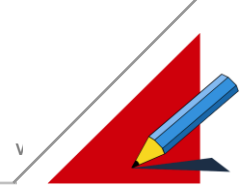


Kommunikation

INHALT

KOMMUNIKATION.....	1
ERSTELLUNG EINER NETZWERKVERBINDUNG.....	3
DAS KOMMUNIKATIONSKONZEPT (EONLINE / ZONLINE)	4
<i>Kommunikationsnetze.....</i>	4
<i>Schnittstellen</i>	5
ENTWICKLUNG VON GERÄTE- UND NETZWERKTREIBERN.....	6
<i>Das EOnline Treiber Konzept</i>	6
Einführung.....	6
Driver	6
Device.....	6
<i>EOnline Scheduler.....</i>	7
<i>Der Festplattentreiber DRIVER.DLL.....</i>	7
Driver_WindowDriverParameter.....	8
Driver_InitDriverSlot	8
Driver_GetSchedulerCycle.....	8
Driver_GetPreTime	9
Driver_SizeDataList	9
Driver_SlotTransmitData	9
Driver_SlotReceiveData	11
Driver_GetRestartTelegram	12
Driver_GetRestartTime	12
Driver_StartScan	12
Driver_ProgressScan	13
Driver_EscapeScan.....	13
<i>Der Gerätetreiber für Dateien DEVICE.DLL</i>	13
Device_SelectDevice.....	13
Device_WindowSelectDataPoint.....	14
Device_WindowMultiSelectDataPoint.....	14
Device_CheckDriver	15
Device_DataPointInfo	15
Device_ReInitDevice	16
DIENSTE DER KOMMUNIKATION	17
<i>Bearbeitung der Dienste.....</i>	17
<i>Prioritätenliste</i>	18
<i>Verfügbare Dienste: PC - Zielsystem.....</i>	18
<i>Verfügbare Dienste: Zielsystem mit Zielsystem.....</i>	19
<i>Voraussetzungen, damit diverse Datenpunkte nutzbar werden</i>	20
<i>Schichtenmodell</i>	21
RS232 SCHNITTSTELLE.....	22
<i>Über den RS232 Netzwerktreiber.....</i>	22
<i>Der Treiberdialog</i>	22
<i>Download und dabei andere Geräte deaktivieren.....</i>	24
<i>Modemverbindung.....</i>	25
Softwareeinstellungen auf PC Seite.....	25
Anwahl auf PC Seite	27
Softwareeinstellungen am Modul.....	29
<i>Download und dabei andere Geräte deaktivieren.....</i>	30
CAN SCHNITTSTELLE	31
<i>Über CAN.....</i>	31
<i>Der Treiberdialog</i>	31
<i>Die „Parameter“ Seite des Treiberdialogs.....</i>	32
CAN/MPC-Slot und I/O-Adresse	32
Übertragungsrate	36
<i>Download und dabei andere Geräte deaktivieren.....</i>	36
<i>Die „Sperrbereiche“ Seite Treiberdialogs.....</i>	37
<i>Übersicht.....</i>	37



Polling-Verfahren	39
Interrupt-Verfahren.....	40
<i>Einstellungen</i>	42
Lesezykluszeit	42
Übertragungsrate	42
CAN-distance	44
<i>Kommunikationsroutinen</i>	45
Low-Level Kommunikationsroutine.....	45
<i>Protokoll 1.x:</i>	46
Master/Slave Protokoll	47
Intermodul Protokoll	48
Single- Master Betrieb bis zur Firmware-Version 1.19.....	49
Multi-Master Betrieb mit der Firmware-Version ab 1.20	53
Single- Master Betrieb mit der Firmware-Version ab 1.20.....	59
<i>Protokoll 2.0:</i>	60
Intermodul Protokoll	61
Single- Master Betrieb mit der Firmware-Version ab 1.30.....	61
Multi- Master Betrieb mit der Firmware-Version ab 1.30.....	62
<i>Sonstige Protokolle</i>	64
<i>Download</i>	64
DIE GERÄTETREIBER	65
<i>Geräteauswahl Dialog</i>	66
<i>Datenpunktauswahl für Single/Multi Select</i>	67
Selbstdefinierte Datenpunkte.....	70
RAM-Variablen (ElaSim).....	72
Firmware-Variablen	74
Userware-Variablen.....	75
Parameter (ElaGraph).....	76
Speicher Direktzugriff (ElaSim/ElaGraph).....	77
Sondertelegramme (ElaSim/ElaGraph)	79
CAN ID	80
PROFIBUS SCHNITTSTELLE	81
<i>Die physikalischen Profibus Schnittstelle</i>	81
<i>Die Software Protokolle der Profibus Schnittstelle</i>	81
PERFORMANCE TEST	81
<i>RS232 Schnittstelle direkt</i>	81
<i>RS232 Schnittstelle via Modem</i>	81
<i>CAN Schnittstelle</i>	81

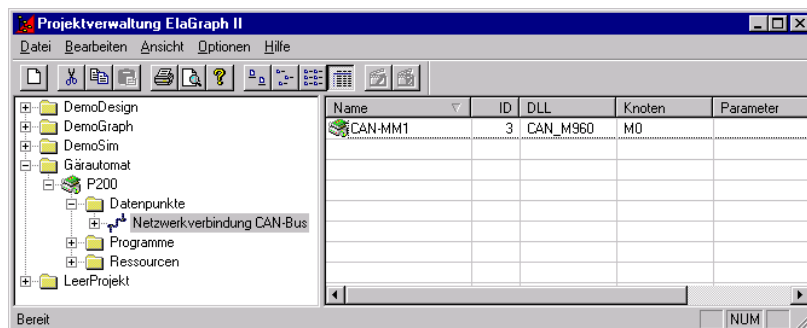


Erstellung einer Netzwerkverbindung

Die Kommunikation zwischen mehreren Geräten kann über zwei Verbindungsarten realisiert werden. Die Verbindung zwischen PC und Zielsystem wird als PC-Verbindung, zwischen mehreren Zielsystemen als Netzwerkverbindung bezeichnet.

Die PC-Verbindung kann über CAN-Bus oder RS232 umgesetzt werden. Eine Netzwerkverbindung ist zur Zeit nur über CAN-Bus möglich. Weitere Verbindungsarten sind in Vorbereitung.

Der Zugriff auf Datenpunkte weiterer Geräte, wird über eine Netzwerkverbindung realisiert. Die Netzwerke lassen sich individuell parametrieren. Alle im Projekt implementierte Geräte lassen sich somit miteinander verbinden.



Der PC ist über die RS232 mit dem Zielsystem P200 verbunden. Das P200 kommuniziert über eine CAN-Bus Netzwerkverbindung mit dem zweiten Zielsystem CAN-MM1.

Im Zielsystem CAN-MM1 können Datenpunkte projiziert werden. Das Zielsystem P200 kann somit über die Netzwerkverbindung auf Datenpunkte anderer Geräte zugreifen.

Der PC ist somit in der Lage über das Zielsystem P200 (RS232) mit dem Zielsystem CAN-MM1 (CAN-Bus) zu kommunizieren.

Hinweis:

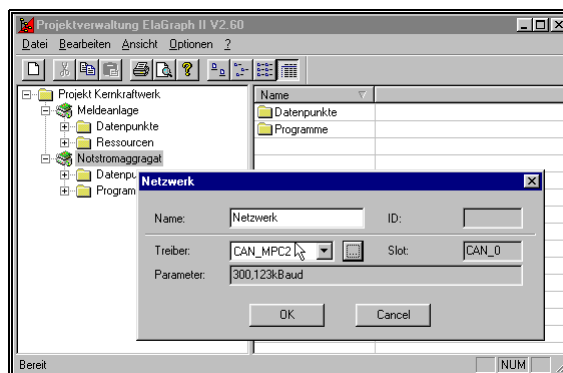
Der Zugriff auf Datenpunkte über mehrere Geräte ist nur dann möglich, wenn die Datenpunkte über eine Netzwerkverbindung zwischen diesen Geräten projiziert sind.

Netzwerkverbindungen werden in der Projektverwaltung wie folgt erstellt:

1. Auswahl des zuvor erstellten Projektes
2. Auswahl des zuvor projizieren Gerät
3. Neues Objekt einfügen: Netzwerk
4. Allgemeine Projektierung des Netzwerkes vornehmen

Unter <Name> kann ein beliebiger Name für das Netzwerk vergeben werden.

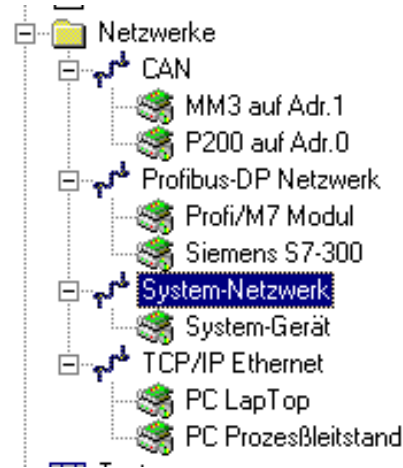
Im Feld <Treiber> muß der Treiber parametrieren werden, der in diesem Netzwerk eingesetzt werden soll.





Das Kommunikationskonzept (EOnline / ZOnline)

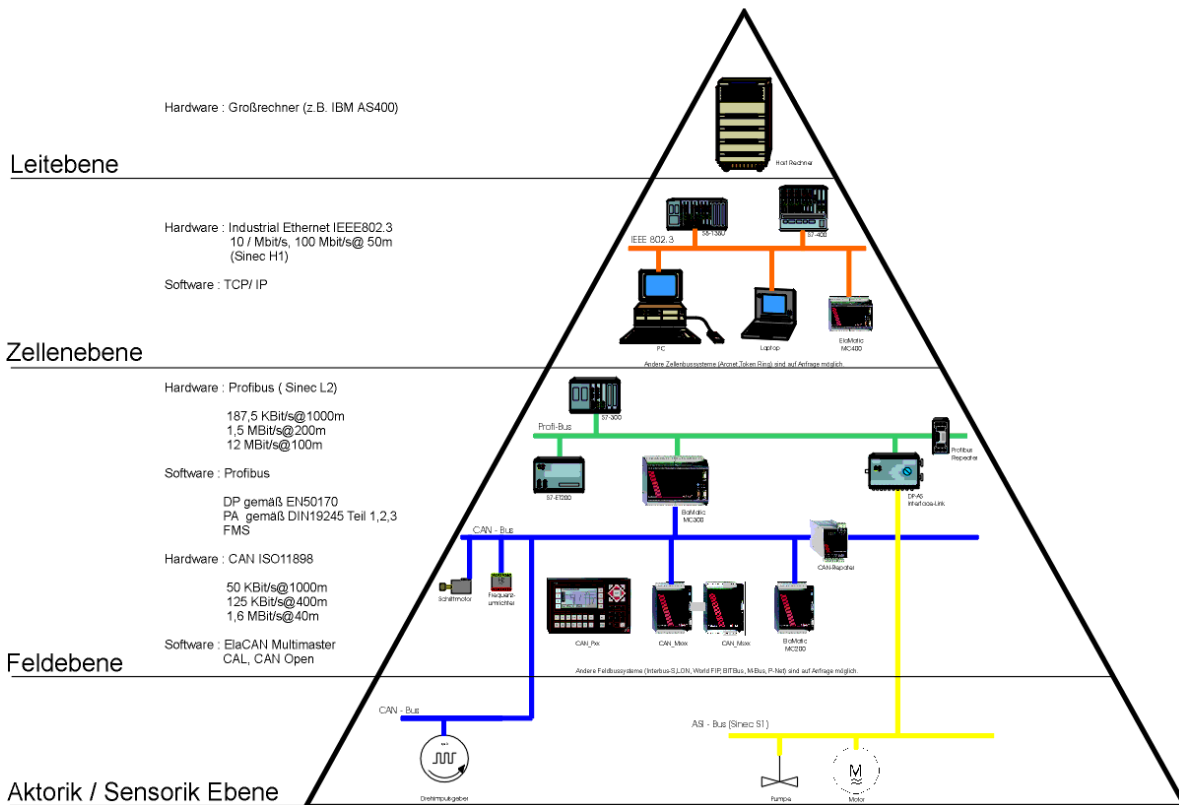
- EOnline ist eine eigenständige Applikation, welche im Hintergrund die gewünschte Kommunikation zwischen den Nutzerprogrammen und der höchsten Schicht des Kommunikationstreibers (z.B. Schicht 7 gemäß ISO/OSI Referenzmodells bzw. der TCP- Schicht) abwickelt. EOnline besitzt kein Ausgabefenster und wird vom ersten Zugriff eines Nutzerprogrammes selbständig gestartet und mit dem letzten Nutzerprogramm beendet.



