan	Europe/Monaco
Africa/Ouagadougou	Asia/Chongqing	Europe/Moscow
Africa/Porto-Novo	Asia/Chungking	Europe/Nicosia
Africa/Sao_Tome	Asia/Colombo	Europe/Oslo
Africa/Timbuktu	Asia/Dacca	Europe/Paris
Africa/Tripoli	Asia/Damascus	Europe/Podgorica
Africa/Tunis	Asia/Dhaka	Europe/Prague
Africa/Windhoek	Asia/Dili	Europe/Riga
America/Adak	Asia/Dubai	Europe/Rome
America/Anchorage	Asia/Dushanbe	Europe/Samara
America/Anguilla	Asia/Gaza	Europe/San_Marino
America/Antigua	Asia/Harbin	Europe/Sarajevo
America/Araguaina	Asia/Ho_Chi_Min	Europe/Simferopol
America/Argentina/Buenos_Aires	Asia/Hong_Kong	Europe/Skopje
America/Argentina/Catamarca	Asia/Hovd	Europe/Sofia
America/Argentina/ComodRivadavia	Asia/Irkutsk	Europe/Stockholm
America/Argentina/Cordoba	Asia/Istanbul	Europe/Tallinn
America/Argentina/Jujuy	Asia/Jakarta	Europe/Tirane
America/Argentina/La_Rioja	Asia/Jayapura	Europe/Tiraspol
America/Argentina/Mendoza	Asia/Jerusalem	Europe/Uzhgorod
America/Argentina/Rio_Gallegos	Asia/Kabul	Europe/Vaduz
America/Argentina/Salta	Asia/Kamchatka	Europe/Vatican
America/Argentina/San_Juan	Asia/Karachi	Europe/Vienna
America/Argentina/San_Luis	Asia/Kashgar	Europe/Vilnius
America/Argentina/Tucuman	Asia/Kathmandu	Europe/Volgograd
America/Argentina/Ushuaia	Asia/Katmandu	Europe/Warsaw
America/Aruba	Asia/Kolkata	Europe/Zagreb
America/Asuncion	Asia/Krasnoyarsk	Europe/Zaporozhye
America/Atikokan	Asia/Kuala_Lumpur	Europe/Zurich
America/Atka	Asia/Kuching	Factory
America/Bahia	Asia/Kuwait	GB
America/Bahia_Banderas	Asia/Macao	GB-Eire
America/Barbados	Asia/Macau	GMT
America/Belem	Asia/Magadan	GMT+0
America/Belize	Asia/Makassar	GMT-0
America/Blanc-Sablon	Asia/Manila	GMT0
America/Boa_Vista	Asia/Muscat	Greenwich
America/Bogota	Asia/Nicosia	HST
America/Boise	Asia/Novokuznetsk	Hongkong
America/Buenos_Aires	Asia/Novosibirsk	Iceland
America/Cambridge_Bay	Asia/Omsk	Indian/Antananarivo
America/Campo_Grande	Asia/Oral	Indian/Chagos
America/Cancun	Asia/Phnom_Penh	Indian/Christmas
America/Caracas	Asia/Pontianak	Indian/Cocos
America/Catamarca	Asia/Pyongyang	Indian/Comoro
America/Cayenne	Asia/Qatar	Indian/Kerguelen
America/Cayman	Asia/Qyzylorda	Indian/Mahe
America/Chicago	Asia/Rangoon	Indian/Maldives
America/Chihuahua	Asia/Riyadh	Indian/Mauritius
America/Coral_Harbour	Asia/Riyadh87	Indian/Mayotte
America/Cordoba	Asia/Riyadh88	Indian/Reunion
America/Costa_Rica	Asia/Riyadh89	Iran
America/Cuiaba	Asia/Saigon	Israel
America/Curacao	Asia/Sakhalin	Jamaica
America/Danmarkshavn	Asia/Samarkand	Japan

America/Dawson	Asia/Seoul	Kwajalein
America/Dawson_Creek	Asia/Shanghai	Libya
America/Denver	Asia/Singapore	MET
America/Detroit	Asia/Taipei	MST
America/Dominica	Asia/Tashkent	MST7MDT
America/Edmonton	Asia/Tbilisi	Mexico/BajaNorte
America/Eirunepe	Asia/Tehran	Mexico/BajaSur
America/El_Salvador	Asia/Tel_Aviv	Mexico/General
America/Ensenada	Asia/Thimbu	Mideast/Riyadh87
America/Fort_Wayne	Asia/Thimphu	Mideast/Riyadh88
America/Fortaleza	Asia/Tokyo	Mideast/Riyadh89
America/Glace_Bay	Asia/Ujung_Pandang	NZ
America/Godthab	Asia/Ulaanbaatar	NZ-CHAT
America/Goose_Bay	Asia/Ulan_Bator	Navajo
America/Grand_Turk	Asia/Urumqi	PRC
America/Grenada	Asia/Vientiane	PST8PDT
America/Guadeloupe	Asia/Vladivostok	Pacific/Apia
America/Guatemala	Asia/Yakutsk	Pacific/Auckland
America/Guayaquil	Asia/Yekaterinburg	Pacific/Chatham
America/Guyana	Asia/Yerevan	Pacific/Chuuk
America/Halifax	Atlantic/Azores	Pacific/Easter
America/Havana	Atlantic/Bermuda	Pacific/Efate
America/Hermosillo	Atlantic/Canary	Pacific/Enderbury
America/Indiana/Indianapolis	Atlantic/Cape_Verde	Pacific/Fakaofu
America/Indiana/Knox	Atlantic/Faeroe	Pacific/Fiji
America/Indiana/Marengo	Atlantic/Faroe	Pacific/Funafuti
America/Indiana/Petersburg	Atlantic/Jan_Mayen	Pacific/Galapagos
America/Indiana/Tell_City	Atlantic/Madeira	Pacific/Gambier
America/Indiana/Vevay	Atlantic/Reykjavik	Pacific/Guadalcanal
America/Indiana/Vincennes	Atlantic/South_Georgia	Pacific/Guam
America/Indiana/Winamac	Atlantic/St_Helena	Pacific/Honolulu
America/Indianapolis	Atlantic/Stanley	Pacific/Johnston
America/Inuvik	Australia/ACT	Pacific/Kiritimati
America/Iqaluit	Australia/Adelaide	Pacific/Kosrae
America/Jamaica	Australia/Brisbane	Pacific/Kwajalein
America/Jujuy	Australia/Broken_Hill	Pacific/Majuro
America/Juneau	Australia/Canberra	Pacific/Marquesas
America/Kentucky/Louisville	Australia/Currie	Pacific/Midway
America/Kentucky/Monticello	Australia/Darwin	Pacific/Nauru
America/Knox_IN	Australia/Eucla	Pacific/Niue
America/La_Paz	Australia/Hobart	Pacific/Norfolk
America/Lima	Australia/LHI	Pacific/Noumea
America/Los_Angeles	Australia/Lindeman	Pacific/Pago_Pago
America/Louisville	Australia/Lord_Howe	Pacific/Palau
America/Maceio	Australia/Melbourne	Pacific/Pitcairn
America/Managua	Australia/NSW	Pacific/Pohnpei
America/Manaus	Australia/North	Pacific/Ponape
America/Marigot	Australia/Perth	Pacific/Port_Moresby
America/Martinique	Australia/Queensland	Pacific/Rarotonga
America/Matamoros	Australia/South	Pacific/Saipan
America/Mazatlan	Australia/Sydney	Pacific/Samoa
America/Mendoza	Australia/Tasmania	Pacific/Tahiti
America/Menominee	Australia/Victoria	Pacific/Tarawa
America/Merida	Australia/West	Pacific/Tongatapu
America/Metlakatla	Australia/Yancowinna	Pacific/Truk
America/Mexico_City	Brazil/Acre	Pacific/Wake
America/Miquelon	Brazil/DeNoronha	Pacific/Wallis
America/Moncton	Brazil/East	Pacific/Yap
America/Monterrey	Brazil/West	Poland
America/Montevideo	CET	Portugal
America/Montreal	CST6CDT	ROC
America/Montserrat	Canada/Atlantic	ROK
America/Nassau	Canada/Central	Singapore
America/New_York	Canada/East-Saskatchewan	SystemV/AST4
America/Nipigon	Canada/Eastern	SystemV/AST4ADT
America/Nome	Canada/Mountain	SystemV/CST6
America/Noronha	Canada/Newfoundland	SystemV/CST6CDT
America/North_Dakota/Beulah	Canada/Pacific	SystemV/EST5
America/North_Dakota/Center	Canada/Saskatchewan	SystemV/EST5EDT
America/North_Dakota/New_Salem	Canada/Yukon	SystemV/HST10
America/Ojinaga	Chile/Continental	SystemV/MST7
America/Panama	Chile/EasterIsland	SystemV/MST7MDT
America/Pangnirtung	Cuba	SystemV/PST8
America/Paramaribo	EET	SystemV/PST8PDT
America/Phoenix	EST	SystemV/YST9
America/Port-au-Prince	EST5EDT	SystemV/YST9YDT
America/Port_of_Spain	Egypt	Turkey
America/Porto_Acre	Eire	UCT
America/Porto_Velho	Etc/GMT	US/Alaska
America/Puerto_Rico	Etc/GMT+0	US/Aleutian
America/Rainy_River	Etc/GMT+1	US/Arizona

America/Rankin_Inlet	Etc/GMT+10	US/Central
America/Recife	Etc/GMT+11	US/East-Indiana
America/Regina	Etc/GMT+12	US/Eastern
America/Resolute	Etc/GMT+2	US/Hawaii
America/Rio_Branco	Etc/GMT+3	US/Indiana-Starke
America/Rosario	Etc/GMT+4	US/Michigan
America/Santa_Isabel	Etc/GMT+5	US/Mountain
America/Santarem	Etc/GMT+6	US/Pacific
America/Santiago	Etc/GMT+7	US/Pacific-New
America/Santo_Domingo	Etc/GMT+8	US/Samoa
America/Sao_Paulo	Etc/GMT+9	UTC
America/Scoresbysund	Etc/GMT-0	Universal
America/Shiprock	Etc/GMT-1	W-SU
America/Sitka	Etc/GMT-10	WET
America/St_Barthelemy	Etc/GMT-11	Zulu
America/St_Johns	Etc/GMT-12	

15.3.10. Plm_GetTimezone (PLM_Time.lib)

Dieser Baustein hat nur auf PLM700-A Systemen eine Funktion.

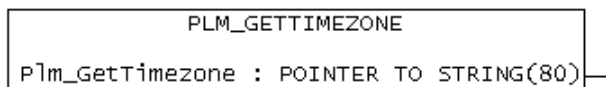


Abb. 15-10: Baustein Plm_GetTimezone () (PLM_Time.lib)

Rückgabewert:		
	POINTER TO STRING	Adresse ADR () eines Strings mit dem Namen der aktuellen Zeitzone

Der Baustein liefert beim Aufruf einen Zeiger auf einen String mit der aktuellen Zeitzone der Steuerung. Mögliche Werte für die Zeitzone sind beim Baustein Plm_SetTimezone () in Abschnitt 15.3.9 aufgelistet.

15.3.11. Plm_GetUtcTime (PLM_Time.lib)

Dieser Baustein hat nur auf PLM700-A Systemen eine Funktion.

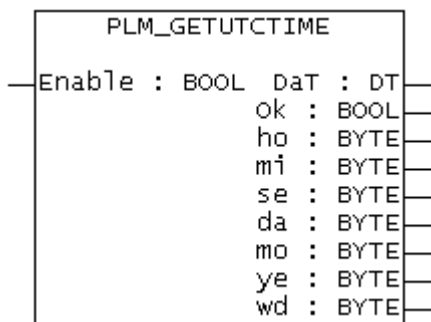


Abb. 15-11: Baustein Plm_GetUtcTime () (PLM_Time.lib)

Input-Parameter:		
Enable	BOOL	Auslesen der RTC erfolgt bei Enable = TRUE
Output-Parameter:		
DaT	DATE_AND_TIME	Datum und Uhrzeit im DATE_AND_TIME-Format
Ok	BOOL	Ausgabewerte sind gültig
ho	BYTE	Stunden (0...23)
mi	BYTE	Minuten (0...59)

se	BYTE	Sekunden (0...59)
da	BYTE	Tag (1...31)
mo	BYTE	Monat (1...12)
ye	BYTE	Jahr (0...99 für die Jahre 2000...2099)
wd	BYTE	Wochentag (0=Montag, 1=Dienstag ... 6=Sonntag)

Dieser Baustein verhält sich ähnlich wie `Plm_GetRtc()` (siehe Abschnitt 15.3.2), liefert jedoch anstelle der lokalen Zeit die ortsunabhängige UTC. Lokale Zeit und UTC sind in Abhängigkeit von der eingestellten Zeitzone gegeneinander verschoben.

15.4. Programmbeispiele (FUP und ST)

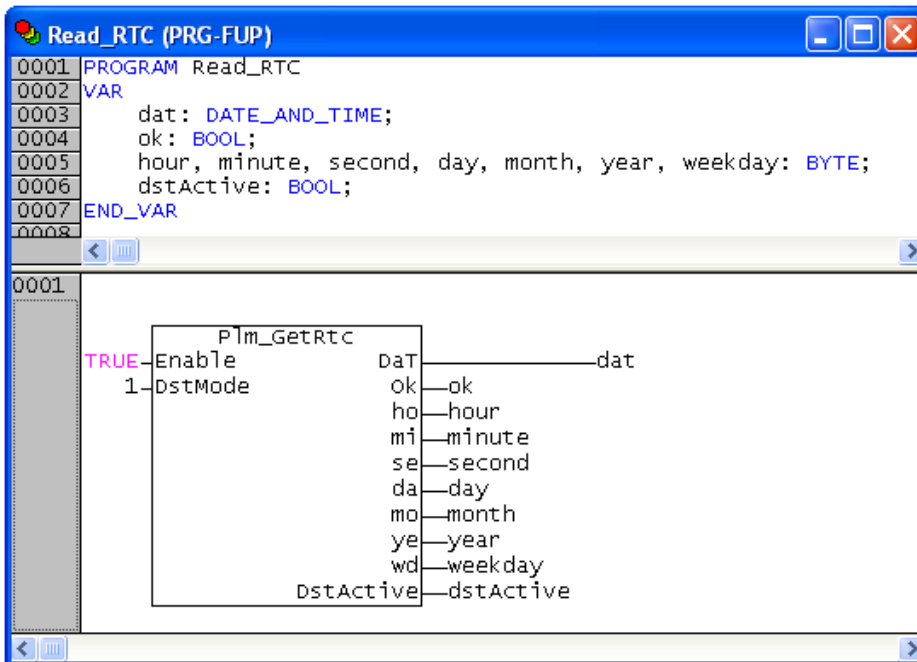


Abb. 15-12: Abfrage der Echtzeituhr (FUP)

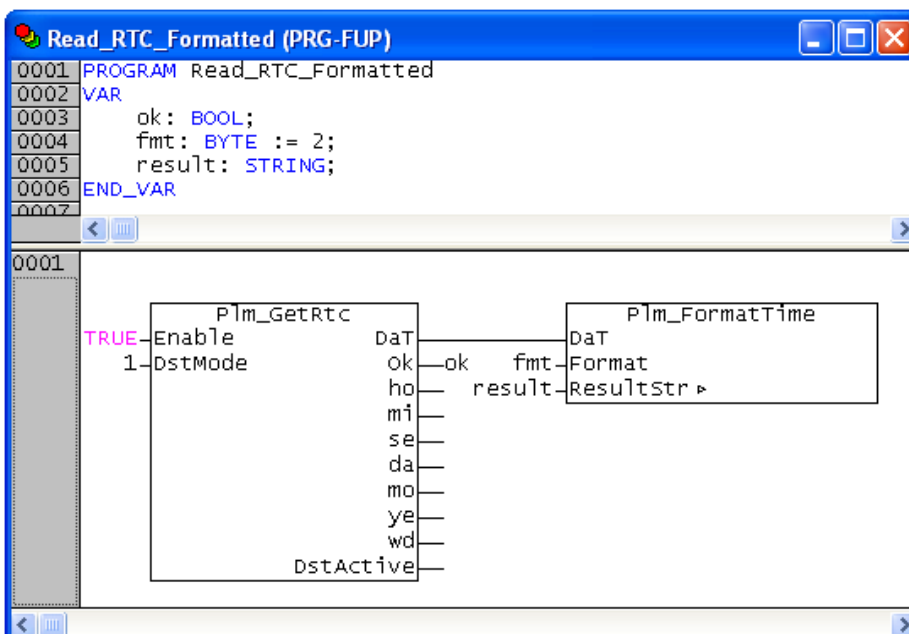


Abb. 15-13: Erzeugen eines Strings mit formatierter Zeitangabe (FUP)

Die Beispiele in Abb. 15-12 und Abb. 15-13 fragen die Echtzeituhr (RTC) ab.

Nur Coldfire: Durch `DstMode = 1` erfolgt die automatische Berücksichtigung der deutschen Sommerzeit bei Abfrage der RTC. Dies setzt voraus, dass die RTC stets auf Winterzeit läuft.

Nur PLM700-A: Der Eingang `DstMode` hat bei PLM700-A keine Bedeutung und wird ignoriert.

In Abb. 15-13 wird zusätzlich ein formatierter String `result` erzeugt, dessen Format mit `fmt = 2` aus der vorgegebenen Formatliste ausgewählt wird. Bei `fmt = 2` entsteht eine deutsche Datums- und Zeitangabe, z.B. '31.01.2009 16:25:44'. Weitere verfügbare Formate sind in der Bibliothek ersichtlich.

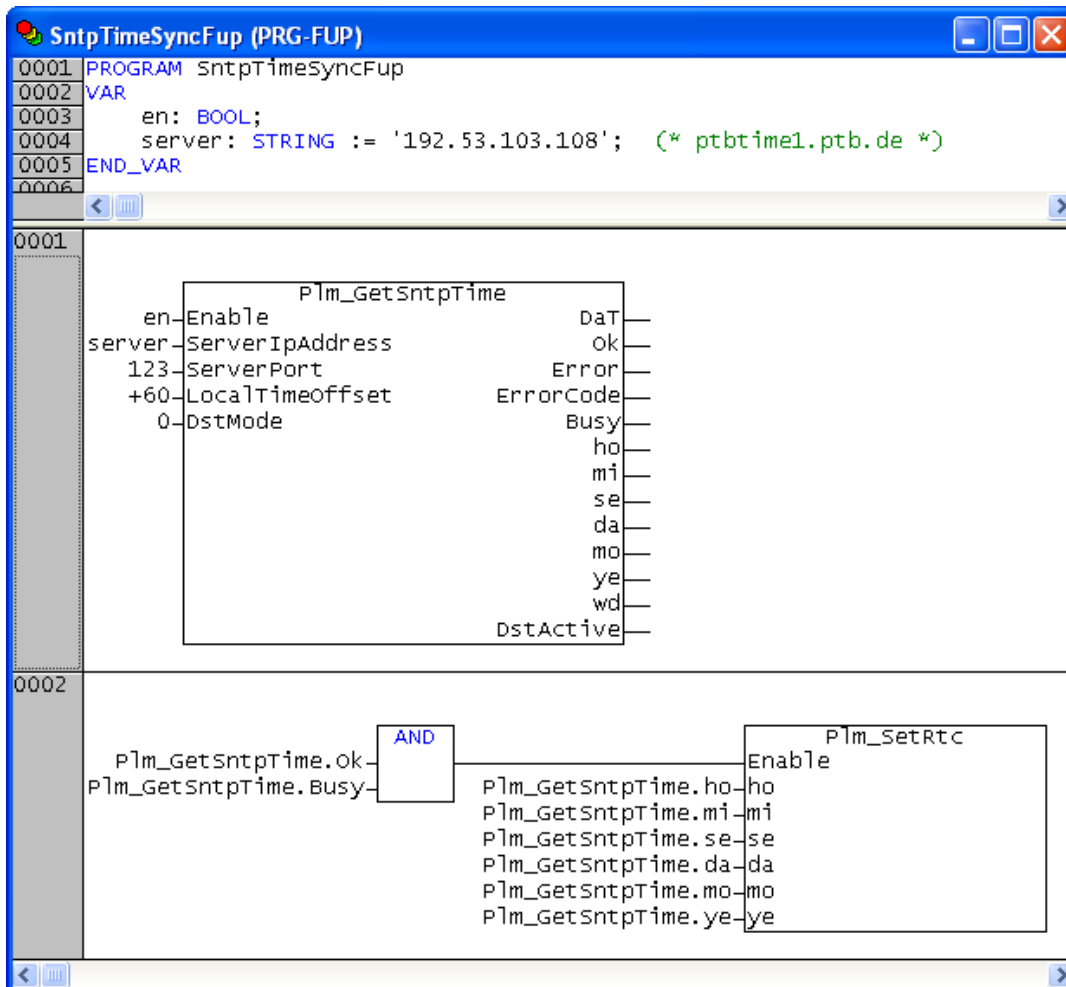


Abb. 15-14: Abfrage eines NTP/SNTP-Servers und anschließendes Stellen der RTC (FUP)

```

SntpTimeSync_ST (PRG-ST)
0001 PROGRAM SntpTimeSync_ST
0002 VAR
0003     en: BOOL;
0004     server: STRING := '192.53.103.108'; (* ptbtime1.ptb.de *)
0005 END_VAR
0006
0001 Plm_GetSntpTime(
0002     Enable := en,
0003     ServerIpAddress := server,
0004     ServerPort := 123,
0005     LocalTimeOffset := +60,
0006     DstMode := 0,
0007 );
0008
0009 Plm_SetRtc(
0010     Enable := (Plm_GetSntpTime.ok AND Plm_GetSntpTime.Busy), (* TRUE for 1 cycle *)
0011     ho := Plm_GetSntpTime.ho,
0012     mi := Plm_GetSntpTime.mi,
0013     se := Plm_GetSntpTime.se,
0014     da := Plm_GetSntpTime.da,
0015     mo := Plm_GetSntpTime.mo,
0016     ye := Plm_GetSntpTime.ye
0017 );
0018

```

Abb. 15-15: Abfrage eines NTP/SNTP-Servers und anschließendes Stellen der RTC (ST)

Das in Abb. 15-14 bzw. Abb. 15-15 gezeigte Beispiel fragt die aktuelle Uhrzeit bei einem NTP/SNTP-Server ab und stellt damit im Erfolgsfall die Echtzeituhr (RTC). Die Abfrage erfolgt einmalig, wenn `enable` von FALSE auf TRUE gesetzt wird.

Eine derartige Abfrage kann z.B. einmal am Tag durchgeführt werden, um eine genaue Synchronisation der RTC mit anderen Uhren sicherzustellen. In diesem Fall ist `enable` über einen entsprechenden Timer anzusteuern.

Listen der verfügbaren NTP/SNTP-Server finden sich im Internet. Die Nutzung der Server ist fast immer kostenlos, einige Serverbetreiber bitten jedoch um Benachrichtigung, wenn der Dienst regelmäßig in Anspruch genommen wird.

Nur Coldfire: Die vom Server gelieferte UTC wird durch die Angabe von `LocalTimeOffset = +60` in die lokale Zeit von Berlin/Deutschland umgerechnet. Da die RTC stets auf Winterzeit laufen soll, ist `DstMode = 0` angegeben.

Nur PLM700-A: Die Eingänge `LocalTimeOffset` und `DstMode` haben keine Bedeutung und werden ignoriert.

15.5. Verwendung der Visu-Bibliothek

Die Bibliothek `Plm_TimeVisu.lib` ergänzt die oben beschriebene Bibliothek `Plm_Time.lib` und enthält folgende vorgefertigte Blöcke:

- Baustein `Plm_SetClockPrg()`
- SubVisu `PLM_SETCLOCKVISU`

Mit diesen beiden Blöcken kann auf einfache Weise ein Einstelldialog für die Echtzeituhr in eigene Projekte eingebaut werden.

15.5.1. Blöcke der Bibliothek Plm_TimeVisu.lib

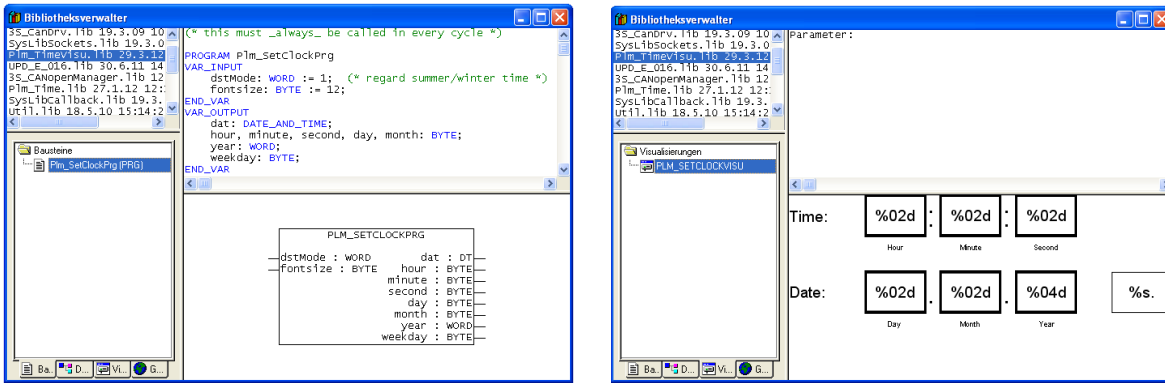


Abb. 15-16: Blöcke der Bibliothek *Plm_TimeVisu.lib*

Der Baustein `Plm_SetClockPrg()` muss zyklisch aufgerufen werden. Er übernimmt die Auswertung der Eingaben in die SubVisu und stellt die Uhrzeit in Variablen zur Verfügung.

Der Parameter `dstMode` entspricht dem gleichnamigen Parameter aus der Funktion `Plm_GetRtc()` (siehe Abschnitt 15.3.2).

Als Ausgabewerte werden aktuelles Datum und Uhrzeit in verschiedenen Formaten geliefert, auch diese entsprechen den gleichnamigen Parametern aus der Funktion `Plm_GetRtc()` (siehe Abschnitt 15.3.2).

Der Baustein ruft intern die Funktion `Plm_EnableSysTime()` auf (siehe Abschnitt 15.3.1).

Die Subvisu `PLM_SETCLOCKVISU` stellt sechs Eingabefelder mit Beschriftung zur Verfügung, in denen zunächst die aktuelle Uhrzeit der Echtzeituhr angezeigt wird.

Durch Eingabe einer Zahl in eines der Felder wird diese sofort als neue Uhrzeit der Echtzeituhr übernommen. In das Sekundenfeld ist keine Eingabe möglich, hier bewirkt ein Drücken des Feldes ein Rückstellen der Sekunden auf den Wert Null. Der Wochentag kann nicht eingegeben werden, sondern wird intern aus dem aktuellen Datum berechnet.

Falls die Beschriftungen nicht gewünscht sind, können diese nach dem Einbinden der SubVisu mit weißen Flächen überdeckt werden.

15.5.2. Programmbeispiel zur Bibliothek Plm_TimeVisu.lib (ST)

Das folgende Beispiel verwendet die Bibliothek *Plm_TimeVisu.lib*.

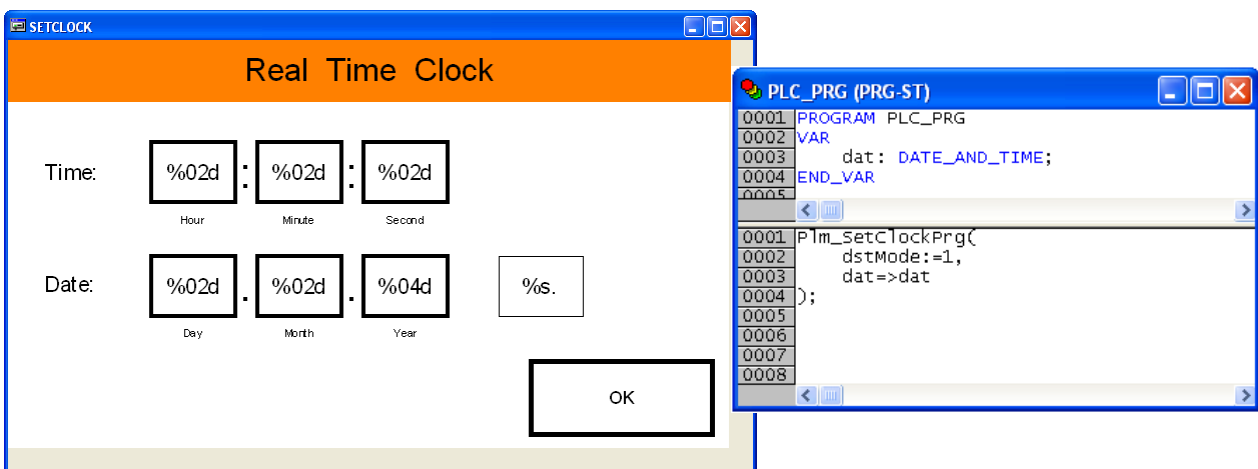


Abb. 15-17: Verwendung der Bibliothek *Plm_TimeVisu.lib* (ST)

Die SubVisu `PLM_SETCLOCKVISU` wurde in eine eigene Visu-Seite eingefügt und mit einer Überschrift und einem OK-Button zum Verlassen der Visu-Seite ergänzt. Der

OK-Button führt lediglich eine Zoom-Funktion zu einer anderen Visu-Seite aus (*Eingabe* → *Zoomen nach Visu*).

Der Baustein `Plm_SetClockPrg()` wird zyklisch aufgerufen und stellt für den Rest des Programms die aktuelle Uhrzeit zur Verfügung. In diesem Beispiel wird nur der Wert `dat` im Format `DATE_AND_TIME` verwendet.

16. M-Bus

16.1. Allgemeines

Der M-Bus (Meter-Bus) wird in der Gebäudeautomation zur Fernabfrage von Verbrauchszählern, wie z.B. Wasser-, Strom- oder Wärmemengenzählern, eingesetzt. Eine Beschreibung erfolgt in der EN13757. Weitere Informationen sind auf der Website <http://www.m-bus.com/> verfügbar.

Die Datenübertragung erfolgt seriell mit niedriger Baudrate auf einer verpolungssicheren Zweidrahtleitung. Dabei übernimmt ein Master die Abfrage von bis zu 250 Slaves, die durch ihre Primäradresse ausgewählt werden. Üblicherweise erfolgt die Abfrage in großen Zeitintervallen, z.B. einmal pro Stunde oder einmal pro Monat.

Die von einem Slave zur Verfügung gestellten Datenwerte hängen vom Gerätehersteller und vom Slave-Typ ab und können bei einigen Slaves zusätzlich konfiguriert werden. Ein Datenwert besteht immer aus dem Zahlenwert, z.B. "8,23" und einer codierten Einheit, z.B. "x 0,1 kWh". Innerhalb einer Abfrage werden üblicherweise mehrere Datenwerte übertragen.

Für das PLM-System steht ein M-Bus-Mastermodul SIM.730.20 zur Verfügung, an welches bis zu 20 Slaves angeschlossen werden können. Bei Bedarf können mehrere Mastermodule gleichzeitig betrieben werden.

Die Programmierung in CoDeSys erfolgt mittels der Bibliothek `Plm_Mbus.lib`, die im folgenden erläutert wird. Darüber hinaus wird auf das Datenblatt zum SIM.730.20 verwiesen.

16.2. Spezifikation

- M-Bus Master, bestehend aus M-Bus-Mastermodul SIM.730.20 und CoDeSys-Bibliothek `Plm_Mbus.lib`
- Beliebige Baudrate, u.a. 300, 2400 und 9600 Baud
- Unterstützt M-Bus-Slaves mit Primäradressierung
- Anschluss von bis zu 20 Standard-Slaves an einem M-Bus-Mastermodul SIM.730.20, mehrere Module können gleichzeitig verwendet werden
- Bibliotheksbausteine zum automatischen Abwickeln der M-Bus-Zugriffe und zur Auswertung der Slave-Antworten
- Vorgefertigte Bausteine für Wärmemengenzähler, Elektrizitätszähler, Wasserzähler, Gaszähler
- Optionale Initialisierungssequenz mit `SND_NKE`
- Auswertung der Antworttypen *Variable Data Structure* und *Fixed Data Structure*
- Debug-Möglichkeiten zur Fehlersuche

16.3. Vorgehensweise

Die Vorgehensweise wird in Abschnitt 16.6 anhand eines Programmierbeispiels illustriert.

Das M-Bus-Mastermodul SIM.730.20 wird wie eine serielle Schnittstellenerweiterung in die Steuerungskonfiguration von CoDeSys eingebunden. Die M-Bus-Baudrate wird in den *Service Data Objects* des Moduls festgelegt.

Beim Start des Programms muss zunächst die Schnittstelle einmalig mit dem Baustein `SIM_730_COM` initialisiert werden (vgl. Abschnitt 2.3). Das dabei gelieferte Handle (`comm_nr`) dient als Kennung für alle Bausteine, die den M-Bus-Master benötigen.

Die Bausteine der Bibliothek `Plm_Mbus.lib` sind in Abb. 16-1 dargestellt.

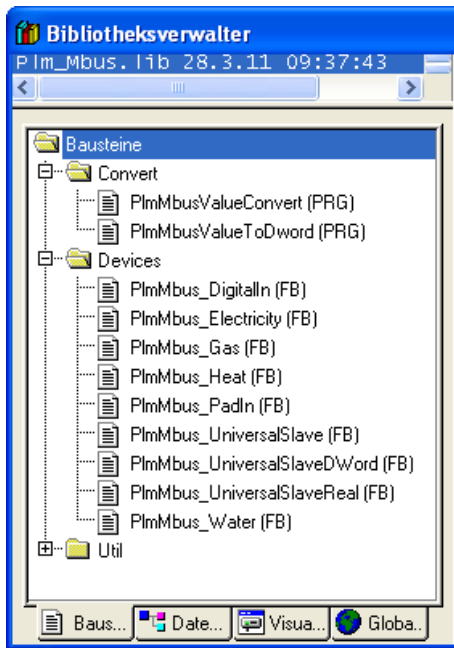


Abb. 16-1: Bausteine der Bibliothek Plm_Mbus.lib

Der Bereich *Devices* enthält Bausteine, die einen M-Bus-Slave repräsentieren:

Baustein	Slave-Typ	Ausgänge
DigitalIn	Digital Input	<ul style="list-style-type: none"> Zustand der Digitaleingänge im Format <code>PlmMbusValue</code>
Electricity	Elektrizitätszähler	<ul style="list-style-type: none"> Energie Leistung im Format <code>PlmMbusValue</code>
Gas	Gaszähler	<ul style="list-style-type: none"> Volumen Durchfluss im Format <code>PlmMbusValue</code>
Heat	Wärmemengenzähler	<ul style="list-style-type: none"> Energie Leistung Volumen Durchfluss Vorlauftemperatur Rücklauftemperatur Differenztemperatur im Format <code>PlmMbusValue</code>
Water	Wasserzähler	<ul style="list-style-type: none"> Volumen Durchfluss im Format <code>PlmMbusValue</code>
UniversalSlave	Universal	<ul style="list-style-type: none"> 30 Ausgangswerte des Slaves im Format <code>PlmMbusValue</code>
UniversalSlaveDWord	Universal	<ul style="list-style-type: none"> 30 Ausgangswerte des Slaves im Format <code>DWORD</code>
UniversalSlaveReal	Universal	<ul style="list-style-type: none"> 30 Ausgangswerte des Slaves im Format <code>REAL</code>

Die tatsächlich gelieferten Werte hängen vom jeweiligen Slave ab.

Für jeden Slave wird ein Funktionsblock aus dem Bereich *Devices* eingefügt. Dieser ruft intern Bausteine aus dem Bereich *Util* auf und wickelt den M-Bus-Zugriff für seinen Slave ab.

Die Slave-Funktionsblöcke einigen sich automatisch untereinander, so dass immer nur ein Baustein den M-Bus belegt und die Bus-Zugriffe sequentiell abgearbeitet werden.

16.4. PlmMbusValue

An den Ausgängen der Slave-Funktionsblöcke stehen nach erfolgreichem M-Bus-Zugriff die vom Slave gelieferten Datenwerte als `PlmMbusValue` zur Verfügung (siehe Abb. 16-2).

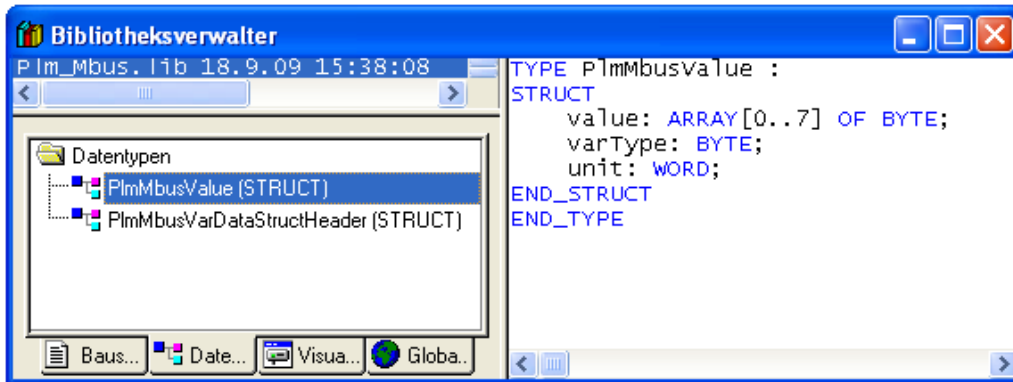


Abb. 16-2: Datentyp `PlmMbusValue`

Ein `PlmMbusValue` entspricht dem Format, in dem die Daten vom M-Bus-Slave geliefert werden und enthält neben dem Datenwert noch Angaben zur Codierung und zur Einheit des Datenwerts.

Ein `PlmMbusValue` muss durch Nachschalten eines Bausteins vom Typ `PlmMbusValueConvert()` in eine REAL-Zahl in der gewünschten Einheit, z.B. kWh oder °C, umgewandelt werden (siehe Abschnitt 16.5.3). Für spezielle Zwecke steht auch ein Baustein `PlmMbusValueToDword()` zur Verfügung.