- Dynamisch verwaltete Datenpunkte. Die Beschreibung zu einem Datenpunkt (physikalische Adresse, zugehöriges Gerät, zugehöriges Netzwerk, etc.) wird in der Ressource Datei gespeichert.

- EOnline ist lauffähig unter Microsoft® Windows 95™, Windows 98™ oder Windows NT™
- ZOnline ist das Gegenstück zu EOnline in den Zielsystemen. ZOnline entspricht funktionell EOnline

Kommunikationsnetze





Schnittstellen

RS232 (ca. +/- 12V Pegel, TxD, RxD, GND) als Standard.

RS232 erweitert (ca. +/- 12V Pegel, CTS, RTS, TxD, RxD, GND) als Option.

RS422 (ca. 1V Differenzpegel, T+, T-, R+, R-, GND) als Option.

RS485 (ca. 1V Differenzpegel, Low, High, GND) als Option.

TTY (ca. 20mA Stromschleife, I+, I-) als Option.

Freie Protokolle, programmiert durch Userware (kundenspezifische Protokolle).

Siemens AS511 Protokoll: Zugriff auf Siemens S5 Familie .

Siemens 3964R Protokoll: Rechnerkopplung zu Siemens Baugruppen CP-525.

Die Anschlußbelegung der Schnittstellen entnehmen Sie bitte aus der Gerätebeschreibung.



Entwicklung von Geräte- und Netzwerktreibern

Das EOnline Treiber Konzept

Einführung

Wenn es nötig wird, fremde Hardware in das EOnline Treiber Konzept einzubinden, muß der Benutzer zwei Treiber zur Verfügung stellen. Der erste wird Driver und der zweite Device Treiber genannt. Diese Programme, die als DLL („dynamic link library“ für Windows) mit C-Schnittstelle ausgeführt werden müssen, sind für folgende Aufgaben zuständig:

Driver

Ein Driver ist definiert durch die Eigenschaften der Netzwerkkarte, die er repräsentiert :

- Kommunikation mit den Geräten über die Netzwerkhardware
- Berechnung hardwarespezifischer Parameter
- „Abklopfen“ des Netzwerkes nach Kommunikationspartner (Autoscan)
- Parametrierung des Treibers durch den Anwender
- Mehrfache Instanz eines Prozesses (Threads) pro Steckplatz (Slot) auf der Netzwerkkarte

Device

Ein Device repräsentiert den Kommunikationspartner – das Gerät

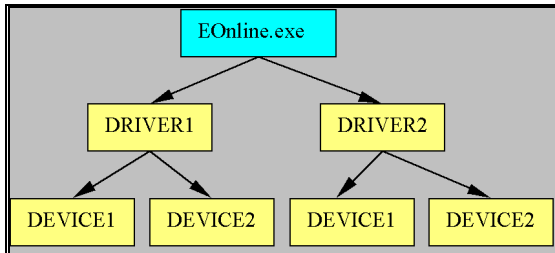
- Definition der möglichen Kommunikationspartner (welche Hardware verstehe ich ?)
- Festlegung des Gesprächspartners
- Definition möglicher Datenpunkte in einem Gerät. Dabei stehen drei mögliche Verwaltungsprinzipien zur Verfügung :
 1. Verwaltung in einer festen Liste, die in DEVICE.DLL programmiert wird.
 2. Die Liste wird aus einer Konfigurationsdatei ermittelt.
 3. Die Liste wird mittels einer Konfigurationsdatei ermittelt, die aktuell aus dem Zielsystem gelesen wird.
- Auswahl der Datenpunkte in einfacher und mehrfacher Selektion.



EOnline Scheduler

Die Device und Driver Treiber bilden mit EOnline die Kommunikation auf dem Netzwerk ab. EOnline ist ein Scheduler (Ereignisverwaltung), der die Verwaltung der Kommunikation übernimmt.

Schematisch kann der Zusammenhang (die virtuellen Verbindungen zwischen den Treibern) wie folgt dargestellt werden :



EOnline verwaltet sämtliche Anfragen aller Programme. Die Anfrage wird über das projektierte Netzwerk an das projektierte Gerät weitergeleitet.

Zur Veranschaulichung werden im folgenden zwei Beispiele zur Erstellung eines Treibers beschrieben. Diese Treiber sind nicht bei der Installation Ihres Produktes enthalten. Falls Sie an den Quellen interessiert sind, wenden Sie sich bitte an unsere Hotline.

Die Beispieldreiber wurden in C++ mit Hilfe der MFC (Microsoft Foundation Classes) erstellt. Es gelten folgende Konventionen:

```
#define DLLEXPORT extern "C" __declspec(dllexport)
```

Wird die DLL mit der MFC erzeugt, ist außerdem darauf zu achten, daß alle exportierten C-Funktionen, die auf globale Variable zugreifen mit

```
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

beginnen. Dieser Befehl wurde im Beispiel in das #define START_DLL_FUNCTION integriert. Beginnen wir nun mit einer systematischen Beschreibung der Funktionen:

Der Festplattentreiber DRIVER.DLL

Dieser Treiber leitet eine Kommunikation in eine Datei um. Die hier besprochenen Funktionen finden sie im Beispiel DRIVER in der Datei extern.cpp. Alle zu erledigenden Aufgaben sind durch den Kommentar gekennzeichnet.

```

////////////////////////////////////
// TO DO :
// Der Scan wird unterbrochen
und
// END TO DO
////////////////////////////////////
    
```

Damit eine DLL als Treiber von EOnline.exe akzeptiert wird sind folgende exportierte Funktionen nötig :



Driver_WindowDriverParameter

DLLEXPORT BOOL Driver_WindowDriverParameter(HWND hwnd, int language, char *Slot, int SizeSlot, char *ParaString, int SizeParaString)

Diese Funktion wird dazu verwendet, um den Treiber zu parametrieren. So kann hier zum Beispiel die Übertragungsrate usw. der Hardware eingestellt werden. Es ist üblich in dieser Funktion einen Dialog zu öffnen, der dies ermöglicht.

hwnd	Handle des „Parent“ Window (wird benutzt um den Dialog zu zentrieren)
language	Index der Sprache, falls der Dialog mehrsprachig erscheinen soll. Elrest intern wird der Index 0 für deutsch und 1 für englisch verwendet.
Slot	Repräsentiert die Bezeichnung des Steckplatzes in der Schnittstellenkarte.
SizeSlot	Länge des Speicherbereiches für „Slot“ einschließlich terminierender 0.
ParaString	Zeichenkette in der die Parametrisierung des Netzwerktreibers codiert sein muß.
SizeParaString	Länge des Speicherbereiches für „ParaString“ einschließlich terminierender 0.

Driver_InitDriverSlot

DLLEXPORT BOOL Driver_InitDriverSlot (char *Slot, char * ParaString)

In dieser Funktion findet die Initialisierung eines Steckplatzes (Slot) in der Schnittstellenkarte mit den Parametern (ParaString)

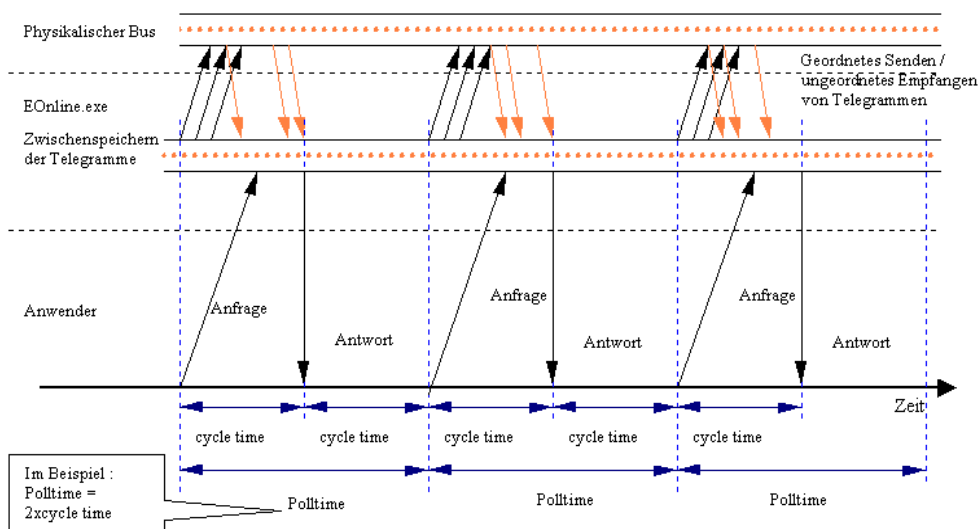
Slot	Kennzeichner für den Schnittstellenkarte
ParaString	Parameter für die Initialisierung der Schnittstellenkarte

Driver_GetSchedulerCycle

DLLEXPORT ULONG Driver_GetSchedulerCycle (char *Slot)

Slot	Kennzeichner für den Schnittstellenkarte
------	--

Diese Funktion definiert das kleinste Zeitraster, in dem die Schnittstellenkarte abgefragt wird. EOnline hört kontinuierlich in diesem Zeitraster den Bus ab und speichert sich die



empfangenen Telegramme in einem Zwischenspeicher ab. Diese Zeit wird „cycle time“ genannt und wird durch diese Funktion definiert. Slot definiert hier wieder den Steckplatz der Schnittstellenkarte.

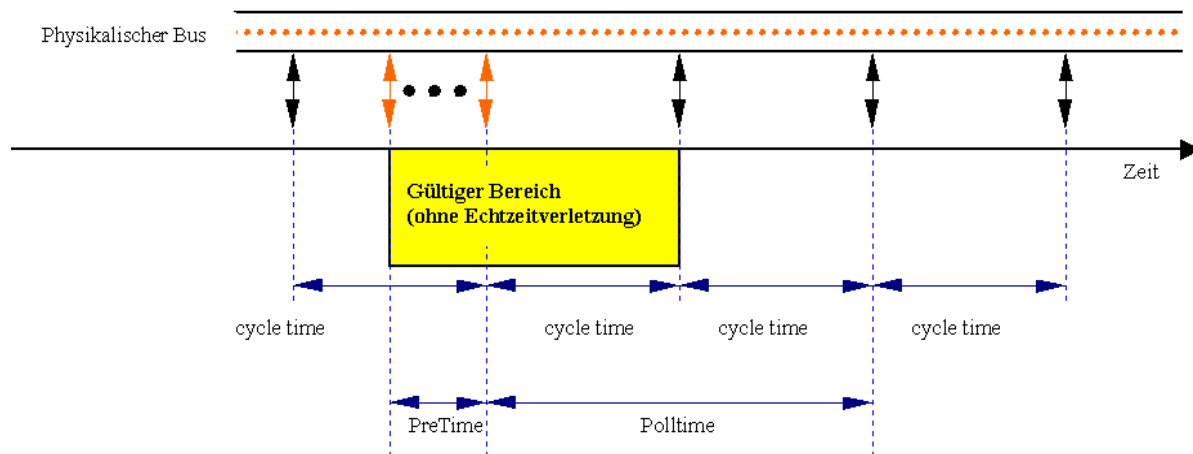


Driver_GetPreTime

DLLEXPORT ULONG WINAPI Driver_GetPreTime (ULONG cycle, ULONG databytes)

cycle Die cycle Time in ms
 databytes Anzahl der Bytes des Telegramms

Diese Funktion definiert ein zusätzliches „Toleranzband“, das die Kommunikation auf dem Netz effektiver gestaltet. Wie oben schon geschildert, wird der Bus zyklisch in der cycle time abgefragt. Erfragt sich der Anwender über EOnline nun Daten vom Gerät, so sind nach der obigen Zeichnung nur die Daten akzeptabel, die nach der Anfrage eingelesen wurden. Die von dieser Funktion zurückgegebene Zeit gibt nun an, daß auch die Telegramme noch gültig sind, die <RETURN> ms vor der Anfrage eingingen.



Wird eine Anfrage erst außerhalb den Grenzen des gültigen Bereichs beantwortet, so kommt es zu einer Echtzeitverletzung.

Driver_SizeDataList

DLLEXPORT Int Driver_SizeDataList (ULONG cycle, char * Slot)

Diese Funktion gibt die Anzahl der Telegramme zurück, die pro cycle time bearbeitet werden können.

Cycle cycle time in ms
 Slot Bezeichnung des Steckplatzes der Schnittstellenkarte

Driver_SlotTransmitData

DLLEXPORT BOOL Driver_SlotTransmitData (char* Slot, DATAPOINT_STRUCT **pdatapoint, int CountDatapoint)

Slot Bezeichnung des Steckplatzes der Schnittstellenkarte
 pdatapoint Anzahl der zu sendenden Telegramme
 CountDatapoint Feld von < CountDatapoint > Telegrammen.

Diese Funktion ist die zentralste im ganzen Treiber. Sie ist nämlich für die Kommunikation an sich zuständig. Bevor jedoch auf die Details eingegangen wird, ist es notwendig zu verstehen welche Treiber Typen es gibt.



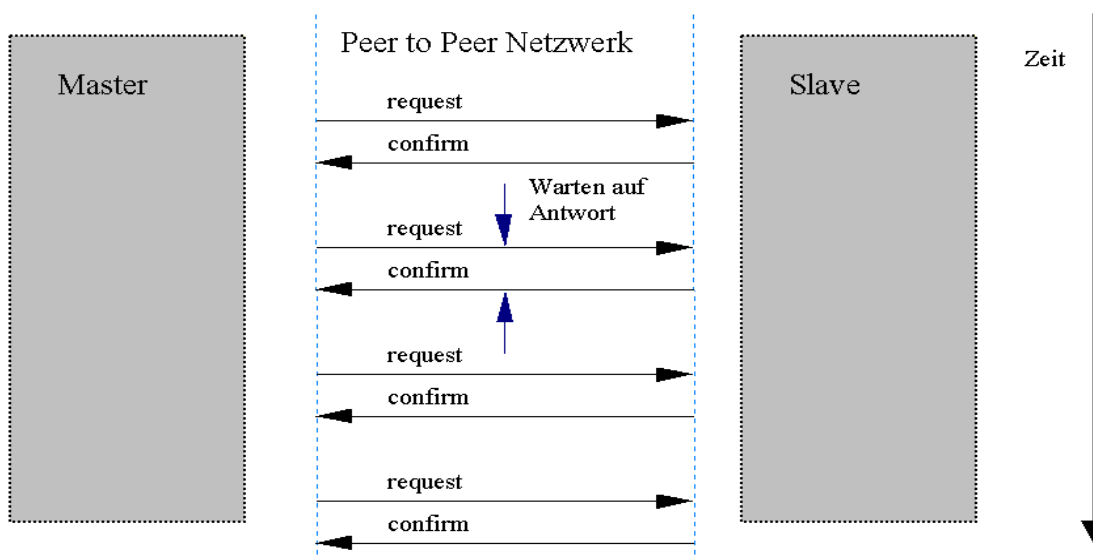
Treibertypen

Grundsätzlich von Interesse sind an dieser Stelle zwei verschiedene Arten von Treibern :

1. Treiber für Peer to Peer Netzwerke (Punkt zu Punkt Verbindungen wie z.B. RS232)
2. Treiber für Netzwerke LAN (Local Area Network wie z.B. Ethernet, CAN etc.)

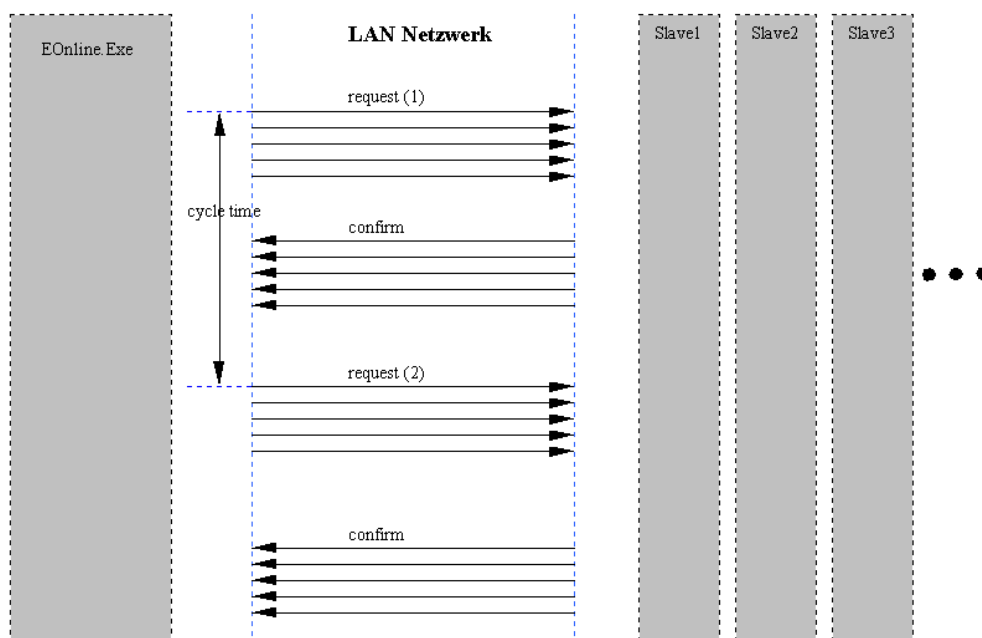
Peer to Peer:

Die grundlegendste Eigenschaft dieser Netzwerktopologie besteht darin, daß diese Kommunikation nur zwei Teilnehmer hat. Auf der einen Seite der Master, der für die Kommunikationslogistik zuständig ist und auf der anderen das Gerät aus dem Daten ausgelesen werden. Außerdem belegt eine Anfrage des Masters den ganzen Bus. Die ganze Kommunikation wird mit der Funktion Driver_SlotTransmitData nachgebildet. In der Funktion wird die Anfrage geschickt und auf eine Antwort gewartet.



Netzwerk

Hier gibt es mehrere Teilnehmer, die „gleichzeitig“ auf dem Bus kommunizieren. Mit dem EOnline Treiberkonzept sieht dies folgendermaßen aus :





Implementierung

Request wird durch die Funktion Driver_SlotTransmitData und confirm durch Driver_SlotReceiveData realisiert. Wichtig ist noch, daß nach Driver_SlotTransmitData noch eine gewisse Zeit gewartet werden muß, daß der Slave Zeit bekommt zum Antworten. Der Inhalt von pdatapoint legt die Art und Weise des Telegrammes fest:

```
typedef struct
{
    unsigned char    AddressMode;        // Adressierungsart
    ULONG           Address;            // physische Adresse
    char            *Node;              // Pointer auf Node-Name
    int             OnlineDataType;     // Datentyp auf der ElaOnline Seite (DATATYPE_xxx)
    int             TargetDataType;     // Datentyp im entspr. Target (DATATYPE_xxx)
    int             Count;              // Anzahl des Types ab Adresse
    WORD           Access;              // Zugriff (ACCESS_xx_yyy)
    unsigned char   *DataBufferRead;    // Pointer auf Pufferbereich
    unsigned char   *DataBufferWrite;  // Pointer auf Pufferbereich
    int             SizeDataBuffer;     // Pufferbereich in Bytes
    unsigned char   Send;              // angefordert
    unsigned char   Receive;           // empfangen (== ok)
    unsigned char   SemaClearDataPoint; // wenn ==1 Lesen abbrechen
    unsigned long   Timestamp;         // letzter Zeitstempel beim Lesen
    unsigned long   CountOfRead;       // Anzahl Lesevorgänge dieses Datenpunktes
} DATAPOINT_STRUCT;
```

Für nähere Informationen sehen Sie sich bitte im Beispieldriver DRIVER.DLL die Datei extern.cpp an.

Driver_SlotReceiveData

DLLEXPORT BOOL Driver_SlotReceiveData(char* Slot, DATAPOINT_STRUCT **pdatapoint, int CountDatapoint)

Slot	Bezeichnung des Steckplatzes der Schnittstellenkarte
pdatapoint	Anzahl der zu sendenden Telegramme
CountDatapoint	Feld von < CountDatapoint > Telegrammen.

Analog der Funktion Driver_SlotTransmitData ,jedoch für Netzwerke bei Confirmed Services.



Driver_GetRestartTelegram

DLLEXPORT BOOL Driver_GetRestartTelegram (char * Device, char * pMode, ULONG * pAddress, DATABUFFER4 * pData, BOOL * pCheck)

Device	Name des Gerätetreibers
pMode	Eindeutige Kennung für Restart (muß im Treiber selbst definiert sein)
pAddress	Adresse des Datenpunktes (im Prinzip zur freien Verwendung)
pData	Zeiger auf gelesene Daten
pCheck	Zusätzlicher BOOL Wert

Antwortet ein Gerät nicht mehr auf die Anfragen von EOnline, spricht man von einem Geräteausfall. An diese Geräte wird in zeitlichen Abständen, die durch Driver_GetRestartTime definiert wird, ein sogenanntes "Anklopftelegramm" gesendet. Dieses soll die Initialisierung des Gerätes bewirken. Dieses Telegramm wird durch Driver_GetRestartTelegram definiert. Es wird von dem Gerät erwartet, daß es eine Rückantwort (Confirmed Service) sendet. Diese Rückantwort wird nicht weiter verarbeitet.

Driver_GetRestartTime

DLLEXPORT ULONG Driver_GetRestartTime()

Rückgabe der Zeit in Millisekunden, die für einen Wiederanlauf verwendet wird. D.h. alle x Millisekunden wird ein Anklopftelegramm gesendet (näheres siehe Driver_GetRestartTelegram).

Driver_StartScan

DLLEXPORT BOOL Driver_StartScan(char * Slot, char * FileName, char * ParaString)

Slot	Steckplatz der Schnittstellenkarte
FileName	Name der Scan Datei
ParaString	Parameter der Initialisierung des Netzwerkes

In dieser Funktion wird die automatische Suche im Netzwerk nach aktiven Geräten realisiert. Das Ergebnis der Suche muß in eine Datei mit dem Namen < FileName > (standard : scan.out) geschrieben werden. Es ist darauf zu achten, daß ein neuer Prozeß gestartet wird. Die Datei hat folgendes Format :

```
<Kommentar>
<...>
<Kommentar>
[<Parameterstring des Drivers>]
  <Name der Device DLL>
  <Kommentar 1>
  <...>
  <Kommentar 5>
...
```

(mehr Informationen über das Thema entnehmen Sie bitte dem Beispieldriver in der Datei extern.cpp in der Funktion StartSearchThread)



Driver_ProgressScan

DLLEXPORT Driver_ProgressScan(char * Buffer, int BufferSize)

Buffer Zeiger auf den Textbuffer, der während des Scans am Autoscan Dialog dargestellt wird.

BufferSize Größe des Buffers

Rückgabe eines Statusstrings, der in CAN-Hex, während des Scanvorgangs ausgegeben wird. Wird als Text "End of Scan" zurückgegeben, interpretiert dies CAN-Hex als Ende des Scans.

Driver_EscapeScan

DLLEXPORT Driver_EscapeScan

Diese Funktion beendet den Autoscan Threat, wenn der Benutzer in CAN-Hex die Abbruch Taste drückt.

Der Gerätetreiber für Dateien DEVICE.DLL

Die hier implementierte Beispiel DEVICE kennt nur zwei Datenpunkte:

- Datei1
- Datei2

Werden Werte gelesen, oder geschrieben, so wird im Verzeichnis <SLOT>\<ParaStringDevice> jeweils die Datei Datei?.drv angelegt und die Werte geschrieben. Für die DLL gelten die gleichen Konventionen wie für die DRIVER DLL.

Device_SelectDevice

DLLEXPORT Device_SelectDevice (HWND hwnd, int language, char * Node, int SizeNode, char * ParaString, int SizeParaString)

Hwnd	Handle des Parent Fensters
language	Index der Sprache, falls der Dialog mehrsprachig erscheinen soll. Elrest intern wird der Index 0 für deutsch und 1 für englisch verwendet
Node	Gerätename
SizeNode	Länge des Gerätenamens
ParaString	Parameterstring für das Gerät (kann frei verwendet werden)
SizeParaString	Länge des Parameterstrings.

Diese Funktion wird aufgerufen um das Gerät zu spezifizieren. Jedes Gerät wird durch einen eindeutigen Namen (Node) gekennzeichnet, der in der gleichnamigen Zeichenkette hinterlegt wird. Üblicherweise wird in dieser Funktion ein Dialog geöffnet, der den Anwender das Gerät parametrieren läßt.



Device_WindowSelectDataPoint

DLLEXPORT BOOL Device_WindowSelectDataPoint(HWND hwnd, int language, char * Node, char * DataPoint, int SizeDataPoint, int * Length, char * ParaString)

Hwnd	Handle des Parent Fensters
Language	Sprach ID (0=deutsch 1=englisch)
Node	Gerätebezeichnung
DataPoint	Eindeutiger String der den Datenpunkt identifiziert
SizeDataPoint	Länge des Speicherbereiches von DataPoint
Length	Länge der Daten des selektierten Datenpunktes
ParaString	Parameterstring des Gerätes

<RETURN> TRUE wenn ein Datenpunkt ausgewählt wurde, FALSE bei Abbruch

Diese Funktion öffnet einen Dialog, der den Anwender einen Datenpunkt auswählen läßt (single selection).

Device_WindowMultiSelectDataPoint

DLLEXPORT BOOL Device_WindowMultiSelectDataPoint (int language, char *Node, char * ParaString, S_DATAPOINT *ppDataPoint[], long *count)

Language	Sprach ID (0=deutsch, 1=englisch)
Node	Gerätebezeichnung
ParaString	Parameterstring des Geräts (z.B. Dateinamen einer Map-Datei von einem Compiler + Projektgerätepfad für spez. Gerätebezogene Datenpunkte)
PpDataPoint[]	Feld auf selektierte Datenpunkte
count	Anzahl der selektierten Datenpunkte

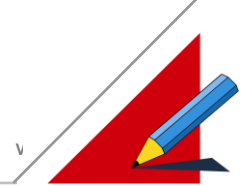
Diese Funktion öffnet einen Dialog, der den Anwender mehrere Datenpunkte auswählen läßt

(multi selection). Die Struktur S_DATAPOINT definiert einen selektierten Datenpunkt :