16.5. Benötigte Bibliotheken

Plm_Mbus.lib
SIM_COM.lib
UPD_E_005.lib (oder spätere Version)
Plm_Std.lib

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

16.5.1. PlmMbus_Electricity, ..._Gas, ..._Heat, ..._Water, ..._DigitalIn (Plm_Mbus.lib)

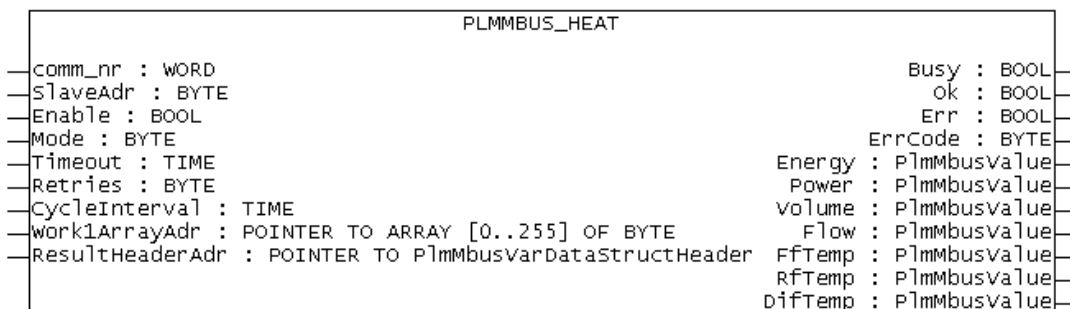


Abb. 16-3: Funktionsblock `PlmMbus_Heat` (`Plm_Mbus.lib`)

Der Funktionsblock `PlmMbus_Heat` dient der Abfrage eines Wärmemengenzählers und wird stellvertretend für die Funktionsblöcke `PlmMbus_Electricity`, `PlmMbus_Gas`, `PlmMbus_Water` und `PlmMbus_DigitalIn` erläutert, die sich lediglich durch die anderen Ausgangswerte unterscheiden.

Input-Parameter:		
<code>comm_nr</code>	WORD	Zugewiesene COM-Nummer des M-Bus-Masters aus <code>SIM_730_COM</code>
<code>SlaveAdr</code>	BYTE	Primäradresse des Slaves, z.B. 2 (siehe unten)
<code>Enable</code>	BOOL	TRUE = Slave abfragen
<code>Mode</code>	BYTE	0 = Einmalige Slaveabfrage bei <code>Enable FALSE</code> ⇒ TRUE auslösen 1 = Zyklische Slaveabfrage gemäß <code>CycleInterval</code> solange <code>Enable = TRUE</code> + 32 = ggf. zusätzliche Antworttelegramme anfordern, erste Abfrage erfolgt mit <code>FCB = 0</code> + 64 = ggf. zusätzliche Antworttelegramme anfordern, erste Abfrage erfolgt mit <code>FCB = 1</code> + 128 = vor der Abfrage <code>SND_NKE</code> senden (Reset Slave-Kommunikation)
<code>Timeout</code>	TIME	Timeout für Slave-Abfrage, z.B. T#5s
<code>Retries</code>	BYTE	Anzahl der Wiederholungen nach Timeout, z.B. 2
<code>CycleInterval</code>	TIME	Zyklisches Abfrageintervall bei <code>Mode = 1</code> , z.B. T#24h
<code>WorklArrayAdr</code>	POINTER TO ARRAY[] OF BYTE	Sollte den Wert 0 haben
<code>ResultHeaderAdr</code>	POINTER TO PlmMbusVar DataStructHeader	Adresse einer Variablen vom Typ <code>PlmMbusVarDataStructHeader</code> . In diese wird bei erfolgreicher Slave-Abfrage der <i>Fixed Data Header</i> des M-Bus-Protokolls eingetragen. Werden die Angaben nicht benötigt, so kann anstelle einer Adresse der Wert 0 angegeben werden.
Output-Parameter:		
<code>Busy</code>	BOOL	TRUE = M-Bus-Übertragung aktiv
<code>Ok</code>	BOOL	TRUE = Message-Übertragung und -Auswertung erfolgreich
<code>Err</code>	BOOL	TRUE = Fehler bei Message-Übertragung oder -Auswertung, siehe <code>ErrCode</code>
<code>ErrCode</code>	BYTE	0 = ok 1 = Slave Timeout 2 = Invalid Frame Boundary 3 = Frame Length Error, Längenangaben im Frame nicht konsistent

		<p>4 = Frame Checksum Error 5 = Rx Buffer overflow 6 = interner Parameterfehler 7 = Frame Inconsistency Error bei Long Frame 11 = Konfigurationsfehler SIM.730.20 20 = Unbekannter Antworttyp, weder Fixed Data Structure noch Variable Data Structure 23 = unbekanntes Datenformat bei Variable Length Data 24 = kein Datenframe empfangen 25 = Fixed Data Header fehlt 27 = Var Length Data zu lang 28 = weitere Datenübertragung erforderlich</p>
<p>Energy Power Volume Flow FfTemp RfTemp DifTemp</p>	<p>PlmMbusValue</p>	<p>Vom Slave gelieferte Werte als Datentyp <code>PlmMbusValue</code>. Zur Umwandlung in eine Zahl mit einer bestimmten Einheit ist der Baustein <code>PlmMbusValueConvert()</code> nachzuschalten.</p>

Als `SlaveAdr` wird die M-Bus-Primäradresse des Slaves angegeben, diese liegt im Bereich 1...250 und wird am Slave üblicherweise durch Codierschalter oder mittels Konfigurations-Software eingestellt.

Die Adressen 254 und 255 werden von den Slaves als Broadcast interpretiert, der an alle Slaves gerichtet ist. Bei `SlaveAdr = 254` antworten alle angeschlossenen Slaves; diese Adresse kann deshalb nur verwendet werden, wenn nur ein einziger Slave am M-Bus angeschlossen ist. Bei `SlaveAdr = 255` antwortet kein Slave, diese Adresse kann daher nicht für die Datenabfrage verwendet werden.

Der Eingang `Mode` sollte bei den meisten M-Bus-Slaves den Wert 192 bzw. 16#C0 (Einzelabfrage + FCB=1 + SND_NKE) oder 193 bzw. 16#C1 (Zyklische Abfrage + FCB=1 + SND_NKE) haben. Diese Einstellung führt bei den meisten Slaves dazu, dass die gewünschten Daten in der Antwort erscheinen. Details sind der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen.

Der Baustein versucht, aus der Slave-Antwort die gewünschten Datenwerte zu extrahieren, z.B. bei `PlmMbus_Heat()` die Werte für Energie, Leistung, Volumen, Durchfluss, Vorlauftemperatur, Rücklauftemperatur und Differenztemperatur. Ob dies möglich ist, hängt vom jeweiligen Slave ab und ist der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen.

Bei erfolgreicher Übertragung werden die vom Slave empfangenen Datenwerte in die entsprechenden Ausgangsvariablen kopiert. Diese haben den Datentyp `PlmMbusValue` und müssen mit Bausteinen vom Typ `PlmMbusValueConvert()` in die gewünschte Einheit umgewandelt werden (siehe Abschnitt 16.5.3).

Nicht in der Slave-Antwort enthaltene Ausgangswerte werden auf 0 gesetzt; ein nachgeschalteter Baustein vom Typ `PlmMbusValueConvert()` zeigt in diesem Fall einen Fehler an (siehe Abschnitt 16.5.3).

Die Eingänge `Work1ArrayAdr` und `ResultHeaderAdr` können in den meisten Fällen auf 0 gesetzt werden.

Der Baustein muss zyklisch aufgerufen werden, auch wenn gerade keine Abfrage läuft, damit die internen Abläufe korrekt bearbeitet werden.

Hinweise zur Fehlersuche finden sich in Abschnitt 16.7.

16.5.2. PlmMbus_UniversalSlave (Plm_Mbus.lib)

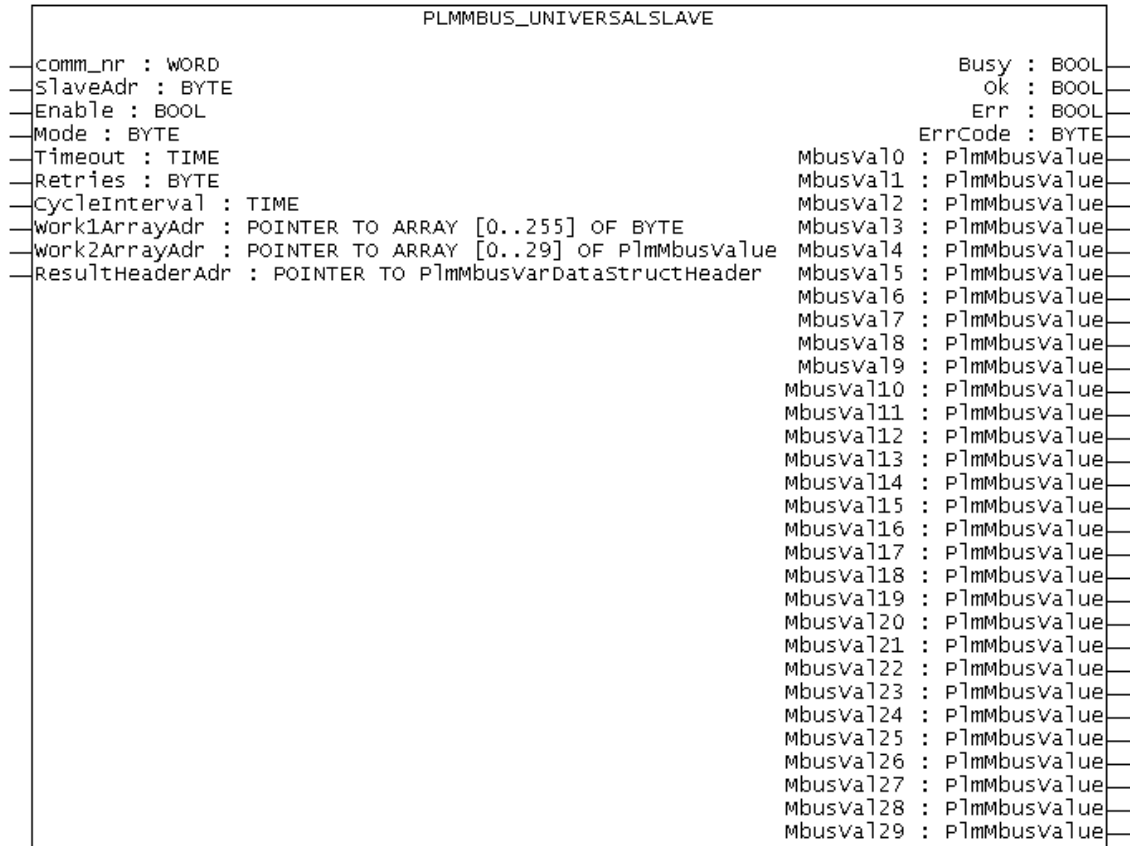


Abb. 16-4: Funktionsblock PlmMbus_UniversalSlave (Plm_Mbus.lib)

Dieser Funktionsblock dient zur Abfrage unbekannter Slaves oder Slaves, die keinem der vorhandenen Standardtypen zuzuordnen sind.

Er wird stellvertretend für die Funktionsblöcke PlmMbus_UniversalSlaveDWord und PlmMbus_UniversalSlaveReal erläutert.

Input-Parameter:		
comm_nr	WORD	Zugewiesene COM-Nummer des M-Bus-Masters aus SIM_730_COM
SlaveAdr	BYTE	Primäradresse des Slaves, z.B. 2 (siehe unten)
Enable	BOOL	TRUE = Slave abfragen
Mode	BYTE	0 = Einmalige Slaveabfrage bei Enable FALSE ⇒ TRUE auslösen 1 = Zyklische Slaveabfrage gemäß CycleInterval solange Enable = TRUE + 32 = ggf. zusätzliche Antworttelegramme anfordern, erste Abfrage erfolgt mit FCB = 0 + 64 = ggf. zusätzliche Antworttelegramme anfordern, erste Abfrage erfolgt mit FCB = 1 + 128 = vor der Abfrage SND_NKE senden (Reset Slave-Kommunikation)
Timeout	TIME	Timeout für Slave-Abfrage, z.B. T#5s
Retries	BYTE	Anzahl der Wiederholungen nach Timeout, z.B. 2

CycleInterval	TIME	Zyklisches Abfrageintervall bei Mode = 1, z.B. T#24h
Work1ArrayAdr	POINTER TO ARRAY[] OF BYTE	Sollte den Wert 0 haben
Work2ArrayAdr	POINTER TO ARRAY[] OF PlmMbusValue	Adresse eines PlmMbusValue-Arrays als temporärer Speicherbereich (siehe unten)
ResultHeaderAdr		Adresse einer Variablen vom Typ PlmMbusVarDataStructHeader. In diese wird bei erfolgreicher Slave-Abfrage der <i>Fixed Data Header</i> des M-Bus-Protokolls eingetragen. Werden die Angaben nicht benötigt, so kann anstelle einer Adresse der Wert 0 angegeben werden.
Output-Parameter:		
Busy	BOOL	TRUE = M-Bus-Übertragung aktiv
Ok	BOOL	TRUE = Message-Übertragung und -Auswertung erfolgreich
Err	BOOL	TRUE = Fehler bei Message-Übertragung oder -Auswertung, siehe ErrCode
ErrCode	BYTE	0 = ok 1 = Slave Timeout 2 = Invalid Frame Boundary 3 = Frame Length Error, Längenangaben im Frame nicht konsistent 4 = Frame Checksum Error 5 = Rx Buffer overflow 6 = interner Parameterfehler 7 = Frame Inconsistency Error bei Long Frame 11 = Konfigurationsfehler SIM.730.20 20 = Unbekannter Antworttyp, weder Fixed Data Structure noch Variable Data Structure 23 = unbekanntes Datenformat bei Variable Length Data 24 = kein Datenframe empfangen 25 = Fixed Data Header fehlt 27 = Var Length Data zu lang 28 = weitere Datenübertragung erforderlich
MbusVal0 ... MbusVal29	PlmMbusValue	Vom Slave gelieferte Werte als Datentyp PlmMbusValue. Zur Umwandlung in eine Zahl mit einer bestimmten Einheit ist der Baustein PlmMbusValueConvert() nachzuschalten.

Als SlaveAdr wird die M-Bus-Primäradresse des Slaves angegeben, diese liegt im Bereich 1...250 und wird am Slave üblicherweise durch Codierschalter oder mittels Konfigurations-Software eingestellt.

Die Adressen 254 und 255 werden von den Slaves als Broadcast interpretiert, der an alle Slaves gerichtet ist. Bei SlaveAdr = 254 antworten alle angeschlossenen Slaves; diese Adresse kann deshalb nur verwendet werden, wenn nur ein einziger

Slave am M-Bus angeschlossen ist. Bei `SlaveAdr = 255` antwortet kein Slave, diese Adresse kann daher nicht für die Datenabfrage verwendet werden.

Der Eingang `Mode` sollte bei den meisten M-Bus-Slaves den Wert 192 (Einzelabfrage + `FCB=1` + `SND_NKE`) oder 193 (Zyklische Abfrage + `FCB=1` + `SND_NKE`) haben. Diese Einstellung führt bei den meisten Slaves dazu, dass die gewünschten Daten in der Antwort erscheinen. Details sind der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen.

Der Baustein versucht, aus der Slave-Antwort die ersten 30 Datenwerte zu extrahieren. Ob dies überhaupt möglich ist, hängt vom jeweiligen Slave ab und ist der M-Bus-Dokumentation des jeweiligen Slaves zu entnehmen. Nicht in der Slave-Antwort enthaltene Ausgangswerte werden auf 0 gesetzt.

An den Eingang `Work2ArrayAdr` muss die Adresse eines Arrays zur Übergabe der aus der Message extrahierten Daten angelegt werden. Das Array enthält nach einem erfolgreichen Zugriff die Antwort-Werte des Slaves vom Typ `PlmMbusValue`. Das Array muss ausreichend groß dimensioniert sein, Deklaration z.B.

```
VAR
    work2array: ARRAY[0..30] OF PlmMbusValue;
END_VAR
```

Bei erfolgreicher Übertragung werden die vom Slave empfangenen Datenwerte in die Ausgangsvariablen `MbusVal0...MbusVal29` kopiert. Diese haben beim Baustein `PlmMbus_UniversalSlave` den Datentyp `PlmMbusValue` und müssen mit Bausteinen vom Typ `PlmMbusValueConvert()` in die gewünschte Einheit umgewandelt werden (siehe Abschnitt 16.5.3).

Beim Baustein `PlmMbus_UniversalSlaveDWord()` erfolgt bereits intern eine Umwandlung in den Zahlentyp `DWORD`, ohne Berücksichtigung einer Einheit.

Beim Baustein `PlmMbus_UniversalSlaveReal()` erfolgt bereits intern eine Umwandlung in den Zahlentyp `REAL`, ohne Berücksichtigung einer Einheit.

Der Baustein muss zyklisch aufgerufen werden, auch wenn gerade keine Abfrage läuft, damit die internen Abläufe korrekt bearbeitet werden.

Hinweise zur Fehlersuche finden sich in Abschnitt 16.7.

16.5.3. PlmMbusValueConvert (Plm_Mbus.lib)

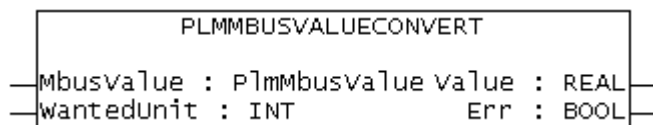


Abb. 16-5: Baustein `PlmMbusValueConvert` (`Plm_Mbus.lib`)

Input-Parameter:		
MbusValue	PlmMbusValue	Ausgabewert eines Slave-Funktionsblocks
WantedUnit	INT	Gewünschte physikalische Einheit, in der der Wert ausgegeben werden soll (siehe unten)
Output-Parameter:		
Value	REAL	Zahlenwert im Format <code>REAL</code> in der durch <code>WantedUnit</code> vorgegebenen physikalischen Einheit
Err	BOOL	TRUE = Umwandlung kann nicht durchgeführt werden, die globale Variable <code>PlmMbusValueConvertError</code> enthält einen Fehlercode: 0 = kein Fehler 50 = angeforderter Wert ist nicht in Slave-Antwort enthalten

		51 = fehlerhaft BCD-codierter Wert 52 = Umwandlung in gewünschte Einheit (WantedUnit) nicht möglich 53 = unbekanntes Codierungsformat (MbusValue.varType)
--	--	---

Der Baustein `PlmMbusValueConvert()` wandelt Datenwerte aus dem Format `PlmMbusValue` in einen Zahlenwert in der gewünschten Einheit.

Der Baustein erlaubt nur sinnvolle Umwandlungen, z.B. kann ein Datenwert, der in der Einheit "Liter/Minute" geliefert wird, in "Kubikmeter/Stunde" umgewandelt werden aber nicht in "Kilowatt".

Im Fehlerfall wird der Ausgang `Value` auf den Wert 0 und der Ausgang `Err` auf TRUE gesetzt. Zusätzlich enthält die globale Bibliotheksvariable `PlmMbusValueConvertError` einen Fehlercode.

Am Eingang `WantedUnit` wird eine konstante Zahl angelegt, mit der die gewünschte Ausgabeeinheit von `Value` festgelegt wird. Anstelle einer Zahl kann auch eine entsprechende globale Konstante aus der Bibliothek `Plm_Mbus.lib` verwendet werden; dies führt zu einer übersichtlicheren Darstellung (vgl. das Beispiel in Abschnitt 16.6). Die verfügbaren globalen Konstanten sind aus der Bibliothek ersichtlich.

Folgende Ausgabeeinheiten `WantedUnit` sind möglich:

Konstanter Faktor:
0 ⇒ x1, 1 ⇒ x10, 2 ⇒ x100, 3 ⇒ x1000, 4 ⇒ x1e4, 5 ⇒ x1e5, 6 ⇒ x1e6, 7 ⇒ x1e7, 8 ⇒ x1e8, 9 ⇒ x1e9 -1 ⇒ ÷10, -2 ⇒ ÷100, -3 ⇒ ÷1000, -4 ⇒ ÷1e4, -5 ⇒ ÷1e5, -6 ⇒ ÷1e6, -7 ⇒ ÷1e7, -8 ⇒ ÷1e8, -9 ⇒ ÷1e9
Energie:
20 ⇒ mJ, 21 ⇒ J, 22 ⇒ kJ, 23 ⇒ MJ, 24 ⇒ GJ, 25 ⇒ mWxs, 26 ⇒ mWxmin, 27 ⇒ mWxhour, 28 ⇒ Wxs, 29 ⇒ Wxmin, 30 ⇒ Wxhour, 31 ⇒ kWxs, 32 ⇒ kWxmin, 33 ⇒ kWxhour, 34 ⇒ MWxs, 35 ⇒ MWxmin, 36 ⇒ MWxhour, 37 ⇒ GWxs, 38 ⇒ GWxmin, 39 ⇒ GWxhour
Leistung:
50 ⇒ mW, 51 ⇒ W, 52 ⇒ kW, 53 ⇒ MW, 54 ⇒ GW, 55 ⇒ mJ/s, 56 ⇒ mJ/min, 57 ⇒ mJ/hour, 58 ⇒ J/s, 59 ⇒ J/min, 60 ⇒ J/hour, 61 ⇒ kJ/s, 62 ⇒ kJ/min, 63 ⇒ kJ/hour, 64 ⇒ MJ/s, 65 ⇒ MJ/min, 66 ⇒ MJ/hour, 67 ⇒ GJ/s, 68 ⇒ GJ/min, 69 ⇒ GJ/hour
Volumen:
80 ⇒ ml, 81 ⇒ l, 82 ⇒ m ³
Volumenfluss:
90 ⇒ ml/s, 91 ⇒ ml/min, 92 ⇒ ml/hour, 93 ⇒ l/s, 94 ⇒ l/min, 95 ⇒ l/hour, 96 ⇒ m ³ /s, 97 ⇒ m ³ /min, 98 ⇒ m ³ /hour
Masse:
110 ⇒ mg, 111 ⇒ g, 112 ⇒ kg, 113 ⇒ ton
Massenfluss:
120 ⇒ mg/s, 121 ⇒ mg/min, 122 ⇒ mg/hour, 123 ⇒ g/s, 124 ⇒ g/min, 125 ⇒ g/hour, 126 ⇒ kg/s, 127 ⇒ kg/min, 128 ⇒ kg/hour,

129 ⇨ ton/s, 130 ⇨ ton/min, 131 ⇨ ton/hour
Temperatur:
140 ⇨ °C, 141 ⇨ °F,
Temperaturdifferenz:
145 ⇨ mK, 146 ⇨ K
Druck:
150 ⇨ mbar, 151 ⇨ bar, 152 ⇨ kbar, 153 ⇨ mPa, 154 ⇨ Pa, 155 ⇨ hPa, 156 ⇨ kPa, 157 ⇨ psi, 158 ⇨ atm
Zeitintervall:
170 ⇨ s, 171 ⇨ min, 172 ⇨ hours, 173 ⇨ days

16.5.4. PlmMbusValueToDWord (Plm_Mbus.lib)

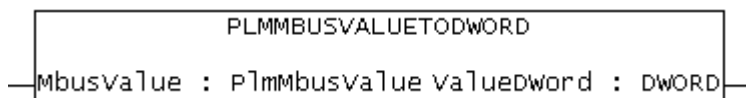


Abb. 16-6: Baustein PlmMbusValueToDWord (Plm_Mbus.lib)

Input-Parameter:		
MbusValue	PlmMbusValue	Ausgabewert eines Slave-Funktionsblocks
Output-Parameter:		
ValueDWord	DWORD	Zahlenwert im Format DWORD ohne weitere Umwandlung

Dieser Baustein wandelt den am Eingang angelegten PlmMbusValue direkt in einen Wert vom Typ DWORD, ohne Berücksichtigung einer Einheit.

Der Baustein kann zu Debugging-Zwecken verwendet werden, oder um große Integer-Werte, wie z.B. Seriennummern, zu konvertieren.

Falls die Umwandlung nicht möglich ist, hat ValueDWord den Wert 0; eine weitergehende Fehlerdiagnose ist mit diesem Baustein nicht möglich.

16.6. Programmbeispiel (FUP)

Das folgende Beispiel demonstriert die Verwendung des M-Bus-Masters in einem IEC-Programm. Das Beispiel wurde in FUP ausgeführt, kann aber ebenso in CFC oder ST implementiert werden.

Die Beispiel-Bausteine InitOnce() und TestHeat() müssen zyklisch aus PLC_PRG() heraus aufgerufen werden.

Zunächst wird das M-Bus-Mastermodul SIM.730.20 in die Steuerungskonfiguration eingetragen (siehe Abb. 16-7).

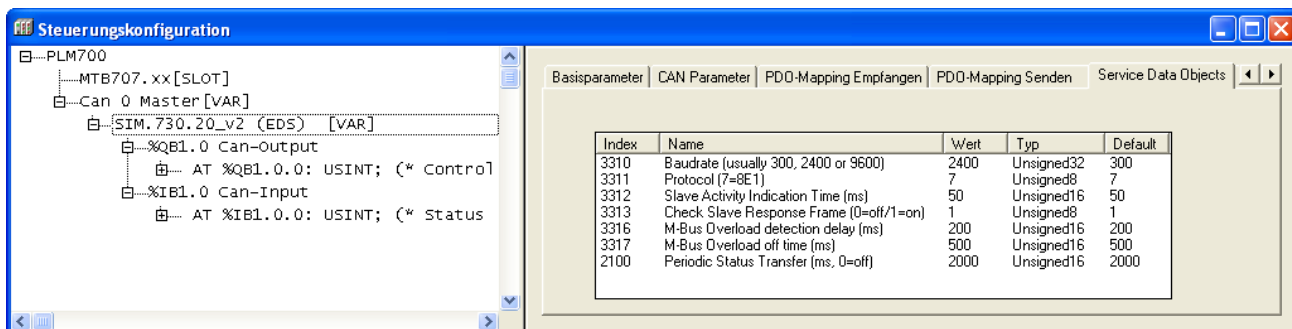


Abb. 16-7: Steuerungskonfiguration mit M-Bus-Master SIM.730.20

Der Baustein `InitOnce()` initialisiert die Verbindung zum SIM.730.20 (siehe Abb. 16-8).

`SIM_730_COM()` erhält am Eingang `NodeID` die CAN-ID des M-Bus-Mastermoduls, die in der Steuerungskonfiguration unter *CAN Parameter* eingestellt ist (hier: 2). Am Eingang `FirstQbAddress` wird die erste QB-Adresse des Moduls aus der Steuerungskonfiguration angelegt (hier: `%QB1.0.0`, vgl. Abb. 16-7).

Die Baudrate auf dem M-Bus wird unter *Service Data Objects* eingestellt. Die üblichen Baudraten sind 300, 2400 und 9600 Baud. Viele M-Bus-Slaves erkennen die Baudrate automatisch. Im Zweifelsfall ist die M-Bus-Dokumentation des entsprechenden Slaves heranzuziehen.

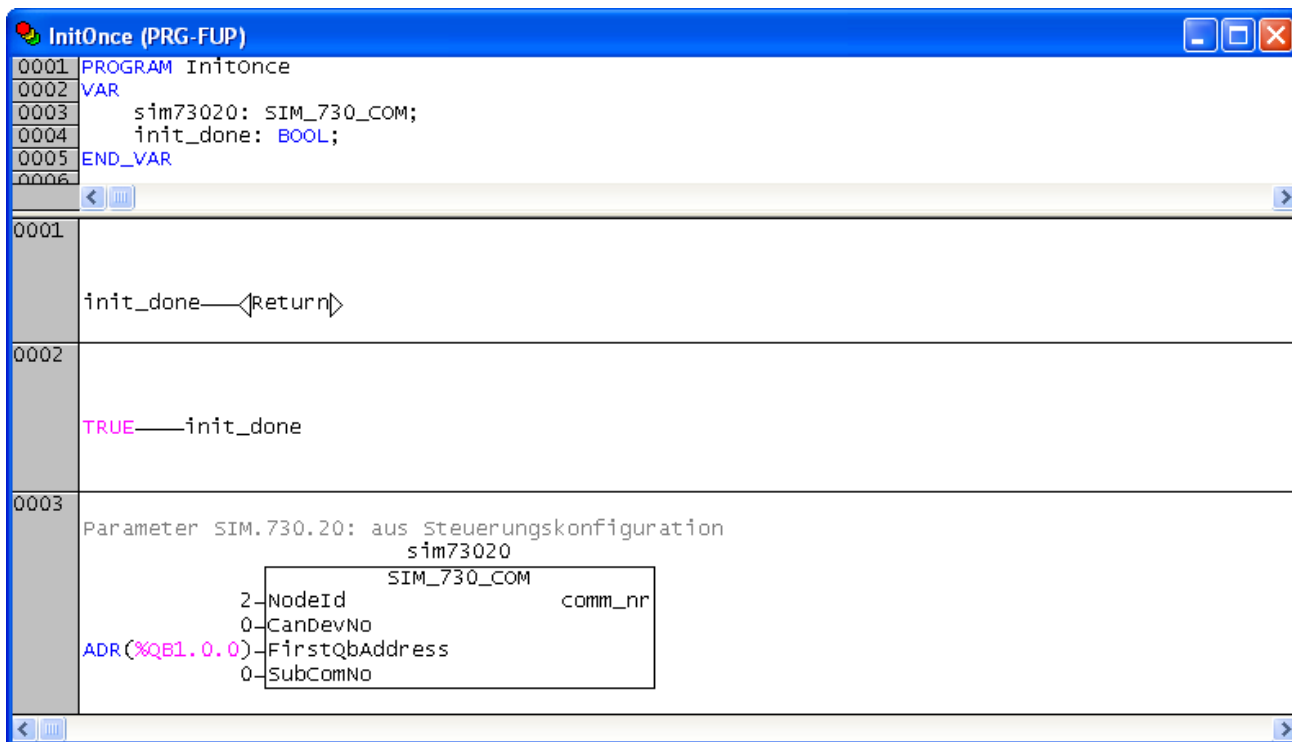


Abb. 16-8: Einmalige Initialisierung der Kommunikation mit dem M-Bus-Mastermodul SIM.730.20 (FUP)

Der Baustein `TestHeat()` realisiert die zyklische Slave-Abfrage (siehe Abb. 16-9).

Der Eingang `comm_nr` am Baustein `PlmMbus_Heat()` wird im Beispiel direkt aus `InitOnce()` entnommen (siehe Abb. 16-8). Alternativ könnte `comm_nr` in eine globale Variable gespeichert werden.

Im Beispiel wird ein M-Bus-Wärmemengenzähler mit der Primäradresse 2 abgefragt. Falls bei der M-Bus-Abfrage ein Fehler auftritt, ist `transfErr = TRUE`, andernfalls ist `transfOk = TRUE`.

Die Ausgangswerte vom Typ `PlmMbusValue` werden durch die Bausteine `PlmMbusConvert` in Zahlenwerte in der gewünschten Einheit gewandelt. Falls bei der Umwandlung Fehler auftreten, sind diese in `err[]` vermerkt.

Die Ergebnisse in den Variablen `energy`, `power`, `volume`, `flow`, `ffTemp`, `rfTemp` und `difTemp` sind vom Typ `REAL`.

Die Abfrage des Slaves erfolgt im Beispiel zyklisch alle 10 Sekunden; in der Praxis sind Abfragen meistens eher einmal pro Tag o.ä. vorzunehmen.

The screenshot shows the TestHeat (PRG-FUP) software interface. The top part displays the program code, and the bottom part shows a hardware connection diagram.

```

0001 PROGRAM TestHeat
0002 VAR
0003   heat: PlmMbus_Heat;
0004   err: ARRAY[0..6] OF BOOL;
0005   transfBusy, transfok, transfErr: BOOL;
0006   transfErrCode: BYTE;
0007   energy, power, volume, flow, ffTemp, rfTemp, difTemp: REAL;
0008 END_VAR
0009

```

The hardware connection diagram shows the following connections:

- 0001:** A block named `heat` (PlmMbus_Heat) is connected to the `heat` variable. Its inputs are `comm_nr` (2), `SlaveAdr` (2), `Enable` (TRUE), `Mode` (16#C1), `Timeout` (t#1s), `Retries` (2), `CycleInterval` (t#10s), `work1ArrayAdr` (0), and `ResultHeaderAdr` (0). Its outputs are `Busy` (transfBusy), `ok` (transfok), `Err` (transfErr), `ErrCode` (transfErrCode), `Energy` (energy), `Power` (power), `Volume` (volume), `Flow` (flow), `FfTemp` (ffTemp), `RfTemp` (rfTemp), and `DifTemp` (difTemp).
- 0002:** A block named `PlmMbusValueConvert` is connected to `heat.Energy`. Its inputs are `MbusValue` (heat.Energy) and `wantedUnit` (PLMMBUS_CONVERT_kwh). Its outputs are `value` (energy) and `Err` (err [0]).
- 0003:** A block named `PlmMbusValueConvert` is connected to `heat.Power`. Its inputs are `MbusValue` (heat.Power) and `wantedUnit` (PLMMBUS_CONVERT_kw). Its outputs are `value` (power) and `Err` (err [1]).
- 0004:** A block named `PlmMbusValueConvert` is connected to `heat.Volume`. Its inputs are `MbusValue` (heat.Volume) and `wantedUnit` (PLMMBUS_CONVERT_m3). Its outputs are `value` (volume) and `Err` (err [2]).
- 0005:** A block named `PlmMbusValueConvert` is connected to `heat.Flow`. Its inputs are `MbusValue` (heat.Flow) and `wantedUnit` (PLMMBUS_CONVERT_lph). Its outputs are `value` (flow) and `Err` (err [3]).
- 0006:** A block named `PlmMbusValueConvert` is connected to `heat.FfTemp`. Its inputs are `MbusValue` (heat.FfTemp) and `wantedUnit` (PLMMBUS_CONVERT_DegC). Its outputs are `value` (ffTemp) and `Err` (err [4]).
- 0007:** A block named `PlmMbusValueConvert` is connected to `heat.RfTemp`. Its inputs are `MbusValue` (heat.RfTemp) and `wantedUnit` (PLMMBUS_CONVERT_DegC). Its outputs are `value` (rfTemp) and `Err` (err [5]).
- 0008:** A block named `PlmMbusValueConvert` is connected to `heat.DifTemp`. Its inputs are `MbusValue` (heat.DifTemp) and `wantedUnit` (PLMMBUS_CONVERT_DegC). Its outputs are `value` (difTemp) and `Err` (err [6]).

Abb. 16-9: Beispiel für Abfrage eines M-Bus-Wärmemengenzählers (FUP)

16.7. Fehlersuche

Im Fehlerfall, d.h. es erscheinen keine Werte an den Ausgängen der Slave-Bausteine, ist zunächst der Ausgang `ErrCode` zu betrachten, der häufig Aufschluss über die Art des Fehlers gibt. Die Fehlercodes sind in der Bibliothek vermerkt.

- Falls es sich um Timeout-Fehler handelt, ist zu prüfen, ob überhaupt eine Kommunikation mit dem betreffenden Slave zustande kommt.
- Die Antwort eines Slaves ist beim SIM.730.20 am kurzzeitigen Blinken der Status-LED *Slave-Aktivität* erkennbar.
- Eine testweise Kommunikation kann direkt am SIM.730.20 ausgelöst werden, indem der Fronttaster am Modul vier Sekunden lang gedrückt wird. Dadurch wird eine Datenabfrage an alle M-Bus-Slaves ausgelöst (an Broadcast-Adresse 254). Die Status-LED *Slave-Aktivität* des Moduls muss blinken als Zeichen einer Slave-Antwort. Falls die LED nicht blinkt: M-Bus falsch angeschlossen, falsche Baudrate, Slave falsch konfiguriert (z.B. antwortet nicht auf Primäradressierung), Slave zu häufig abgefragt (s.u.).
- Prüfen Sie, ob die Baudrate korrekt eingestellt ist. Falls ein Slave keine automatische Baudratenerkennung besitzt, muss diese im Slave passend konfiguriert oder beim M-Bus-Master (in der Steuerungskonfiguration) passend eingestellt werden.
- Bei einigen Slaves ist nur eine begrenzte Anzahl von Abfragen pro Zeitintervall möglich (z.B. max. 10 Abfragen pro Tag), vornehmlich bei Slaves mit Batteriespeisung. Dies kann dazu führen, dass die Abfrage zunächst funktioniert, später aber scheinbar nicht mehr. Abhilfe: Abfrageintervall ausreichend groß wählen, siehe Slave-Dokumentation.
- Falls die Datenübertragung mit dem Slave-Baustein funktioniert (`Ok = TRUE`), aber die Wertenumwandlung mit dem Baustein `PlmMbusValueConvert()` einen Fehler meldet (`Err = TRUE`), kann der Wert nicht in der gewünschten Weise aus den übertragenen Daten extrahiert werden. Hinweise auf die Ursache liefert in diesem Fall die globale Bibliotheksvariable `PlmMbusValueConvertError`; diese muss direkt nach dem Aufruf des betreffenden Bausteins `PlmMbusValueConvert()` ausgewertet werden. Die möglichen Fehlercodes sind in Abschnitt 16.5.3 aufgelistet und als globale Konstanten in der Bibliothek definiert.
- Debug-Logfile einschalten: globale Bibliotheksvariable `PlmMbusLogEnable` in `PLC_PRG` auf `TRUE` setzen, dies erzeugt ein Logfile `a/mbus_debug.txt`. Diese Datei kann mit einem FTP-Client von der Steuerung geladen und mit einem Texteditor untersucht werden. Senden Sie uns die Datei ggf. nach Rücksprache zu Diagnosezwecken zu. Unbedingt `PlmMbusLogEnable` im späteren Betrieb wieder deaktivieren, da das Logfile sonst unbegrenzt wächst.

17. KNX/EIB

17.1. Allgemeines

KNX ist ein Standard zur Vernetzung von Elektroinstallationskomponenten in der Gebäudeautomation. Entsprechende Komponenten werden von zahlreichen großen Herstellern angeboten. Das Protokoll ist der Nachfolger von EIB und zu diesem vollständig kompatibel. KNX wird zentral von der KNX Association CVBA in Belgien verwaltet; weitere Informationen unter <https://www.knx.org/>.

KNX kennt mehrere physikalische Realisierungen, nämlich KNX TP (Twisted Pair), KNX PL (Powerline), KNX RF (Funk) und KNX IP (Ethernet). Davon ist KNX TP die am häufigsten verwendete. Das Gateway-Modul SIM.730.27 der Fa. SABO Elektronik GmbH ist für den Anschluss an KNX TP geeignet.

Alle KNX-Geräte, wie Lichtschalter oder Relais' zum Schalten der Raumbeleuchtung, besitzen einen Prozessor und sind über den KNX-Bus miteinander vernetzt. Durch Konfiguration wird jedem Gerät eine eindeutige sog. physikalische Adresse zugewiesen; diese wird allerdings meist nur zur Anlagenkonfiguration benötigt und spielt im späteren Betrieb keine Rolle.

Die Funktionen jedes Geräts werden sog. Gruppenadressen zugewiesen, diese sind das Ziel von Schalt- oder Steuerbefehlen. Anhand der Gruppenadressen erfolgt die Verknüpfung zwischen Schaltern und Aktoren.

Die Konfiguration des Systems erfolgt über das von der KNX Association vertriebene PC-Programm ETS (Engineering Tool Software). Der Einsatz des Moduls SIM.730.27 setzt eine mit ETS konfigurierte Anlage voraus. Das Modul hat auf dem KNX-Bus eine physikalische Adresse, braucht jedoch nicht in die Anlagenkonfiguration aufgenommen zu werden.

Die Programmierung in CoDeSys erfolgt mittels der Bibliothek `KNX_SIM73027.lib`. Mithilfe der Bibliothek und des Moduls kann eine PLM-Steuerung die auf dem KNX-Bus gesendeten Telegramme und damit den Anlagenzustand verfolgen oder selbst Telegramme senden und damit Aktionen in der KNX-Anlage auslösen.

17.2. Spezifikation

- Gateway zu KNX TP (Twisted Pair), bestehend aus Busanschaltungsmodul SIM.730.27 und CoDeSys-Bibliothek `KNX_SIM73027.lib`
- Mitlesen der Gruppenadressen (Befehle) auf dem KNX-Bus
- Senden von Gruppenadressen auf den KNX-Bus
- Keine Beschränkung der Objektanzahl
- Voraussetzung: Mit ETS konfigurierte KNX-Anlage
- Keine Integration des Gateways in ETS erforderlich

17.3. KNX-Adressen und -Telegramme

17.3.1. Physikalische Adressen

Jedes Gerät am KNX-Bus benötigt eine sog. physikalische Adresse. Die physikalischen Adressen von KNX TP bilden die Hierarchie der Anlage ab und müssen nach bestimmten Regeln durch ETS vergeben werden; hierauf kann an dieser Stelle nicht weiter eingegangen werden.

Die physikalische Adresse dient zur eindeutigen Identifizierung eines KNX-Geräts. Sie steht als Absenderadresse in jedem KNX-Telegramm, spielt aber für den Betrieb der Anlage keine Rolle.

Physikalische Adressen haben die Form 'x.y.z' und liegen im Bereich '0.0.0' bis '15.15.255'.

Das Gateway-Modul SIM.730.27 hat die voreingestellte physikalische Adresse '15.15.255', diese kann jedoch bei Bedarf mittels der Bibliothek auf einen beliebigen Wert geändert werden.

17.3.2. Gruppenadressen

Schalt- und Steuerbefehle auf dem KNX-Bus werden normalerweise an sog. Gruppenadressen gesendet. Die entsprechend konfigurierten Busteilnehmer reagieren darauf durch Ausführen der gewünschten Aktion.

Gruppenadressen können (bei identischer Bedeutung) auf drei Arten dargestellt werden:

- dreistufig (Format 'a/b/c', Bereich '0/0/0' bis '15/7/255')
- zweistufig (Format 'a/d', Bereich '0/0' bis '15/2047')
- frei (Format 'nnn', Bereich '0' bis '32767')

Die Bausteineingänge der Bibliothek `KNX_SIM73027.lib` können alle drei Formate verarbeiten.

Die Bibliothek `KNX_SIM73027.lib` ermöglicht das "Mithören" am KNX-Bus, wobei die auf dem Bus gesendeten Gruppenadressen aufgezeichnet und gespeichert werden können. Das in diesem Fall gewünschte Format der Gruppenadressen kann über die globale Bibliotheksvariable `PlmKnx_GroupAdrFormat` eingestellt werden; möglich sind die Werte 3 (dreistufig, Voreinstellung), 2 (zweistufig) und 1 (frei).

17.3.3. Telegramminhalt

Ein KNX-Telegramm enthält im Wesentlichen die

- physikalische Adresse des Absenders, die
- Zieladresse (entweder physikalische Adresse eines einzelnen Empfängers oder Gruppenadresse für eine Gruppe von Empfängern) und
- 1...16 Datenbytes.

Die Klassifizierung der KNX-Knoten und die Codierung der Datenbytes ist im KNX-Standard festgelegt (Dokument "KNX System Specifications – Interworking – Datapoint Types").

Die bei den meisten Schalt- oder Steuervorgängen verwendeten Variablentypen benötigen nur 1...4 Datenbytes. Beispielsweise sendet ein einfacher KNX-Schalter (Datapoint Type B₁) nur ein Datenbyte, von dem nur das erste Bit für die Ein/Aus-Information verwendet wird.

17.3.4. Übertragung

Die Übertragung eines Telegramms auf KNX-TP erfolgt mit 9600 Baud.

Der Empfänger mit der passenden Zieladresse muss den korrekten Empfang des Telegramms bestätigen (ACK), ansonsten wird es automatisch bis zu dreimal wiederholt. Diese Funktionen werden automatisch vom Gateway-Modul `SIM.730.27` ausgeführt.

Das Senden zusammen mit dem Abwarten der Empfangsbestätigung belegt den Bus für ca. 20 ms.

Die `Value`-Ausgänge der Bausteine aus der Bibliothek `KNX_SIM73027.lib` werden beim Senden (Write) erst aktualisiert, wenn das Telegramm durch den Empfänger mit ACK bestätigt wurde.

17.4. Bibliothek `KNX_SIM73027.lib`

Die Bausteine der Bibliothek `KNX_SIM73027.lib` sind in der folgenden Abbildung dargestellt:

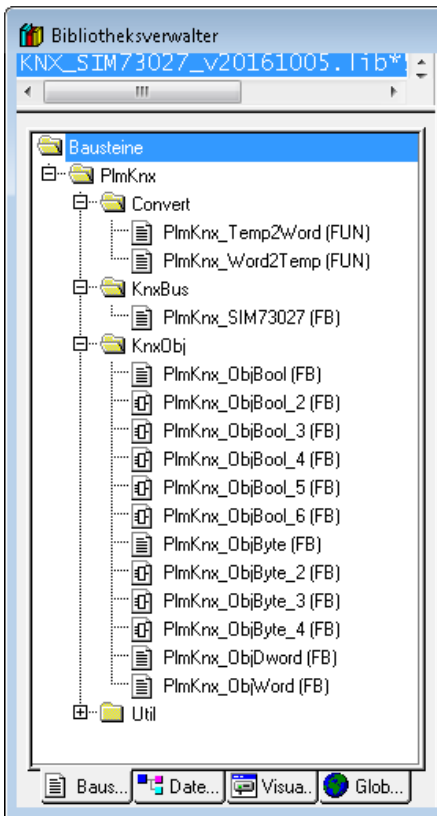


Abb. 17-1: Bausteine der Bibliothek KNX_SIM73027.lib

Der Baustein `PlmKnx_SIM73027` stellt die Verbindung zum KNX-Gatewaymodul SIM.730.27 her. Er muss zyklisch aufgerufen werden und liefert ein Handle, das zum Anhängen eines oder mehrerer KNX-Objekte dient.

Der Bereich `KnxObj` enthält Bausteine, die an `PlmKnx_SIM73027` angehängt werden und verschiedene KNX-Objekte der Anlage repräsentieren.

Wegen der Vielfalt an spezifizierten KNX Datapoint Types (der aktuelle Standard definiert ca. 350 verschiedene) orientieren sich die Bausteine im Bereich `KnxObj` nur an den zugehörigen Datenlängen. Je nach Codierung der Datenwerte sind ggf. Konverterfunktionen aus dem Bereich `Convert` vor- oder nachzuschalten.

17.4.1. PLMKNX_SIM73027 (KNX_SIM73027.lib)

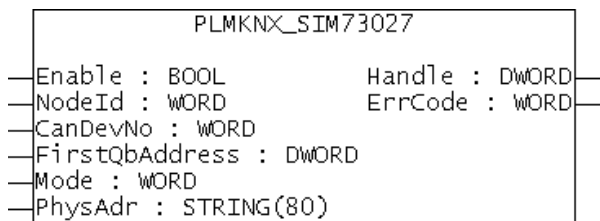


Abb. 17-2: Funktionsblock `PlmKnx_SIM73027`

Input-Parameter:		
Enable	BOOL	bei TRUE wird eine Verbindung zu einem Modul SIM.730.27 aufgebaut und zunächst das Modul sowie sämtliche angehängten Objekt-Bausteine initialisiert
NodeId	WORD	CAN-NodeID des SIM.730.27 aus der Steuerungskonfiguration
CanDevNo	WORD	Index des CAN-Masters, 0 oder 1
FirstQbAddress	DWORD	Adresse <code>ADR()</code> des ersten <code>%QB</code> des

		SIM.730.27 aus der Steuerungskonfiguration
Mode	WORD	Summe von verschiedenen Steuerbits: +1 = beim Initialisieren (nach Enable) für alle angehängten Objektbausteine den aktuellen Zustand vom KNX-Bus anfordern +2 = bei Werteänderung am Eingang eines Objektbausteins automatisch Senden auslösen +4 = FALSE ⇨ TRUE: einmaliges Anfordern des Zustands aller angehängten Objekte vom KNX-Bus +8 = FALSE ⇨ TRUE: einmaliges Senden des aktuellen Zustands aller angehängten Objekte auf den KNX-Bus +16 = Mitprotokollieren aller Ereignisse auf dem KNX-Bus und Ablegen im Globalen Array <code>PlmKnx_LogRxArray[]</code> +32 = Mitprotokollieren aller Ereignisse auf dem KNX-Bus und Speichern in eine Datei, Dateiname aus der Globalen Variablen <code>PlmKnx_LogFileName</code>
PhysAdr	STRING	Physikalische Adresse des SIM-Moduls auf dem KNX-Bus. Wenn keine Adresse angegeben ist (String leer oder nicht angegeben), wird die Adresse '15.15.255' verwendet
Output-Parameter:		
Handle	DWORD	Referenz (Handle) für das Anhängen von Objektbausteinen
ErrCode	WORD	0 = kein Fehler 100 = Ausgangswert <code>Value</code> eines Objektbausteins ungültig (noch nicht initialisiert) 200 = Modul-Initialisierung noch nicht abgeschlossen 201 = CAN-Status des Moduls ist nicht 5, d.h. NodeGuarding-Fehler oder CAN-Bus noch nicht gestartet 202 = Die am Eingang <code>PhysAdr</code> anliegende Adresse kann nicht vom SIM-Modul realisiert werden

Der Baustein `PlmKnx_SIM73027` muss zyklisch aufgerufen werden.

An den Eingang `Mode` wird häufig der feste Wert 3 (= +1 +2) angelegt:

Der Wert +1 bewirkt, dass beim Starten des IEC-Programms zunächst für alle angehängten Objektbausteine der aktuelle Zustand (`Value`) vom KNX-Bus angefordert wird. Dazu wird eine Leseanfrage an die entsprechende Gruppenadresse geschickt. Diese wird von allen entsprechend konfigurierten KNX-Knoten beantwortet. Die empfangenen Werte werden als Vorbesetzung für den Objektbaustein verwendet und damit die Anlagenkonsistenz hergestellt. Es kann dabei vorkommen, dass scheinbar ein falscher Wert als Vorbesetzung gelesen wird. Dies deutet jedoch entweder auf eine falsche ETS-Konfiguration hin (die entsprechenden KNX-Objekte in der Anlage lassen sich nicht auslesen) oder die Anlage befindet sich in einem inkonsistenten Zustand (die angesprochenen KNX-Objekte in der Anlage antworten trotz gleicher Gruppenadresse mit verschiedenen Zuständen).

Der Wert +2 in `Mode` bewirkt, dass jede Änderung am Eingang `WriteValue` eines Objektbausteins automatisch zum Senden des neuen Werts auf den KNX-Bus führt. Dies erleichtert die IEC-Programmierung.

Zur einfachen Erkundung bestehender KNX-Anlagen und zur Fehlersuche kann der Wert +16 zu Mode addiert werden. In diesem Fall werden alle auf dem KNX-Bus erkannten Anlagentelegramme in eine globale Bibliotheksvariable vom Typ

```
PlmKnx_LogRxArray: ARRAY[] OF PLMKNX_VAL_STR
```

protokolliert. Die Länge des Arrays wird durch die globale Konstante PLMKNX_LEN_LOGRXARRAY voreingestellt. Diese hat den Wert 20, kann aber durch Neudeklaration auf einen beliebigen Wert geändert werden.

Der Strukturtyp PLMKNX_VAL_STR ist wie folgt aufgebaut:

```
TYPE PLMKNX_VAL :
STRUCT
  GroupAdr: STRING(20); (* group address *)
  PhysAdr: STRING(20); (* physical address of sender *)
  val: DWORD; (* value *)
END_STRUCT
END_TYPE
```

Das Format der GroupAdr kann mit der globalen Bibliotheksvariablen PlmKnx_GroupAdrFormat eingestellt werden (siehe Abschnitt 17.3.2).

Das Array wird bei Index 0 mit neuen Telegrammen befüllt. Alte Telegramme werden nach hinten geschoben und fallen schließlich hinten heraus. Damit stellt das Array eine Tabelle mit den letzten 20 Telegrammen dar.

17.4.2. PLMKNX_OBJBOOL (KNX_SIM73027.lib)

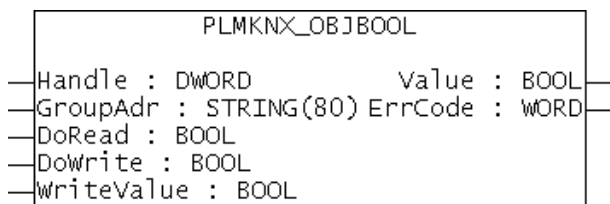


Abb. 17-3: Funktionsblock PlmKnx_ObjBool

Input-Parameter:		
Handle	DWORD	Dieser Eingang ist mit dem entsprechenden Ausgang des Bausteins PlmKnx_SIM73027 zu verbinden
GroupAdr	STRING	Gruppenadresse dieses Objektbausteins, Format: siehe Abschnitt 17.3.2
DoRead	BOOL	FALSE ⇒ TRUE löst einmalige Abfrage der GroupAdr auf dem KNX-Bus aus
DoWrite	BOOL	FALSE ⇒ TRUE löst einmaliges Schreiben des WriteValue mit der angegebenen GroupAdr aus
WriteValue	BOOL	Neuer Wert, der mit DoWrite gesendet wird.
Input-Parameter:		
Value	BOOL	Aktueller Zustand des KNX-Objekts
ErrCode	WORD	0 = kein Fehler 100 = der Ausgangswert Value des Objektbausteins ist ungültig (Baustein noch nicht initialisiert)

Dieser Baustein repräsentiert ein 1-Bit-Objekt (z.B. DPT_Switch, DPT_Bool, DPT_Ramp, DPT_UpDown etc.) mit der angegebenen Gruppenadresse.

Wenn am Baustein PlmKnx_SIM73027 der Wert +2 zum Eingang Mode addiert ist, löst eine Änderung des Eingangs WriteValue automatisch das Senden des Werts aus.

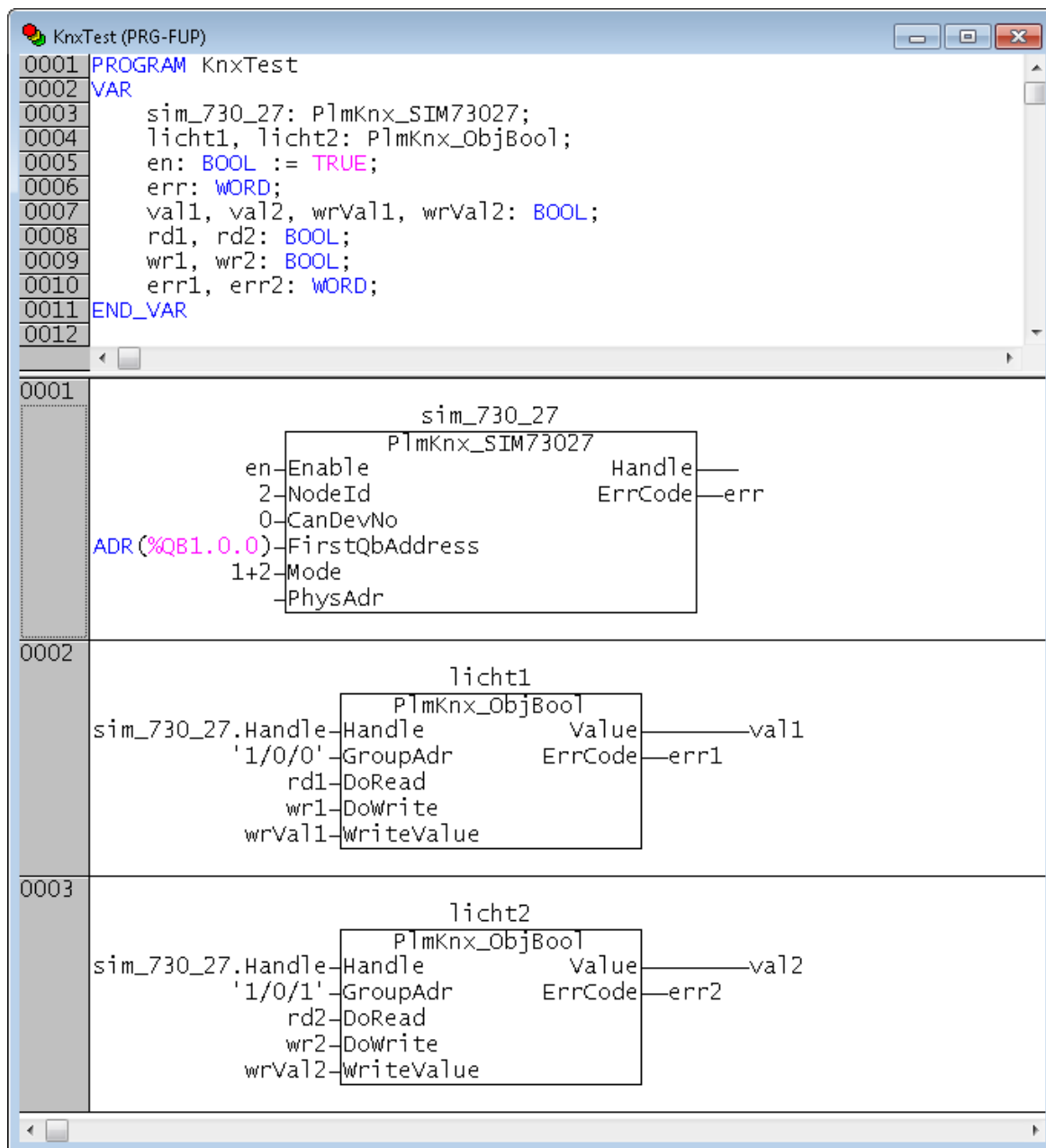
17.5. Programmbeispiel (FUP)

Das erste Programmbeispiel realisiert zwei KNX-Lichtschalter.

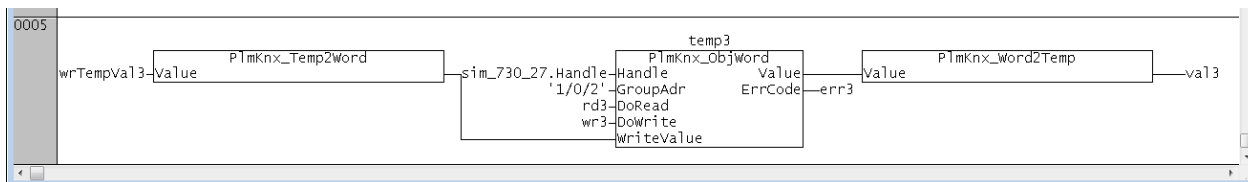
Zuvor muss mit ETS die KNX-Anlage konfiguriert worden sein, in diesem Fall z.B. ein Lichtschalter und ein Lichtschaltrelais auf Gruppenadresse '1/0/0' und ein Lichtschalter und ein Lichtschaltrelais auf Gruppenadresse '1/0/1'. Die physikalischen Adressen der beteiligten Komponenten spielen keine Rolle.

Die Lichtschalter und Schaltrelais' müssen so konfiguriert sein, dass sie alle die Eigenschaften LESEN und SCHREIBEN unterstützen.

Die aktuellen Objektzustände (Ein/Aus) liegen in den Ausgangsvariablen val1 und val2 vor. Bei Änderung der Eingangsvariablen wrVal1 oder wrVal2 werden die neuen Werte mit der jeweiligen Gruppenadresse auf den KNX-Bus gesendet und lösen den entsprechenden Schaltvorgang aus.



Das zweite Beispiel zeigt die Sollwertvorgabe für einen Einzelraumregler. Der Datenpunkt-Typ des Temperatureingangs des Reglers ist "2-Octet Float Value" (DPT_Value_Temp). Dadurch ist sowohl die Anzahl der Datenbytes im KNX-Telegramm als auch die Codierung festgelegt.



Die Variablen `wrTempVal3` und `val3` sind in diesem Fall vom Typ `REAL`, während die Datenübertragung auf dem KNX-Bus als 16-Bit-Word (2 Octets) geschieht.

Der aktuelle Objektzustand (Temperaturssollwert) liegt in der Ausgangsvariablen `val3` vor. Bei Änderung der Eingangsvariablen `wrTempVal3` wird der neue Wert mit der jeweiligen Gruppenadresse auf den KNX-Bus gesendet und ändert entsprechend die Sollwertvorgabe des Reglers.

Bei häufiger Verwendung der dargestellten Kombination kann der Programmierer sich leicht selbst einen neuen Baustein mit der Gesamtfunktion erstellen.

18. FTP-Client

18.1. Allgemeines

Das FTP-Protokoll wird verwendet, um beliebige Dateien über eine TCP-Netzwerkverbindung zu übertragen. FTP basiert auf einem Client-Server-Modell. Dabei meldet sich der Client zunächst auf dem Server an und führt dann eine Folge von Befehlen aus, um Dateien zum Server oder von diesem zu übertragen.

Das ursprüngliche FTP-Protokoll stammt aus dem Jahre 1971. Die Verbindung zwischen Client und Server ist weder verschlüsselt noch authentifiziert. Anmeldenamen und Passwörter werden im Klartext übertragen, daher genügt das Protokoll für viele Zwecke nicht den heutigen Sicherheitsanforderungen.

Die meisten Server erlauben einen anonymen Zugang unter dem Anmeldenamen "anonymous" mit beliebigem Passwort. Im Internet existieren zahlreiche FTP-Server, die unter diesem anonymen Zugang den Download von Software und Daten erlauben, daher ist das FTP-Protokoll immer noch populär und wird von allen gängigen Browsern unterstützt (URLs beginnen mit `ftp://...`).

Eine Besonderheit des FTP-Protokolls ist die Verwendung von getrennten Verbindungen für die Befehls- und Datenübertragung. Vorgesehen ist, dass zunächst die Befehlsverbindung vom Client zum Server aufgebaut wird. Wenn dann durch einen entsprechenden FTP-Befehl eine Datenübertragung angefordert wird, baut entweder der Server eine zusätzliche Verbindung zum Client auf ("Active Mode"), oder der Client eine zusätzliche Verbindung zum Server ("Passive Mode"). Da die Rückwärtsverbindung im Active Mode häufig durch Firewall-Einstellungen blockiert wird, ist der Passive Mode vorzuziehen und wird vom hier dargestellten Client unterstützt.

Der FTP-Client für PLM-Steuerungen unterstützt die wichtigsten FTP-Befehle und erlaubt damit eine Datenübertragung zwischen den internen Laufwerken der Steuerung (Local) und einem externen FTP-Server (Remote).

Der FTP-Client wird durch zyklischen Aufruf des eigentlichen Client-Bausteins `Plm_FtpClient` und der daran angehängten Befehlsbausteine realisiert. Der Client-Baustein stellt die Netzwerk-Verbindung zum FTP-Server her und übernimmt die Anmeldung (Login). Anschließend werden die Befehlsbausteine in der Programmreihenfolge ausgeführt.

18.2. Vorgehensweise

Im einfachsten Fall werden mind. zwei Bausteine angelegt: Der Client-Baustein vom Typ `Plm_FtpClient` und ein Befehlsbaustein z.B. vom Typ `Plm_FtpGet`. Der Befehlsbaustein erhält an seinem Eingang `FtpClientId` den Wert des gleichnamigen Ausgangs `FtpClientId` des Client-Bausteins und wird diesem dadurch zugeordnet.

Durch Aktivieren des Eingangs `Enable` am Client-Baustein wird die Verbindung aufgebaut und anschließend die angehängten Befehlsbausteine ausgeführt, der Client-Ausgang `Busy` ist solange TRUE. Nach Beendigung aller Befehle geht `Busy` auf FALSE. Wenn alle Befehle fehlerfrei ausgeführt werden konnten, hat der Client-Ausgang `Err` den Wert FALSE, andernfalls den Wert TRUE und in `ErrCode` steht ein Fehlercode, der Rückschlüsse auf den Fehler zulässt.

Alle Befehlsbausteine verfügen ebenfalls über einen `Enable`-Eingang. Wenn dieser konstant auf TRUE gesetzt ist, werden nach dem Herstellen der Server-Verbindung automatisch alle Bausteine der Reihe nach abgearbeitet. Alternativ können die `Enable`-Eingänge der Befehlsbausteine durch das Programm gesteuert werden. In diesem Fall wird ein Befehlsbaustein einmalig ausgeführt, sobald er an der Reihe ist und sein `Enable`-Eingang von FALSE auf TRUE gewechselt hat.

Im Detail verhalten sich verschiedene FTP-Server oft etwas unterschiedlich, daher kann das Verhalten des Clients durch Bausteine vom Typ `Plm_FtpConfig` modifiziert werden. Außerdem stehen verschiedene Möglichkeiten zur Fehlersuche zur Verfügung (siehe Abschnitt 18.6).

18.3. Spezifikation

- FTP Client gemäß RFC 959
- Unterstützte Befehle: Change Remote Directory, Create Remote Directory, Delete Remote Directory, List Remote Directory, Put File (Local → Remote), Get File (Remote → Local), Delete Remote File, Rename Remote File, Quit
- Datentransfer im "Passive Mode"
- Beliebige Anzahl von gleichzeitig aktiven FTP-Clients
- Standard-Port für Kontrollverbindung: 21 (konfigurierbar)
- Wahlweise vollautomatische Abarbeitung einer FTP-Befehlssequenz oder Ausführen von FTP-Einzelbefehlen

18.4. Benötigte Bibliotheken

Plm_FtpClient.lib
SysLibSockets.lib*
PLM_Sockets.lib*
SysLibMem.lib*
SysLibCallback.lib*
UPD_E_006.lib* (oder spätere Version)
Plm_Std.lib*

* wird von Plm_FtpClient.lib benötigt

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster → Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

18.4.1. PLM_FTPCLIENT (Plm_FtpClient.lib)

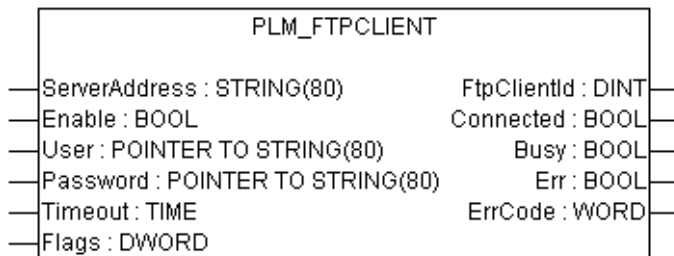


Abb. 18-1: Funktionsblock PLM_FTPCLIENT (Plm_FtpClient.lib)

Input-Parameter:		
ServerAddress	STRING	IP-Adresse des FTP-Servers, zu dem eine Verbindung hergestellt werden soll, z.B. '10.1.1.156'; nach einem Doppelpunkt kann eine Portnummer angegeben werden, z.B. '10.1.1.156:21'
Enable	BOOL	Enable-Flag, bei TRUE wird eine Verbindung zum Server aufgebaut, bei FALSE wird diese wieder beendet
User	POINTER TO STRING(80)	Adresse eines Strings mit dem Anmeldenamen. Wenn keine String-Adresse angegeben wird oder String leer ist, wird als User "anonymous" verwendet.
Password	POINTER TO STRING(80)	Adresse eines Strings mit dem Anmeldekennwort. Wenn keine String-

		Adresse angegeben wird, wird ein leeres Password verwendet. Allerdings akzeptieren einige FTP-Server beim anonymen Login zwar ein beliebiges, jedoch kein leeres Password.
Timeout	TIME	Timeout-Zeit für alle Server-Reaktionen, z.B. $t\#10s$
Flags	DWORD	Momentan nur $16\#8000$ zum Einschalten des Debug-Log-Modus'.
Output-Parameter:		
FtpClientId	DINT	Referenz (Handle) für das Anhängen der Befehlsbausteine
Connected	BOOL	TRUE = Verbindung zum FTP-Server hergestellt
Busy	BOOL	TRUE = Verbindung wird hergestellt, Anmeldung läuft oder es wird ein Befehlsbaustein ausgeführt
Err	BOOL	TRUE = Fehler beim Herstellen der Verbindung zum Server, beim Anmelden (Login) oder bei der Abarbeitung eines Befehlsbausteins, <code>ErrCode</code> enthält Fehlernummer
ErrCode	BYTE	<p>0 = kein Fehler</p> <p>1 = TIMEOUT: Der Server hat nicht innerhalb der eingestellten Timeout-Zeit geantwortet</p> <p>2 = SERVDISCONNECT: Die Verbindung wurde durch den Sever getrennt</p> <p>3 = NOFTPCLIENTVISUTASK: Der Baustein <code>Plm_FtpClientVisuTask</code> wird nicht vom Programm aufgerufen</p> <p>4 = SOCKCREATE: Es konnte kein Socket für die Verbindung zum Server angelegt werden</p> <p>5 = NOCMDBLOCKS: An den Client sind keine Befehlsblöcke angehängt</p> <p>6 = CONFIGPAR: Es wurde ein Config-Block mit einem ungültigen Wert am Eingang <code>Par</code> ausgeführt</p> <p>7 = CONFIGVAL: Es wurde ein Config-Block mit einem ungültigen Wert am Eingang <code>Val</code> ausgeführt</p> <p>8 = UNKNOWNCMD: interner Fehler</p> <p>9 = CMDFAILED: Serverantwort war negativ, letzter FTP-Befehl fehlgeschlagen</p> <p>10 = CONNECT1:</p> <p>11 = CONNECT2:</p> <p>12 = CONNECT3:</p> <p>13 = CONNECT4:</p> <p>14 = CONNECTTIMEOUT: Es konnte keine Verbindung zum FTP-Server unter <code>ServerAddress</code> hergestellt werden.</p> <p>15 = READ: Lesen vom Server fehlgeschlagen</p> <p>16 = WRITE: Schreiben zum Server fehlgeschlagen</p> <p>17 = WELCOME: negative Serverantwort in Welcome-Message</p> <p>18 = LOGINFAILED: Anmeldung mit User</p>

		und Password fehlgeschlagen 19 = PASVONLY: Datenübertragung nur im PASV-Modus möglich 20 = DATACONFAILED: Datenübertragung wurde unterbrochen 21 = PASVFAILED: Server lehnt Umschaltung der Datenverbindung in den Passive-Mode ab 22 = TYPEFAILED: Server lehnt Umschaltung der Datenverbindung mit TYPE ab 23 = LOCALFILEREADERROR: lokale Datei auf PLM-Steuerung kann nicht gelesen werden 24 = LOCALFILEWRITEERROR: lokale Datei auf PLM-Steuerung kann nicht geschrieben werden 25 = NOLOCALFILENAME: lokale Datei nicht angegeben 26 = NOREMOTEFILENAME: Remote-Datei nicht angegeben 27 = NOREMOTEDIR: Remote-Directory nicht angegeben 28 = NOFILENAMEFROM: FilenameFrom Parameter nicht angegeben 29 = NOFILENAMEFROM: FilenameTo Parameter nicht angegeben 30 = ILLEGALSTATE: interner Fehler
--	--	--

Der Baustein `PLM_FTPCLIENT` muss zyklisch aufgerufen werden.

Die angehängten Befehlsbausteine müssen mindestens einmal (z.B. in `InitOnce`) aufgerufen werden, können jedoch auch zyklisch aufgerufen werden, ohne nennenswert Ressourcen zu verschwenden.

Die Befehlsbausteine werden in genau der Reihenfolge ausgeführt, in der sie beim ersten Aufruf ausgeführt wurden, d.h. in der Programmreihenfolge.

Das Aufbauen der Verbindung zum FTP-Server geschieht, indem der Eingang `Enable` auf `TRUE` gesetzt wird. Sind der anschließende Verbindungsaufbau und das Anmelden erfolgreich, werden die an diesen Client-Baustein angehängten Befehlsblöcke ausgeführt. Anschließend muss die Verbindung wieder beendet werden, indem der Eingang `Enable` auf `FALSE` gesetzt wird.

Sobald ein Fehler auftritt, werden die folgenden Befehlsbausteine nicht mehr ausgeführt und die Verbindung automatisch beendet. Dieses Verhalten kann durch den Config-Parameter `StopOnError` modifiziert werden. Das ist insbesondere sinnvoll, wenn die `Enable`-Eingänge einzelner Befehlsbausteine nicht konstant auf `TRUE` gesetzt sind, sondern vom Programm gesteuert werden.

Der Ausgang `Busy` ist `TRUE`, wenn die Verbindung zum FTP-Server aufgebaut wird und solange Befehlsbausteine abgearbeitet werden. Wenn alle Befehlsbausteine abgearbeitet sind, geht `Busy` auf `FALSE`; das Anwenderprogramm sollte dann `Enable` auf `FALSE` setzen und damit die Verbindung beenden.

Der Ausgang `ErrCode` zeigt den zuletzt aufgetretenen Fehlercode des Clients oder eines Befehlsbausteins an.

Das Verhalten des Ausgangs `Err` hängt vom Config-Parameter `StopOnError` ab. Bei `StopOnError = TRUE` (Voreinstellung) zeigt `Err` an, dass entweder im Client oder in einem der Befehlsbausteine ein Fehler aufgetreten ist, der zum vorzeitigen Beenden der FTP-Verbindung geführt hat. Bei `StopOnError = FALSE` zeigt `Err` nur Fehler des Clients an. Fehler in einem der Befehlsbausteine können dann nur am `Err`-Ausgang des entsprechenden Befehlsbausteins erkannt werden.

Das FTP-Protokoll sieht vor, dass sich der Client vor dem Beenden der Verbindung mit dem Befehl `QUIT` beim Server abmeldet. Der `QUIT`-Befehl wird daher

automatisch gesendet, sobald `Enable` auf `FALSE` gesetzt wird. Dieses Verhalten kann durch den Config-Parameter `AutoQuit` modifiziert werden.

Manche FTP-Server beenden die Verbindung nach Ablauf einer bestimmten Zeit (z.B. 15 Minuten) automatisch. Dies kann vom Client erst bemerkt werden, wenn die Ausführung eines weiteren Befehls fehlschlägt. Die Verbindung wird in diesem Fall auch vom Client beendet, der Fehlercode am Ausgang `ErrCode` ist dann `2` (`SERVDISCONNECT`). Es empfiehlt sich generell, die Verbindung zum Server sofort nach Abarbeitung der gewünschten Befehle wieder zu trennen.

Am Eingang `ServerAddress` wird die IP-Adresse des FTP-Servers angelegt, z.B. '10.1.1.156'. Der Client versucht dann, auf Port 21 die Verbindung zum FTP-Server aufzubauen. Falls der FTP-Server einen anderen Port verwendet, kann dieser nach einem Doppelpunkt an die IP-Adresse angehängt werden, z.B. für Port 2223 '10.1.1.156:2223'.

Die Fehlercodes am Ausgang `ErrCode` sind als globale Konstanten in der Bibliothek `Plm_FtpClient.lib` definiert.

Alle Eingänge des Clients und der Befehlsbausteine, die einen String benötigen, erwarten die Angabe einer String-Adresse `ADR(str)`. Dadurch wird die Ausführungsgeschwindigkeit der Bibliotheksfunktionen erhöht.

18.4.2. PLM_FTPCLIENTVISUTASK (Plm_FtpClient.lib)

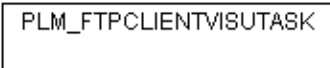


Abb. 18-2: Funktionsblock `PLM_FTPCLIENTVISUTASK` (`Plm_FtpClient.lib`)

Input-Parameter:
-
Output-Parameter:
-

Der Baustein `PLM_FTPCLIENTVISUTASK` wird nur benötigt, falls Bausteine vom Typ `Plm_FtpGet` verwendet werden. Er übernimmt das Speichern der lokalen Datei, die vom FTP-Server gesendet wird. Der Baustein muss zyklisch aufgerufen werden.

Da ein Schreibzugriff auf die Speichermedien der PLM-Steuerung u.U. länger als ein IEC-Zyklus dauert, kann hiermit der Schreibvorgang in einen zeitunkritischen Task verlagert werden. Es empfiehlt sich, den Baustein an den VISU-TASK anzuhängen.

Auch bei mehreren FTP-Clients darf der Bausteinaufruf nur einmal im Programm vorkommen.

Falls der Baustein *nicht explizit* an den VISU_TASK angehängt wird, wird er *automatisch* aus dem IEC-Task heraus ausgeführt. In diesem Fall ist jedoch mit Auswirkungen auf die Zykluslänge zu rechnen, solange ein Baustein `Plm_FtpGet` abgearbeitet wird.

18.4.3. PLM_FTPCHDIR (Plm_FtpClient.lib)

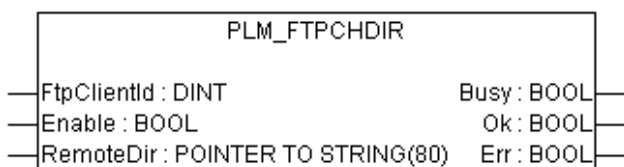


Abb. 18-3: Funktionsblock `PLM_FTPCHDIR` (`Plm_FtpClient.lib`)

Input-Parameter:

FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
RemoteDir	POINTER TO STRING (80)	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, in das gewechselt werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein Plm_FtpChdir veranlasst den FTP-Server, das aktuelle Remote-Arbeitsverzeichnis zu wechseln. Auf dieses Verzeichnis beziehen sich alle folgenden Dateioperationen auf dem Server, sofern bei diesen kein absoluter Pfad angegeben ist.