```
struct S_DATAPOINT
{
    char    *DataPoint;           // Name des Datenpunktes
    int     SizeDataPoint;       // Länge des allocierten Speicherbereichs
    int     Length;              // Länge der Daten des selektierten Datenpunktes
};
```

Zu bemerken ist, daß die Funktion den Speicher für die selektierten Datenpunkte mit der Windows Funktion *GlobalAlloc* bereitstellen muß. Sinngemäß wie folgt :

```
for (n=0;n<dlg.m_nSelected;n++)
{
    // String in der Struktur freigeben (GlobalAlloc ist wichtig)
    (*ppDataPoint)[n].DataPoint=(char*) GlobalAlloc(GPTR, sizeof(char)*50);
    strcpy((*ppDataPoint)[n].DataPoint,dlg.m_sdpSelected[n].DataPoint);
    delete dlg.m_sdpSelected[n].DataPoint;
    (*ppDataPoint)[n].Length=dlg.m_sdpSelected[n].Length;
    (*ppDataPoint)[n].SizeDataPoint=50;
}
```

	• DATATYPE_FLOAT	0x0080
	• DATATYPE_STRING	0x0100
	• DATATYPE_TIME	0x0200
	• DATATYPE_DATE	0x0400
	• DATATYPE_ERRORTELEGRAMM	0x0800
	// Telegramm zum Restart (darf nur ElaOnline verwenden!)	
	• DATATYPE_3BYTES	0x1000
Access	Zugriffsrechte auf den Datenpunkt. Folgende sind definiert :	
	• ACCESS_RD	0x0001
	• ACCESS_WR_CONFIRMED	0x0002
	• ACCESS_RD_WR_CONFIRMED	0x0003
	• ACCESS_WR_UNCONFIRMED	0x0004
	• ACCESS_CODE	0x0008
	• ACCESS_OLD_RAM_ADDRESSING	0x0010
ConvertFunctionRead	Konvertierungsfunktion beim Lesen des Datenpunktes	
ConvertFunctionWrite	Konvertierungsfunktion beim Schreiben des Datenpunktes. Wird bei den beiden Funktionen NULL übergeben, so werden. Diese Funktionen ignoriert.	
SigMin	Kleinster Wert des Signalbereichs. (0=ignorieren)	
SigMax	Größter Wert des Signalbereichs (0=ignorieren)	
ValueMin	Kleinster Wert des Wertebereichs (0=ignorieren)	
ValueMax	Größter Wert des Wertebereichs (0=ignorieren)	
PhysUnit	Speicherbereich für die Physikalische Einheit des Datenpunktes.	
SizePhysUnit	Größe des reservierten Speicherbereichs für die physikalische Einheit (mit terminierender \0)	
DefaultValue	Standardwert	
Length	Länge des Wertes in Anzahl Datentyp	
Check_C_Variable	Elrest interne Spezialvariable. Bitte False (=0) zurückgeben.	

RETURN:

TRUE = Der Datenpunkt ist bekannt. Die benötigten Informationen wurden der Funktion übergeben.

FALSE= Der Datenpunkt ist nicht bekannt.

Device_ReInitDevice

DLLEXPORT BOOL Device_ReInitDevice (WORD AdressMode, ULONG * PhysAdr, DATABUFFER4* Send, DATABUFFER4* Receive)

AdressMode	Adressierungsmodus für Wiederanlauf.
PhysAdr	PhysAdr (kann nach belieben verwendet werden)
Send	Telegramm, das empfangen wurde
Receive	Dieses Telegramm wird zurückgeschickt.

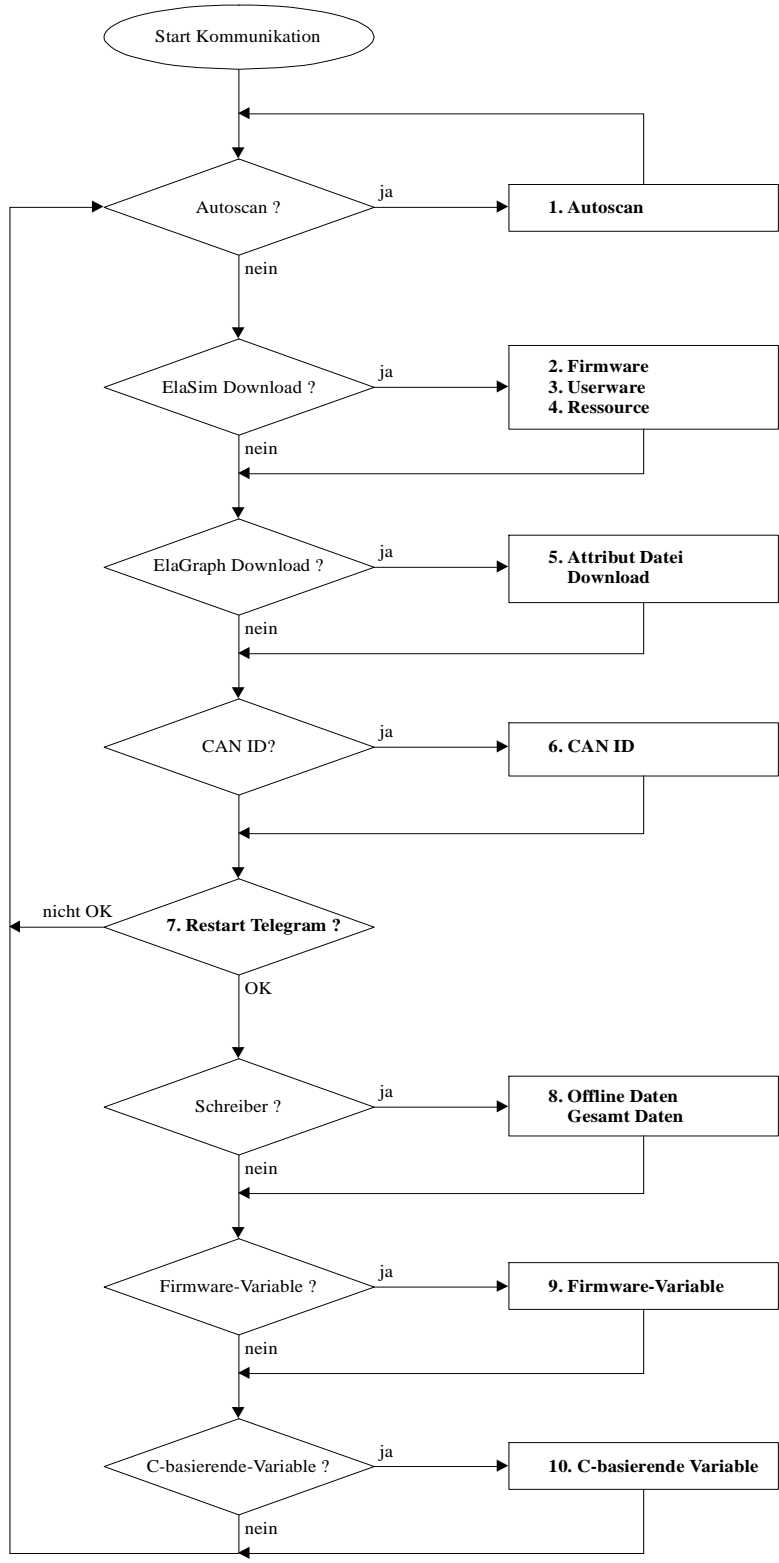
Diese Funktion wird aufgerufen, wenn ein Gerät neu initialisiert werden soll.



Dienste der Kommunikation

Der Aufbau einer Kommunikation wird über Dienste realisiert. Ein Dienst stellt eine bestimmte Funktionalität des Treibers dar.

Bearbeitung der Dienste





Prioritätenliste

Folgende Tabelle beschreibt die Dienste:

Dienste	Adressierungsmodus	Beschreibung
1. Autoscan		Automatische Suche nach Zielsystemen
2. ElaSim Firmware Download	ADRESSMODE_C_DOWNLOAD	Programmierung des Zielsystems mit der Firmware
3. ElaSim Userware Download	ADRESSMODE_C_DOWNLOAD	Programmierung des Zielsystems mit der Userware
4. ElaSim Ressource Download	ADRESSMODE_C_DOWNLOAD	Programmierung des Zielsystems mit der Ressource-Datei
5. ElaGraph Attributfile und Download	ADRESSMODE_ELAGRAPH_DIRECT	Programmierung des Zielsystems mit ElaGraph-Programmen
6. CAN ID direct	ADRESSMODE_CAN_ID_DIRECT	Kommunikation zwischen Geräten über einen definierten CAN-Identifizier
7. Restart Telegram	ADRESSMODE_DIRECT DATATYPE_ERRORELEGRAMM	
8. Schreiber	ADRESSMODE_OFFLINE_SCHREIBER ADRESSMODE_GESAMT_SCHREIBER	
9. Selbstdefinierte Datenpunkte FW-Variablen Parameter (ElaGraph) Sondertelegramme	ADRESSMODE_DIRECT	Datenaustausch von Firmware-Variablen
10. RAM-Variablen USERWARE-Variablen Speicher-Direktzugriff	ADRESSMODE_USERWARE ADRESSMODE_C_VARIABLE ADRESSMODE_ELAGRAPH	Datenaustausch von C-Variablen

Verfügbare Dienste: PC - Zielsystem

Treiber	CAN-MPC1		CAN-MPC2		RS232	
	CPU515 - V1.12 CPU515 - V1.23	CPU960 CPU167	CPU515 - V1.12 CPU515 - V1.23	CPU960 - V1.3x CPU167 - V1.4x	CPU515 - V1.12 CPU515 - V1.23	CPU960 - V1.3x CPU167 - V1.4x
Dienste	Prot. 1		Prot. 1	Prot. 1 u. 2	Prot. 1	Prot. 2
Autoscan	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ElaSim Firmware Download	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ElaSim Userware Download	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ElaSim Ressource Download	<input type="checkbox"/>	nicht	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ElaGraph Attributfile u. Download	<input type="checkbox"/>	möglich	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CAN ID direct	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Restart Telegram	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Schreiber	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Selbstdefinierte Datenpunkte	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FW-Variablen	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>



Parameter (ElaGraph)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Sondertelegramme	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
RAM-Variablen	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
USERWARE-Variablen	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Speicher-Direktzugriff	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Verfügbare Dienste: Zielsystem mit Zielsystem

Treiber	CAN Schnittstelle		RS232 Schnittstelle	
	CPU515 - V1.12/V1.23 mit CPU515 – V1.12/V1.23	CPU960 V1.30, CPU167V1.40 mit CPU960 V1.30, CPU167V1.40	CPU515 - V1.12/V1.23 mit CPU515 – V1.12/V1.23	CPU960 V1.30, CPU167V1.40 mit CPU960 V1.30, CPU167V1.40
Dienste	Prot. 1	Prot. 1 u. 2	Prot. 1	Prot. 1 u. 2
Autoscan	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ElaSim Firmware Download	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ElaSim Userware Download	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ElaSim Ressource Download	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ElaGraph Attributfile u. Download	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAN ID direct	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Restart Telegram	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Schreiber	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Selbstdefinierte Datenpunkte	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FW-Variablen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Parameter (ElaGraph)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sondertelegramme	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RAM-Variablen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