Der angegebene Pfad RemoteDir muss als ADR(str) angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.4. PLM_FTPMKDIR (Plm_FtpClient.lib)

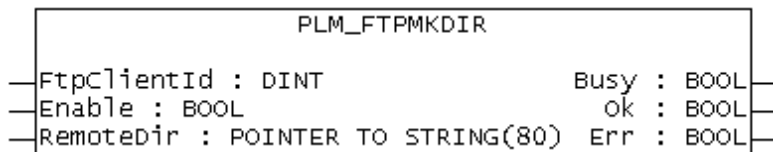


Abb. 18-4: Funktionsblock PLM_FTPMKDIR (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
RemoteDir	POINTER TO STRING (80)	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das erzeugt werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein `Plm_FtpMkdir` veranlasst den FTP-Server, das angegebene Remote-Verzeichnis zu erstellen. Der angegebene Pfad `RemoteDir` muss als `ADR(str)` angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.5. PLM_FTPRMDIR (Plm_FtpClient.lib)

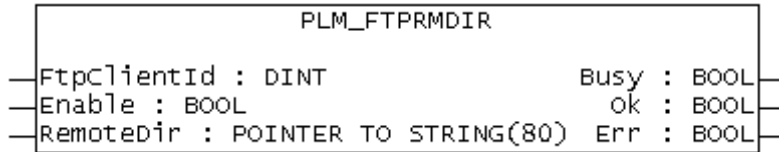


Abb. 18-5: Funktionsblock `PLM_FTPRMDIR` (`Plm_FtpClient.lib`)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
RemoteDir	POINTER TO STRING(80)	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das gelöscht werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang <code>ErrCode</code> am Client-Baustein enthält Fehlernummer

Der Baustein `Plm_FtpRmdir` veranlasst den FTP-Server, das angegebene Remote-Verzeichnis zu löschen. Der angegebene Pfad `RemoteDir` muss als `ADR(str)` angegeben werden.

Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.6. PLM_FTPDELETE (Plm_FtpClient.lib)

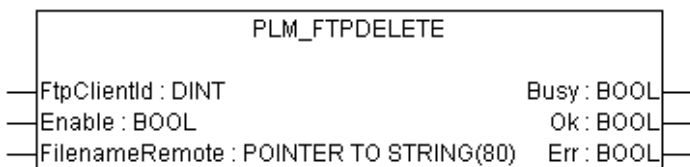


Abb. 18-6: Funktionsblock `PLM_FTPDELETE` (`Plm_FtpClient.lib`)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.

FilenameRemote	POINTER TO STRING (80)	Adresse eines Strings, der den Namen der Remote-Datei enthält, die gelöscht werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein Plm_FtpDelete veranlasst den FTP-Server, die angegebene Datei auf dem FTP-Server zu löschen.

Falls der Dateiname keinen absoluten Pfad enthält, bezieht sich FilenameRemote auf das aktuelle Remote-Arbeitsverzeichnis.

Der angegebene Dateiname FilenameRemote muss als ADR(str) angegeben werden.

Ob der Dateiname eine Pfadangabe enthalten darf oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.7. PLM_FTPRENAME (Plm_FtpClient.lib)

PLM_FTPRENAME	
— FtpClientId : DINT	Busy : BOOL
— Enable : BOOL	Ok : BOOL
— FilenameFrom : POINTER TO STRING(80)	Err : BOOL
— FilenameTo : POINTER TO STRING(80)	

Abb. 18-7: Funktionsblock PLM_FTPRENAME (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
FilenameFrom	POINTER TO STRING (80)	Adresse eines Strings, der den alten Namen der Remote-Datei enthält
FilenameTo	POINTER TO STRING (80)	Adresse eines Strings, der den neuen Namen der Remote-Datei enthält
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein Plm_FtpRename veranlasst den FTP-Server, die angegebene Datei auf dem FTP-Server von FilenameFrom in FilenameTo umzubenennen.

Falls keine absoluten Pfad angegeben sind, beziehen sich FilenameFrom und FilenameTo auf das aktuelle Remote-Arbeitsverzeichnis.

Die angegebenen Dateinamen `FilenameFrom` und `FilenameTo` müssen als `ADR(str)` angegeben werden.

Ob durch das Umbenennen die Datei in ein anderes Directory verschoben werden kann oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

18.4.8. PLM_FTPGET (Plm_FtpClient.lib)

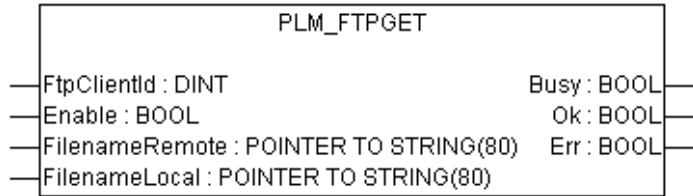


Abb. 18-8: Funktionsblock `PLM_FTPGET (Plm_FtpClient.lib)`

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
FilenameRemote	POINTER TO STRING (80)	Adresse eines Strings, der den Namen der Datei auf dem FTP-Server enthält
FilenameLocal	POINTER TO STRING (80)	Adresse eines Strings mit dem Namen, unter dem die Datei auf der PLM-Steuerung gespeichert werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang <code>ErrCode</code> am Client-Baustein enthält Fehlernummer

Der Baustein `Plm_FtpGet` veranlasst die Übertragung einer Datei vom FTP-Server auf die PLM-Steuerung.

Bei Verwendung dieses Bausteins sollte der Baustein `PLM_FTPCLIENTVISUTASK` an den VISU-TASK angehängt werden. Er übernimmt das lokale Speichern der Datei, die vom FTP-Server gesendet wird. Für weitere Informationen siehe Abschnitt 18.4.2.

Die Datei wird als Binärdatei übertragen, d.h. es erfolgt keine Umwandlung von Zeilenendesequenzen etc.

Falls der `FilenameRemote` keinen absoluten Pfad enthält, bezieht er sich auf das aktuelle Remote-Arbeitsverzeichnis.

Die angegebenen Dateinamen `FilenameRemote` und `FilenameLocal` müssen als `ADR(str)` angegeben werden.

Ob `FilenameRemote` eine Pfadangabe enthalten darf oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

Die Remote-Datei ersetzt (überschreibt) die lokale Datei. Falls stattdessen die Remote-Datei an die lokale Datei angehängt werden soll, kann dies durch Setzen des Config-Parameter `AppendMode` auf 1 erreicht werden.

18.4.9. PLM_FTPLIST (Plm_FtpClient.lib)

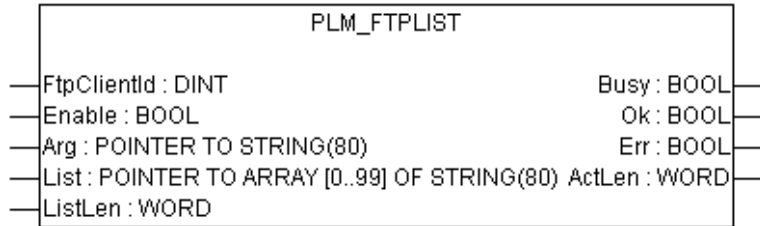


Abb. 18-9: Funktionsblock PLM_FTPLIST (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
Arg	POINTER TO STRING (80)	Adresse eines Strings, der den Namen des Remote-Verzeichnisses enthält, das gelistet werden soll.
List	POINTER TO ARRAY OF STRING (80)	Adresse eines String-Arrays, in das die Listing-Zeilen geschrieben werden. Kann 0 sein, in diesem Fall werden die Listing-Zeilen nicht gespeichert.
ListLen	WORD	Länge des List-Arrays, d.h. max. Anzahl von Listing-Zeilen, die gespeichert werden können. Kann 0 sein.
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer
ActLen	WORD	Tatsächliche Anzahl von Listing-Zeilen (unabhängig von ListLen)

Der Baustein `Plm_FtpList` veranlasst den FTP-Server, die Dateien im durch `Arg` angegebenen Verzeichnis aufzulisten.

Die Auswertung des `Arg`-Parameters und das Listing-Ergebnis hängen stark vom jeweiligen FTP-Server ab.

Wenn der Parameter `List` angegeben ist, muss der Parameter `ListLen` angegeben sein und die max. Länge des Arrays enthalten. In diesem Fall wird das Array mit den Ergebniszeilen des FTP-Servers gefüllt.

Wenn der Parameter `List` nicht oder mit 0 angegeben ist, wird `ListLen` nicht ausgewertet. Die Ergebniszeilen des FTP-Servers werden in diesem Fall nicht gespeichert.

In beiden Fällen enthält der Ausgang `ActLen` die tatsächliche Anzahl der Ergebniszeilen und kann in jedem Fall ausgewertet werden. Dies kann z.B. sinnvoll sein, wenn `Arg` den Namen einer bestimmten Remote-Datei enthält und lediglich geprüft werden soll, ob die Datei auf dem FTP-Server vorhanden ist oder nicht. In

diesem Fall brauchen die Listing-Zeilen nicht gespeichert zu werden und ActLen enthält entweder den Wert 0 oder 1.

Das Listing-Format kann mit dem Config-Parameter *WideListing* eingestellt werden: Bei *WideListing* = 0 werden nur die Dateinamen des Remote-Verzeichnisses aufgelistet. Bei *WideListing* = 1 werden weitere Angaben mit aufgelistet, die vom Betriebssystem des FTP-Servers abhängen.

Achtung: Das genaue Ergebnis des Bausteins variiert stark zwischen verschiedenen FTP-Servern und ist auch abhängig vom Listing-Format (Config-Parameter *WideListing*). Dies betrifft die Anzahl der Ergebniszeilen, die Auflistung von Verzeichnissen und das Auflisten von fehlenden Dateien. Es muss daher experimentell ermittelt werden!

18.4.10. PLM_FTTPUT (Plm_FtpClient.lib)

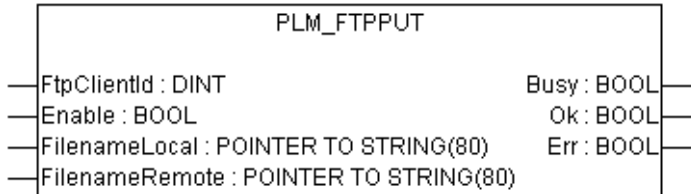


Abb. 18-10: Funktionsblock PLM_FTTPUT (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
FilenameLocal	POINTER TO STRING(80)	Adresse eines Strings, der den Namen der Datei auf der PLM-Steuerung enthält
FilenameRemote	POINTER TO STRING(80)	Adresse eines Strings mit dem Namen, unter dem die Datei auf dem FTP-Server gespeichert werden soll
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein *Plm_FtpPut* veranlasst die Übertragung einer Datei von der PLM-Steuerung auf den FTP-Server.

Die Datei wird als Binärdatei übertragen, d.h. es erfolgt keine Umwandlung von Zeilenendesequenzen etc.

Falls der *FilenameRemote* keinen absoluten Pfad enthält, bezieht er sich auf das aktuelle Remote-Arbeitsverzeichnis.

Die angegebenen Dateinamen *FilenameLocal* und *FilenameRemote* müssen als *ADR(str)* angegeben werden.

Ob *FilenameRemote* eine Pfadangabe enthalten darf oder nicht hängt vom FTP-Server ab. Als Pfadtrennzeichen ist in jedem Fall der Slash (/) zu verwenden, nicht der unter Windows übliche Backslash (\).

Die lokale Datei ersetzt (überschreibt) die Remote-Datei auf dem FTP-Server. Falls stattdessen die lokale Datei an die Remote-Datei angehängt werden soll, kann dies durch Setzen des Config-Parameter *AppendMode* auf 1 erreicht werden.

18.4.11. PLM_FTPQUIT (Plm_FtpClient.lib)

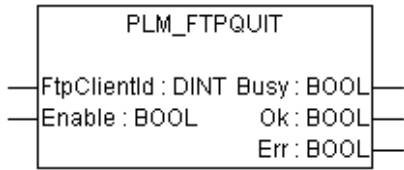


Abb. 18-11: Funktionsblock PLM_FTPQUIT (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein *Plm_FtpQuit* veranlasst eine Abmeldung auf dem FTP-Server. Anschließend muss die Verbindung durch den Client geschlossen werden.

Der QUIT-Befehl wird automatisch ausgeführt, sobald der *Enable*-Eingang am Client-Baustein auf FALSE gesetzt wird um die Verbindung zu beenden. Dieses Verhalten kann durch den Config-Parameter *AutoQuit* modifiziert werden.

18.4.12. PLM_FTPCONFIG (Plm_FtpClient.lib)

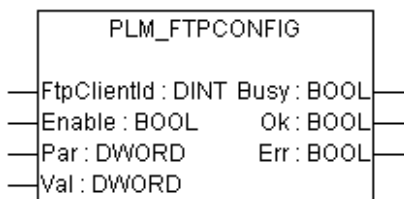


Abb. 18-12: Funktionsblock PLM_FTPCONFIG (Plm_FtpClient.lib)

Input-Parameter:		
FtpClientId	DINT	Referenz (Handle) des zugehörigen Client-Bausteins
Enable	BOOL	Enable-Flag, bei steigender Flanke FALSE → TRUE wird der Baustein einmalig ausgeführt; oder fest mit TRUE verbinden.
Par	DWORD	Nummer des Config-Parameters, der geändert werden soll (siehe globale Konstanten in <i>Plm_FtpClient.lib</i>)

Val	DWORD	Neuer Wert des Config-Parameters
Output-Parameter:		
Busy	BOOL	TRUE = Baustein wird abgearbeitet
Ok	BOOL	TRUE = Abarbeitung war erfolgreich
Err	BOOL	TRUE = Fehler bei Abarbeitung, Ausgang ErrCode am Client-Baustein enthält Fehlernummer

Der Baustein `Plm_FtpConfig` ermöglicht das Einstellen verschiedener Parameter des FTP-Clients. Im Gegensatz zu den anderen Befehlsbausteinen führt er keinen Befehl auf dem FTP-Server aus.

Wenn mehrere Parameter umgestellt werden sollen, ist für jeden ein eigener Config-Baustein zu instanzieren und einzusetzen. Dies gilt auch, wenn ein Parameter mehrfach umgestellt werden soll.

Für die Nummer des gewünschten Config-Parameters am Eingang `Par` existieren globale Konstantendefinitionen in `Plm_FtpClient.lib`. Die möglichen Werte am Eingang `Val` hängen vom gewählten Config-Parameter ab.

Die folgende Tabelle zeigt, welche Config-Parameter eingestellt werden können:

Par	Val	Voreinst.	Funktion
AppendMode (101)	DWORD = 0/1	0 (aus)	Schaltet um zwischen Überschreiben/Anhängen bei <code>Plm_FtpGet</code> und <code>Plm_FtpPut</code>
WideListing (102)	DWORD = 0/1	0 (aus)	Schaltet das Listing-Format um für <code>Plm_FtpList</code>
AutoQuit (103)	DWORD = 0/1	1 (ein)	Veranlasst das automatische Senden eines QUIT-Befehls vor dem Trennen der Verbindung durch den Client
StopOnError (104)	DWORD = 0/1	1 (ein)	Stoppt nach einem Fehler die Ausführung weiterer Befehlsbausteine. Wenn <code>DisconnectOnError</code> (109) TRUE ist, wird zusätzlich die Verbindung beendet.
PassiveMode (105)	DWORD = 0/1	1 (ein)	Legt fest, ob Datenübertragungen im Active oder Passive Mode ausgeführt werden. Momentan wird nur der Passive Mode unterstützt.
TransferMode (106)	ADR(string) = 'i/a'	'i' (binär)	Legt den Transfer Mode fest. Momentan wird nur 'i' (binär) unterstützt.
- (107)	-	-	(nicht verwendet)
DataPort (108)	DWORD = port	20	Legt den TCP-Port für Datenübertragungen fest (nur im Active Mode)
DisconnectOnError (109)	DWORD = port	1 (ein)	Wenn <code>StopOnError</code> (104) TRUE ist und ein Fehler auftritt, wird die Verbindung beendet

18.5. Programmbeispiel (FUP)

Das folgende Programmbeispiel realisiert einen FTP-Client, der sich nach dem Setzen von `do_connect` auf TRUE als User 'anonymous' und mit leerem Password mit dem FTP-Server an der IP-Adresse 127.0.0.1 verbindet.

Bei erfolgreicher Anmeldung wird die Datei 'a/local.txt' übertragen und auf dem FTP-Server unter 'a/remote.txt' gespeichert.

Nachdem der Ausgang `client.Busy` auf FALSE gegangen ist, kann `do_connect` wieder auf FALSE gesetzt und die Verbindung zum FTP-Server dadurch beendet werden.

Die im Beispiel gewählte IP-Adresse 127.0.0.1 (localhost) erlaubt die Verwendung des FTP-Servers derselben PLM-Steuerung, auf der der FTP-Client getestet wird.

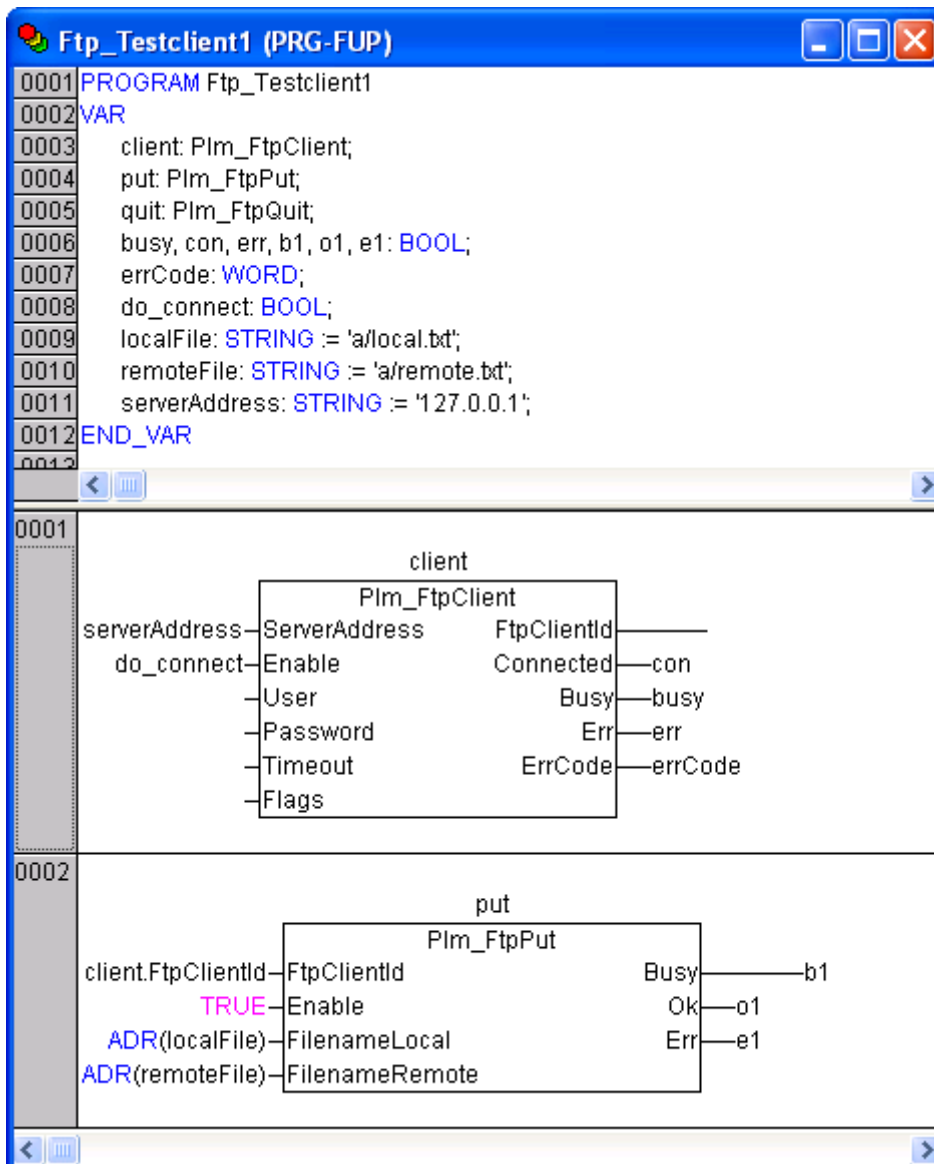


Abb. 18-13: Beispiel für Datenübertragung durch FTP-Client (FUP)

18.6. Fehlersuche

Für die Fehlersuche und Problemdiagnose stehen zwei Möglichkeiten zur Verfügung.

1. Die zuletzt aufgetretene negative Serverantwort kann als String gespeichert und dann vom Anwender ausgewertet werden.

Dazu ist der globalen Variablen `Plm_FtpLastServerErrMsg` die Adresse einer String-Variablen zuzuweisen, in der die negative Serverantwort im Klartext gespeichert wird.

Beispiel:

```
VAR
    serverMsg: STRING(80);
END_VAR

Plm_FtpLastServerErrMsg := ADR(serverMsg);
```

Nach der erfolgreichen Ausführung eines Befehlsbausteins bleibt der String in `serverMsg` leer. Falls jedoch der FTP-Server einen Befehl nicht ausführen konnte, erscheint die entsprechende Serverantwort in `serverMsg`, z.B. nach fehlgeschlagenem `Plm_Chdir`:

```
'550 mydir: No such file or directory.'
```

Die genauen Serverantworten hängen vom Typ des FTP-Servers ab.

Dieses Verfahren ersetzt nicht die Auswertung von `ErrCode`, da nur diejenigen Fehler im Klartext erscheinen, die vom FTP-Server gemeldet werden. Lokale Fehler des FTP-Clients können nur anhand des Wertes von `ErrCode` identifiziert werden.

Es werden maximal 80 Zeichen in die String-Variable geschrieben; eine längere Fehlermeldung des FTP-Servers wird abgeschnitten.

2. Die Kommunikation zwischen FTP-Client und FTP-Server kann im Klartext aufgezeichnet und anschließend analysiert werden (Log-File).

Zum Einschalten der Protokollierung ist am Eingang `Flags` des FTP-Clients der Wert `16#8000` anzulegen. Die globale Variable `Plm_FtpLogFilename` enthält den Namen der anzulegenden Log-Datei. Der Name ist voreingestellt auf

```
'a/ftp_debug.txt',
```

er kann jedoch vor dem ersten Aufruf des FTP-Client-Bausteins geändert werden.

Nach dem Ausführen des FTP-Clients kann die Log-Datei per FTP von der PLM-Steuerung auf einen PC heruntergeladen und dort mit einem Texteditor analysiert werden.

Die Aufzeichnung der Log-Datei sollte nur in der Entwicklungsphase verwendet werden, da dies erheblich zusätzliche Rechenzeit benötigt und dadurch evtl. den IEC-Zyklus verlangsamt.

19. MySQL Datenbank-Client

19.1. Allgemeines

MySQL ist ein kostenloses, leistungsstarkes und daher weit verbreitetes relationales Open-Source-Datenbanksystem. Es wurde ursprünglich von der schwedischen Firma MySQL AB entwickelt, ging danach an SUN Microsystems über und gehört seit 2010 der amerikanischen Oracle Corporation. Weitere Informationen und Downloads unter <https://www.oracle.com/>.

Die ursprünglichen Entwickler von MySQL initiierten 2009 einen Fork von MySQL unter dem Namen MariaDB, der nahezu vollständig kompatibel zu MySQL und ebenfalls kostenlos erhältlich ist. Weitere Informationen hierzu unter <https://mariadb.org/>.

Beide Datenbanksysteme bestehen generell aus einem Datenbank-Server, der die Daten speichert und über Ethernet zur Verfügung stellt, und einem oder mehreren Clients, die auf den Server zugreifen und die Daten auslesen oder manipulieren. Server und Client befinden sich üblicherweise auf verschiedenen Computern.

Nach dem Verbindungsaufbau sendet der Client Befehle in der Datenbanksprache SQL, die dann vom Server ausgeführt werden, z.B. den Befehl `INSERT` zum Einfügen von Daten oder `SELECT` zum Auslesen. Ein solcher Befehl wird als "Query" bezeichnet. An dieser Stelle kann nicht auf Details der SQL-Programmierung eingegangen werden.

Die Daten in der Datenbank sind immer in Form von zweidimensionalen Tabellen organisiert. Eine Tabelle besteht aus mindestens einer Spalte und Null oder mehr Zeilen. Bei der Abfrage von Daten mit `SELECT` erhält man auch die Ergebnisdaten als zweidimensionale Tabelle, dies wird in Abschnitt 19.4 genauer ausgeführt.

Da die Ausführungszeit der MySQL-Funktionen unbestimmt ist und ggf. mehrere Sekunden in Anspruch nehmen kann, dürfen die Funktionen nicht direkt im Zyklus aufgerufen werden. Stattdessen sind sie aus dem auf PLM-Systemen verfügbaren `SLOW_TASK` aufzurufen. Ein Beispiel findet sich in Abschnitt 19.6.

19.2. Spezifikation

- Datenbank-Client für MySQL und MariaDB
- Funktionen und Funktionsblöcke zur einfachen Verwendung in IEC 61131-Programmen
- Beliebige Anzahl von gleichzeitig aktiven MySQL-Clients
- Benötigt PLM-Steuerung aus der Serie PLM 700-A und Laufzeitsystem ab LZS v21611020

19.3. Benötigte Bibliotheken

Plm_MySQL_E_v21611020.lib (oder spätere Version)
Plm_MySQL_I_v21611290.lib (oder spätere Version)
Plm_Std.lib*

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

19.3.1. PLM_mysql_simple_query (Plm_MySQL_I.lib)

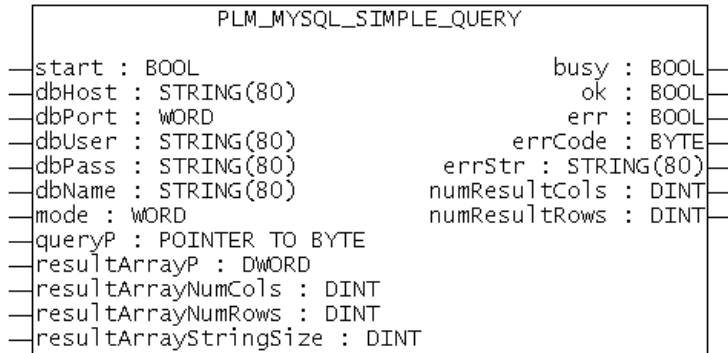


Abb. 19-1: Funktionsblock PLM_mysql_simple_query

Input-Parameter:		
start	BOOL	Bei Wechsel von FALSE ⇒ TRUE wird eine Datenbankabfrage gestartet
dbHost	STRING	IP-Adresse oder Netzwerkname des Datenbank-Servers, z.B. '10.1.1.156'
dbPort	WORD	TCP-Port des Datenbank-Servers. Bei Angabe von 0 wird der MySQL-Standard-Port 3306 verwendet.
dbUser	STRING	Anmeldename des Users an der Datenbank
dbPass	STRING	Passwort des Users für die Anmeldung, nur falls erforderlich
dbName	STRING	Name der zu verwendenden Datenbank auf dem Datenbank-Server (SQL-Befehl USE)
mode	WORD	Summe von verschiedenen Steuer-Flags: +1 ⇒ löscht das resultArray vor der Abfrage, +2 ⇒ verwirft überschüssige Spalten stillschweigend, wenn mehr Ergebnisspalten geliefert werden, als das resultArray aufnehmen kann (ansonsten Abbruch mit Fehler), +4 ⇒ verwirft überschüssige Zeilen stillschweigend, wenn mehr Ergebniszeilen geliefert werden, als das resultArray aufnehmen kann (andernfalls Abbruch mit Fehler)
queryP	POINTER TO STRING	Adresse ADR () eines Strings mit der durchzuführenden Datenbankabfrage (Query), z.B. 'SELECT * FROM Cars' oder 'INSERT INTO Cars VALUES (NULL, "Audi", 1986)'
resultArrayP	DWORD	Adresse ADR () eines zweidimensionalen String-Arrays zur Aufnahme der Abfrageergebnisse. Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
resultArray NumCols	DINT	Anzahl der Spalten des Ergebnis-Arrays (Größe in x-Richtung). Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
resultArray NumRows	DINT	Anzahl der Zeilen des Ergebnis-Arrays (Größe in y-Richtung). Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.

<code>resultArray StringSize</code>	DINT	Länge der Strings des Ergebnis-Arrays, z.B. 80 bei <code>ARRAY OF STRING(80)</code> . Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
Output-Parameter:		
<code>busy</code>	BOOL	TRUE = Datenbankabfrage läuft
<code>ok</code>	BOOL	TRUE = Datenbankabfrage erfolgreich beendet
<code>err</code>	BOOL	TRUE = Datenbankabfrage mit Fehler beendet, <code>errCode</code> und <code>errStr</code> enthalten weitere Informationen
<code>errCode</code>	BYTE	0 = kein Fehler 10 = interne Initialisierung fehlgeschlagen 20 = Verbindung zum Datenbank-Server konnte nicht aufgebaut werden 25 = Query-String <code>queryP</code> fehlt oder ist leer 30 = Query fehlgeschlagen (z.B. unzulässige Abfrage oder SQL-Syntaxfehler) 41 = <code>resultArrayP</code> ungültig 42 = <code>resultArrayNumCols</code> ungültig oder <code>resultArrayNumRows</code> ungültig oder <code>resultArrayStringSize</code> ungültig 43 = Leeres Ergebnis, keine Ergebnisspalten 44 = Zu viele Ergebnisspalten, mehr als in das <code>resultArray</code> passen 50 = Zu viele Ergebniszeilen, mehr als in das <code>resultArray</code> passen
<code>errStr</code>	STRING	Fehlertext, falls <code>err = TRUE</code>
<code>numResultCols</code>	DINT	Anzahl der im <code>resultArray</code> tatsächlich ausgefüllten Ergebnisspalten
<code>numResultRows</code>	DINT	Anzahl der im <code>resultArray</code> tatsächlich ausgefüllten Ergebniszeilen

Der Baustein `PLM_mysql_simple_query` führt eine vollständige Datenbankabfrage durch, indem er eine Netzwerkverbindung zum Datenbank-Server herstellt, sich unter dem angegebenen User-Namen anmeldet und dann die angegebene Query ausführt. Ggf. anfallende Ergebnisdaten werden im `resultArray` gespeichert. Anschließend wird die Verbindung zum Server beendet.

Der Baustein muss im `SLOW_TASK` zyklisch aufgerufen werden.

Falls die Query mehr Ergebnisspalten oder -zeilen liefert, als das `resultArray` aufnehmen kann (`resultArrayNumCols` und `resultArrayNumRows`), erfolgt normalerweise ein Abbruch mit `errCode = 44` oder `50`. Dies kann verhindert werden, indem `mode` entsprechend gesetzt wird (+2 und/oder +4). In diesem Fall werden überschüssige Abfrageergebnisse stillschweigend verworfen. In jedem Fall enthalten `numResultCols` und `numResultRows` die tatsächlich im `resultArray` ausgefüllte Anzahl Spalten und Zeilen.

Abfrageergebnisse liegen immer in String-Form vor und müssen ggf. in das gewünschte Zahlenformat umgewandelt werden (z.B. mittels `STRING_TO_INT`).

Falls die Query einen Wert in einer Auto-Inkrement-Spalte automatisch erzeugt, kann dieser über die lokale Bausteinvariable `lastInsertId` ausgelesen werden.

Der Baustein `PLM_mysql_simple_query` führt nur eine einzige Query aus und beendet anschließend die Verbindung zum Datenbank-Server. Für komplexere oder schnellere Datenbankoperationen sind die Funktionen `PLM_mysql_db_open`, `PLM_mysql_db_query` und `PLM_mysql_db_close` aus derselben Bibliothek zu verwenden.

19.3.2. PLM_mysql_db_open (Plm_MySQL_I.lib)

```

PLM_MYSQL_DB_OPEN
mysqlStructP : POINTER TO PLM_MYSQL_STRUCT
plm_mysql_db_open : INT
dbHost : STRING(80)
dbPort : WORD
dbUser : STRING(80)
dbPass : STRING(80)
dbName : STRING(80)
    
```

Abb. 19-2: Funktion PLM_mysql_db_open

Input-Parameter:		
mysqlStructP	POINTER TO PLM_MYSQL_STRUCT	Adresse ADR() einer Variablen vom Typ PLM_MYSQL_STRUCT
dbHost	STRING	IP-Adresse oder Netzwerkname des Datenbank-Servers, z.B. '10.1.1.156'
dbPort	WORD	TCP-Port des Datenbank-Servers. Bei Angabe von 0 wird der MySQL-Standard-Port 3306 verwendet.
dbUser	STRING	Anmeldename des Users an der Datenbank
dbPass	STRING	Passwort des Users für die Anmeldung, nur falls erforderlich
dbName	STRING	Name der zu verwendenden Datenbank auf dem Datenbank-Server (SQL-Befehl USE)
Rückgabewert:		
	INT	0 = Fehler aufgetreten 1 = ok

Die Funktion PLM_mysql_db_open stellt eine Netzwerkverbindung zum Datenbank-Server her und meldet sich unter dem angegebenen User-Namen an. Die Verbindung bleibt danach geöffnet.

Alle relevanten Informationen zu dieser Datenbankverbindung werden in der übergebenen Variablen vom Typ PLM_MYSQL_STRUCT gespeichert. Dieselbe Variable muss daher auch an folgende Aufrufe von PLM_mysql_db_query und PLM_mysql_db_close übergeben werden. Der Typ PLM_MYSQL_STRUCT ist wie folgt definiert:

```

TYPE PLM_MYSQL_STRUCT :
STRUCT
    errCode: BYTE; (* last error code, see Global constants PLM_MYSQL_ERR... *)
    errStr: STRING; (* last error text *)
    numResultCols: DINT; (* actual num of columns filled in resultArray[] after query *)
    numResultRows: DINT; (* actual num of rows filled in resultArray[] after query *)
    lastInsertId: DWORD; (* auto increment value after INSERT query *)
    elapsedTime: DWORD; (* duration in ms of last db_open() or db_query() *)
    pCon: DWORD;
END_STRUCT
END_TYPE
    
```

Im Fehlerfall (Rückgabewert = 0) stehen der Fehlercode und ein detaillierter Fehlertext in errCode und errStr. Die Parameter entsprechen den Output-Parametern von PLM_mysql_simple_query aus Abschnitt 19.3.1.

Da die Ausführungsdauer unbestimmt ist, darf die Funktion nur aus dem SLOW_TASK heraus aufgerufen werden.

19.3.3. PLM_mysql_db_close (Plm_MySQL_I.lib)

```

PLM_MYSQL_DB_CLOSE
mysqlStructP : POINTER TO PLM_MYSQL_STRUCT PLM_mysql_db_close : INT
    
```

Abb. 19-3: Funktion PLM_mysql_db_close

Input-Parameter:		
mysqlStructP	POINTER TO PLM_MYSQL_STRUCT	Adresse ADR() einer Variablen vom Typ PLM_MYSQL_STRUCT
Rückgabewert:		
	INT	0 = Fehler aufgetreten 1 = ok

Die Funktion PLM_mysql_db_close beendet und schließt eine zuvor mit PLM_mysql_db_open geöffnete Netzwerkverbindung zum Datenbank-Server.

Da die Ausführungsdauer unbestimmt ist, darf die Funktion nur aus dem SLOW_TASK heraus aufgerufen werden.

19.3.4. PLM_mysql_db_query (Plm_MySQL_I.lib)

```

PLM_MYSQL_DB_QUERY
mysqlStructP : POINTER TO PLM_MYSQL_STRUCT PLM_mysql_db_query : INT
mode : WORD
queryP : POINTER TO BYTE
resultArrayP : DWORD
resultArrayNumCols : DINT
resultArrayNumRows : DINT
resultArrayStringSize : DINT
    
```

Abb. 19-4: Funktion PLM_mysql_db_query

Input-Parameter:		
mysqlStructP	POINTER TO PLM_MYSQL_STRUCT	Adresse ADR() einer Variablen vom Typ PLM_MYSQL_STRUCT
mode	WORD	Summe von verschiedenen Steuer-Flags: +1 ⇒ löscht das resultArray vor der Abfrage, +2 ⇒ verwirft überschüssige Spalten stillschweigend, wenn mehr Ergebnisspalten geliefert werden, als das resultArray aufnehmen kann (ansonsten Abbruch mit Fehler), +4 ⇒ verwirft überschüssige Zeilen stillschweigend, wenn mehr Ergebniszeilen geliefert werden, als das resultArray aufnehmen kann (andernfalls Abbruch mit Fehler)
queryP	POINTER TO STRING	Adresse ADR() eines Strings mit der durchzuführenden Datenbankabfrage (Query), z.B. 'SELECT * FROM Cars' oder 'INSERT INTO Cars VALUES (NULL, "Audi", 1986)'
resultArrayP	DWORD	Adresse ADR() eines

		zweidimensionalen String-Arrays zur Aufnahme der Abfrageergebnisse. Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
resultArray NumCols	DINT	Anzahl der Spalten des Ergebnis-Arrays (Größe in x-Richtung). Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
resultArray NumRows	DINT	Anzahl der Zeilen des Ergebnis-Arrays (Größe in y-Richtung). Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
resultArray StringSize	DINT	Länge der Strings des Ergebnis-Arrays, z.B. 80 bei <code>ARRAY OF STRING(80)</code> . Nur erforderlich, wenn die angegebene Query eine Abfrage durchführt.
Rückgabewert:		
	INT	0 = Fehler aufgetreten 1 = ok

Die Funktion `PLM_mysql_db_query` führt die angegebene Query auf dem Datenbank-Server aus. Die Verbindung zum Server muss zuvor mittels `PLM_mysql_db_open` hergestellt worden sein. Falls Ergebnisse anfallen, werden diese in dem angegebenen `resultArray` gespeichert.

Alle relevanten Informationen zu dieser Datenbankverbindung werden in der übergebenen Variablen vom Typ `PLM_MYSQL_STRUCT` gespeichert. Zur Definition des Datentyps `PLM_MYSQL_STRUCT` siehe Abschnitt 19.3.2.

Im Fehlerfall (Rückgabewert = 0) stehen der Fehlercode und ein detaillierter Fehlertext in `errCode` und `errStr`.

Falls die Query mehr Ergebnisspalten oder -zeilen liefert, als das `resultArray` aufnehmen kann (`resultArrayNumCols` und `resultArrayNumRows`), erfolgt normalerweise ein Abbruch mit `errCode = 44` oder `50`. Dies kann verhindert werden, indem `mode` entsprechend gesetzt wird (+2 und/oder +4). In diesem Fall werden überschüssige Abfrageergebnisse stillschweigend verworfen. In jedem Fall enthalten `numResultCols` und `numResultRows` die tatsächlich im `resultArray` ausgefüllte Anzahl Spalten und Zeilen.

Abfrageergebnisse liegen immer in String-Form vor und müssen ggf. in das gewünschte Zahlenformat umgewandelt werden (z.B. mittels `STRING_TO_INT`).

Falls die Query einen Wert in einer Auto-Inkrement-Spalte automatisch erzeugt, kann dieser über die lokale Bausteinvariable `lastInsertId` ausgelesen werden.

Da die Ausführungsdauer unbestimmt ist, darf die Funktion nur aus dem `SLOW_TASK` heraus aufgerufen werden.

19.4. Ergebnis-Array

Beispiel:

Das Ergebnis-Array soll maximal 4 Spalten (0...3) und 6 Zeilen (0...5) aufnehmen können. Die Arrayelemente müssen immer vom Typ `STRING` sein. In diesem Fall wurden die Elemente mit `STRING(80)` deklariert (Länge max. 80 Zeichen).

```
resArr: ARRAY[0..3,0..5] OF STRING(80);
```

Der Aufruf von `PLM_mysql_db_query` muss dann wie folgt aussehen:

```
PLM_mysql_db_query(
  mysqlStructP := ADR(mysqlStruct),
  queryP := ADR(query),
```

```

resultArrayP := ADR(resArr),
resultArrayNumCols := 4,
resultArrayNumRows := 6,
resultArrayStringSize := 80
);

```

Die Arrayelemente von `resArr[]` können wie folgt grafisch dargestellt werden:

	Spalte 0	Spalte 1	Spalte 2	Spalte 3
Zeile 0	[0,0]	[1,0]	[2,0]	[3,0]
Zeile 1	[0,1]	[1,1]	[2,1]	[3,1]
Zeile 2	[0,2]	[1,2]	[2,2]	[3,2]
Zeile 3	[0,3]	[1,3]	[2,3]	[3,3]
Zeile 4	[0,4]	[1,4]	[2,4]	[3,4]
Zeile 5	[0,5]	[1,5]	[2,5]	[3,5]

Jedes Array-Element besteht dabei aus einem String mit 80 Zeichen maximaler Länge.

Anschließend werde z.B. eine Query durchgeführt, die 3 Ergebnisspalten und 2 Ergebniszeilen liefert:

```
SELECT Firstname, Lastname, Age FROM People;
```

Die Ergebnisse der Query würden wie folgt in `resArr[]` abgelegt:

	FirstName	LastName	Age	Spalte 3
Zeile 0	Peter	Miller	27	[3,0]
Zeile 1	Paul	Meyer	42	[3,1]
Zeile 2	[0,2]	[1,2]	[2,2]	[3,2]
Zeile 3	[0,3]	[1,3]	[2,3]	[3,3]
Zeile 4	[0,4]	[1,4]	[2,4]	[3,4]
Zeile 5	[0,5]	[1,5]	[2,5]	[3,5]

Die nicht blau markierten Felder würden nicht belegt.

Im Allgemeinen gilt:

Die Anzahl der Ergebnisspalten wird durch die Query festgelegt und ist daher dem Programmierer bekannt.

Die Anzahl der Ergebniszeilen hängt vom Inhalt der Datenbank ab und ist dem Programmierer daher zunächst unbekannt. Generell muss er damit rechnen, entweder keine, eine oder mehrere Ergebniszeilen zu erhalten.

Entsprechend ist die Größe von `resArr[]` festzulegen.

Falls die Query mehr Spalten oder Zeilen liefert, als das Array aufnehmen kann, erfolgt normalerweise ein Abbruch mit `errCode = 44` oder `errCode = 50`. Dies kann verhindert werden, indem `mode` entsprechend gesetzt wird (+2 und/oder +4). In diesem Fall werden überschüssige Abfrageergebnisse stillschweigend verworfen. In jedem Fall enthalten `numResultCols` und `numResultRows` die tatsächlich in `resArr[]` ausgefüllte Anzahl Spalten und Zeilen.

Sonderfall: Nur eine Spalte oder nur eine Zeile

In einigen Fällen wird nur eine Spalte oder nur eine Zeile als Ergebnis erwartet bzw. benötigt. In diesem Fall kann das Ergebnis-Array mit nur einer Dimension deklariert werden, z.B.

```
resArr2: ARRAY[0..5] OF STRING(80);
```

Um mit diesem Array 1 Spalte und bis zu 6 Zeilen aufnehmen zu können, muss der Aufruf von `PLM_mysql_db_query` wie folgt aussehen:

```
PLM_mysql_db_query(
    mysqlStructP := ADR(mysqlStruct),
    queryP := ADR(query),
    resultArrayP := ADR(resArr2),
    resultArrayNumCols := 1,
    resultArrayNumRows := 6,
    resultArrayStringSize := 80
);
```

Die bis zu 6 Ergebniszeilen stehen anschließend in den Array-Elementen `resArr2[0..5]`.

Um mit demselben Array 6 Spalten, aber nur 1 Zeile aufnehmen zu können, muss der Aufruf von `PLM_mysql_db_query` wie folgt aussehen:

```
PLM_mysql_db_query(
    mysqlStructP := ADR(mysqlStruct),
    queryP := ADR(query),
    resultArrayP := ADR(resArr2),
    resultArrayNumCols := 6,
    resultArrayNumRows := 1,
    resultArrayStringSize := 80
);
```

Die 6 Ergebnisspalten stehen anschließend in den Array-Elementen `resArr2[0..5]`.

Array mit STRING-Elementen

Das Ergebnis-Array muss immer als Datentyp `STRING` oder `STRING(n)` deklariert werden. Ggfs. sind die String-Ergebnisse nach der Query in einen anderen Datentyp umzuwandeln.

Ohne explizite Deklaration einer Länge beträgt die (max.) Länge eines Strings 80 Zeichen. Alternativ kann eine bestimmte Länge in der Deklaration vorgegeben werden, z.B. `STRING(20)`.

In jedem Fall ist die deklarierte Länge im Parameter `resultArrayStringSize` zu übergeben.

Die String-Länge kann *nicht* direkt mit dem Operator `sizeof()` ermittelt werden, da dieser nicht die String-Länge, sondern den internen Speicherbedarf zurückliefert; z.B. den Wert 81 ist bei einem `STRING(80)`.

Der Programmierer sollte berücksichtigen, dass große Arrays mit großen Strings erhebliche Mengen an Variablenspeicher auf der Steuerung belegen.

Falls der Ergebnis-String aus der Datenbank länger ist als die deklarierte String-Länge, wird er stillschweigend abgeschnitten, es erfolgt keine Warnung oder Fehlerabbruch.

19.5. Hilfsfunktionen zum Erstellen des Query-Strings

Im Ordner "Util" der Bibliothek finden sich Hilfsfunktionen, die das Erstellen des Query-Strings erleichtern.

Der Query-String ist häufig nicht konstant, sondern muss aus einzelnen Teilen zusammengesetzt werden, z.B. beim Schreiben von aktuellen Werten in die Datenbank. Man legt dazu für die Query eine String-Variable ausreichender Länge an und hängt an diese die gewünschten MySQL-Befehlssteile und Werte als String an. Diese Vorgehensweise wird von den Funktionsbausteinen `PLM_mysql_append...()` unterstützt.

Die Funktionsbausteine sehen vor, dass im Input-Parameter `dstStrP` die Adresse `ADR()` des Query-Strings übergeben wird und im Input-Parameter `dstStrSize` die

maximale Länge des Query-Strings (z.B. `STRING(80)` \Rightarrow `dstStrSize := 80`). Der Parameter `dstStrSize` dient dazu, String-Überläufe zu verhindern.

Zu Beginn muss der Query-String leer sein oder mit dem Anfang der SQL-Query initialisiert sein. Anschließend werden die fehlenden Query-Teile mit `PLM_mysql_append_...()` einzeln angehängt.

Der für die Query vorgesehene String muss eine ausreichende Länge haben, um in jedem Fall auch die dynamischen Inhalte vollständig aufnehmen zu können. Andernfalls wird die Query bei Verwendung der `PLM_mysql_append_...()`-Funktionen abgeschnitten, was zu einer Fehlermeldung der Datenbank oder zu ungewollten Ergebnissen führt. Je nach Anwendung kann eine Deklaration wie `query: STRING(100);` oder sogar `query: STRING(500);` sinnvoll sein.

Folgende Funktionsbausteine stehen zur Verfügung:

19.5.1. **PLM_mysql_append_dint()**

Hängt an den angegebenen Query-String eine vorzeichenbehaftete Variable oder Zahl vom Typ `SINT`, `INT` oder `DINT` an.

19.5.2. **PLM_mysql_append_dword()**

Hängt an den angegebenen Query-String eine Variable oder Zahl ohne Vorzeichen vom Typ `BYTE`, `WORD` oder `DWORD` an.

19.5.3. **PLM_mysql_append_real()**

Hängt an den angegebenen Query-String eine Variable oder Zahl vom Typ `REAL` an. Zusätzlich muss das Ausgabeformat angegeben werden, z.B. `'%.3f'` für eine Darstellung mit drei Nachkommastellen.

19.5.4. **PLM_mysql_append_string()**

Hängt an den angegebenen Query-String eine Variable vom Typ `STRING` oder einen konstanten String an.

19.5.5. **PLM_mysql_append_stringp()**

Hängt an den angegebenen Query-String eine Variable vom Typ `STRING` an. In diesem Fall muss die Adresse `ADR()` der anzuhängenden String-Variablen übergeben werden.

19.5.6. **PLM_mysql_append_string_quote()**

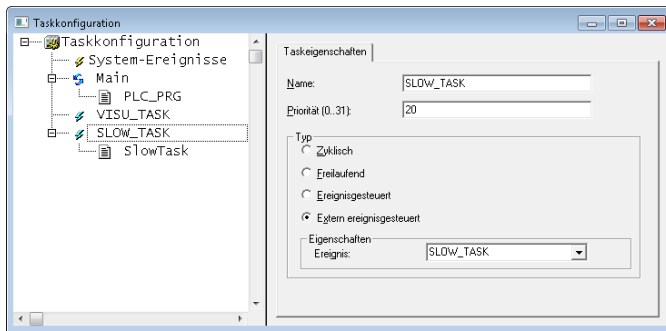
Hängt an den angegebenen Query-String eine Variable vom Typ `STRING` oder einen festen String an, der dabei in Hochkommas `'...'` eingeschlossen wird.

19.5.7. **PLM_mysql_append_stringp_quote()**

Hängt an den angegebenen Query-String eine Variable vom Typ `STRING` an, der dabei in Hochkommas `'...'` eingeschlossen wird. In diesem Fall muss die Adresse `ADR()` der anzuhängenden String-Variablen übergeben werden.

19.6. **Programmbeispiel (ST)**

Das folgende Programmbeispiel realisiert einen MySQL-Client im `SLOW_TASK`. Dieser ist in der Taskkonfiguration einzurichten:



An den SLOW_TASK wird ein Programmbaustein namens SlowTask angehängt.
Dieser sieht wie folgt aus:

```
PROGRAM SlowTask
VAR_INPUT
    doQuery: BOOL;
END_VAR
VAR
    query: STRING := 'SELECT * from User';
    mysqlStruct: PLM_MYSQL_STRUCT;
    retn: INT;
    t1, t2: DWORD;
    resArr: ARRAY[0..3,0..5] OF STRING(80);
    doQuery: BOOL;
END_VAR

IF doQuery THEN

    retn := PLM_mysql_db_open(
        mysqlStructP := ADR(mysqlStruct),
        dbHost := '10.1.1.149',
        dbPort := 3306,
        dbUser := 'cgi',
        dbPass := 'cgipassw',
        dbName := 'customerdb',
    );
    t1 := mysqlStruct.elapsedTime;

    IF retn > 0 THEN

        PLM_mysql_db_query(
            mysqlStructP := ADR(mysqlStruct),
            mode := 7,
            queryP := ADR(query),
            resultArrayP := ADR(resArr),
            resultArrayNumCols := 4,
            resultArrayNumRows := 6,
            resultArrayStringSize := 80
        );
        t2 := mysqlStruct.elapsedTime;

        PLM_mysql_db_close(
            mysqlStructP := ADR(mysqlStruct),
        );

    END_IF

    doQuery := FALSE;

END_IF
```

Die Datenbank-Abfrage wird durch Setzen des Input-Parameters `doQuery` gestartet; dies kann z.B. im `PLC_PRG` geschehen.

Das `PLC_PRG` wartet im Zyklus darauf, dass `doQuery` wieder `FALSE` wird als Zeichen, dass die Datenbankabfrage beendet ist.

Anschließend liegen alle relevanten Informationen in den Variablen `mysqlStruct` (Fehlercode bzw. Anzahl Spalten/Zeilen des Ergebnisses) und `resArr[]` (Ergebnistabelle).

19.7. Fehlersuche und MySQL-Konsole

Zum Lieferumfang von MySQL gehört ein leistungsstarker Kommandozeilen-Client, der auf einem PC mit Windows oder Linux installiert wird. Dieser Client wird auch als

MySQL-Konsole bezeichnet und kann bei der Fehlersuche sehr hilfreich sein. Mit der MySQL-Konsole können z.B. die Anmeldeprozedur ausprobiert oder testweise SQL-Queries abgesetzt werden. Die folgende Beschreibung kann nur in Ansätzen auf die Möglichkeiten eingehen.

Im Folgenden wird vorausgesetzt, dass der Client auf einem PC installiert ist (hier: Ubuntu-Linux) und nach dem Öffnen eines Terminals durch Eingabe des Befehls `mysql` aufgerufen wird.

Viele Probleme bei der Verwendung von MySQL hängen mit dem Verbindungsaufbau und der Anmeldung an der Datenbank zusammen. Der Fehlercode der in den Abschnitten 19.3.1 und 19.3.2 beschriebenen Bibliotheksfunktionen (`errCode`) ist in diesem Fall 20.

Obwohl die Fehlersuche mittels PC sehr hilfreich sein kann, muss bedacht werden, dass die Steuerung, auf der der MySQL-Client läuft, eine andere IP-Adresse besitzt. Das kann einerseits dazu führen, dass sich die Netzwerkverbindung von der Steuerung zum Datenbank-Server anders gestaltet als vom PC aus (physikal. Layer, Routing), andererseits können auf dem Datenbank-Server Client-Zugriffe auf bestimmte IP-Adressen beschränkt sein, so dass der Zugriff nur von bestimmten IP-Adressen aus möglich ist. Entsprechende User-Privileges müssen durch den Datenbank-Administrator konfiguriert und mitgeteilt werden.

Zunächst sollte mit `ping` geprüft werden, oder der Datenbank-Serverrechner überhaupt über das Netzwerk erreichbar ist (was voraussetzt, dass der Server entsprechende Anfragen beantwortet). Um von der PLM-Steuerung aus ein `ping` absetzen zu können, verwenden Sie das WebConfig der Steuerung, Menüpunkt *Netzwerk*.

Die relevanten User-Einstellungen sind in der MySQL-Datenbank selbst gespeichert. Um sie auszulesen oder zu ändern meldet sich der Datenbank-Administrator in der MySQL-Konsole als User `root` an. Dann:

```
mysql> use mysql;
Database changed
mysql> SELECT Host, User, Password FROM user;
+-----+-----+-----+
| Host      | User          | Password                                     |
+-----+-----+-----+
| localhost | root          | *114ED8E81AF733F40E7648119609EA067EB7E |
| 127.0.0.1 | root          | *114ED8E81AF733F40E7648119609EA067EB7E |
| localhost | debian-sys-maint | *9DB678631C586B842411C2185D454B458E576 |
| %         | cgi           | *DF951EBBB335505895B275567923B2B1B6921 |
| localhost | testuser      | *00E247985F9AF26AE0194B41E19DEE1429A29 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> show grants for 'testuser';
+-----+-----+-----+
| Grants for testuser@localhost                                     |
+-----+-----+-----+
| GRANT USAGE ON *.* TO 'testuser'@'localhost' IDENTIFIED BY PASSWORD '*00E247985F9AF26A51429A29' |
| GRANT SELECT, INSERT, UPDATE, DELETE, LOCK TABLES ON `customerdb`.* TO 'testuser'@'localhost' |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

In der Ausgabe ist ersichtlich, welcher User von welcher Netzwerk-Adresse aus zugreifen darf ("% bedeutet "alle") und in welcher Konstellation ein Passwort erforderlich ist (die Passwords werden verschlüsselt gespeichert und angezeigt).

Zudem können Einstellungen in der Konfigurationsdatei des MySQL-Servers (`/etc/mysql/my.cnf`) Netzwerkzugriffe verhindern, z.B. `skip-networking` oder `bind-address`. Details sind der Dokumentation zum MySQL-Server zu entnehmen.

Änderungen der User-Privileges durch den Datenbank-Administrator müssen mit dem SQL-Befehl `flush privileges` aktiviert werden. Ggf. ist ein Neustart des MySQL-Server-Prozesses erforderlich.

Wenn die User-Privileges korrekt vergeben sind, kann mit der MySQL-Konsole versucht werden, eine Verbindung zum Datenbank-Server herzustellen. Dabei werden die Aufrufparameter `-h host` zur Angabe eines Server-Hosts und `-u user` zur

Angabe des User-Namens benötigt. Falls ein Passwort erforderlich ist, muss der Aufrufparameter `-p` angegeben werden, in diesem Fall wird das Passwort mit einer unsichtbaren Eingabe abgefragt:

```
$ mysql -h localhost -u testuser
ERROR 1045 (28000): Access denied for user 'testuser'@'localhost' (using password: NO)
$ mysql -h localhost -u testuser -p
Enter password:
Welcome to the MySQL monitor.
[...]
```

Wenn der Konsolen-Prompt `mysql>` erscheint, konnte der User erfolgreich angemeldet werden.

Bereits ohne Auswahl einer Datenbank und ohne Kenntnis der vorhandenen Tabellen können jetzt interne MySQL-Funktionen aufgerufen werden, z.B.

```
mysql> select version();
+-----+
| version() |
+-----+
| 5.1.61-0ubuntu0.10.10.1 |
+-----+
1 row in set (0.00 sec)

mysql> select curdate();
+-----+
| curdate() |
+-----+
| 2016-11-30 |
+-----+
1 row in set (0.00 sec)

mysql>
```

Da ein Datenbank-Server mehrere Datenbanken verwalten kann, wird normalerweise mit `use` die gewünschte Datenbank ausgewählt. Die Datenbank mit dem Namen `mysql` dient internen Zwecken der Datenbank und bleibt dem Datenbank-Administrator vorbehalten. Beim Datenbanknamen ist Groß-/Kleinschreibung relevant.

```
mysql> show databases;
+-----+
| Database |
+-----+
| information schema |
| customerdb |
| mysql |
| test |
+-----+
4 rows in set (0.00 sec)

mysql> use customerdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

Es können nun weitere Informationen über die gewählte Datenbank und ihre Tabellen abgerufen werden:

```
mysql> show tables;
+-----+
| Tables_in_customerdb |
+-----+
| Address |
| Attachment |
| BusinessKind |
| Company |
| Contact |
| [...] |
| User |
| UserSettings |
+-----+
21 rows in set (0.00 sec)

mysql> describe Company;
+-----+-----+-----+-----+-----+-----+
|
```

```

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CompanyID | int(10) unsigned | NO | PRI | NULL | auto_increment |
| NodeID | int(10) unsigned | NO | | NULL | |
| LocalName | varchar(100) | YES | | NULL | |
| SalesVolume | int(11) | YES | | NULL | |
| BusinessKind | varchar(64) | YES | | NULL | |
| Url | varchar(100) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
    
```

Schließlich ist auch die Abfrage von Tabelleninhalten möglich:

```

mysql> SELECT CompanyID, LocalName, SalesVolume, Url FROM Company;
+-----+-----+-----+-----+
| CompanyID | LocalName | SalesVolume | Url |
+-----+-----+-----+-----+
| 1 | SABO | 0 | |
| 2 | Siemens AG | 0 | |
[...
+-----+-----+-----+-----+
37 rows in set (0.00 sec)

mysql>
    
```

An einigen Stellen der SQL-Query ist Groß-/Kleinschreibung relevant. Dies gilt z.B. für Datenbank- und Tabellennamen, nicht jedoch für Spaltennamen. Im Zweifelsfall sollte Groß-/Kleinschreibung in der Query immer beachtet werden. Bei den MySQL-Schlüsselwörtern, wie `SELECT`, `FROM` usw., spielt Groß-/Kleinschreibung keine Rolle. Die MySQL-Konsole kann auch hier als Hilfsmittel zum Ausprobieren von Queries dienen, bevor diese aus der Steuerung abgesetzt werden.

Die MySQL-Konsole kann durch Eingabe von `quit` verlassen werden.

20. Gateway-Modul SIM.730.99 Automotive-CAN

20.1. Allgemeines

Einige Anwendungen erfordern einen Datenaustausch zwischen Steuerungen der PLM700-Familie mit Steuergeräten aus dem Automotive-Bereich, z.B. zur Steuerung von BHKW-Verbrennungsmotoren oder bei Verwendung der PLM-Steuerungen als Zusatzgerät auf Kraftfahrzeugen. Die Verbindung erfolgt in diesem Fall über den CAN-Bus des Motorsteuergeräts, der hier als Automotive-CAN bezeichnet wird.

Da auf dem Automotive-CAN, trotz ähnlicher CAN-Hardware, andere Baudraten, andere Message-Formate und andere Protokolle als auf den CAN-Bussen der Steuerung (CANopen) verwendet werden, muss ein Gateway zwischen beiden Bussen eingesetzt werden. Hierfür steht das Gateway-Modul SIM.730.99 zur Verfügung.

Das Modul SIM.730.99 besitzt dazu zwei unabhängige CAN-Schnittstellen, CAN 0 und CAN 1. CAN 0 wird mit dem System-CAN der Steuerung verbunden, CAN 1 mit dem Automotive-CAN. Während auf dem System-CAN (CAN 0) die Kommunikation mit der Steuerung unter Verwendung von 11-Bit-Identifiern mit dem Protokoll CANopen läuft, werden auf dem Automotive-CAN üblicherweise andere Protokolle, ggf. eine andere Baudrate und ggf. 29-Bit-Identifizier verwendet.

Der Begriff "Sende-Message" bezeichnet hier Nachrichten von der PLM-Steuerung auf den Automotive-CAN, der Begriff "Empfangs-Message" Nachrichten vom Automotive-CAN, die auf der Steuerung ausgewertet werden.

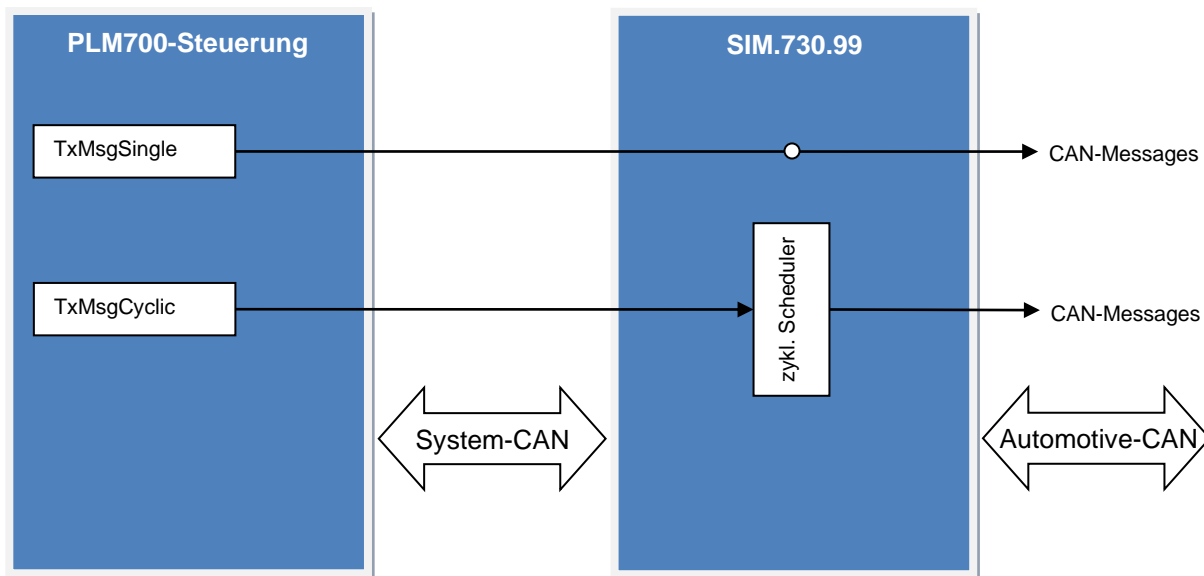
Die Verwendung des Moduls erfordert die im Lieferumfang der Steuerung enthaltene CoDeSys-Bibliothek `PLM_SIM73099_CAN.lib`.

20.2. Spezifikation

- Senden und Empfangen von 11-Bit- und 29-Bit-Messages auf dem Automotive-CAN
- Senden von zyklischen Messages mit 0,1 ms Zeitauflösung
- Bis zu 32 Filter für Empfangsnachrichten

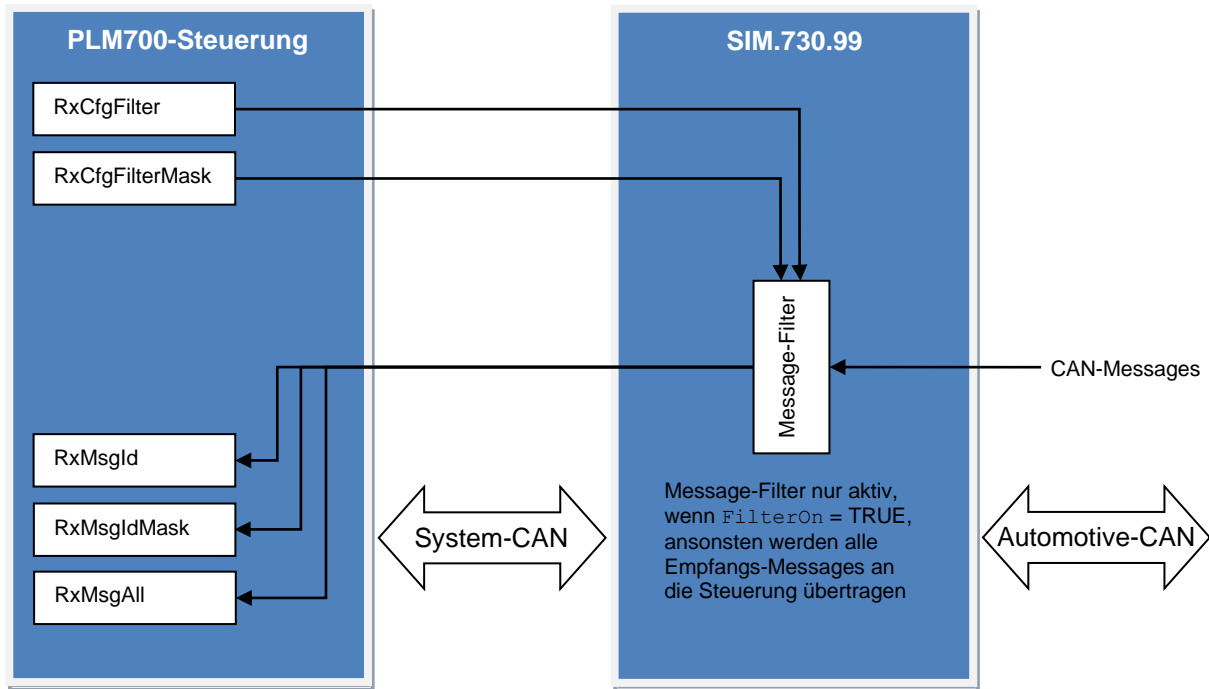
20.3. Funktionsweise

20.3.1. Senden von CAN-Messages auf den Automotive-CAN



CAN-Messages auf dem Automotive-CAN können entweder als Einzel-Message oder als zyklische Message erzeugt werden. Im zweiten Fall wird das millisekundengenaue Sende-Timing vom zyklischen Scheduler im SIM.730.99 sichergestellt.

20.3.2. Empfang von CAN-Messages vom Automotive-CAN



Die Einrichtung von Message-Filtern ist (meistens) notwendig, um den System-CAN zu entlasten. Da auf dem Automotive-CAN häufig mit vielen zyklischen Messages bei höheren Baudraten gearbeitet wird und eine einzige CAN-Message des Automotive-CANs in mehrere CAN-Messages auf dem System-CAN umgesetzt wird, kann es sonst zu einer Überlastung des System-CANs mit unkontrolliertem Datenverlust kommen.

Üblicherweise existiert zu jedem Baustein `RxMsgId` ein passender Baustein `RxCfgFilter`.

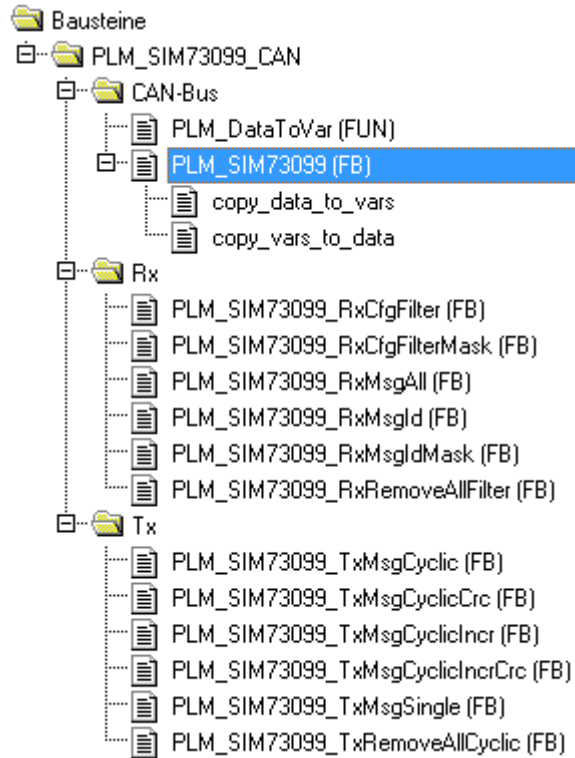
Zusätzlich zur Einrichtung der Message-Filter mit den entsprechenden Bausteinen müssen diese mit dem Eingang `FilterOn` des Bausteins `PLM_SIM73099` eingeschaltet werden.

20.4. Benötigte Bibliotheken

Plm_SIM73099_CAN_v20160906.lib (oder spätere Version)

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

20.4.1. Übersicht über die Bibliotheksbausteine



Für jedes am System-CAN angeschlossene Modul SIM.730.99 muss zunächst ein Kommunikationsbaustein `PLM_SIM73099` eingerichtet werden, der ständig zyklisch aufzurufen ist.

Dieser Baustein liefert einen Ausgang `Handle`, der zur Verknüpfung mit den angehängten Tx- oder Rx-Bausteinen dient.

Hinweis: Bedingt durch die komplexe interne Funktionsweise der Bibliothek kann es passieren, dass nach einem Online-Change mit Änderungen von Sende- oder Empfangsbausteinen der Empfang einzelner Nachrichten nicht mehr funktioniert. In diesem Fall schafft ein "Reset Kalt" auf der Steuerung Abhilfe.

20.4.2. PLM_SIM73099

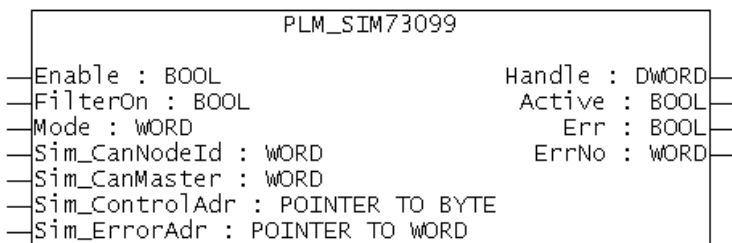


Abb. 20-1: Funktionsblock `PLM_SIM73099`

Input-Parameter:		
Enable	BOOL	TRUE ⇒ Baustein aktivieren
FilterOn	BOOL	TRUE ⇒ Angehängte Empfangsfilterbausteine (RxCfgFilter und RxCfgFilterMask) sind aktiv, FALSE ⇒ Alle Empfangsfilterbausteine sind inaktiv, d.h. alle Nachrichten vom Automotive-CAN werden empfangen (kann zur Überlastung des System-CANs führen)
Mode	WORD	0 ⇒ Bausteine vom Typ RxCfgFilter und RxCfgFilterMask sind explizit im Programm

		<p>angelegt, oder es werden keine Filter verwendet</p> <p>1/2 ⇒ Für jeden angehängten RxMsgId- und RxMsgIdMask-Baustein wird automatisch ein passendes Empfangsfilter im SIM.730.99 eingerichtet, zusätzlich muss der Eingang FilterOn = TRUE eingeschaltet werden (s.o.)</p> <p>1 ⇒ AlwaysUpdate = TRUE,</p> <p>2 ⇒ AlwaysUpdate = FALSE (vgl. Abschnitte 20.4.6 und 20.4.7)</p>
Sim_CanNodeId	WORD	CAN-NodeID des SIM.730.99 aus der Steuerungskonfiguration
Sim_CanMaster	WORD	Index des CAN-Masters, 0 oder 1
Sim_ControlAdr	POINTER TO BYTE	Adresse ADR () des ersten %QB des SIM.730.99 aus der Steuerungskonfiguration
Sim_ErrorAdr	POINTER TO WORD	Adresse ADR () des ersten %IB des SIM.730.99 aus der Steuerungskonfiguration
Output-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit Rx- oder Tx-Bausteinen
Active	BOOL	TRUE ⇒ Baustein und Datenaustausch mit SIM.730.99 sind aktiv
Err	BOOL	TRUE ⇒ Fehler aufgetreten, die Fehlernummer steht in ErrNo
ErrNo	WORD	<p>0 = kein Fehler</p> <p>1 = Kann System-CAN-Kommunikation zum Modul nicht herstellen (PLM_SIM73099ERR_NOCOM)</p> <p>2 = Keine Rx- oder Tx-Bausteine mit Handle verknüpft (PLM_SIM73099ERR_NOFUNCBLOCKS)</p> <p>3 = Sim_ControlAdr nicht angegeben oder Null (PLM_SIM73099ERR_ILLCtrlADR)</p> <p>4 = Sim_ErrorAdr nicht angegeben oder Null (PLM_SIM73099ERR_ILLErrADR)</p> <p>5 = Sim_CanNodeID nicht in Steuerungskonfiguration gefunden (PLM_SIM73099ERR_MODNOTLISTED)</p> <p>6 = CAN-Status des Moduls ist ungleich fünf, d.h. das Modul ist nicht am System-CAN angeschlossen oder Node-Guarding-Fehler (PLM_SIM73099ERR_MODNODEGUARD)</p> <p>7 = Das Modul meldet, dass es neu initialisiert werden muss, dies wird von der Bibliothek automatisch durchgeführt (PLM_SIM73099ERR_MODNEEDINIT)</p> <p>8/9/10/13 = Interner Protokollfehler bei Kommunikation mit SIM.730.99</p> <p>11/12 = Ungültige Angabe von CanMsgLen und/oder CanMsgData bei einem TxMsgSingle- oder TxMsgCyclic-Baustein</p> <p>13 = Interner Protokollfehler bei Kommunikation mit SIM.730.99</p> <p>14 = Das Modul meldet, dass versucht wurde, zuviele zyklische Tx-Messages einzurichten (PLM_SIM73099ERR_TOOMANYCYCLIC)</p> <p>15 = Das Modul meldet, dass versucht wurde, zuviele Empfangsfilter einzurichten (PLM_SIM73099ERR_TOOMANYFILTERS)</p> <p>16 = Ein RxMsgAll-Baustein kann nicht</p>

		zusammen mit RxCfgFilter-Bausteinen verwendet werden (PLM_SIM73099ERR_CANTUSERXALLWITHFILTERS)
--	--	---

Der Baustein PLM_SIM73099 ist zyklisch aufzurufen.

Der Ausgang Handle liefert einen Wert, der zum Anhängen von Rx- und Tx-Bausteinen benötigt wird.

20.4.3. PLM_SIM73099_RxMsgId

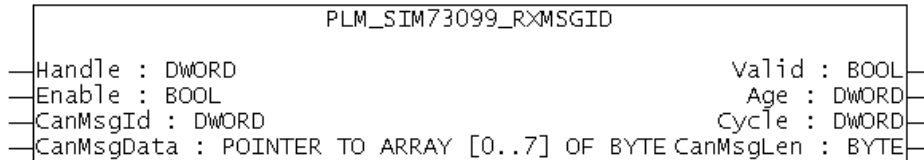


Abb. 20-2: Funktionsblock PLM_SIM73099_RxMsgId

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇒ Baustein aktivieren
CanMsgId	DWORD	CAN-Identifizier der zu empfangenen Message
CanMsgData	POINTER TO ARRAY [0..7] OF BYTE	Adresse ADR() eines Arrays für die max. 8 zu empfangenen CAN-Datenbytes
Output-Parameter:		
Valid	BOOL	CAN-Message mit angegebener CanMsgId wurde tatsächlich empfangen
Age	DWORD	Anzahl Millisekunden, seitdem die Message zuletzt empfangen wurde
Cycle	DWORD	Anzahl Millisekunden zwischen den beiden zuletzt empfangenen Messages
CanMsgLen	BYTE	Tatsächliche Anzahl Datenbytes, die in CanMsgData eingetragen wurde (0...8)

Der Baustein PLM_SIM73099_RxMsgId ist mit dem Eingang Handle an einen Baustein vom Typ PLM_SIM73099 anzuhängen.

An den Eingang CanMsgId wird die (normalerweise konstante) Message-ID der CAN-Message gelegt, die vom Automotive-CAN empfangen werden soll. Der Identifizier kann 11 oder 29 gültige Bits haben und wird üblicherweise in Hexdezimal angegeben, z.B. 16#702 für einen 11-Bit-Identifizier oder 16#15061234 für einen 29-Bit Identifizier.

Der Ausgang Valid wird TRUE beim Empfang einer passenden CAN-Message. Er fällt nach 9999 ms (ca. 10 Sekunden) auf FALSE zurück (Timeout). Die Timeout-Zeit ist eine lokale DWORD-Variable im Baustein PLM_SIM73099_RxMsgId, die ggf. mittels Schreibzugriff über einen Pointer geändert werden kann.

Die Angaben in Valid, Age und Cycle können ungültig sein, wenn ein zugehöriger Message-Filter-Baustein mit AlwaysUpdate = FALSE konfiguriert ist.