USERWARE-Variablen	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Speicher-Direktzugriff	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



Voraussetzungen, damit diverse Datenpunkte nutzbar werden

Verschiedene Datenpunkttypen	Voraussetzungen
<i>Firmware-Variablen</i>	Die Firmware Variable sind ohne weitere Schritte für die Prozessoren CPU515, CPU167 und CPU960 gleichermaßen verfügbar.
<i>Selbstdefinierte Datenpunkte</i>	Die selbstdefinierte Datenpunkte können im Zielsystem nur verwendet werden, falls in die USERWARE die Datei Uwmodul.c eingebunden ist. Siehe dazu die Beschreibung Uwbase.lib im Handbuch von ElaSim.
<i>Parameter ElaGraph</i>	Die Parameter ElaGraph können nur verwendet werden, falls in die USERWARE der ElaGraph Kernel eingebunden ist. Siehe dazu die Beschreibung Graph.lib im Handbuch von ElaSim.
<i>Sondertelegramme</i>	Die Sondertelegramme können nur verwendet werden, falls in der USERWARE in die Routine „can_rcv_telegram()“ ein entsprechender C-Code eingetragen wurde, der die Speicherung dieser Kommunikationsdatenpunkte erlaubt.
<i>RAM-Variablen</i>	Der direkte Zugriff auf Speicheradressen zwischen verschiedener Zielsysteme ist aus Sicherheitsgründen unterbunden.
<i>USERWARE-Variablen</i>	Der direkte Zugriff auf Speicheradressen zwischen verschiedener Zielsysteme ist aus Sicherheitsgründen unterbunden.
<i>Speicher-Direktzugriff</i>	Der direkte Zugriff auf Speicheradressen zwischen verschiedener Zielsysteme ist aus Sicherheitsgründen unterbunden.
<i>Layer 2 CAN Telegramme</i>	Nicht möglich.



Schichtenmodell

Der Austausch von Dateneinheiten zwischen Partnerinstanzen im Schichtenmodell der Datenkommunikation ist in folgender Grafik abgebildet:

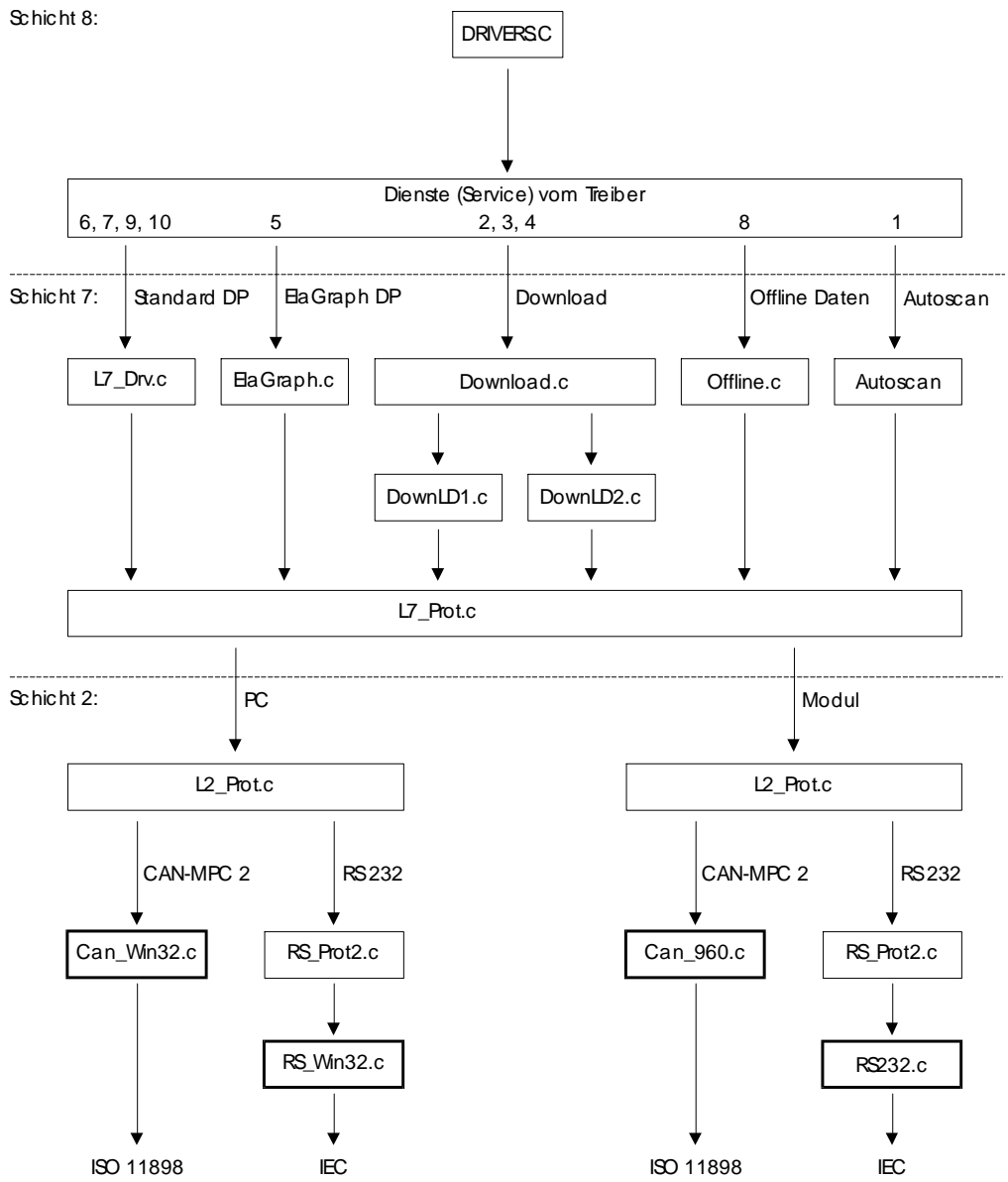


Abbildung: Export_Funktionen



RS232 Schnittstelle

Über den RS232 Netzwerktreiber

Der überwiegende Teil der Peripheriegeräte für PCs und überhaupt für Computer nutzt eine ganz besondere serielle Schnittstelle : Die RS232 Schnittstelle. Sie ist wahrscheinlich die universellste und am meisten verbreitete Schnittstelle, die es im Computerbereich gibt. Jeder PC hat mindestens eine davon, allermeistens sogar zwei. Unter Windows lassen sich bis zu vier solcher Schnittstellen verwenden. Ursprünglich wurde diese Schnittstelle dazu entwickelt, um über ein Modem mit anderen Rechnern zu kommunizieren.

Der Vorzug der RS232 Schnittstelle liegt darin, daß nicht einfach nur Daten übertragen werden. Es kann zusätzlich eine Paritätsprüfung stattfinden, d.h. ein übertragenes Wort wird auf seine Richtigkeit hin überprüft. Zudem beinhaltet die RS232 Schnittstelle zahlreiche zusätzliche Steuer- und Meldeleitungen.

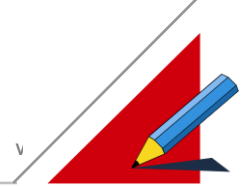
Der Treiberdialog

Zum Verbindungsaufbau mit der RS232 Schnittstelle zwischen PC und einem CAN-Modul ist der Schnittstellen-Treiber **RS232.dll** notwendig. Die serielle Schnittstelle ist interruptgesteuert und mit einem Empfangsspeicher von 256 Bytes wie folgt ausgeführt:

Die Parametrierung des Netzwerks erfolgt über folgenden Dialog:



- ComPort:** Nummer der RS232 Schnittstelle (maximal 4 Stück werden von Windows unterstützt). Standard ist die COM2 (Normalerweise wird über COM1 die Maus angesteuert)
- Baudrate:** Die Übertragungsrate von 110 Baud bis 256000 Baud ¹⁾
- Datenbits:** Anzahl der Bits in einem Telegramm, die für reine Benutzerdaten reserviert sind. Standard sind 8 Bits.
- Parität:** Ist die Geradzahligkeit der Anzahl der zu 1 gesetzten Datenbits im Datenbitblock. Standard : ‚keine‘ ²⁾
- Stopbits:** Kennzeichnet das Ende eines RS232 Telegramms. Hier stellen Sie die Anzahl und Länge des beendenden Telegrammimpulses ein. Standard ist 1Bit
- Routing:** Standard : nicht aktiviert(Siehe ³⁾)



Da Sender und Empfänger immer mit der gleichen Netzwerkeinstellung parametrieren müssen um miteinander kommunizieren zu können, ist es wichtig die richtigen Parameter auszuwählen. Bei Elrest sind alle Geräte werksseitig auf folgender Einstellung parametrieren :

- ComPort : COM_2
- Baudrate : 9600Baud
- Datenbits : 8Bits
- Parität : keine
- Stopbits : 1Bits
- Routing : nicht aktiviert

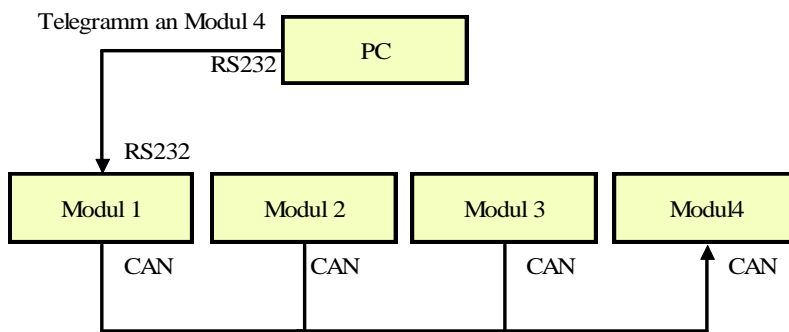
¹⁾**Definition Baud nach ANSI** : Maßeinheit für die Schrittgeschwindigkeit : 1/Sekunde. Der Schritt ist der vereinbarte kürzeste Abstand zwischen aufeinanderfolgenden Übergängen des Signalzustandes. Bei Binärer Übertragung ist die Schrittgeschwindigkeit (Baud) gleich der Übertragungsgeschwindigkeit (Bit/s).

²⁾Parität:

- Gerade Paritätsbit ist gesetzt, wenn die Anzahl der zu 1 gesetzten Datenbits gerade ist.
- Marke Als Parität wird immer eine 1 ausgegeben.
- Keine Paritätsbit wird nicht verwendet (Telegramm ist somit ein Bit kürzer und schneller)
- Ungerade Paritätsbit ist gesetzt, wenn die Anzahl der zu 1 gesetzten Datenbits ungerade ist.

³⁾Routing:

Da über die RS232 nur mit einem Gerät kommuniziert werden kann („peer to peer“ Verbindung), ergibt sich unter Umständen eine Redundanz der Geräteadressierung. D.h. das Gerät das mit dem PC verbunden ist kommuniziert evtl. über eine andere Schnittstelle mit weiteren Geräten. Ist das Routing ausgewählt, so wird das Telegramm das über die RS232 empfangen wurde, an das Gerät weitergeleitet, welches über den Gerätetreiber adressiert ist. So ist es möglich alle Geräte in einem Netzwerk über eine Schnittstelle zu erreichen. Ist das Routing ausgeschaltet, so nimmt das Modul immer das empfangene Telegramm entgegen :



Der Telegrammverkehr zwischen PC und Modul 4 erfolgt über Modul 1. Das Modul 1 wirkt wie ein virtuelles Modul. Es empfängt Daten vom PC (mit der Adresse von Modul 4) und leitet diese Daten an das Modul 4 weiter. Als Antwort empfängt Modul 1 die Daten vom Modul 4 (mit der eingestellten Routing) und leitet diese Daten an den PC weiter.

Mögliche Geräteadressierung:

Protokoll 1.0	Protokoll 2.0	
	Single- Master Betrieb	Multi- Master Betrieb
M0 bis M63	M0 bis M63	M0 bis M(Interm.Master)

Hinweis:

Routing ist ab Firmware Version 1.30 verfügbar. Alle Module im Netzwerk müssen mit mindestens dieser Firmware Version bestückt sein.



Für das Routing wird automatisch die Modulnummer 63 vergeben. Diese Modulnummer darf im Netzwerk nicht vergeben werden. Bei einem Autoscan nicht diese Modulnummer erkannt, da sie separat behandelt wird. Routing ist pro Netzwerk ein mal möglich.