20.4.4. PLM_SIM73099_RxMsgIdMask

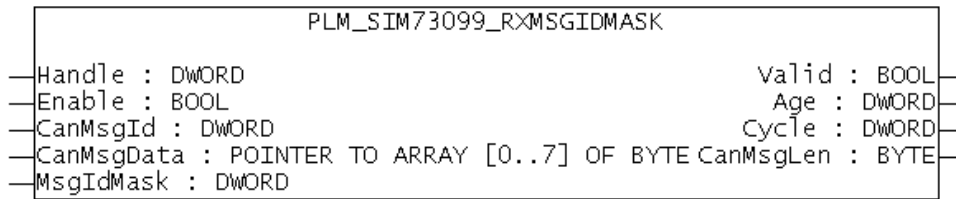


Abb. 20-3: Funktionsblock PLM_SIM73099_RxMsgIdMask

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇔ Baustein aktivieren
CanMsgId	DWORD	CAN-Identifizier der zu empfangenen Message
CanMsgData	POINTER TO ARRAY [0..7] OF BYTE	Adresse ADR () eines Arrays für die max. 8 zu empfangenen CAN-Datenbytes
MsgIdMask	DWORD	Filtermaske für die CanMsgId, 16#ffffffff = Maske unwirksam.
Output-Parameter:		
Valid	BOOL	CAN-Message wurde tatsächlich empfangen
Age	DWORD	Anzahl Millisekunden, seitdem die Message zuletzt empfangen wurde
Cycle	DWORD	Anzahl Millisekunden zwischen den beiden zuletzt empfangenen Messages
CanMsgLen	BYTE	Tatsächliche Anzahl Datenbytes, die in CanMsgData eingetragen wurde (0..8)

Der Baustein PLM_SIM73099_RxMsgId ist mit dem Eingang Handle an einen Baustein vom Typ PLM_SIM73099 anzuhängen.

Wenn ein Rx-Filter mit Maske verwendet wird, kann dieser Baustein verwendet werden, um alle Messages zu empfangen, die das entsprechende Filter passieren konnten.

An den Eingang CanMsgId wird die (normalerweise konstante) Message-ID der CAN-Message gelegt, die vom Automotive-CAN empfangen werden soll. Die ankommende Message-ID wird zuerst mit der MsgIdMask UND-verknüpft (0-Bits in der Maske werden in der Message-ID gelöscht) und dann mit der angegebenen CanMsgId verglichen.

Der Identifizier kann 11 oder 29 gültige Bits haben und wird üblicherweise in Hexdezimal angegeben, z.B. 16#702 für einen 11-Bit-Identifizier oder 16#15061234 für einen 29-Bit Identifizier.

Der Ausgang Valid wird TRUE beim Empfang einer passenden CAN-Message. Er fällt nach 9999 ms (ca. 10 Sekunden) auf FALSE zurück (Timeout). Die Timeout-Zeit ist eine lokale DWORD-Variable im Baustein PLM_SIM73099_RxMsgId, die ggf. mittels Schreibzugriff über einen Pointer geändert werden kann.

Die Angaben in Valid, Age und Cycle können ungültig sein, wenn ein zugehöriger Message-Filter-Baustein mit AlwaysUpdate = FALSE konfiguriert ist.

20.4.5. PLM_SIM73099_RxMsgAll

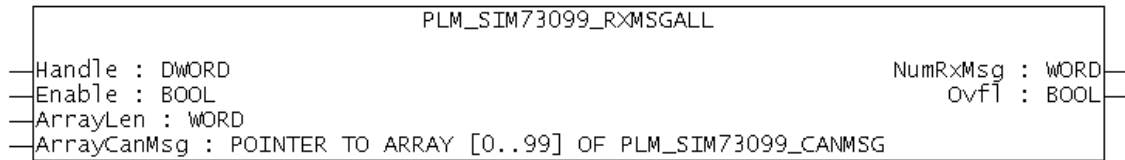


Abb. 20-4: Funktionsblock PLM_SIM73099_RxMsgAll

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇒ Baustein aktivieren
ArrayLen	WORD	Max. Anzahl der Elemente im Empfangsarray am Eingang ArrayCanMsg
ArrayCanMsg	POINTER TO ARRAY [0..99] OF PLM_SIM73099_CANMSG	Adresse ADR() eines Arrays für die zu empfangenen CAN-Messages
Output-Parameter:		
NumRxMsg	WORD	Anzahl der im letzten Zyklus empfangenen CAN-Messages
Ovfl	BOOL	TRUE ⇒ Es wurden mehr als ArrayLen CAN-Messages empfangen

Der Baustein PLM_SIM73099_RxMsgAll ist mit dem Eingang Handle an einen Baustein vom Typ PLM_SIM73099 anzuhängen.

Dieser Baustein empfängt alle Messages, die nicht einem Baustein PLM_SIM73099_RxMsgId zugeordnet werden können. Wenn dieser Baustein verwendet wird, dürfen keine Filter konfiguriert sein.

Am Eingang ArrayCanMsg wird die Adresse eines Arrays angegeben, in dem die empfangenen Messages gespeichert werden. Die Anzahl der Messages wird in NumRxMsg zurückgeliefert.

Das Array wird beim Aufruf von PLM_SIM73099 zunächst jedesmal gelöscht, so dass es nach Aufruf nur die seit dem letzten Aufruf neu empfangenen CAN-Messages enthält. Benötigte Daten müssen also vom Anwender nach Aufruf des Bausteins PLM_SIM73099 wegekopiert werden.

Das Empfangsarray wird z.B. als

```
rxAllArray: ARRAY [0..9] OF PLM_SIM73099_CANMSG
```

deklariert. In diesem Fall erhält der Eingang ArrayLen den Wert 10. Der Strukturtyp PLM_SIM73099_CANMSG zum Speichern einer CAN-Message ist wie folgt deklariert:

```
TYPE PLM_SIM73099_CANMSG :
STRUCT
    MsgId: DWORD;
    MsgLen: BYTE;
    MsgData: ARRAY [0..7] OF BYTE;
END_STRUCT
END_TYPE
```

20.4.6. PLM_SIM73099_RxCfgFilter

PLM_SIM73099_RXCFGFILTER
—Handle : DWORD
—Enable : BOOL
—AlwaysUpdate : BOOL
—CanMsgId : DWORD

Abb. 20-5: Funktionsblock PLM_SIM73099_RxCfgFilter

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇔ Filter aktivieren
AlwaysUpdate	BOOL	FALSE ⇔ Übertragung nur bei Datenänderung (Entlastung des Systembus'), TRUE ⇔ neue CAN-Message immer übertragen
CanMsgId	DWORD	CAN-Identifizier der zu empfangenen Message

Der Baustein PLM_SIM73099_RxCfgFilter ist mit dem Eingang Handle an einen Baustein vom Typ PLM_SIM73099 anzuhängen.

Dieser Baustein konfiguriert ein Empfangsfilter im Modul SIM.730.99 und öffnet es für die angegebene CanMsgId, d.h. die Nachrichten mit dieser Message-ID können das Empfangsfilter des Moduls passieren und werden auf dem System-CAN an die Steuerung weitergeleitet.

Damit die angegebenen Filter wirksam werden, muss zusätzlich am Baustein PLM_SIM73099 der Eingang FilterOn auf TRUE gesetzt werden.

Die Konfiguration von Empfangsfiltern kann notwendig sein, um den System-CAN vor Überlastung zu schützen.

Normalerweise existiert zu jedem Baustein PLM_SIM73099_RxCfgFilter ein passender Baustein PLM_SIM73099_RxMsgId, der die Messages auf der Steuerung entgegennimmt.

20.4.7. PLM_SIM73099_RxCfgFilterMask

PLM_SIM73099_RXCFGFILTERMASK
—Handle : DWORD
—Enable : BOOL
—AlwaysUpdate : BOOL
—CanMsgId : DWORD
—MsgIdMask : DWORD

Abb. 20-6: Funktionsblock PLM_SIM73099_RxCfgFilterMask

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇔ Filter aktivieren
AlwaysUpdate	BOOL	FALSE ⇔ Übertragung nur bei Datenänderung (Entlastung des Systembus'), TRUE ⇔ neue CAN-Message immer übertragen
CanMsgId	DWORD	CAN-Identifizier der zu empfangenen

		Message
MsgIdMask	DWORD	Filtermaske für die CanMsgId, 16#ffffffff = Maske unwirksam.

Der Baustein `PLM_SIM73099_RxCfgFilterMask` ist mit dem Eingang `Handle` an einen Baustein vom Typ `PLM_SIM73099` anzuhängen.

Dieser Baustein konfiguriert ein Empfangsfilter im Modul SIM.730.99. Ankommende Message-IDs werden zuerst mit der `MsgIdMask` UND-verknüpft (0-Bits in der Maske werden in der Message-ID gelöscht) und dann mit der angegebenen `CanMsgId` verglichen. Für zutreffende Nachrichten wird das Filter geöffnet, d.h. Nachrichten mit dieser Message-ID können das Empfangsfilter des Moduls passieren und werden auf dem System-CAN an die Steuerung weitergeleitet.

Damit die angegebenen Filter wirksam werden, muss zusätzlich am Baustein `PLM_SIM73099` der Eingang `FilterOn` auf `TRUE` gesetzt werden.

Die Konfiguration von Empfangsfiltern kann notwendig sein, um den System-CAN vor Überlastung zu schützen.

Normalerweise existiert zu jedem Baustein `PLM_SIM73099_RxCfgFilterMask` ein passender Baustein `PLM_SIM73099_RxMsgIdMask`, der die Messages auf der Steuerung entgegennimmt.

20.4.8. PLM_SIM73099_TxMsgSingle

PLM_SIM73099_TXMSGSSINGLE	
—Handle	: DWORD
—Enable	: BOOL
—CanMsgId	: DWORD
—CanMsgLen	: BYTE
—CanMsgData	: POINTER TO ARRAY [0..7] OF BYTE

Abb. 20-7: Funktionsblock `PLM_SIM73099_TxMsgSingle`

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein <code>PLM_SIM73099</code>
Enable	BOOL	Steigende Flanke FALSE ⇒ TRUE bewirkt einen einzelnen Versand der angegebenen CAN-Message
CanMsgId	DWORD	CAN-Identifizier der zu sendenden Message
CanMsgLen	BYTE	Anzahl der Datenbytes der CAN-Message (0...8)
CanMsgData	POINTER TO ARRAY [0..7] OF BYTE	Adresse <code>ADR()</code> eines Arrays mit den max. 8 Datenbytes der CAN-Message

Der Baustein `PLM_SIM73099_TxMsgSingle` ist mit dem Eingang `Handle` an einen Baustein vom Typ `PLM_SIM73099` anzuhängen.

Dieser Baustein ermöglicht das Senden von CAN-Messages auf den Automotive-CAN. Die Messages werden in diesem Fall einzeln durch die Steuerung ausgelöst.

20.4.9. PLM_SIM73099_TxMsgCyclic

```

    PLM_SIM73099_TXMSGCYCLIC
—Handle : DWORD
—Enable : BOOL
—CanMsgId : DWORD
—CanMsgLen : BYTE
—CanMsgData : POINTER TO ARRAY [0..7] OF BYTE
—Cycle : WORD
    
```

Abb. 20-8: Funktionsblock PLM_SIM73099_TxMsgCyclic

Input-Parameter:		
Handle	DWORD	Handle zur Verknüpfung mit einem Baustein PLM_SIM73099
Enable	BOOL	TRUE ⇒ die angegebene CAN-Message wird zyklisch gesendet, FALSE ⇒ Abschalten der zyklischen Message
CanMsgId	DWORD	CAN-Identifizier der zu sendenden Message
CanMsgLen	BYTE	Anzahl der Datenbytes der CAN-Message (0...8)
CanMsgData	POINTER TO ARRAY [0..7] OF BYTE	Adresse ADR() eines Arrays mit den max. 8 Datenbytes der CAN-Message
Cycle	WORD	Sendezykluszeit in Millisekunden (1...65535)

Der Baustein PLM_SIM73099_TxMsgCyclic ist mit dem Eingang Handle an einen Baustein vom Typ PLM_SIM73099 anzuhängen.

Dieser Baustein ermöglicht das regelmäßige, zyklische Senden von CAN-Messages auf den Automotive-CAN. Die Messages werden in diesem Fall zyklisch durch das Modul SIM.730.99 ausgelöst. Die zeitliche Auflösung beträgt modulintern ca. 0,1 ms. Änderungen der Datenbytes werden in der nächsten gesendeten Message übernommen, ohne dabei die vorgegebene Zykluszeit zu verändern. Übliche Zykluszeiten sind z.B. 20 ms oder 100 ms.

CAN-Messages mit einer präzisen zeitlichen Wiederkehr werden von vielen Automotive-Steuergeräten erwartet. Abweichungen von der erwarteten Zykluszeit können dabei als Fehler interpretiert werden.

20.5. Programmbeispiel

(in Vorbereitung)

20.6. Fehlersuche

Bei Empfangsfehlern ist zunächst zu prüfen, ob der Baustein PLM_SIM73099 überhaupt Daten vom Modul empfängt. Dazu ist zunächst der Ausgang ErrNo zu prüfen, dieser muss den Wert 0 liefern. Desweiteren sind die Eingänge Sim_CanNodeId, Sim_CanMaster, SimControlAdr und SimErrorAdr zu überprüfen.

Nach Online-Changes sollte ein "Reset Kalt" der Steuerung versucht werden.

Im Baustein PLM_SIM73099 existiert eine interne WORD-Variable rawRxMsgCnt, die um eins inkrementiert wird, sobald eine CAN-Nachricht vom Modul empfangen wird. Außerdem existieren im Modul ein SDO 0x3010 sub 0, der um eins inkrementiert wird, sobald eine neue Message vom Automotive-CAN empfangen wird und ein SDO 0x3011 sub 0, der um eins inkrementiert wird, sobald diese Message vom Modul auf dem System-CAN an die PLM-Steuerung weitergeleitet wird.

Wenn der SDO 0x3010 sub 0 nicht zählt, sind entweder keine Messages auf dem Automotive-CAN vorhanden oder der Automotive-CAN ist falsch angeschlossen

(Leitungen falsch, fehlender Abschlusswiderstand) oder der im Modul konfigurierte CAN-Message-Typ (11/29 Bit) passt nicht.

Wenn der SDO 0x3010 sub 0 zählt, aber der SDO 0x3011 sub 0 nicht, sind die Filter im Modul evtl. falsch konfiguriert.

Wenn der SDO 0x3011 sub 0 zählt, aber die WORD-Variable `rawRxMsgCnt` im Baustein `PLM_SIM73099` nicht, so kann das daran liegen, dass zwar Nachrichten auf dem Automotive-CAN erkannt werden, der Inhalt sich jedoch nicht ändert und die zugehörigen Filter mit `AlwaysUpdate = FALSE` konfiguriert sind.

Wenn der SDO 0x3011 sub 0 zählt, aber `rawRxMsgCnt` immer den Wert 0 hat, ist möglicherweise die Übertragung zwischen Modul und PLM-Steuerung auf dem System-CAN gestört. Wenn der Moduls-CAN-Status 5 ist (d.h. das Node-Guarding arbeitet korrekt und die Modul-LED ist dauergrün) kann es trotzdem an einer Überlastung des System-CAN liegen. In diesem Fall sind geeignete Message-Filterbausteine zu konfigurieren bzw. ggf. die Baudrate auf dem System-CAN zu erhöhen.

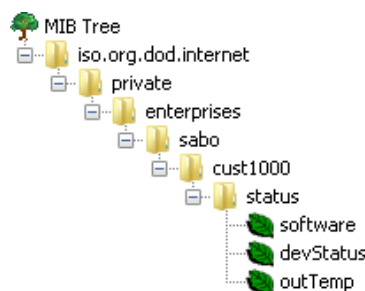
21. SNMP

21.1. Allgemeines

Das Simple Network Management Protocol (SNMP) in den aktuell relevanten Versionen v2c und v3 ist ein offener Kommunikationsstandard zur Verwaltung und Überwachung von Netzwerkkomponenten wie Switches, Routern, Server-Rechnern u.ä. Die zu überwachenden Geräte stellen dazu eine Reihe von Datenobjekten über einen sog. SNMP-Agent zur Verfügung.

Die Abfrage erfolgt durch SNMP-Clients, z.B. eine Managementkonsole auf einem PC. Es existieren zahlreiche, zum Teil kostenlose Applikationen, die eine Verwendung von SNMP auch außerhalb von Zwecken der Netzwerküberwachung interessant machen.

Die Datenobjekte eines Geräts sind nach einem weitgehend standardisierten Schema streng hierarchisch in einer Baumstruktur strukturiert. Diese Struktur wird als MIB (Management Information Base) bezeichnet. Die folgende Abbildung veranschaulicht beispielhaft die Baumstruktur einer MIB:



Jedes Datenobjekt innerhalb der MIB hat eine bestimmte Objekt-ID (OID), mit der es eindeutig referenziert werden kann. Eine OID besteht aus einer Folge von durch Punkten getrennten Zahlen, z.B.

```
.1.3.6.1.4.1.46984.1000.1.1
```

Die Zahlen in einer OID können einzeln durch symbolische Namen ersetzt werden, z.B. für die obige OID

```
.iso.org.dod.internet.private.enterprises.sabo.
cust1000.status.software
```

Eigene Geräte, deren Datenobjekte keinem vordefinierten Standard entsprechen, können unterhalb von `.1.3.6.1.4.1` (`.iso.org.dod.internet.private.enterprises`) angehängt werden. Dies setzt eine von der IANA (Internet Assigned Numbers Authority, <https://www.iana.org/>) vergebene Herstellerkennung voraus. Die Zahl 46984 ist die durch die IANA vergebene Herstellerkennung der SABO Elektronik GmbH. Unterhalb dieser Kennung legt SABO eine eigene MIB-Struktur fest. An unsere Kunden ohne eigene Herstellerkennung vergeben wir in Absprache die OIDs `sabo.1000` bis `sabo.9999`. Die Kunden können unterhalb ihrer jeweiligen Kennung wiederum eigene MIB-Strukturen festlegen.

Die Zuordnung zwischen Zahlen und Namen in der MIB erfolgt durch sog. MIB-Dateien. Hersteller von SNMP-fähigen Geräten liefern zu ihren Geräten passende MIB-Dateien, die die Struktur des eigenen Teilbaums, die Position im Gesamtbaum und die vorhandenen eigenen Datenobjekte beschreiben.

Die SNMP-Implementierung auf Steuerungen der Serie PLM 700-A ist in der Lage, während der Konfiguration der Datenobjekte durch das IEC-Programm automatisch eine passende MIB-Datei zu erzeugen, die über FTP von der Steuerung heruntergeladen werden kann.

21.2. Spezifikation

- SNMP-Agent (Server) für Systeme der Reihe PLM700-A
- Protokollstandards SNMP v2c und v3

- Kommunikation über UDP, Standard-Port 161
- Konfiguration der MIB und des Agents durch das IEC-Programm
- Beliebige Anzahl von Datenobjekten
- Versenden von Traps
- Automatische Erzeugung der MIB-Datei

21.3. Installation auf der Steuerung

Zur Installation des SNMP-Agents auf der Steuerung ist das Einspielen eines Update-Pakets über das WebConfig (Menü *System-Updates*) erforderlich. Das Update-Paket ist auf Anfrage bei uns erhältlich. Außerdem muss der SNMP-Dienst im WebConfig im Menü *Services* aktiviert werden.

Anschließend ist ein Neustart der Steuerung erforderlich.

Die weitere Konfiguration erfolgt durch das IEC-Programm unter Verwendung der aktuellen CoDeSys-Bibliothek `PLM_SNMP_v.....lib`.

Die Bausteine der Bibliothek müssen einmalig (z.B. in `InitOnce()`) in bestimmter Reihenfolge aufgerufen werden und konfigurieren dabei die MIB und alle wesentlichen Parameter des SNMP-Agents.

Die Konfiguration wird im Folgenden beschrieben und in einem Programmbeispiel in Abschnitt 21.7 demonstriert.

21.4. Konfiguration des SNMP-Agents

Die Konfiguration des SNMP-Agents startet durch Aufruf des Funktionsbausteins `Plm_SNMP_Init()`.

21.4.1. Plm_SNMP_Init()

```

PLM_SNMP_INIT
—SysDescr : STRING(80)
—SysLocation : STRING(80)
—SysContact : STRING(80)
—SysObjId : STRING(80)
—Port : WORD
—MibName : STRING(80)
—MibFilename : STRING(80)
    
```

Abb. 21-1: Funktionsbaustein `Plm_SNMP_Init`

Input-Parameter:		
SysDescr	STRING	Ein beschreibender Text für die Anlage, der über SNMP unter der OID <code>.1.3.6.1.2.1.1.1</code> abgerufen werden kann
SysLocation	STRING	Ein beschreibender Text für den Standort der Anlage, der über SNMP unter der OID <code>.1.3.6.1.2.1.1.6</code> abgerufen werden kann
SysContact	STRING	Ein beschreibender Text mit einer Kontaktangabe für die Anlage, der über SNMP unter der OID <code>.1.3.6.1.2.1.1.4</code> abgerufen werden kann
SysObjId	STRING	Die Basis-OID, unter der die MIB dieser Anlage eingeordnet ist
Port	WORD	UDP-Port des SNMP-Agents, Standard: 161
MibName	STRING	Symbolische Bezeichnung der MIB zur Verwendung innerhalb der MIB-Datei
MibFilename	STRING	Dateiname zum autom. Erzeugen einer MIB-Datei auf der Steuerung. Falls kein Dateiname angegeben ist (Leerstring), wird keine MIB-Datei erzeugt.

Beispiel:

```
snmpAgent (
  SysDescr      := 'YornbloSch Heater Controller',
  SysLocation   := 'Muwumi Building, 1st floor',
  SysContact    := 'service@yornbloSch.com',
  SysObjId      := '.1.3.6.1.4.1.46984.1000',
  Port          := 161,
  MibName       := 'YornbloSchHeaterMib',
  MibFilename   := 'a/YornbloSchHeater.mib'
);
```

Nach der Konfiguration des Agents kann die erzeugte MIB-Datei `YornbloSchHeater.mib` von FTP-Laufwerk `a/` der Steuerung heruntergeladen werden.

Die `SysObjId` stellt die Basis-OID der MIB für diese Anlage dar. Ihre Einordnung in die globale MIB wird durch entsprechende Header-Zeilen in der MIB-Datei festgelegt. Diese geschieht z.B. wie folgt:

```
PLM_SNMP_mibimports( Import := 'ENTERPRISES FROM SNMPv2-SMI' );
PLM_SNMP_mibdef( Def := 'sabo OBJECT IDENTIFIER ::= { enterprises 46984 }' );
PLM_SNMP_mibdef( Def := 'cust1000 OBJECT IDENTIFIER ::= { sabo 1000 }' );
PLM_SNMP_mibdef( Def := 'status OBJECT IDENTIFIER ::= { cust1000 1 }' );
PLM_SNMP_mibdef( Def := 'control OBJECT IDENTIFIER ::= { cust1000 2 }' );
```

Der Aufruf von `PLM_SNMP_mibimports()` fordert das Einbinden einer Standard-MIB an, auf der die vorliegende anlagenspezifische MIB basiert. In diesem Fall wird die Standard-MIB *ENTERPRISES* eingebunden, die die Definition bis einschließlich `.1.3.6.1.4.1` (`.iso.org.dod.internet.private.enterprises`) enthält.

Die anschließenden Aufrufe von `PLM_SNMP_mibdef()` definieren rekursiv die Struktur der anlagenspezifischen MIB, in diesem Fall:

```
sabo          = enterprises.46984,
cust1000      = sabo.1000,
status        = cust1000.1 und
control       = cust1000.2
```

Unterhalb von `status` und `control` erfolgt die weitere MIB-Definition durch das Anlegen von Datenobjekten (siehe Abschnitt 21.6).

21.5. Zugriffsrechte

Um auf die Datenobjekte eines SNMP-Agents zugreifen zu können, muss ein Client die entsprechende Zugriffsberechtigung besitzen. Für die diesbezügliche Konfiguration des Agents stehen die *Communities* gemäß SNMP v2c und das *User Based Security Model* gemäß SNMP v3 zur Verfügung.

Von der Verwendung des Community-Modells gemäß SNMP v2c ist aus Sicherheitsgründen dringend abzuraten. Allerdings ist dieses Modell sehr weit verbreitet und wird deshalb auf PLM 700-Systemen unterstützt, da die Auswahl des Zugriffmodells häufig mit Rücksicht auf die Möglichkeiten der in vielen Fällen vorgegebenen SNMP-Clients erfolgen muss.

21.5.1. SNMP v2c Communities

SNMP v2c unterscheidet lediglich zwischen einer Gruppe von Usern mit Nur-Lese-Zugriff und einer weiteren Gruppe mit Lese-und-Schreibzugriff. Diese Gruppen werden als *Communities* bezeichnet.

Das Anlegen von SNMP v2c Communities geschieht mit folgenden Bibliotheksfunktionsaufrufen:


```
PLM_SNMP_V2_User( Name := 'Public', RW := FALSE );
PLM_SNMP_V2_User( Name := 'Private', RW := TRUE );
```

Im ersten Aufruf wird eine Community *Public* eingerichtet, die Nur-Lese-Zugriff auf alle Datenobjekte hat. Im zweiten Aufruf wird zusätzlich eine Community *Private* eingerichtet mit Schreib- und Lese-Berechtigung. Groß-/Kleinschreibung bei den Community-Namen wird unterschieden.

Die Verwendung von SNMP v2c Communities ist aus heutiger Sicht sicherheitstechnisch nur innerhalb von geschlossenen, privaten Netzwerken vertretbar, da der Zugriff unverschlüsselt und ohne Authentifizierung erfolgt.

Auf keinen Fall dürfen Anlagen bei Verwendung von SNMP v2c Communities direkt mit dem Internet verbunden werden, insbesondere wenn Anlagensteuerfunktionen (Schreibzugriffe) über SNMP zugelassen sind.

21.5.2. SNMP v3 User Based Security Model

SNMP v3 bietet Möglichkeiten für verbesserten Zugriffsschutz durch Einsatz von Protokollverschlüsselung und starker Authentifizierung. Dabei können drei Stufen konfiguriert werden:

1. Keine Authentifizierung, keine Verschlüsselung (NoAuthNoPriv)
2. Authentifizierung, aber keine Verschlüsselung (AuthNoPriv)
3. Authentifizierung und Verschlüsselung (AuthPriv)

Die beiden ersten Varianten sind aus verschiedenen, u.a. historischen und Kompatibilitätsgründen verfügbar, sollten jedoch aus Sicherheitsgründen nicht verwendet werden.

Für die Authentifizierung stehen die beiden Verfahren MD5 und SHA zur Verfügung, für die Verschlüsselung DES und AES. Wenn möglich, sollte den beiden moderneren Verfahren SHA und AES der Vorzug gegeben werden.

Die Einrichtung zweier SNMP v3-User mit Authentifizierung und Verschlüsselung geschieht mit folgenden Bibliotheksfunktionsaufrufen:

```
PLM_SNMP_V3_User(
  Name      := 'paul',
  LoginPass := 'paulpassw',
  Auth      := 'SHA',          (* '' or 'MD5' or 'SHA' *)
  Priv      := 'AES',          (* '' or 'DES' or 'AES' *)
  RW        := FALSE
);

PLM_SNMP_V3_User(
  Name      := 'greg',
  LoginPass := 'gregpassw',
  Auth      := 'SHA',          (* '' or 'MD5' or 'SHA' *)
  Priv      := 'AES',          (* '' or 'DES' or 'AES' *)
  RW        := TRUE
);
```

Im ersten Aufruf wird ein User *paul* mit Passwort *paulpassw* eingerichtet, der Nur-Lese-Zugriff auf alle Datenobjekte hat. Im zweiten Aufruf wird zusätzlich ein User *greg* mit Passwort *gregpassw* eingerichtet mit Schreib- und Lese-Berechtigung.

Das bei `LoginPass` angegebene Passwort wird sowohl für die Authentifizierung als auch für die Verschlüsselung verwendet.

21.6. SNMP-Datenobjekte

Das Anlegen der Datenobjekte wird mit der Bibliotheksfunktion

```
PLM_SNMP_Baseoid()
```

eingeleitet. Hierdurch wird die Position der folgenden Datenobjekte innerhalb der Hierarchie der MIB definiert.

Die folgenden Datenobjekte werden mit den drei Bibliotheksbausteinen

```
PLM_SNMP_DataLine()
PLM_SNMP_DataLineString()
PLM_SNMP_DataLineReal()
```

angelegt. Die Bausteine unterscheiden sich geringfügig je nach Variablentyp.

Die Konfiguration des Agents wird abgeschlossen durch Aufruf der Funktion PLM_SNMP_DataLine() mit leeren Parametern:

```
PLM_SNMP_DataLine( Name := '', Descr := '', Id := '', RW := FALSE, ValType := 0, ValAddress := 0 );
```

21.6.1. PLM_SNMP_Baseoid()

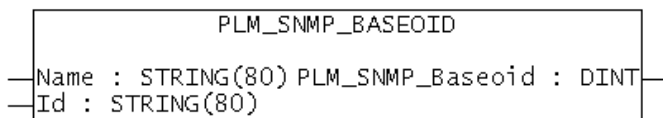


Abb. 21-2: Funktion PLM_SNMP_Baseoid

Input-Parameter:		
Name	STRING	Symbolische Bezeichnung des Basisknotens für die folgenden Variablen in der MIB. Dieser muss sich auf eine im MIB-Header vorangegangene Definition beziehen (siehe Abschnitt 21.4.1).
Id	STRING	Vollständige OID dieses Knotens. Diese muss sich auf eine im MIB-Header vorangegangene Definition beziehen (siehe Abschnitt 21.4.1).

Beispiel:

```
PLM_SNMP_Baseoid(
  Name := 'status',
  Id := '.1.3.6.1.4.1.46984.1000.1'
);
```

21.6.2. PLM_SNMP_DataLine()

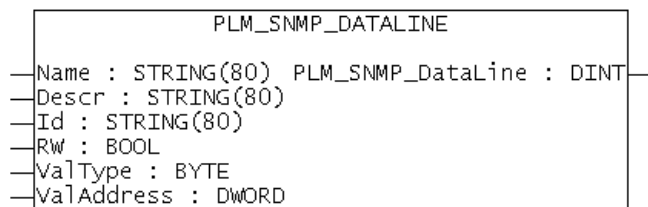


Abb. 21-3: Funktion PLM_SNMP_DataLine

Input-Parameter:		
Name	STRING	Symbolische Bezeichnung der Variablen in der MIB
Descr	STRING	Textuelle Beschreibung der Variablen
Id	STRING	Letzte Stelle der OID dieser Variablen, Zahl im Bereich '1' bis '65535'

RW	BOOL	FALSE = Nur-Lese-Zugriff TRUE = Lese-Schreib-Zugriff
ValType	BYTE	Kennung für den bei ValAddress angelegten Variablentyp: 0 = BOOL 1 = INT 2 = BYTE 3 = WORD 4 = DINT 5 = DWORD 14 = SINT 15 = USINT 16 = UINT 17 = UDINT
ValAddress	DWORD	Adresse ADR() der CoDeSys-Variablen

Die durch ValType angegebenen Variablentypen werden wie folgt in SNMP-Datentypen umgesetzt:

ValType	CoDeSys	SNMP
0	BOOL	INTEGER (0..1)
14	SINT	INTEGER (-128..127)
2 / 15	BYTE / USINT	INTEGER (0..255)
1	INT	INTEGER (-32768..32767)
3 / 16	WORD / UINT	INTEGER (0..65535)
4	DINT	INTEGER (-2147483648..2147483647)
5 / 17	DWORD / UDINT	INTEGER (0..4294967295)

Beispiel:

```

PLM_SNMP_DataLine (
    Name      := 'devStatus',
    Descr     := 'Device Status (0=Off, 1=On, 2=Error)',
    Id        := '2',
    RW        := FALSE,
    ValType   := 1, (* 1 = INT *)
    ValAddress := ADR( DeviceStatus )
);
    
```

Im Beispiel wird die CoDeSys-Variablen DeviceStatus vom Typ INT über SNMP unter dem Namen devStatus bereitgestellt. Der angegebene ValType muss unbedingt zum CoDeSys-Typ der Variablen passen.

Der beschreibende Text bei Descr wird nur für die MIB-Datei benötigt und kann ggf. leer sein.

Die OID der Variablen ergibt sich aus dem zuletzt vorangegangenen Aufruf von PLM_SNMP_Baseoid(), an den die hier angegebene Id angehängt wird.

21.6.1. PLM_SNMP_DataLineString()

PLM_SNMP_DATALINESTRING	
—Name :	STRING(80) PLM_SNMP_DataLineString : DINT
—Descr :	STRING(80)
—Id :	STRING(80)
—RW :	BOOL
—ValAddress :	DWORD
—ValSize :	DWORD

Abb. 21-4: Funktion PLM_SNMP_DataLineString

Input-Parameter:		
Name	STRING	Symbolische Bezeichnung der Variablen in der MIB
Descr	STRING	Textuelle Beschreibung der Variablen
Id	STRING	Letzte Stelle der OID dieser Variablen, Zahl im Bereich '1' bis '65535'
RW	BOOL	FALSE = Nur-Lese-Zugriff TRUE = Lese-Schreib-Zugriff
ValAddress	DWORD	Adresse ADR() der CoDeSys-Variablen
ValSize	DWORD	Länge sizeof() der String-Variablen im Speicher

Beispiel:

```

PLM_SNMP_DataLineString(
    Name      := 'software',
    Descr     := 'Software Version',
    Id        := '1',
    RW        := FALSE,
    ValAddress := ADR( SoftwareVersion ),
    ValSize   := sizeof( SoftwareVersion )
);
    
```

21.6.2. PLM_SNMP_DataLineReal()

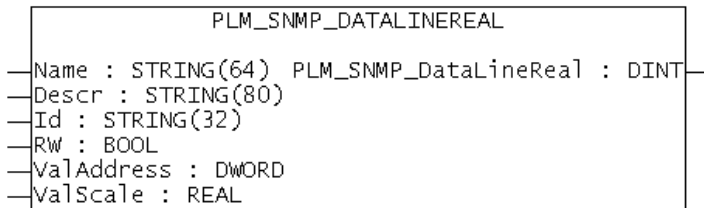


Abb. 21-5: Funktion PLM_SNMP_DataLineReal

Input-Parameter:		
Name	STRING	Symbolische Bezeichnung der Variablen in der MIB
Descr	STRING	Textuelle Beschreibung der Variablen
Id	STRING	Letzte Stelle der OID dieser Variablen, Zahl im Bereich '1' bis '65535'
RW	BOOL	FALSE = Nur-Lese-Zugriff TRUE = Lese-Schreib-Zugriff
ValAddress	DWORD	Adresse ADR() der CoDeSys--Variablen
ValScale	REAL	Skalierungsfaktor

Für den CoDeSys-Variablentyp REAL gibt es unter SNMP keine direkte Entsprechung, da dort keine Fließkommazahlen definiert sind. Die Umsetzung erfolgt daher immer in den SNMP-Datentyp INTEGER (-2147483648..2147483647).

Um die Übertragung von Nachkommastellen zu ermöglichen, kann ein Skalierungsfaktor ValScale angegeben werden, mit dem der CoDeSys-Wert beim Lesen durch SNMP multipliziert wird. Beim Schreiben der Variablen über SNMP wird der SNMP-Wert durch ValScale dividiert.

Beispiel:

```

PLM_SNMP_DataLineReal(
    Name      := 'outTemp',
    Descr     := 'Outside Temperature (DegC x 10)',
    ...
);
    
```

```

    Id      := '3',
    RW      := FALSE,
    ValAddress := ADR( OutTemp ),
    ValScale := 10.0 (* scaling REAL -> SNMP integer *)
);

```

Der Temperaturwert `OutTemp`, der in `CoDeSys` als `REAL` mit 1/10 °C Auflösung verwendet wird, kann mit `ValScale = 10.0` unter `SNMP` als Integer-Zahl dargestellt werden, wobei eine Temperatur von z.B. 23,7 °C einem `SNMP`-Wert von 237 entspricht.

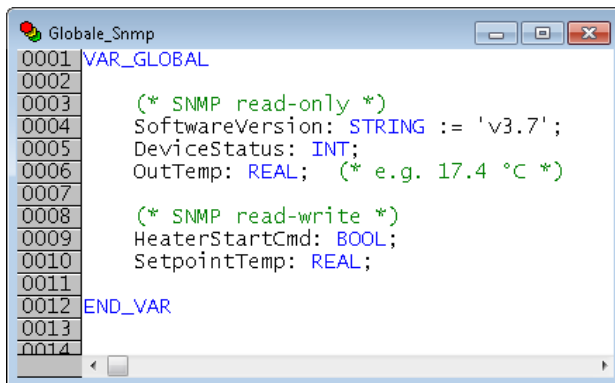
21.7. Programmbeispiel

Das folgende Programmbeispiel illustriert die Konfiguration eines `SNMP`-Agents. Es ist auf Nachfrage als vollständiges `CoDeSys`-Projekt bei uns erhältlich.

Aus dem `PLC_PRG()` oder aus `InitOnce()` heraus muss lediglich einmalig der unten angegebene Programmbaustein `SnmpInit()` aufgerufen werden. Danach können die dort konfigurierten Variablen über `SNMP` gelesen bzw. geschrieben werden.

Der Programmbaustein `SnmpInit()` ist zu Demonstrationszwecken einmal in `ST` und mit gleicher Funktion alternativ in `FUP` realisiert.