Übertragungsraten:

<i>CPU515</i>	<i>CPU167</i>	<i>CPU960</i>
9600 Baud	9600 Baud	9600 Baud
	38400 Baud	

Die minimale Lesezykluszeit beträgt:

<i>Übertragungsrate</i>	<i>min. Lesezykluszeit</i>
38400 Baud	100 ms
9600 Baud	500 ms

Die maximale Anzahl der Telegramme / Sekunde beträgt in Abhängigkeit der Übertragungsrate:

<i>Übertragungsrate</i>	<i>Telegramme / Sekunde</i>
38400 Baud	70
9600 Baud	15

Download und dabei andere Geräte deaktivieren

Hinweis:

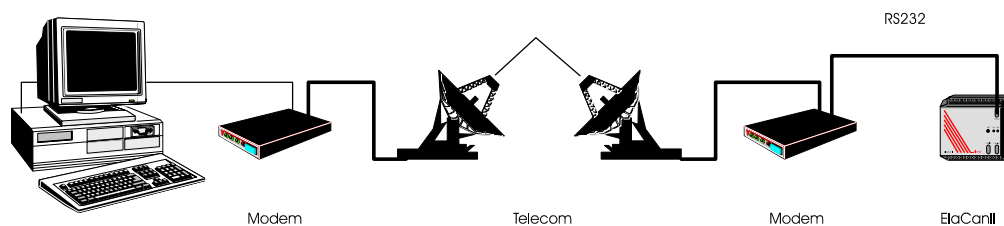
Ein Download auf ein Zielsystem kann unter diversen Umständen fehlschlagen.
 Für die CPU515 genügt es das Modul in der Servicestellung aus- und einzuschalten.
 Für die CPU167 und CPU960 muß das Modul ebenso in der Servicestellung aus- und einzuschalten werden und zusätzlich mit dem Programm „can_hex.exe“ die USERWARE gelöscht werden.
 Tritt bei den Modulen mit CPU167 und CPU960 der Fall ein, daß die FIRMWARE gelöscht wurde, bleibt nur das Einsenden des Moduls ins Werk Elrest.

Soll ein ElaGraphII Projekt in eine Zielhardware gebracht werden, so wird ein sogenannter „Download“ auf das Zielsystem mit den entsprechenden Daten durchgeführt. Dies kann bei laufendem Betrieb an einer Anlage zu Problemen führen, weil in der Regel im Netzwerk noch andere Geräte kommunizieren. Dies hat zur Folge, daß die anderen Geräte den Download stören. Um dies vermeiden zu können, gibt es eine Option, die bewirkt, daß der Treiber allen anderen Geräten bei einem Download eine Meldung schickt, die bewirkt, daß alle Geräte während dem Vorgang die Kommunikation unterbrechen.



Modemverbindung

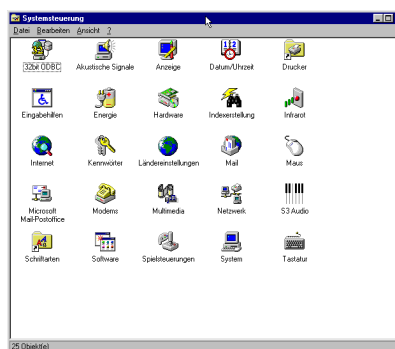
Zum Verbindungsaufbau zwischen PC und CAN- Modulen über eine analoge Telefonleitung ist der Schnittstellen- Treiber **RS232.dll** notwendig. Daten können somit über die serielle Schnittstelle übertragen werden.



Softwareeinstellungen auf PC Seite

Diese Modem Schnittstelle wurde mit der original Windows Modem TAPI Schnittstelle realisiert.

Dazu muß unter dem Thema „Systemsteuerung“ das Modem konfiguriert werden.



Innerhalb des Modem Dialoges sollten Sie für jedes verwendete Modem ein entsprechenden TAPI-Treiber installieren.

Über den Knopf Hinzufügen – Keine autom. Erkennung – können Sie aus eine Liste zu Verfügung stehende TAPI-Treiber des jeweiligen Modems auswählen. Außerdem können externe Treiber über Disketten,... geladen werden.

Über die autom. Erkennung können Sie versuchen, das Modem über das Betriebssystem suchen zu lassen.

Besitzen Sie keinen Treiber, können Sie einen Standardtreiber auswählen und versuchen, diesen zu verwenden. Da aber jedes Modem andere Einstellbefehle verwendet, kann dieser Versuch fehlschlagen.

Getestet wurden die Tapi-Treiber unter Windows 95 für die Modems:

- Creatix SG 2834 Turbo mit der frei erhältlichen Treiberinformation mdmctx1.inf
- Robotics Sportster Voice 33.6k Faxmodem mit dem vorgeschlagenen Defaulttreiber
3Com: US Robotics 33.6k Voice EXT
- 3COM US Robotics 56K Faxmodem mitgelieferter Treiber
- Dr. Neuhaus Smarty 19.2 TI mit dem vorgeschlagenen Defaulttreiber
Dr. Neuhaus: Dr. Neuhaus SMARTY 19.2 TI, 19.2 Voice





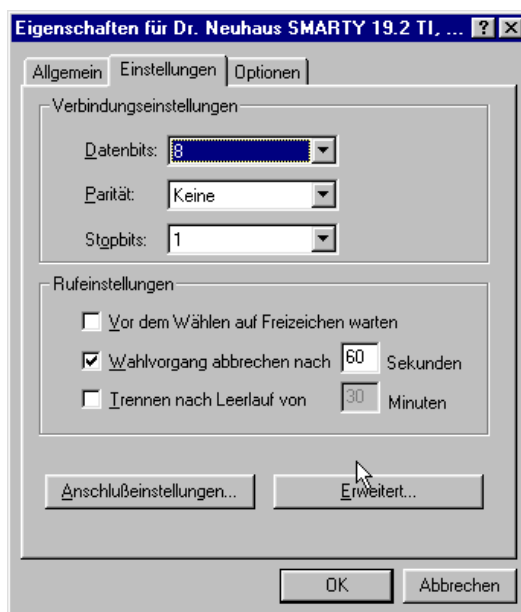
Stellen Sie die Ihren PC im Standortdialog korrekt auf Ton- oder Impulswahl ein.

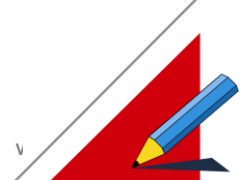


Weitere Einstellungen können Sie über die Eigenschaftsdialoge vornehmen.

Geben Sie unter Allgemein die verwendete COM-Schnittstelle an und unter maximale Geschwindigkeit die mögliche Geschwindigkeit zwischen Rechner und Modem. Bei neuen Systemen sollte 115200Baud verwendbar sein.

Bei Nebenstellenanlagen erhalten Sie u.U. kein Freizeichen nach dem Abheben. Deaktivieren Sie in den Rufeinstellungen das entsprechende Auswahlfeld.





Anwahl auf PC Seite

Im Anwendungsprogramm ist als Netzwerk der RS232-Treiber auszuwählen. In den Parametereinstellungen wechseln Sie auf die Seite „Modem Einstellungen“ und können hier Parameter wie Telefonnummer, Automatisches Trennen von Verbindungen bei Unterbrechungen anwählen. Bei 0 Minuten ist diese Funktionalität deaktiviert.

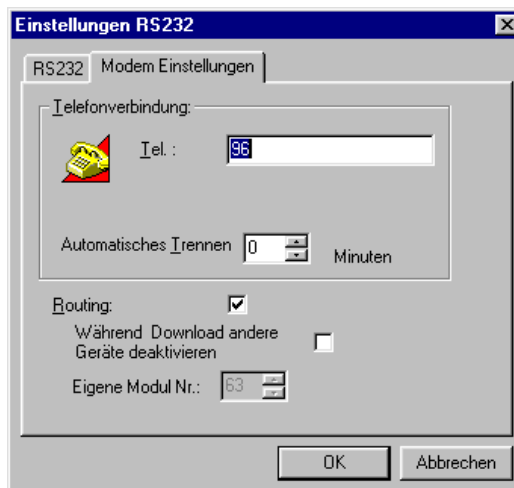
Nach Initialisierung des Netzwerks kann bereits eine Modemverbindung aufgebaut werden.

Die weitere Modemverbindung können Sie nach Initialisierung durch Aufruf des Modemservers beeinflussen. Hierzu wählen Sie in der Taskleiste das Modem-Server Icon per Doppelklick an:



Über diesen Kontrolldialog können Sie Verbindungen vornehmen, trennen, die Telefonnummer ändern und Eigenschaften der verwendeten Tapi-Verbindung ändern.

Hier erfolgt im Moment auch die Auswahl der verwendeten Tapi-Schnittstelle. Wählen Sie hierzu die entsprechende Telefonverbindung (Line) aus.

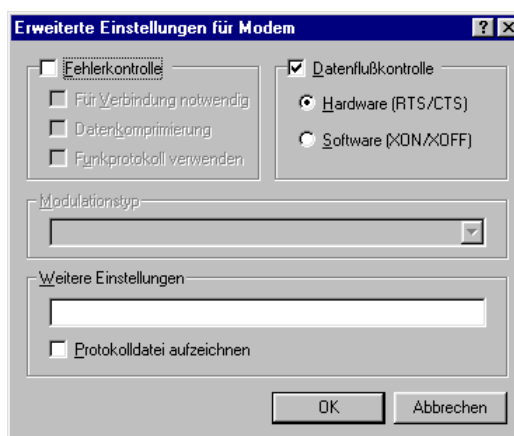


Achtung!

Beachten Sie, daß in den erweiterten Eigenschaften der Telefonverbindung (Line) als Datenflußkontrolle RTS/CTS oder „Keine“ ausgewählt ist. XON/XOFF ist nicht zulässig!

Außerdem konnten wir bei einigen Modems Fehler in der Datenkomprimierung (Fehlerkontrolle) feststellen. Schalten Sie diese daher unbedingt aus! Für die Performance der Datenübertragung ist diese auch nicht relevant, da das Nadelöhr die Modem-Modul Verbindung ist.

Nebenstehend ist der Einstellungsdialog eines Dr. Neuhaus Smarty-Modems dargestellt. Bei anderen Modems kann dieser Dialog anders aussehen.





Während dem Verbindungsaufbau erscheint dieses Fenster.



Hinweis:

Die Modem-Fähigkeit in der RS232.dll ist eine Option und somit nicht im normalen Lieferumfang enthalten, siehe dazu Freischaltung Ihres Dongles.



Softwareeinstellungen am Modul

Es muß ein Modem auf der Gegenseite verwendet werden, das seine Einstellungen remanent speichert. Die eingestellten Parameter werden in einem Benutzerprofil gespeichert. Dieses wird nach einem Reset automatisch geladen.

Da die Module über eine 3Draht RS232 Leitung verfügen, ist das Modem so einzustellen, daß es für das Modul als normale RS-Schnittstelle sichtbar ist.

Steuerleitungen müssen deaktiviert werden, sowie Meldungen des Modems unterdrückt werden. Die Übertragungsgeschwindigkeit (Baudrate) des Moduls ist fest auf dem Modem einzustellen.

Hinweis:

- **Durch die verwendete 3Draht Leitung nehmen die Modems bei Anruf trotz autom. Abnahme nicht ab. Der Pin 7 ist mit Pin 8 daher zu brücken! (9pol. Stecker)**