Die verwendeten Variablen sind in einer globalen Variablenliste wie folgt deklariert:



```

0001 VAR_GLOBAL
0002
0003     (* SNMP read-only *)
0004     SoftwareVersion: STRING := 'v3.7';
0005     DeviceStatus: INT;
0006     OutTemp: REAL; (* e.g. 17.4 °C *)
0007
0008     (* SNMP read-write *)
0009     HeaterStartCmd: BOOL;
0010     SetpointTemp: REAL;
0011
0012 END_VAR
0013
0014

```

Programmbaustein `SnmpInit()` in `ST`:

```

PROGRAM SnmpInit_ST
VAR
    initDone: BOOL;
    snmpAgent: Plm_SNMP_Init;
END_VAR

IF initDone THEN
    RETURN;
END_IF
initDone := TRUE;

(** SNMP agent **)

snmpAgent (
    SysDescr      := 'Yornblosch Heater Controller',
    SysLocation   := 'Muwumi Building, 1st floor',
    SysContact    := 'service@yornblosch.com',
    SysObjId      := '.1.3.6.1.4.1.46984.1000', (* .iso.org.dod.internet.private.enterprises.sabo.cust1000 *)
    Port          := 161, (* standard UDP port for SNMP agent *)
    MibName       := 'YornbloschHeaterMib', (* Name of the MIB *)
    MibFilename   := 'a/YornbloschHeater.mib' (* create MIB file on FTP drive a/ *)
);

PLM_SNMP_V2_User( Name := 'Public', RW := FALSE ); (* SNMP v2 Community with read-only access *)
PLM_SNMP_V2_User( Name := 'Private', RW := TRUE ); (* SNMP v2 Community with read-write access *)

(*
PLM_SNMP_V3_User (
    Name      := 'paul',
    LoginPass := 'paulpassw',
    Auth      := 'SHA',
    Priv      := 'AES',

```

```

        RW          := FALSE
    );
*)

(** MIB headers **)

PLM_SNMP_mibimports( Import := 'ENTERPRISES FROM SNMPv2-SMI' );
PLM_SNMP_mibdef( Def := 'sabo OBJECT IDENTIFIER ::= { enterprises 46984 }' );
PLM_SNMP_mibdef( Def := 'cust1000 OBJECT IDENTIFIER ::= { sabo 1000 }' );
PLM_SNMP_mibdef( Def := 'status OBJECT IDENTIFIER ::= { cust1000 1 }' );
PLM_SNMP_mibdef( Def := 'control OBJECT IDENTIFIER ::= { cust1000 2 }' );

(** definition of cust1000.status **)

PLM_SNMP_Baseoid( Name := 'status', Id := '.1.3.6.1.4.1.46984.1000.1' );

PLM_SNMP_DataLineString(
    Name      := 'software',
    Descr     := 'Software Version',
    Id        := '1',
    RW        := FALSE,
    ValAddress := ADR( SoftwareVersion ),
    ValSize   := SIZEOF( SoftwareVersion )
);

PLM_SNMP_DataLine(
    Name      := 'devStatus',
    Descr     := 'Device Status (0=Off, 1=On, 2=Error)',
    Id        := '2',
    RW        := FALSE,
    ValType   := 1, (* 1 = INT *)
    ValAddress := ADR( DeviceStatus )
);

PLM_SNMP_DataLineReal(
    Name      := 'outTemp',
    Descr     := 'Outside Temperature (DegC x 10)',
    Id        := '3',
    RW        := FALSE,
    ValAddress := ADR( OutTemp ),
    ValScale  := 10.0 (* scaling REAL -> SNMP integer *)
);

(** definition of cust1000.control **)

PLM_SNMP_Baseoid( Name := 'control', Id := '.1.3.6.1.4.1.46984.1000.2' );

PLM_SNMP_DataLine(
    Name      := 'start',
    Descr     := 'Start/Stop Heater (1=Start, 0=Stop)',
    Id        := '1',
    RW        := TRUE,
    ValType   := 0, (* 0 = BOOL *)
    ValAddress := ADR( HeaterStartCmd )
);

PLM_SNMP_DataLineReal(
    Name      := 'tempSetpoint',
    Descr     := 'Temperature Setpoint (DegC x 10)',
    Id        := '2',
    RW        := TRUE,
    ValAddress := ADR( SetpointTemp ),
    ValScale  := 10.0 (* scaling REAL -> SNMP integer *)
);

(** finish definition **)

PLM_SNMP_DataLine( Name := '', Descr := '', Id := '', RW := FALSE, ValType := 0, ValAddress := 0 );

(** end **)

```

Programmbaustein SnpInit() in FUP:

```

SnmpInit_FUP (PRG-FUP)
0001 PROGRAM SnmpInit_FUP
0002 VAR
0003     initDone: BOOL;
0004     snmpAgent: Plm_SNMP_Init;
0005 END_VAR
0006

```

```

0001
initDone—⟨Return⟩

```

```

0002
TRUE—initDone

```

```

0003
iso.org.dod.internet.private.enterprises.sabo.cust1000
    snmpAgent
        PLM_SNMP_INIT
        'Yornblosch Heater Controller'-SysDescr
        'Muwumi Building, 1st floor'-SysLocation
        'service@yornblosch.com'-SysContact
        '.1.3.6.1.4.1.46984.1000'-SysobjId
        161-Port
        'YornbloschHeaterMib'-MibName
        'a/yornbloschHeater.mib'-MibFilename

```

```

0004
SNMP v2 Community with read-only access
'Public'-Name
FALSE-Rw
    PLM_SNMP_V2_USER

```

```

0005
SNMP v2 Community with read-write access
'Private'-Name
TRUE-Rw
    PLM_SNMP_V2_USER

```

```

0006
MIB headers
'ENTERPRISES FROM SNMPv2-SMI'-Import
    PLM_SNMP_MIBIMPORTS

```

```

0007
'sabo OBJECT IDENTIFIER ::= { enterprises 46984 }'-Def
    PLM_SNMP_MIBDEF

```

```

0008
'cust1000 OBJECT IDENTIFIER ::= { sabo 1000 }'-Def
    PLM_SNMP_MIBDEF

```

```

0009
'status OBJECT IDENTIFIER ::= { cust1000 1 }'-Def
    PLM_SNMP_MIBDEF

```

```

0010
'control OBJECT IDENTIFIER ::= { cust1000 2 }'-Def
    PLM_SNMP_MIBDEF

```

```

0011
definition of cust1000.status
'status'-Name
.1.3.6.1.4.1.46984.1000.1'-Id
    PLM_SNMP_BASEOID

```

```

0012
create a STRING with OID = cust1000.status.1
'software'-Name
'Software Version'-Descr
'1'-Id
FALSE-Rw
ADR(SoftwareVersion)-valAddress
SIZEOF(SoftwareVersion)-valSize
    PLM_SNMP_DATALINESTRING

```

```

0013 create an INT with OID = cust1000.status.2
      'devStatus' Name PLM_SNMP_DATALINE
      'device status (0=off, 1=on, 2=Error)' Descr
      '2' Id
      FALSE Rw
      1 valType
      ADR(DeviceStatus) valAddress

0014 create a REAL with OID = cust1000.status.3
      'outTemp' Name PLM_SNMP_DATALINEREAL
      'outside temperature (degC x 10)' Descr
      '3' Id
      FALSE Rw
      ADR(outTemp) valAddress
      10.0 valScale

0015 definition of cust1000.control
      'control' Name PLM_SNMP_BASEOID
      '.1.3.6.1.4.1.46984.1000.2' Id

0016 create a BOOL with OID = cust1000.control.1
      'start' Name PLM_SNMP_DATALINE
      'start/stop heater (1=start, 0=stop)' Descr
      '1' Id
      TRUE Rw
      0 valType
      ADR(HeaterStartCmd) valAddress

0017 create a REAL with OID = cust1000.control.2
      'tempSetpoint' Name PLM_SNMP_DATALINEREAL
      'temperature setpoint (degC x 10)' Descr
      '2' Id
      TRUE Rw
      ADR(setpointTemp) valAddress
      10.0 valScale

0018 end
      Name PLM_SNMP_DATALINE
      Descr
      Id
      FALSE Rw
      0 valType
      0 valAddress
    
```

Nach Aufruf von `SnmpInit()` stehen die Variablen über SNMP zur Verfügung.

Außerdem kann von FTP-Laufwerk `a/` eine MIB-Datei `YornbloschHeater.mib` heruntergeladen werden. Diese wird in einen SNMP-Client oder eine Managementkonsole geladen und macht die Struktur der MIB bekannt.

Anschließend können die deklarierten Variablen über SNMP abgefragt und teilweise geschrieben werden. In einem Test-Client sieht das z.B. so aus:

iReasoning MIB Browser

File Edit Operations Tools Bookmarks Help

Address: 10.1.1.228 Advanced... OID: .1.3.6.1.4.1.46984.1000.2.2.0 Operations: Get Next Go

SNMP MIBs

MIB Tree

- iso.org.dod.internet
 - mgmt
 - private
 - enterprises
 - sabo
 - cust1000
 - status
 - software
 - devStatus
 - outTemp
 - control
 - start
 - tempSetpoint

Result Table

Name/OID	Value	Type	IP:Port
software.0	v3.7	OctetString	10.1.1.228:161
devStatus.0	0	Integer	10.1.1.228:161
outTemp.0	195	Integer	10.1.1.228:161
start.0	0	Integer	10.1.1.228:161
tempSetpoint.0	224	Integer	10.1.1.228:161

Name	tempSetpoint
OID	.1.3.6.1.4.1.46984.1000.2.2
MIB	YornbloschHeaterMib
Syntax	INTEGER (-2147483648..2147483647)
Access	read-write
Status	mandatory
DefVal	
Indexes	
Descr	Temperature Setpoint (DegC x 10)

22. OPC-UA Server

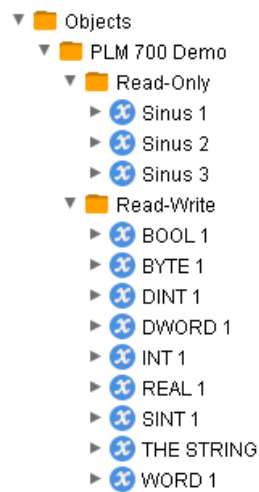
22.1. Allgemeines

Das Protokoll OPC-UA (OPC Unified Architecture) ermöglicht einen geräteunabhängigen Datenaustausch und findet zunehmend Verbreitung in der Industrie. Die Administration erfolgt durch OPC Foundation. Weitere Informationen finden sich auf der Website der Organisation unter <https://opcfoundation.org/>.

Im Gegensatz zum Vorgängerprotokoll OPC beruht OPC-UA nicht auf dem proprietären COM/DCOM-Stack von Microsoft Windows, so dass die Implementierung auch auf Embedded-Systemen möglich ist.

OPC-UA wird zur Verbindung verschiedener Systemebenen eingesetzt, z.B. auf SPS-Steuerungen oder in übergeordneten Leitebenen. Die Implementierungen unterscheiden sich entsprechend im Leistungsumfang.

Das Objektmodell von OPC-UA besteht aus Knotenpunkten, die üblicherweise baumartig strukturiert sind, z.B. wie in folgender Abbildung dargestellt.



Der hier beschriebene OPC-UA-Server erlaubt den Aufbau einer Baumstruktur unterhalb von "Objects" mit benutzerdefinierten Ordnern und Datenpunkten (Variablen).

Jedes Objekt besitzt eine eindeutige Knotennummer. Diese kann beim Aufbau einer Hierarchie verwendet werden, indem beim Anlegen eines Objekts (Ordner oder Datenpunkt) die Knotennummer des übergeordneten Objekts (Elternobjekts) angegeben wird, unter das der neue Knoten eingehängt wird.

OPC-UA-Clients können über einen sog. "Browse"-Befehl zunächst die Objektstruktur beim Server abfragen, daher ist die Konfiguration der Clients in der Regel relativ transparent.

22.2. Spezifikation

- OPC-UA-Server für Systeme der Reihe PLM700-A
- Kommunikation über OPC-UA Binary Encoding, Standard-TCP-Port 4840
- Konfiguration des Servers durch das IEC-Programm
- Beliebige Anzahl von Datenobjekten
- Beliebige Hierarchiestruktur der Ordner und Datenpunkte
- Unterstützt Descriptions und Min/Max-Werteüberwachung
- Bis zu 5 Server mit unterschiedlicher Konfiguration möglich
- Erfordert LZS ab v21804131

22.3. Benötigte Bibliotheken

Plm OPCUA_v2180416.lib (oder spätere Version)

Die angegebene Bibliothek muss vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

22.4. Installation auf der Steuerung

Zur Installation des OPC-UA-Servers auf der Steuerung ist das Einspielen eines Update-Pakets über das WebConfig (Menü *System-Updates*) erforderlich. Das Update-Paket ist auf Anfrage bei uns erhältlich.

Anschließend muss der OPC-UA-Server im WebConfig im Menü *Services* aktiviert werden. Danach ist ein Neustart der Steuerung erforderlich.

Der Server kann anschließend unter der URL

```
opc.tcp://<ip-adresse>:4840
```

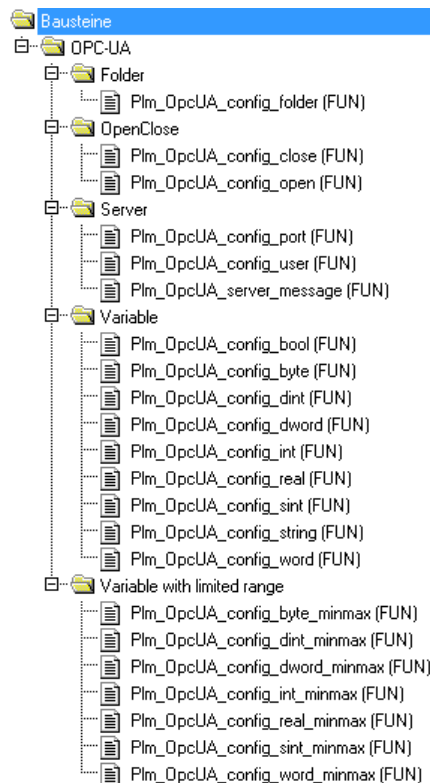
von Clients angesprochen werden.

Die weitere Konfiguration erfolgt durch das IEC-Programm unter Verwendung der aktuellen CoDeSys-Bibliothek `PLM OPCUA_v.....lib`.

Die Konfiguration wird im Folgenden beschrieben und in Abschnitt 22.6 in einem Programmbeispiel demonstriert.

22.5. Konfiguration des OPC-UA-Servers

Die Konfiguration des OPC-UA-Servers erfolgt durch einmaligen Aufruf der benötigten Bibliotheksfunktionen (z.B. in `InitOnce()`):



Die Konfiguration startet mit Aufruf der Funktion `Plm_OpcUA_config_open()`. Es können bis zu fünf verschiedene OPC-UA-Server eingerichtet werden, die getrennt konfiguriert werden können.

Falls erforderlich werden Zugangsberechtigungen mit der Funktion `Plm_OpcUa_config_user()` festgelegt. Dabei können bis zu 20 User mit Passwort festgelegt werden, die Zugang zu diesem OPC-UA-Server haben sollen und wahlweise auch der "Anonymous"-User, der sich ohne Passwort anmelden

kann. Wenn keine Zugangsberechtigungen explizit angelegt werden, wird implizit ein Anonymous-User eingerichtet.

Danach erfolgt das Anlegen von Ordnern und Datenpunkten (Variablen) in beliebiger Reihenfolge mit den Funktionen `Plm_OpcUa_config_folder()` und `Plm_OpcUa_config_int()` etc. Diese Funktionen liefern als Rückgabewert die Knotennummer des soeben erzeugten Objekts, die wiederum als Referenz für das Anhängen weiterer Objekte dienen kann.

Die Konfiguration wird abgeschlossen durch Aufruf der Funktion `Plm_OpcUA_config_close()`.

Wichtiger Hinweis:

Alle Funktionen zum Anlegen von Variablen verwenden Zeiger auf die Variablenwerte, die mit dem `ADR()`-Operator erzeugt werden. Diese Adressen können sich beim Debuggen durch Online-Change ändern, CoDeSys erzeugt dabei Meldungen der Art "Die folgenden Variablen wurden verschoben...". In diesem Fall ist die Server-Konfiguration mit den geänderten Adressen erneut auszuführen. Dieses Problem kann nur während der Programmentwicklung auftreten, im späteren Betrieb der Anlage ändern sich die Variablenadressen nicht.

22.5.1. Plm_OpcUA_config_open()

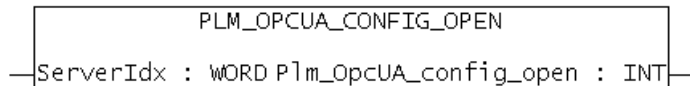


Abb. 22-1: Funktion `Plm_OpcUA_config_open`

Input-Parameter:		
ServerIdx	WORD	Diese Zahl im Bereich 0...4 gibt an, welcher der fünf verfügbaren Server im Folgenden konfiguriert wird.
Return-Wert:		
INT	0 = Fehler, 1 = ok	

Mit Aufruf dieser Funktion startet die Konfiguration eines OPC-UA-Servers. Es können bis zu fünf Server eingerichtet werden. Alle folgenden Funktionsaufrufe zur Konfiguration beziehen sich auf den mit dieser Funktion festgelegten Server.

Durch Anlegen mehrerer Server mit verschiedenen Usern und verschiedenen Datenpunkten kann eine dedizierte Zugriffsrechteverwaltung implementiert werden. In diesem Fall müssen den Servern verschiedene TCP-Ports mit der Funktion `Plm_OpcUa_config_port()` zugewiesen werden.

22.5.1. Plm_OpcUA_config_close()

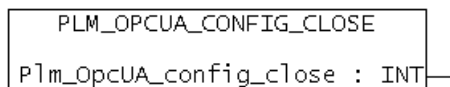


Abb. 22-2: Funktion `Plm_OpcUA_config_close`

Return-Wert:	
INT	0 = Fehler, 1 = ok

Schließt die Konfiguration ab. Der OPC-UA-Server wird ggf. mit der neuen Konfiguration automatisch neu gestartet, die Übernahme von Änderungen kann bis zu zehn Sekunden dauern.

22.5.2. Plm_OpcUa_config_port()

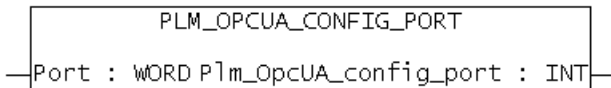


Abb. 22-3: Funktion Plm_OpcUa_config_port

Input-Parameter:		
Port	WORD	TCP-Port dieses Servers (Standard für OPC-UA und Voreinstellung: 4840)
Return-Wert:		
INT		1 = Ok

Falls mehrere Server angelegt werden, müssen sie auf verschiedene Ports gelegt werden.

22.5.1. Plm_OpcUa_config_user()

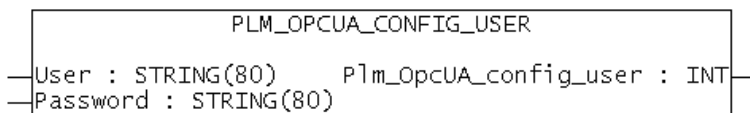


Abb. 22-4: Funktion Plm_OpcUa_config_user

Input-Parameter:		
User	STRING	Anmeldename eines Users, dem Zugriff zu diesem OPC-UA-Server gewährt wird. Der Name kann "anonymous" sein, in diesem Fall wird das Passwort ignoriert.
Password	STRING	Password dieses Users oder Leer-String
Return-Wert:		
INT		1 = Ok

Die Funktion Plm_OpcUa_config_user() kann bis zu 20 mal für einen Server aufgerufen werden. Dadurch werden bis zu 20 verschiedene User angelegt.

Durch Anlegen mehrerer Server mit verschiedenen Users und verschiedenen Datenpunkten kann eine dedizierte Zugriffsrechteverwaltung implementiert werden.

22.5.2. Plm_OpcUa_config_folder()

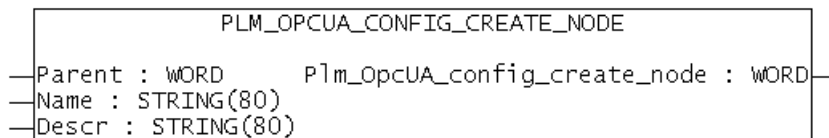


Abb. 22-5: Funktion Plm_OpcUa_config_folder

Input-Parameter:		
Parent	WORD	Knotennummer des übergeordneten (Eltern-) Objekts, unter das dieser Ordner eingehängt wird. Die Nummer muss 0 sein oder der Rückgabewert eines vorhergegangenen Aufrufs von Plm_OpcUa_config_folder(); bei 0 wird der neue Ordner unter den Wurzelordner "Objects" eingehängt.
Name	STRING	Name des Ordners (wird als Browse-Name und als Display-Name verwendet)

Descr	STRING	Eine beliebige kurze Beschreibung des Ordners, oder Leerstring (wird für Description verwendet)
Return-Wert:		
WORD		Knotennummer des neu erzeugten Ordners

Diese Funktion dient zum Aufbau einer Ordnerhierarchie. Dazu wird der Rückgabewert der Funktion in einer CoDeSys-Variablen gespeichert und kann dann bei folgenden Aufrufen als `Parent` verwendet werden.

Beispiel:

```
nBase := Plm_OpcUa_config_folder( Parent:=0, Name:='Machine Variables', Descr := ' ' );
nControl := Plm_OpcUa_config_folder( Parent:=nBase, Name:='Control', Descr := ' ' );
nStatus := Plm_OpcUa_config_folder( Parent:=nBase, Name:='Status', Descr := ' ' );
nSensors := Plm_OpcUa_config_folder( Parent:=nBase, Name:='Sensors', Descr := ' ' );
```

22.5.3. Plm_OpcUa_config_int() etc.

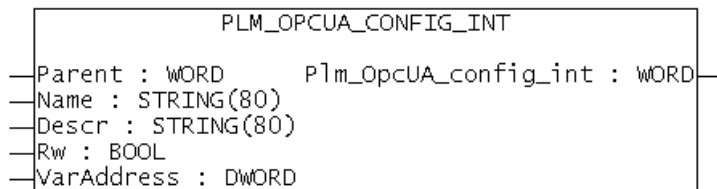


Abb. 22-6: Funktionen `Plm_OpcUa_config_int` etc.

Input-Parameter:		
Parent	WORD	Knotennummer des übergeordneten (Eltern-) Objekts, unter das die Variable eingehängt wird. Die Nummer muss 0 sein oder der Rückgabewert eines vorhergegangenen Aufrufs von <code>Plm_OpcUa_config_folder()</code> ; bei 0 wird die Variable unter den Wurzelordner "Objects" eingehängt.
Name	STRING	Name der Variablen (wird als Browse-Name und als Display-Name verwendet)
Descr	STRING	Eine beliebige kurze Beschreibung der Variablen, oder Leerstring (wird für Description verwendet)
Rw	BOOL	FALSE = Nur Lese-Zugriff möglich TRUE = Lese- und Schreib-Zugriff möglich
VarAddress	DWORD	Adresse <code>ADR()</code> der CoDeSys-Variablen
Return-Wert:		
WORD		Knotennummer der neu erzeugten Variablen

Diese Funktionen legen Variablen vom entsprechenden Typ an gemäß folgender Tabelle:

Funktion	Variablentyp	Wertebereich
<code>Plm_OpcUa_config_bool()</code>	BOOL	TRUE / FALSE
<code>Plm_OpcUa_config_byte()</code>	BYTE	0...255
<code>Plm_OpcUa_config_sint()</code>	SINT	-128...+127
<code>Plm_OpcUa_config_word()</code>	WORD	0...65535
<code>Plm_OpcUa_config_int()</code>	INT	-32768...+32767

Plm_OpcUa_config_dword()	DWORD	0...4294967295
Plm_OpcUa_config_dint()	DINT	-2147483648 ...+2147483647
Plm_OpcUa_config_real()	REAL	-3.402823466e+38 ...+3.402823466e+38

Falls `Rw = FALSE` ist (kein Schreibzugriff) und ein Client dennoch versucht, einen Wert zu schreiben, erhält er den Rückgabewert "Not writeable" und der Wert der CoDeSys-Variablen bleibt unverändert.

22.5.1. Plm_OpcUa_config_int_minmax() etc.

PLM OPCUA CONFIG INT MINMAX	
—Parent : WORD	Plm_OpcUA_config_int_minmax : WORD
—Name : STRING(80)	
—Descr : STRING(80)	
—Rw : BOOL	
—VarAddress : DWORD	
—MinMaxValid : BYTE	
—MinVal : INT	
—MaxVal : INT	

Abb. 22-7: Funktionen Plm_OpcUa_config_int_minmax etc.

Input-Parameter:		
Parent	WORD	Knotennummer des übergeordneten (Eltern-) Objekts, unter das die Variable eingehängt wird. Die Nummer muss 0 sein oder der Rückgabewert eines vorhergegangenen Aufrufs von <code>Plm_OpcUa_config_folder()</code> ; bei 0 wird die Variable unter den Wurzelordner "Objects" eingehängt.
Name	STRING	Name der Variablen (wird als Browse-Name und als Display-Name verwendet)
Descr	STRING	Eine beliebige kurze Beschreibung der Variablen, oder Leerstring (wird für Description verwendet)
Rw	BOOL	FALSE = Nur Lese-Zugriff möglich TRUE = Lese- und Schreib-Zugriff möglich
VarAddress	DWORD	Adresse <code>ADR()</code> der CoDeSys-Variablen
MinMaxValid	BYTE	1 = nur MinVal gültig 2 = nur MaxVal gültig 3 = MinVal und MaxVal gültig
MinVal	...	Minimal zulässiger Wert beim Schreiben durch den OPC-UA-Client
MaxVal	...	Maximal zulässiger Wert beim Schreiben durch den OPC-UA-Client
Return-Wert:		
WORD		Knotennummer der neu erzeugten Variablen

Diese Funktionen legen Variablen vom entsprechenden Typ an, wie `Plm_OpcUa_config_int()` etc., jedoch können zusätzlich Bereichsgrenzen angegeben werden. Diese werden überwacht, wenn ein OPC-UA-Client Werte schreibt. Beim Versuch, einen Wert außerhalb der angegebenen Grenzen zu schreiben, erhält der Client den Rückgabewert "Out of range" und der Wert der CoDeSys-Variablen bleibt unverändert.

Es ist empfehlenswert, die Grenzwerte in der Description der Variablen zu dokumentieren, um den Anwender des OPC-UA-Clients zu informieren.

Falls `Rw = FALSE` ist (kein Schreibzugriff) und ein Client dennoch versucht, einen Wert zu schreiben, erhält er den Rückgabewert "Not writeable" und der Wert der CoDeSys-Variablen bleibt unverändert.

22.5.2. Plm_OpcUa_config_string()

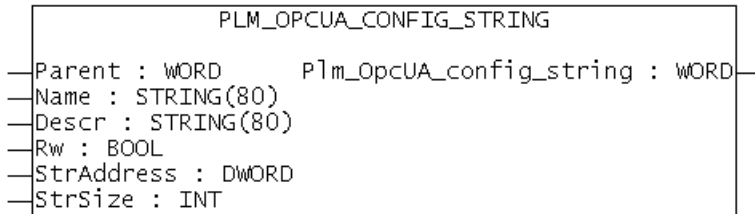


Abb. 22-8: Funktionen `Plm_OpcUa_config_string`

Input-Parameter:		
Parent	WORD	Knotennummer des übergeordneten (Eltern-) Objekts, unter das die String-Variable eingehängt wird. Die Nummer muss 0 sein oder der Rückgabewert eines vorhergegangenen Aufrufs von <code>Plm_OpcUa_config_folder()</code> ; bei 0 wird die Variable unter den Wurzelordner "Objects" eingehängt.
Name	STRING	Name der Variablen (wird als Browse-Name und als Display-Name verwendet)
Descr	STRING	Eine beliebige kurze Beschreibung der Variablen, oder Leerstring (wird für Description verwendet)
Rw	BOOL	FALSE = Nur Lese-Zugriff möglich TRUE = Lese- und Schreib-Zugriff möglich
StrAddress	DWORD	Adresse <code>ADR()</code> der CoDeSys-String-Variablen
StrSize	INT	Länge der CoDeSys-String-Variablen, so wie bei der Deklaration angegeben
Return-Wert:		
WORD		Knotennummer der neu erzeugten Variablen

Diese Funktion legt eine Variable vom Typ `STRING` an.

Als `StrSize` muss die Länge aus der Variablendeklaration angegeben werden, z.B. `STRING(80) → StrSize := 80`. Falls `Rw = TRUE` ist (Schreibzugriff möglich), wird die Länge beim Schreibzugriff überwacht, um Speicherüberläufe zu verhindern. Falls ein Client versucht, einen längeren String zu schreiben, erhält er den Rückgabewert "Out of range" und der Wert der CoDeSys-Variablen bleibt unverändert.

Falls `Rw = FALSE` ist (kein Schreibzugriff) und ein Client dennoch versucht, einen Wert zu schreiben, erhält er den Rückgabewert "Not writeable" und der Wert der CoDeSys-Variablen bleibt unverändert.

22.5.3. Plm_OpcUa_server_message()

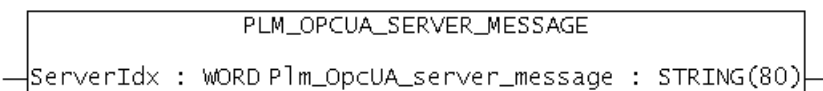


Abb. 22-9: Funktionen `Plm_OpcUa_server_message`

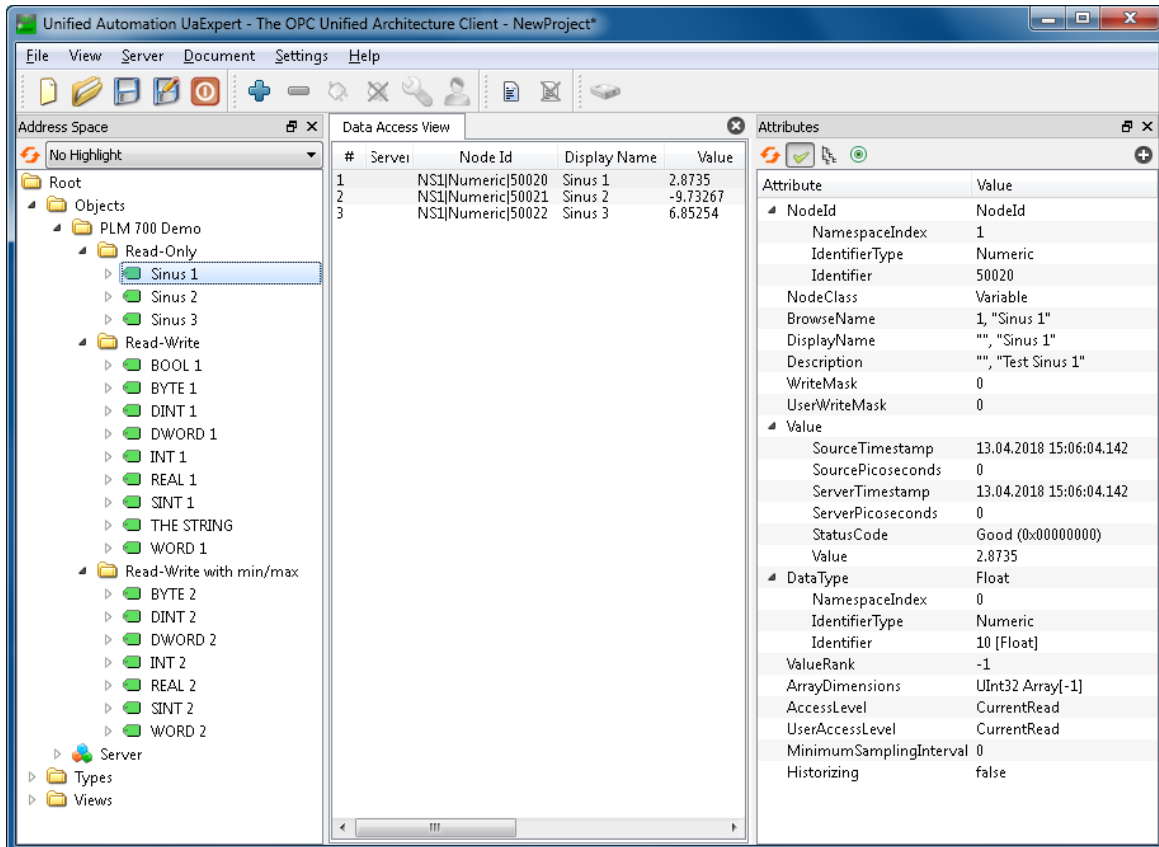
Input-Parameter:		
ServerIdx	WORD	Diese Zahl im Bereich 0...4 gibt an, welcher der fünf

	verfügbaren Server abgefragt wird.
Return-Wert:	
STRING	Kurze Zustandsbeschreibung des Servers

Diese Funktion dient Diagnosezwecken. Der zurückgelieferte String enthält die zuletzt ausgegebene Meldung des jeweiligen Servers. Im Fehlerfall ist dies eine entsprechende Fehlermeldung, andernfalls lautet der Text "server running\$N".

22.6. Programmbeispiel

Das folgende Programmbeispiel legt eine Objekthierarchie mit einigen Variablen an. Diese erscheint in einem OPC-UA-Client z.B. wie in folgender Abbildung:



Der Baustein `OpcUaServerConfig()` braucht nur einmalig aufgerufen zu werden. Das entsprechende CoDeSys-Projekt ist auf Anfrage bei uns erhältlich.

```

VAR_GLOBAL
  Bool1: BOOL;
  Byte1, Byte2: BYTE;
  Sint1, Sint2: SINT;
  Word1, Word2: WORD;
  Int1, Int2: INT;
  Dword1, Dword2: DWORD;
  Dint1, Dint2: DINT;
  Real1, Real2: REAL;
  TheString: STRING := 'Hello, world!';
  Sinus1, Sinus2, Sinus3: REAL;
END_VAR

```

```

PROGRAM OpcUaServerConfig
VAR
  initDone: BOOL;
  n0, nrw, nrw, nro: WORD;
END_VAR

```

```

IF initDone THEN
    RETURN;
END_IF
initDone := TRUE;

(* start configuration *)

Plm_OpcUA_config_open( 0 );
Plm_OpcUa_config_port( 4840 );
Plm_OpcUa_config_user( 'anonymous', '' );

(* create the tree structure *)

n0 := Plm_OpcUA_config_folder( Parent:=0, Name:='PLM 700 Demo', Descr:'' );
nrw := Plm_OpcUA_config_folder( Parent:=n0, Name:='Read-Write', Descr:'' );
nrwx := Plm_OpcUA_config_folder( Parent:=n0, Name:='Read-Write with min/max', Descr:'' );
nro := Plm_OpcUA_config_folder( Parent:=n0, Name:='Read-Only', Descr:'' );

(* create read/write variables *)

Plm_OpcUA_config_bool( Parent:=nrw, Name:='BOOL 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Bool1) );
Plm_OpcUA_config_byte( Parent:=nrw, Name:='BYTE 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Byte1) );
Plm_OpcUA_config_sint( Parent:=nrw, Name:='SINT 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Sint1) );
Plm_OpcUA_config_word( Parent:=nrw, Name:='WORD 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Word1) );
Plm_OpcUA_config_int( Parent:=nrw, Name:='INT 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Int1) );
Plm_OpcUA_config_dword( Parent:=nrw, Name:='DWORD 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Dword1) );
Plm_OpcUA_config_dint( Parent:=nrw, Name:='DINT 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Dint1) );
Plm_OpcUA_config_real( Parent:=nrw, Name:='REAL 1', Descr:'', Rw:=TRUE,
    VarAddress:=ADR(Real1) );
Plm_OpcUA_config_string( Parent:=nrw, Name:='THE STRING', Descr:'', Rw:=TRUE,
    StrAddress:=ADR(TheString), StrSize:=80 );

(* create read/write variables with limited range *)

Plm_OpcUA_config_byte_minmax( Parent:=nrwx, Name:='BYTE 2',
    Descr:='Byte 2 (range: 7...45)', Rw:=TRUE,
    VarAddress:=ADR(Byte2), MinMaxValid:=3, MinVal:=7, MaxVal:=45 );
Plm_OpcUA_config_sint_minmax( Parent:=nrwx, Name:='SINT 2',
    Descr:='Sint 2 (range: -20...+22)', Rw:=TRUE,
    VarAddress:=ADR(Sint2), MinMaxValid:=3, MinVal:=-20, MaxVal:=22 );
Plm_OpcUA_config_word_minmax( Parent:=nrwx, Name:='WORD 2',
    Descr:='Word 2 (range: 7000...40000)', Rw:=TRUE,
    VarAddress:=ADR(Word2), MinMaxValid:=3, MinVal:=7000, MaxVal:=40000 );
Plm_OpcUA_config_int_minmax( Parent:=nrwx, Name:='INT 2',
    Descr:='Int 2 (range: -20000...+22000)', Rw:=TRUE,
    VarAddress:=ADR(Int2), MinMaxValid:=3, MinVal:=-20000, MaxVal:=22000 );
Plm_OpcUA_config_dword_minmax( Parent:=nrwx, Name:='DWORD 2',
    Descr:='Dword 2 (range: 100000...200000)', Rw:=TRUE,
    VarAddress:=ADR(Dword2), MinMaxValid:=3, MinVal:=100000, MaxVal:=200000 );
Plm_OpcUA_config_dint_minmax( Parent:=nrwx, Name:='DINT 2',
    Descr:='Dint 2 (range: -333333...+333333)', Rw:=TRUE,
    VarAddress:=ADR(Dint2), MinMaxValid:=3, MinVal:=-333333, MaxVal:=333333 );
Plm_OpcUA_config_real_minmax( Parent:=nrwx, Name:='REAL 2',
    Descr:='Real 2 (range: -1.234...+1.234)', Rw:=TRUE,
    VarAddress:=ADR(Real2), MinMaxValid:=3, MinVal:=-1.234, MaxVal:=1.234 );

(* create read-only variables *)

Plm_OpcUA_config_real( Parent:=nro, Name:='Sinus 1', Descr:='Test Sinus 1', Rw:=FALSE,
    VarAddress:=ADR(Sinus1) );
Plm_OpcUA_config_real( Parent:=nro, Name:='Sinus 2', Descr:='Test Sinus 2', Rw:=FALSE,
    VarAddress:=ADR(Sinus2) );
Plm_OpcUA_config_real( Parent:=nro, Name:='Sinus 3', Descr:='Test Sinus 3', Rw:=FALSE,
    VarAddress:=ADR(Sinus3) );

(* finished *)

Plm_OpcUA_config_close();

```

23. Fehlertagebuch

[Dokumentation in Vorbereitung]

24. Kurvendarstellung (Trends)

[Dokumentation in Vorbereitung]

25. Target-Visualisierung

25.1. Allgemeines

Die Funktionen der Target-Visualisierung sind in der CoDeSys-Hilfe erläutert.

In der Praxis sind jedoch einige Besonderheiten zu beachten, die teilweise auf Besonderheiten der PLM-Steuerung, teilweise auf Eigenheiten von CoDeSys zurückzuführen sind.

Einige wichtige Besonderheiten sind im folgenden aufgeführt.

25.2. Linienzüge und Polygone

25.2.1. Gefüllte Flächen

Durch einen Fehler in CoDeSys (mind. bis Version 2.3.9.18) werden Linienzüge mit mehr als 2 Punkten immer als gefüllte Polygone dargestellt. Um dies zu verhindern muss im CoDeSys-Eigenschaftsdialog des Linienzugs in der Kategorie *Farbvariablen* bei *FillFlags* eine 1 eingetragen werden.

25.3. Bitmaps

25.3.1. Dateiname

Alle Bilder der Target-Visu müssen im Windows-Bitmap-Format (Dateiendung ".bmp") mit einer Farbtiefe von 256 Farben erstellt werden.

Die Dateinamen müssen vollständig klein geschrieben sein, inklusive der Endung (".bmp").

Die Dateinamen der Bitmaps dürfen eine maximale Länge von 16 Zeichen zuzüglich der Endung (".bmp") haben.

Alle Bitmap-Dateien eines CoDeSys-Projekts dürfen zusammen maximal 2 MB Speicher belegen.

Ab LZS v2090707 sind maximal 128 Bitmaps pro CoDeSys-Projekt zulässig. Vorher lag diese Grenze bei 64 Bitmaps bei Projekt.

Zur Bearbeitung der Bitmaps eignen sich die Programme "Microsoft Paint" (automatisch installiert unter Windows XP und Vista) und "IrfanView" (kostenloser Download unter <http://www.irfanview.net/>).

25.3.2. Farben und Farbpalette

Alle Bitmaps müssen 256 Farben haben (8 Bit Farbtiefe). Bitmaps mit höherer Farbtiefe (16 Bit, 24 Bit) müssen vor der Verwendung auf 256 Farben reduziert werden. Dies kann mit dem Programm IrfanView (s.o.) geschehen.

In jedem Visu-Bildschirm können max. 256 verschiedene Farben gleichzeitig dargestellt werden. Diese Farben stammen aus der Farbpalette der PLM-Steuerung. Die Farbpalette der Steuerung entspricht normalerweise der Standard-Farbpalette von Windows.

Für eine korrekte Farbdarstellung muss die Farbpalette des Bitmaps der Farbpalette der Steuerung entsprechen.

Ist dies nicht der Fall, kann die Farbpalette des Bitmaps gegen die Standard-Farbpalette ausgetauscht werden (z.B. mit IrfanView, s.o.). Die Standard-Farbpalette kann auch im Download-Bereich der Sabo-Website heruntergeladen werden.

Alternativ kann die PLM-Steuerung die Farbpalette aus einem Bitmap zur Laufzeit des IEC-Programms übernehmen. Dies geschieht mit folgendem Befehl:

```
SystemSetParameter( ParameterID:=3012, Value:=ADR(BmpName) );
```

Die Variable `BmpName` muss eine Globale Variable vom Typ `STRING` sein und den Namen der gewünschten Bitmap enthalten, z.B. `BmpName := 'motor2.bmp'`. In diesem Fall werden alle Bitmaps der momentanen Visu mit der Farbpalette von `motor2.bmp` dargestellt.

Der genannte Aufruf von `SystemSetParameter()` braucht nur einmalig beim Systemstart durchgeführt zu werden. Anschließend reicht es aus, den Inhalt der Variablen zu ändern, d.h. einen anderen Bitmap-Namen hinein zu schreiben.

25.3.3. Skalierung und Abschneiden

Eine Skalierung und das Abschneiden von Bitmaps in der Visu wird auf PLM-Steuerungen nicht unterstützt.

Im CoDeSys-Eigenschaftsdialog von Bitmaps darf die Funktion *Abschneiden* nicht aktiviert sein. Andernfalls werden die Positionen verschoben, wenn das Bitmap innerhalb einer Subvisu eingesetzt wird.

25.3.4. Bitmaps mittels String-Variable umschalten

Um Bitmaps im Programm mittels String-Variable umschalten zu können, muss die Funktion `DrawBitmapByString()` durch das LZS zur Verfügung gestellt werden. Dies ist ab LZS v2091212 der Fall.

Außerdem muss anstelle der Bibliothek `SysLibTargetVisu.lib` die Bibliothek `SysLibTargetVisu_neu.lib` eingebunden werden.

25.4. Zeichensatzgrößen

Die Steuerungen der Serie PLM700 verfügen über einen fest vorgegebenen Systemzeichensatz in vier verschiedenen Größen. Die Größen sind:

8, 12, 17 und 22

Die Einstellung der Zeichengröße erfolgt in CoDeSys durch Doppelklick auf ein Visu-Element und anschließend *Kategorie Text* → *Schrift* → *Schriftgrad* (siehe Abb. 25-1).

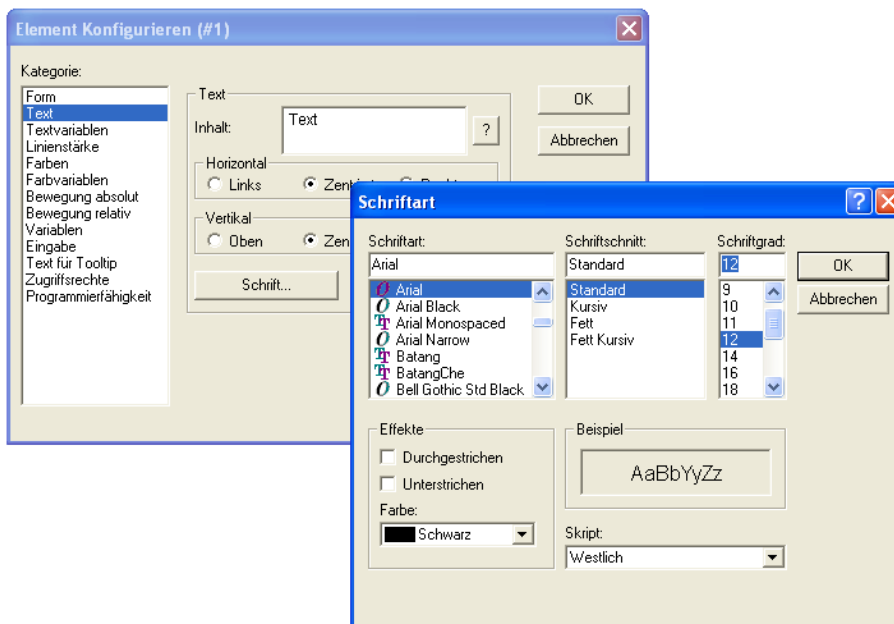


Abb. 25-1: Zeichengrößeneinstellung in CoDeSys; die Angabe wird von Windows zusätzlich skaliert

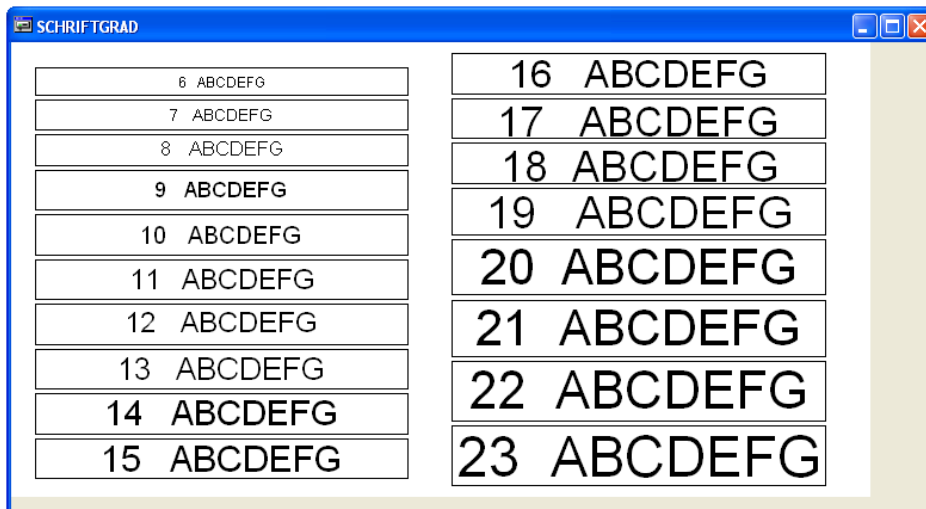


Abb. 25-2: CoDeSys Visu-Editor, verschiedene Zeichengrößen, Einstellung gemäß Abb. 25-1

Leider wird die im Dialog nach Abb. 25-1 eingestellte Zeichengröße noch von Windows umskaliert, und zwar in Abhängigkeit von der DPI-Auflösung des am PC angeschlossenen Bildschirms. Eine Angabe von z.B. Schriftgrad 12 führt deshalb nicht zwangsläufig dazu, dass auch auf der Steuerung die Schriftgröße 12 verwendet wird.

Abb. 25-3 zeigt die Darstellung verschiedener Zeichengrößen in der Target-Visu. Der PC, auf dem CoDeSys lief und das Projekt übersetzt wurde, hatte einen Bildschirm mit einer Auflösung von 96 DPI.

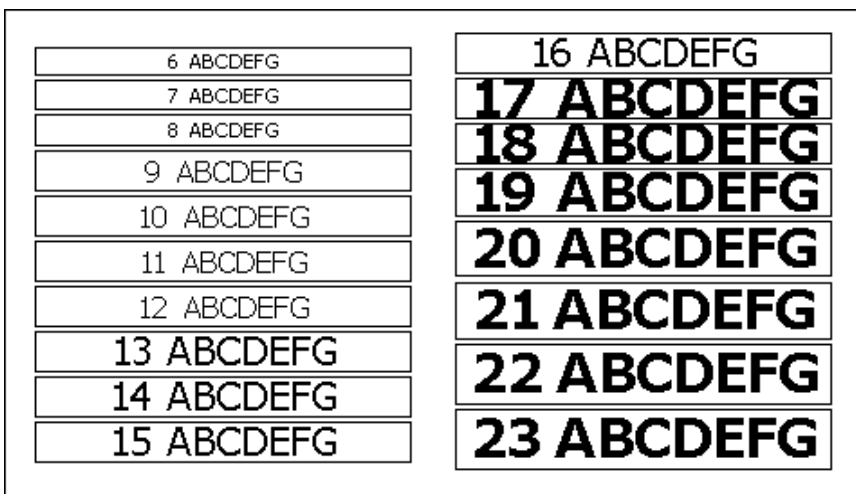


Abb. 25-3: Target-Visu, Zeichengrößeneinstellung gemäß Abb. 25-1; das Ergebnis hängt vom Entwicklungs-PC ab.

Alternativ kann die Zeichengröße durch Doppelklick auf ein Visu-Element und anschließend *Kategorie Textvariablen* → *Fonthöhe* angegeben werden (siehe Abb. 25-4).

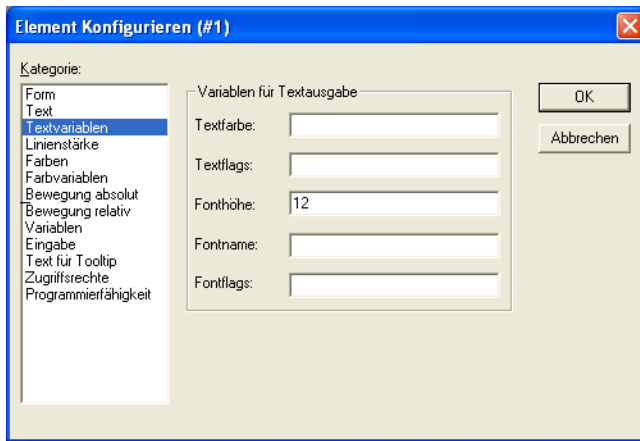


Abb. 25-4: Alternative Zeichengrößeneinstellung ohne zusätzliche Skalierung durch Windows

Die Angabe bei *Textvariablen* gemäß Abb. 25-4 wird nicht umskaliert und direkt in der Target-Visu der Steuerung umgesetzt.

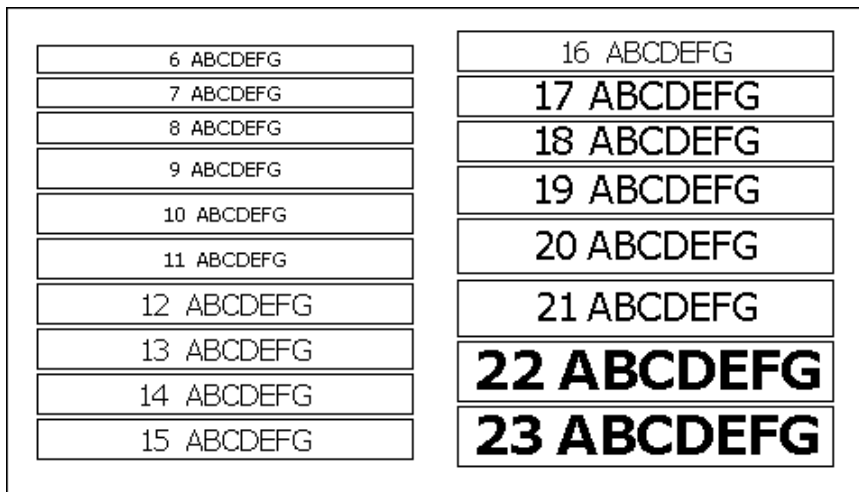


Abb. 25-5: Target-Visu, Zeichengrößeneinstellung über *Textvariablen* gemäß Abb. 25-4

Das Testprogramm, welches in der Target-Visu die Darstellungen gemäß Abb. 25-3 und Abb. 25-5 produziert, ist auf Anfrage bei uns erhältlich.

25.5. Systemzeichensatz und externe Fonts

25.5.1. Systemzeichensatz

PLM-Steuerungen unterstützen standardmäßig einen Zeichensatz nach ISO 8859-1 (siehe Abb. 25-6).

Zeichen	+	1	2	3	4	5	6	7	8	9
30				!	"	#	\$	%	&	'
40	()	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y

90	z	[\]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{		}	~	!	€	.
130	,	f	„	...	†	‡	^	%	Š	<
140	€	.	Ž	.	.	\	'	“	”	•
150	–	—	~	™	š	>	œ	·	ž	ÿ
160	·	ı	ç	£	¤	¥		§	¨	©
170	ª	«	¬	–	®	—	°	±	²	³
180	´	µ	¶	·	,	¹	º	»	¼	½
190	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
200	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
210	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
220	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
230	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
250	ú	û	ü	ý	þ	ÿ				

Abb. 25-6: PLM Systemzeichensatz nach ISO 8859-1

Falls andere Zeichen benötigt werden, kann eine spezielle Font-Datei auf der Steuerung installiert werden (siehe Abschnitt 25.5.2). Anschließend ist z.B. auch die Darstellung asiatischer Schriften möglich.

25.5.2. Externer Font

Auf den Steuerungen der Serie PLM700 kann eine Font-Datei installiert werden, die einerseits weitere Schriftgrößen und andererseits Zugriff auf Unicode-Zeichen bietet. Die Font-Dateien haben die Dateiendung "*.pfe" für Coldfire-Steuerungen und "*.pfa" für ARM-Steuerungen. Sie besitzen ein spezielles Dateiformat und können bei uns bezogen werden.

Bei Verwendung der Font-Datei stehen folgende Funktionen zur Verfügung:

- `SystemSetParameter(2114, 1);`

Durch diesen Befehl wird die Verwendung der installierten Font-Datei anstelle des Systemzeichensatzes erzwungen. Ohne diesen Aufruf werden (aus Kompatibilitätsgründen) alle Texte mit dem Font "Arial" weiterhin mit dem Systemzeichensatz dargestellt.

- `SystemSetParameter(2132, 700);`

Dieser Befehl skaliert die Darstellung der installierten Font-Datei. Die Angabe ist in Promille, d.h. der Wert 700 entspricht 70%. Da die Zeichen der installierten Font-Datei etwas größer dargestellt werden als der ursprüngliche

Systemfont, kann hiermit eine "Größenkompatibilität" zum Systemzeichensatz hergestellt werden. Dadurch wird die Umstellung bestehender Programme erleichtert.

- Verdoppeln der Schriftgröße durch Verdoppeln der Pixel in der Darstellung.
Hierzu muss die Zeichengröße beim Visu-Element spezifiziert werden (durch Doppelklick auf ein Visu-Element und anschließend *Kategorie Textvariablen* → *Fonthöhe*, siehe Abb. 25-4).
Wenn dort zu der gewünschten Zeichengröße der Wert 512 addiert wird, verdoppelt sich die Darstellungsgröße im Display bei allerdings geringerer Darstellungsqualität.

25.6. Sprachumschaltung

CoDeSys bietet zwei verschiedene Möglichkeiten zum Verwalten mehrsprachiger Visualisierungen:

1. Statische Sprachumschaltung (Sprachdatei, *.vis)
2. Dynamische Sprachumschaltung (XML-Datei, *.xml)

Die CoDeSys-Hilfe bietet weitere Hinweise unter dem Stichwort *Sprachumschaltung in Visualisierung*.

25.6.1. Statische Sprachumschaltung (Sprachdatei, *.vis)

Die statische Sprachumschaltung bietet eine primitive Möglichkeit, zur Laufzeit in der Target-Visualisierung Texte sprachabhängig umzuschalten. Das Verfahren benötigt relativ viel Programmspeicherplatz und Rechenleistung auf der Steuerung und ist daher für größere Projekte nicht zu empfehlen. Außerdem wird Unicode nicht unterstützt, so dass dieses Verfahren bei Projekten mit asiatischen oder kyrillischen Zeichen nicht verwendet kann.

Bei Verwendung der statischen Sprachumschaltung wird eine Textdatei (*Sprachdatei*) erstellt, in der die verschiedenen Sprachvarianten gepflegt werden.

Die Textdatei wird beim *Übersetzen* des Programms ausgewertet und braucht nicht auf die Steuerung übertragen zu werden. CoDeSys erzeugt dabei zusätzlichen Programmcode, der die Sprachumschaltung zur Laufzeit ermöglicht.

Für jede Sprache wird ein beliebig zu wählendes Kürzel vergeben. Das Kürzel der ausgewählten Sprache steht in der globalen Variablen `CurrentLanguage`, die von CoDeSys automatisch eingerichtet wird.

Zum Erstellen der Sprachdatei wählt man im fertigen Projekt einen Visu-Bildschirm aus und wählt dann den Menüpunkt *Extras* → *Einstellungen*. Im erscheinenden Dialogfenster ist die Kategorie *Sprache* zu wählen. Nach Anwählen von *Sprachdatei* ist ein Dateiname anzugeben. Die Sprachdatei muss die Endung *.vis* besitzen. Es empfiehlt sich, die Datei im selben Verzeichnis anzulegen in der auch sich auch die Projektdatei befindet.

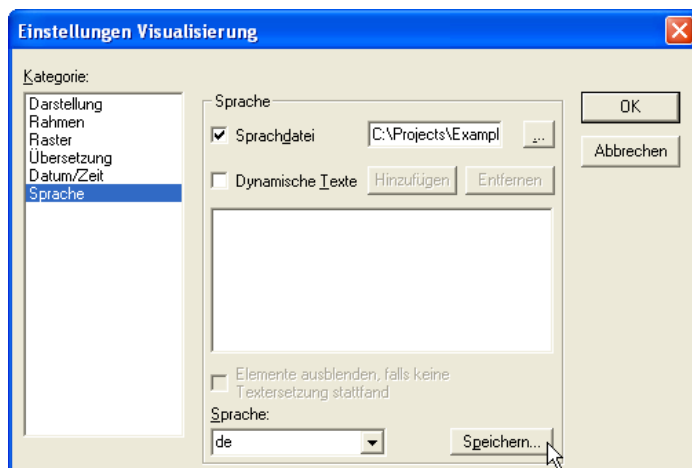


Abb. 25-7: Dialogfenster zur Auswahl der statischen Sprachdatei

Unter *Sprache* ist nun ein beliebiges Kürzel für die momentane Sprache der Visu-Bildschirme anzugeben, z.B. *de*. Anschließend wird durch Anklicken von *Speichern...* die Sprachdatei erzeugt.

Für die in Abb. 25-8 dargestellte Visu-Seite wird beim Speichern die in Abb. 25-9 dargestellte Sprachdatei erzeugt. Als Kürzel wurde *de* angegeben.

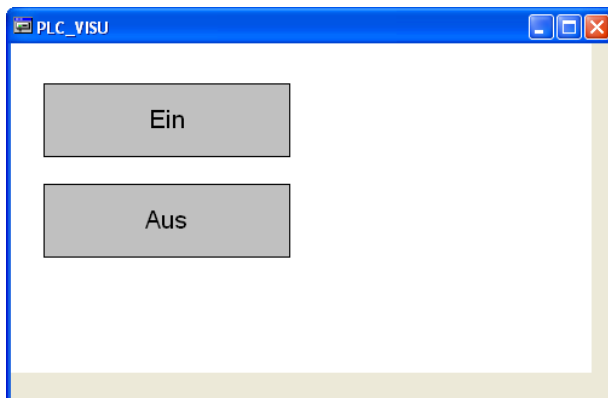


Abb. 25-8: Visualisierung mit statischer Sprachdatei in CoDeSys

```
[Sprachen]
1=de
[de]
PLC_VISU.3="Ein"
PLC_VISU.4="Aus"
```

Abb. 25-9: Inhalt der Sprachdatei für Sprache *de* zur Visu in Abb. 25-8

Für jede weitere Sprache ist die Sprachdatei manuell in einem Texteditor zu erweitern, und zwar

- um einen nummerierten Eintrag unter *[Sprachen]* und
- um einen zugehörigen Abschnitt mit den Übersetzungen.

Das Erweitern der Datei muss sorgfältig geschehen, um die Struktur der Datei nicht zu beschädigen. CoDeSys enthält keinen Editor für diese Datei, der die korrekte Struktur sicherstellen könnte.

Beim Einfügen einer neuen Sprache empfiehlt es sich, zunächst unter *[Sprachen]* ein neues Kürzel festzulegen. Danach kann ein vorhandener Übersetzungsblock kopiert und am Dateiende eingefügt werden, wobei das zu Beginn des Blocks in eckigen Klammern stehende Kürzel angepasst werden muss. Anschließend können die Übersetzungen im neuen Block vorgenommen werden.

Eine um die Sprache *en* erweiterte Sprachdatei könnte aussehen, wie in Abb. 25-10 dargestellt.

```
[Sprachen]
1=de
2=en
[de]
PLC_VISU.3="Ein"
PLC_VISU.4="Aus"
[en]
PLC_VISU.3="On"
PLC_VISU.4="Off"
```

Abb. 25-10: Inhalt der Sprachdatei für die Sprachen *de* und *en* zur Visu in Abb. 25-8

CoDeSys bietet eine eingeschränkte Möglichkeit, die übersetzten Texte aus der Sprachdatei wieder zurück in die Entwicklungsumgebung zu laden. Dieses Verfahren ist jedoch relativ intransparent und fehleranfällig.

Es wird daher empfohlen, die Datei getrennt vom Projekt mit einem Texteditor zu pflegen.

Nach Änderungen an der Visu (neue Textfelder) kann die Sprachdatei erneut gespeichert werden. Dabei wird aber nur der Teil der Datei aktualisiert, der dem im

Dialog angegebenen Kürzel entspricht. Anschließend müssen alle Übersetzungen von Hand nachgepflegt werden.

Falls beim Speichern versehentlich ein falsches Kürzel angegeben wurde, werden eventuell Übersetzungen in der Datei überschrieben; es empfiehlt sich daher unbedingt, vor dem Speichern Backups der Sprachdatei anzulegen.

Die Sprachdatei wird beim Übersetzen des Projekts durch CoDeSys ausgewertet und in Programmcode übersetzt. Damit Änderungen an der Sprachdatei wirksam werden, ist ggf. der Menüpunkt *Projekt* → *Alles Übersetzen* aufzurufen.

Die Umschaltung der Sprache im Projekt erfolgt durch Beschreiben der globalen Variablen `CurrentLanguage`. Diese Variable wird von CoDeSys automatisch angelegt und enthält das Kürzel der aktiven Sprache in Großbuchstaben, z.B. 'DE'. Eine Änderung von `CurrentLanguage`, z.B.

```
CurrentLanguage := 'EN';
```

wirkt sich sofort auf die dargestellte Target-Visualisierung aus.

Ein ungültiger Wert in `CurrentLanguage` führt dazu, dass im Display der Steuerung gar keine Texte angezeigt werden.

Zur Sprachumschaltung durch einen Button in der Target-Visualisierung bietet CoDeSys einen speziellen Befehl an. Dazu ist ein beliebiges Visu-Element anzulegen und der zugehörige Konfigurationsdialog aufzurufen. Anschließend aktiviert man in der Kategorie *Eingabe* die Aktion *Programm ausführen*. Durch Klicken auf den Button mit den drei Punkten gelangt man in einen Konfigurationsdialog. In diesem wählt man als Kommando *LANGUAGE* und gibt als Wert eines der Kürzel aus der Sprachdatei ein (siehe Abb. 25-11). Anschließend auf *Hinzufügen* und im Konfigurationsdialog auf *OK* klicken. Der auf diese Weise konfigurierte Button bewirkt, dass beim Drücken der Wert in `CurrentLanguage` geändert und dadurch die angezeigte Sprache umgeschaltet wird.

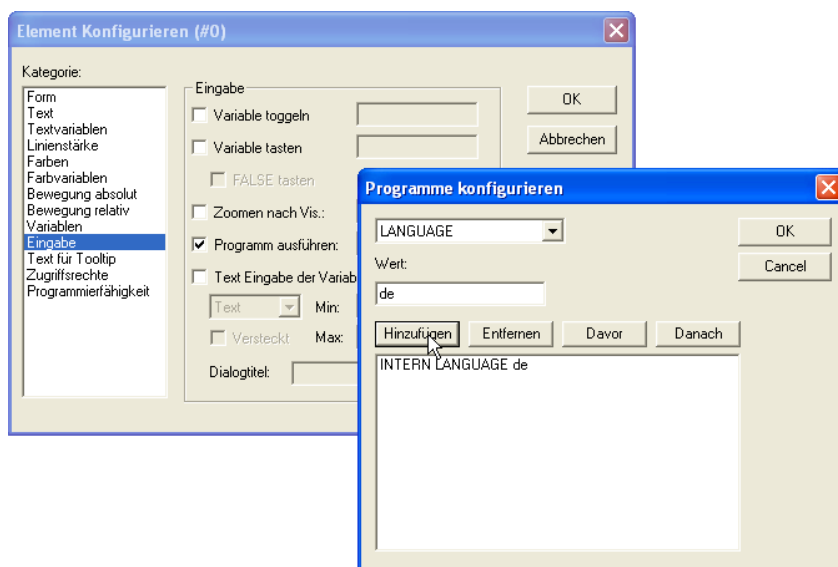


Abb. 25-11: Erzeugen eines Buttons zur Sprachumschaltung

Falls die Variable `CurrentLanguage` nicht vom Programm initialisiert wird und ein Offline-Bootprojekt erzeugt werden soll, muss zunächst ein Einloggen in eine Steuerung und anschließend wieder Ausloggen erfolgen, damit CoDeSys den richtigen Initialwert in das Offline-Bootprojekt aufnimmt.

25.6.2. Dynamische Texte (*.xml)

Bei Verwendung dynamischer Texte müssen alle betroffenen Strings in der Visualisierung durch Platzhalter der Form `%<Name>` ersetzt. Der Platzhalter wird zur Laufzeit durch einen sprachabhängigen Text aus einer XML-Datei ersetzt. Das Kürzel der ausgewählten Sprache steht in der globalen Variablen `CurrentLanguage`, die von CoDeSys automatisch eingerichtet wird.

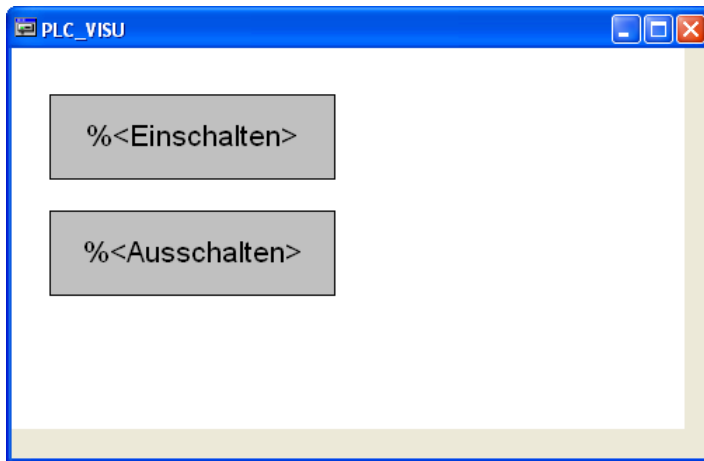


Abb. 25-12: Visualisierung mit dynamischen Texten in CoDeSys

Die XML-Datei muss eine bestimmte Struktur haben und im Unicode-Format UTF-16 vorliegen. Unicode-Dateien beginnen mit einem unsichtbaren Unicode-Zeichen $U+FEFF$, anhand dessen ein XML-Editor das UTF-16-Format und die sog. *Endianness* (Intel- oder Motorola-Byteorder) erkennt.

Das Speicherformat der Datei muss Little-Endian (Intel-Byteorder) sein; die beiden ersten Bytes der Datei (Unicode-Kennung $U+FEFF$) haben dann die Werte $16\#FF$ $16\#FE$ in genau dieser Reihenfolge.

Die XML-Datei wird nicht innerhalb von CoDeSys erstellt und muss außerhalb von CoDeSys gepflegt werden. Wir empfehlen, auf einer vorhandenen Sprach-XML-Datei aufzusetzen und diese für das aktuelle Projekt anzupassen.

Für die Visu in Abb. 25-12 sieht die XML-Datei z.B. so aus:

```

U+FEFF
<?xml version="1.0" encoding="utf-16"?>
<dynamic-text>
  <header>
    <default-language>deu</default-language>
    <default-font>
      <language>deu</language>
      <font-name>Arial</font-name>
      <font-color>0,0,0</font-color>
      <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>0</font-italic>
      <font-underline>0</font-underline>
      <font-strike-out>0</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
    <default-font>
      <language>eng</language>
      <font-name>Arial</font-name>
      <font-color>0,0,0</font-color>
      <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>0</font-italic>
      <font-underline>0</font-underline>
      <font-strike-out>0</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
  </header>
  <text-list>
    <text prefix="Einschalten" id="0">
      <deu>Ein</deu>
      <eng>On</eng>
    </text>
    <text prefix="Ausschalten" id="0">
      <deu>Aus</deu>
      <eng>Off</eng>
    </text>
  </text-list>
</dynamic-text>

```

Abb. 25-13: Beispiel für den Aufbau einer XML-Datei

Im wesentlichen besteht die Datei aus zwei Abschnitten, dem Abschnitt `<header>`, in dem Zeichensatz-Informationen gespeichert sind und dem Abschnitt `<text-list>`, in dem die Übersetzungen stehen. Der Abschnitt `<text-list>` ist üblicherweise deutlich länger als im Beispiel.

Für einen bestimmten Prefix können mehrere Einträge mit verschiedenen IDs vorhanden sein. Die jeweils verwendete ID wird aus der Variablen in `Textausgabe` bestimmt. Im einfachsten Fall wird bei `Textausgabe` der Wert 0 eingetragen.

Wenn das IEC-Programm gestartet ist, werden die Platzhalter durch die entsprechenden Einträge aus der XML-Datei ersetzt (siehe Abb. 25-14 und Abb. 25-15).

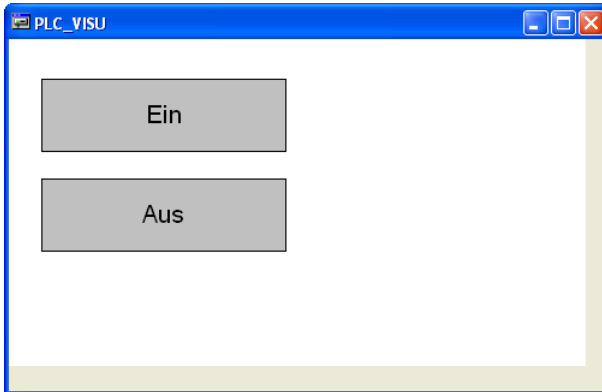


Abb. 25-14: Visualisierung bei gestartetem IEC-Programm (CurrentLanguage = 'deu')

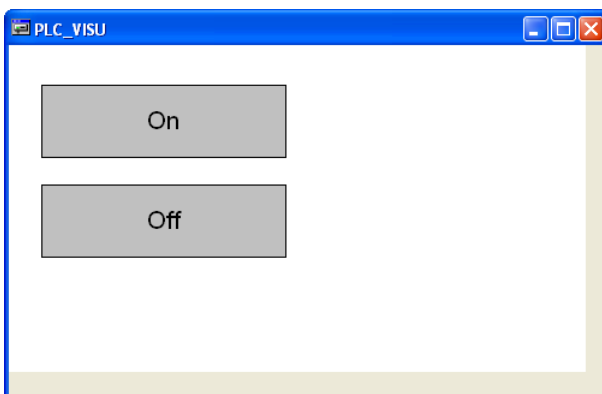


Abb. 25-15: Visualisierung bei gestartetem IEC-Programm (CurrentLanguage = 'eng')

Der Dialog zum Konfigurieren des Visu-Elements sieht dabei aus wie in Abb. 25-5.

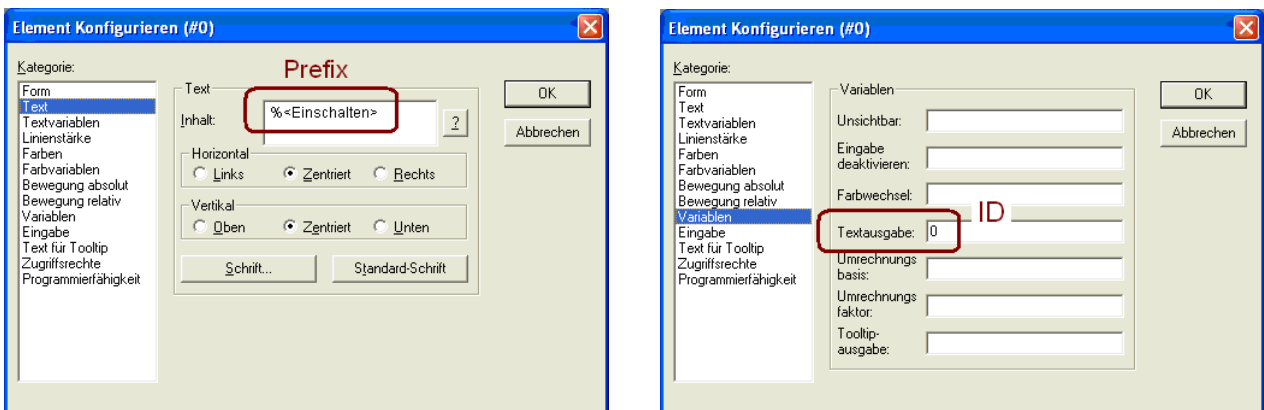


Abb. 25-16: Dialog *Element konfigurieren* bei Verwendung dynamischer Texte

Die XML-Datei wird mit dem aktuellen Projekt verknüpft, indem man in CoDeSys eine Visualisierung öffnet, mit der rechten Maustaste auf den leeren Hintergrund klickt und *Einstellungen* auswählt. Alternativ kann nach dem Öffnen der Visualisierung auch das

Menü *Extras* → *Einstellungen* ausgewählt werden. Im erscheinenden Dialogfenster ist die Kategorie *Sprache* zu wählen. Durch Anwählen von *Dynamische Texte* und Klicken auf *Hinzufügen* kann die gewünschte XML-Datei ausgewählt werden (siehe Abb. 25-17). Diese wird beim Einloggen von CoDeSys zusammen mit dem IEC-Programm automatisch auf die Steuerung übertragen.

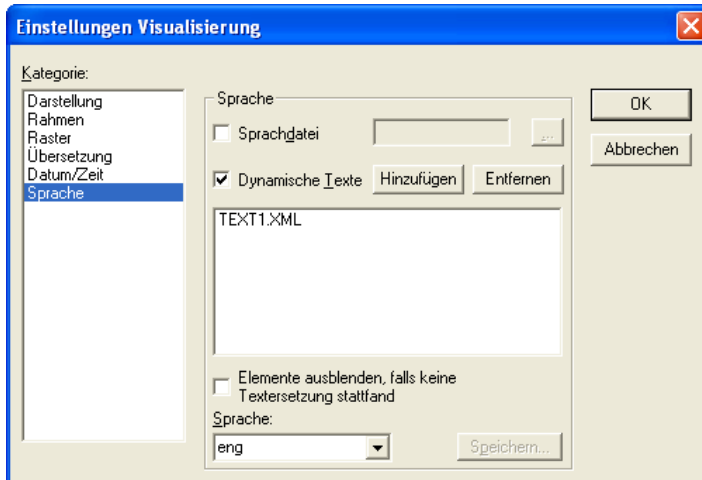


Abb. 25-17: Dialogfenster zur Auswahl der dynamischen Texte

Die CoDeSys-Entwicklungsumgebung enthält keinen XML-Editor. Die XML-Datei muss daher mit einem geeigneten externen Editor erstellt und gepflegt werden.

Zum Lieferumfang von CoDeSys gehört ein Makro für Microsoft Excel, welches die übersichtliche Bearbeitung der XML-Datei in Excel ermöglicht. Das Excel-Makro und die Installationsanleitung finden sich im CoDeSys-Installationsverzeichnis unter

C:\Programme\3S Software\CoDeSys V2.3\Documents\German,

und zwar die Dateien

dyntextmakros.xla (Makro) und
DynText_Macros_D.pdf (Anleitung).

Alternativ kann bei einer vorhandenen Datei die Bearbeitung mit einem XML-Editor, z.B. Microsoft XML-Notepad, erfolgen. Die direkte Bearbeitung der XML-Datei mit einem Texteditor ist zwar möglich, wird aber nicht empfohlen, da die Dateistruktur hierbei leicht zerstört werden kann, wodurch die Datei unbrauchbar wird.

Zur Verwendung der (Unicode-)Texte aus der XML-Datei außerhalb der Target-Visualisierung steht die Funktion

Plm_GetXmlText()

aus der Bibliothek UPD_E_016.lib (oder spätere Version) zur Verfügung. Die Verwendung ist im folgenden Beispiel illustriert:

```
VAR
    buf: STRING(80);
    prefix: STRING;
    id: DWORD;
END_VAR

CurrentLanguage := 'deu';
prefix := '%<Einschalten>';
id := 0;
PLM_GetXmlText( buf, sizeof(buf), prefix, id );
```

Nach dem Aufruf von `Plm_GetXmlText()` steht in der Variablen `buf` der Unicode-Text für den angegebenen Platzhalter.

Hinweis: In älteren Laufzeitsystemversionen muss bei der Verwendung von `Plm_GetXmlText()` mindestens eine Visu-Seite mit der Darstellung eines gültigen Platzhalters und einem Eintrag bei *Textausgabe* stehen.