- **Ist das Modul direkt per RS232 mit dem Rechner verbunden, findet die Drehung des RxD/TxD Signals im Rechner statt. Das Gleiche gilt bei der Modem-Rechner Verbindung. Bei der Modul-Modem Verbindung ist die Drehung in die Leitung einzufügen (Pin 2/3 drehen).**

Änderungen an einem Benutzerprofil des Modems können mittels Terminalprogramm (z.B. Hyperterminal) vorgenommen werden. Dazu ist das Terminalprogramm auf die Parameter 9600/8/1/N einzustellen.

Im einzelnen sind folgende Änderungen am Modem vorzunehmen:

Befehl	Beschreibung	Bemerkung
AT&F0	Werkseinstellung laden	Reset auf Lieferzustand des Modems Profil 0
ATS0=1	Automatische Abnahme nach einmal klingeln	
AT&D0	DTR- Status ignorieren	Achtung! In Österreich nicht zulässig. Hier ist der Pegel des betreffenden Pins 4 auf dem 9pol. Stecker auf High zu setzen (Brücke mit Pin 8)
AT&K0 (Creatix) AT\Q0 (Dr. Neuhaus) AT&R1 (Robotics)	Hardware Handshake mit dem Modul aus	
AT&Y0 (Creatix, Dr. Neuhaus) ATY0 (Robotics)	Profil 0 bei Reset/Einschalten laden	
Default OK (Creatix, Dr. Neuhaus, ansonsten S23 Bit1-3 auf 5 setzen bei beiden Modems) AT&N6 (Robotics)	Feste Übertragungsgeschwindigkeit Modem-Modul einstellen auf 9600 bits/s	
...	Eigene spezielle Einstellungen	

Zum Schluß:

ATE0	Echo aus	
ATQ1	Rückmeldungen aus	Vom Modem erfolgen jetzt keine Rückantworten mehr, also auch nicht OK
AT&W0	Unter Profile 0 speichern	Blind eingeben, OK erfolgt nicht mehr
ATZ0 oder Modem aus- /einschalten	Lädt Profile 0	Blind eingeben, OK erfolgt nicht mehr



Weitere:

ATS30=x (Creatix) ATS19=x (Robotics) - (Smarty)	HW-Timer zur Verbindungsunterbrechung bei Inaktivität Modem-Modem Default=0: deaktiviert Sonst x in Minuten	Diese zusätzliche Trennmöglichkeit durch das Modem ist sehr sinnvoll. Falls die betreffenden Module/Programme nicht mehr reagieren und keine Daten ausgetauscht werden, übernimmt das Modul die Trennung.
AT&V (Creatix) AT&V0 (Dr. Neuhaus) ATI4/ATI5 (Robotics)	Gibt die Einstellungen von Profile 0 aus zur nochmaligen Kontrolle.	
- (Creatix) - (Dr. Neuhaus) ATI3/ATI7/ATI9 (Robotics)	Produktinformationen ausgeben	Robotics meldet mit ATI7 u.a. Länderversion

Die Funktion wurde mit folgenden Modems getestet:

- Modem der Fa. Creatix: SG 2834 Turbo
- Modem der Fa. Robotics (3Com): Sportster Voice 33.6K
Modem der 3COM: 3COM US. Robotics 56K Faxmodem
- Modem der Fa. Dr. Neuhaus: Smarty 19.2 TI

Alle andere Fabrikate sollten zur Funktionsprüfung von der Fa. Elrest geprüft und freigegeben werden.

Sonstiges:

- Das Creatix-Modem hat Probleme nach einem Verbindungsabbruch sich zu reseten. Es kann deshalb nicht gleich angerufen werden. Hier muß in der Regel 1-2 Minuten gewartet werden.
- Die Modems sind in verschiedenen Ländern mit Wahlsperren belegt, falls mehrfach keine Verbindung zustande kam. Danach ist eine längere Zeit zu warten. Deaktiviert werden kann dies durch Aus-/Einschalten des Modems.
 - Deutschland: Nach 12* wählen Pause von 2h
 - Schweiz: Nach 12* wählen Pause von 6h
 - Österreich: Nach 2* wählen Pause von 60sec, nach 12* wählen voll gesperrt

Download und dabei andere Geräte deaktivieren

Hinweis:

Ein Download auf ein Zielsystem kann unter diversen Umständen fehlschlagen. In dieser Fall erzwingt dies ein Vorort gehen an das Zielsystem.

Für die CPU515 genügt es das Modul in der Servicestellung aus- und einzuschalten.

Für die CPU167 und CPU960 muß das Modul ebenso in der Servicestellung aus- und einzuschalten werden und zusätzlich mit dem Programm „can_hex.exe“ die USERWARE gelöscht werden.

Tritt bei den Modulen mit CPU167 und CPU960 der Fall ein, daß die FIRMWARE gelöscht wurde, bleibt nur das Einsenden des Moduls ins Werk Elrest.

Soll ein ElaGraphII Projekt in eine Zielhardware gebracht werden, so wird ein sogenannter „Download“ auf das Zielsystem mit den entsprechenden Daten durchgeführt. Dies kann bei laufendem Betrieb an einer Anlage zu Problemen führen, weil in der Regel im Netzwerk noch andere Geräte kommunizieren. Dies hat zur Folge, daß die anderen Geräte den Download stören. Um dies vermeiden zu können, gibt es eine Option, die bewirkt, daß der Treiber allen anderen Geräten bei einem Download eine Meldung schickt, die bewirkt, daß alle Geräte während dem Vorgang die Kommunikation unterbrechen.



CAN Schnittstelle

Über CAN

Neben dem Einsatz in Personen und Nutzfahrzeugen aller Art werden CAN-Netzwerke heute in zahlreichen industriellen Anwendungen als Maschinen- oder systeminternes Kommunikationssystem eingesetzt. Das ursprünglich für den Einsatz in Kraftfahrzeugen entwickelte „Controller-Area-Network (CAN)“-Protokoll basiert auf dem Prinzip eines Nachrichtenverteilsystems und ermöglicht auf Grund seiner besonderen Leistungsmerkmale sowie der kostengünstigen Protokollbausteine eine Vielfalt neuartiger Systemlösungen.

Der Treiberdialog

Zum Verbindungsaufbau zwischen PC und CAN-Modulen ist der Schnittstellen-Treiber **CAN_MPC1.dll** oder **CAN_MPC2.dll** notwendig. Es existieren zwei voneinander unterschiedliche Ausführungen dieser Schnittstelle:

- ElaCAN I, basierend auf RS-485 Treibern.
- ElaCAN II, nach Eintritt in die CiA Organisation wurden alle Geräte dieser Gerätereihe auf die genormte CAN ISO 11898 Schnittstelle ausgeführt.

Bei der Parametrierung des Netzwerks zeigt sich folgendes Bild :



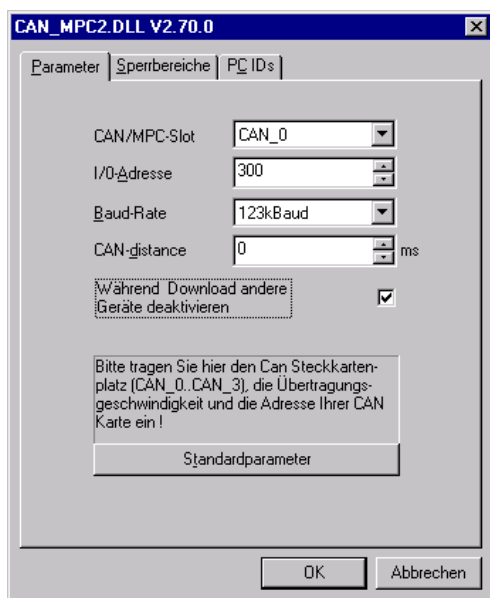
Parameter : Elrest Protokoll unabhängige Parametrisierung eines Steckkartenplatzes

Sperrbereiche : Ausmaskieren von Telegramm Bereichen

PC IDs : Parametrierung des Elrest Protokolls



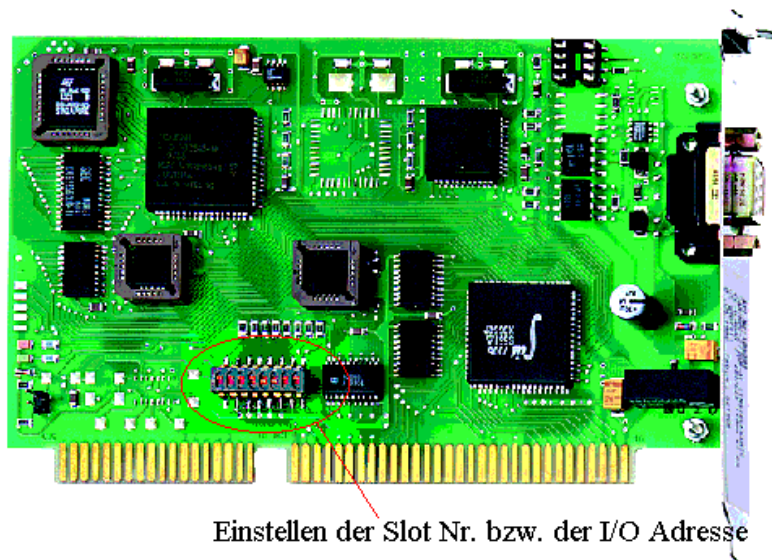
Die „Parameter“ Seite des Treiberdialogs



CAN/MPC-Slot und I/O-Adresse

Über die Einstellungen CAN/MPC-Slot und I/O Adresse wird der Steckkartenplatz der Schnittstellenkarte CAN-MPC2 von Elrest adressiert.

Generell ist es möglich bis zu 4 Einsteckkarten gleichzeitig in einem PC zu betreiben, dazu ist es jedoch notwendig, die Karten so einzustellen, daß sie sich gegenseitig nicht behindern. Dies erfolgt durch Einstellen von DIP Schaltern auf der Schnittstellenkarte CAN-MPC2.



Einstellen der Slot Nr. bzw. der I/O Adresse

VORSICHT : Die DIP Schalter werden mit negativer Logik betrieben. D.h. DIP Schalter „ON“ bedeutet logisches Signal „0“ und DIP Schalter „OFF“ logisches Signal „1“.



Für die Einstellung gelten folgende Regeln :

Einstellung des Karten Slot :

<i>Slot</i>	<i>DIP Schalter 8</i>	<i>DIP Schalter 7</i>
<i>CAN_0</i>	ON=0	ON=0
<i>CAN_1</i>	ON=0	OFF=1
<i>CAN_2</i>	OFF=1	ON=0
<i>CAN_3</i>	OFF=1	OFF=1

Einstellung der I/O Adresse :

<i>I/O Adresse</i>	<i>DIP Schalter 6</i>	<i>DIP Schalter 5</i>	<i>DIP Schalter 4</i>	<i>DIP Schalter 3</i>	<i>DIP Schalter 2</i>	<i>DIP Schalter 1</i>
<i>320h</i>	OFF=1	OFF=1	ON=0	ON=0	OFF=1	ON=0
<i>300h</i>	OFF=1	OFF=1	ON=0	ON=0	ON=0	ON=0
<i>280h</i>	OFF=1	ON=0	OFF=1	ON=0	ON=0	ON=0
<i>240h</i>	OFF=1	ON=0	ON=0	OFF=1	ON=0	ON=0
...

Aus der Angabe des Slots und der I/O Adresse können sich eine Reihe von Problemen bezüglich Doppelbelegung ergeben. Von einer anderen Einsteckkarte kann die Adresse bereits belegt worden sein. Ein typischer Fall ist die Verwendung von einer Netzwerkkarte auf Adresse 300h.

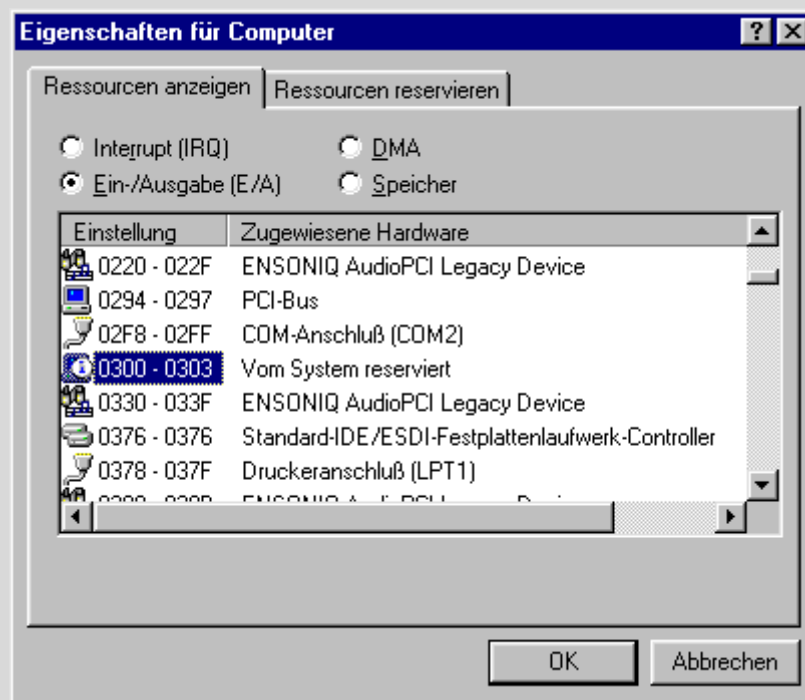
Falls die gleiche Adresse verwendet wird, ist die CAN-Verbindung auf dieser Karte mit einer großen Fehlerrate behaftet und sie erhalten eine äußerst schlechte Verbindung mit Geräteaussetzern.

Zur Lösung bieten sich zwei Möglichkeiten an:

1. Sie ändern die Adresse auf einen anderen unbenutzten Bereich.
2. Sie können versuchen, die I/O – Adresse exklusiv für die Karte zu reservieren. Damit ist der I/O-Bereich für andere Hardware-Komponenten gesperrt. In der Regel stellt sich z.B. eine Netzwerkkarte dann dynamisch selbst auf einen neuen Bereich ein. Verwenden Sie Hardware, die die Adresse fest verwendet, müssen Sie dort die Adresse manuell anpassen.

Reservierung unter Windows95:

Unter „Start – Einstellungen – Systemsteuerung – System“ wählen Sie die Seite Gerätemanager an. Dort wählen Sie „Computer“ aus und klicken auf den Knopf „Eigenschaften“. Mit der Auswahl „Ein-Ausgabe (E/A)“ erhalten Sie eine Übersicht über die belegten I/O Adressen:



Suchen Sie nach dem Bereich, den Sie exklusiv sperren wollen. Ist dieser frei können Sie den Bereich belegen. Führen Sie die notwendigen Schritte unter der Seite „Ressourcen reservieren“ durch.

Der Eingabebereich umfaßt bei der CAN-MPC2 Karte immer 4h also 300-303 (Slot 1), 700-703 (Slot 2: 1*400h+Adresse), B00-B03 (Slot 3), F00-F03 (Slot 4) bei Adressverwendung 300h.

(Der Eingabebereich bei der CAN-MPC1 Karte umfaßt dagegen immer 16h also 300-30F, 700-70F, B00-B0F, F00-F0F. Hier kann jedoch keine weitere Adresse ausgewählt werden, sondern nur der Slot.)

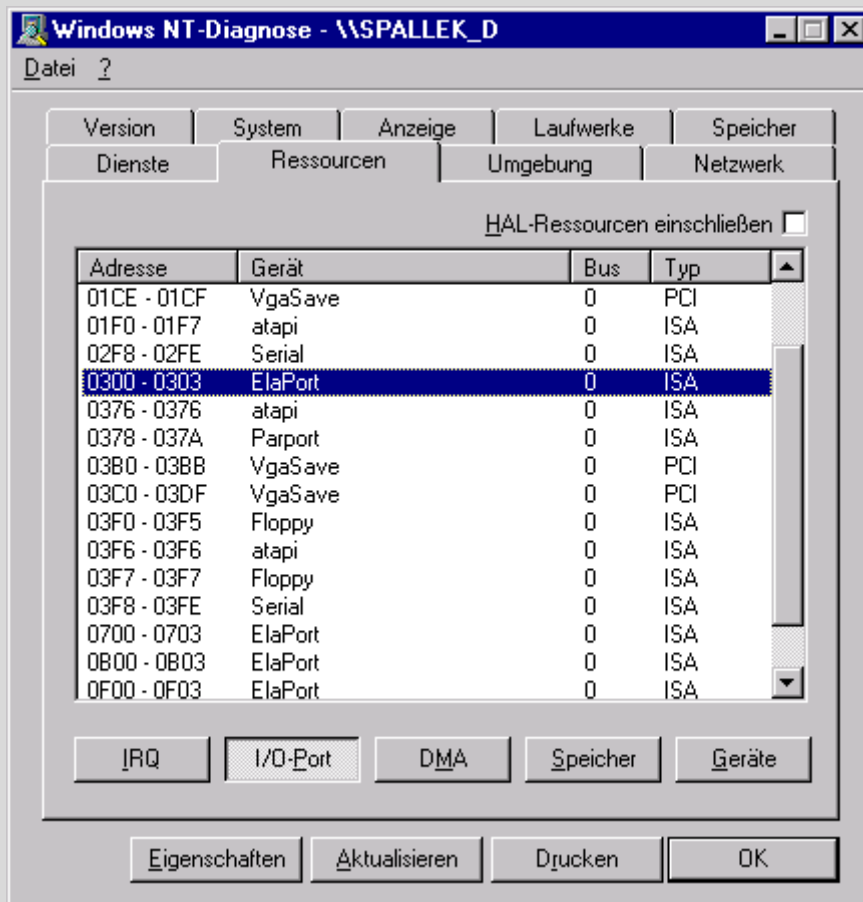
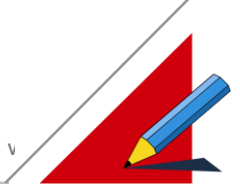
Ist der gewünschte Bereich bereits durch ein anderes Gerät exklusiv belegt worden (Text „Vom System reserviert“, müssen Sie Ihre Karte auf einen anderen I/O Bereich einstellen. Liegt an der Adresse ein anderes Gerät, können Sie versuchen die Adresse zu reservieren. Sie müssen allerdings nach Neustart überprüfen, daß sich das bisherige Gerät auf einen anderen Bereich gelegt hat, ansonsten könnte dieses nicht mehr funktionieren.

Einstellung unter Windows NT:

Unter Windows NT müssen die verwendeten Adressbereiche immer reserviert werden. Dies wird bereits bei der Installation automatisch mit der Installation des CAN-Systemtreibers ElaPort und später durch den CAN-MPC1/2 Gerätetreiber durchgeführt.

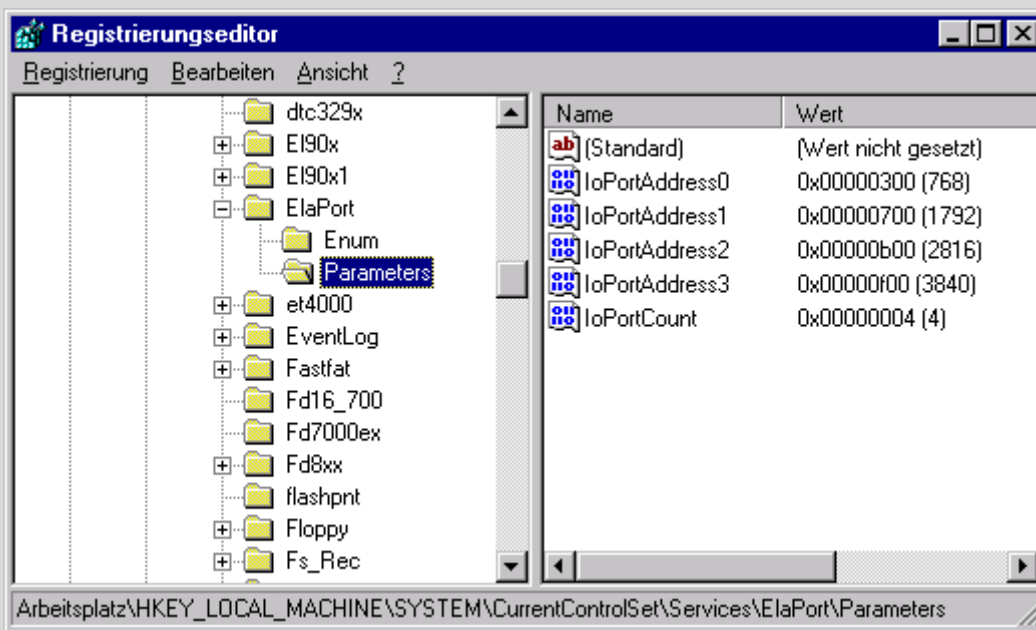
Zunächst sind diese Adressen bis auf eine defaultmäßig deaktiviert (Adresse 000h). Verändern Sie den Slot und die Adresse im Treiberdialog, wird der neue Bereich reserviert und das System muß neu gestartet werden.

Unter „Start – Programme – Verwaltung (Allgemein) – Windows NT-Diagnose“ können Sie sich die verwendeten I/O Adressen anzeigen lassen. Die von ElaGraphII verwendeten Bereiche sind mit „ElaPort“ gekennzeichnet:



Unter bestimmten Umständen kann es dazu führen, daß die Adressreservierung im CAN MPC1/2-Treiber nicht automatisch durchgeführt werden kann, weil eine andere Hardwarekomponente die Reservierung blockiert. Hier müssen Sie den Adressbereich manuell anpassen.

Ermitteln Sie dazu in der NT-Diagnose einen freien Bereich und stellen Sie diesen auf der Karte ein. Laden Sie den Registry-Editor durch „Start – Ausführen – Regedit“. Öffnen Sie den Schlüssel „HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ElaPort\Parameters“.





Die verwendeten Start-Adressen sind unter IOPortAddress0-3 eingetragen. Die Endadresse wird automatisch errechnet. Passen Sie die Startadresse an. Doppelte Adressverwendungen sind nicht zulässig!
Nach einem Neustart sollte der neue Bereich reserviert sein.

Übertragungsrate

Hier stellen Sie die Übertragungsgeschwindigkeit in Baud ¹⁾ ein. Die von Elrest ausgelieferten Module werden standardmäßig mit einer Übertragungsrate von 123kBaude ausgeliefert.

Unterstützte Übertragungsraten:

- 500 Kbaud
- 250 Kbaud
- 125 Kbaud
- 123 Kbaud
- 100 Kbaud
- 50 Kbaud
- 20 Kbaud
- 10 Kbaud

Hinweis:

Übertragungsraten von 10, 20, 50 und 500 kBaude sind nur in Verbindung mit der Schnittstellenkarte Can-MPC 2 ab Version 2.08 möglich

Voraussetzungen für eine ordnungsgemäße Kommunikation :

- Für die Kommunikation mit dem PC ist die Schnittstellenkarte MPC-2 notwendig
- Die physikalische Schnittstelle muß entweder nach Norm ISO 11898 mittels 9-poligem SUB-D Stecker oder nach Elrest- Standard mit RS485 Treiber ausgeführt sein.
- Die Übertragungsrate muß bei allen Teilnehmern identisch eingestellt sein.
- Es dürfen keine CAN-Teilnehmer mit demselben CAN-Identifizier innerhalb eines Segmentes vorhanden sein! Das bedeutet es dürfen nicht mehrere Module mit derselben Moduladresse zusammengeschaltet werden. Bei Verwendung von Fremdkomponenten dürfen sich die CAN-Identifizier keinesfalls mit denen der Fa. Elrest überschneiden.
- Die I/O-Adresse, die Sie auf Ihrer Karte eingestellt haben, darf im System nicht anderweitig – z.B. in einer Netzwerkkarte (typischer Fall!) - verwendet worden sein! Wie Sie dies überprüfen und korrigieren können entnehmen Sie bitte der Beschreibung „Einstellung der I/O Adresse“.

¹⁾Definition Baud nach ANSI : Maßeinheit für die Schrittgeschwindigkeit : 1/Sekunde.; Der Schritt ist der vereinbarte kürzeste Abstand zwischen aufeinanderfolgenden Übergängen des Signalzustandes. Bei Binärer Übertragung ist die Schrittgeschwindigkeit (Baud) gleich der Übertragungsgeschwindigkeit (Bit/s).

Download und dabei andere Geräte deaktivieren

Hinweis:

Ein Download auf ein Zielsystem kann unter diversen Umständen fehlschlagen.
Für die CPU515 genügt es das Modul in der Servicestellung aus- und einzuschalten.
Für die CPU167 und CPU960 muß das Modul ebenso in der Servicestellung aus- und einzuschalten werden und zusätzlich mit dem Programm „can_hex.exe“ die USERWARE gelöscht werden.
Tritt bei den Modulen mit CPU167 und CPU960 der Fall ein, daß die FIRMWARE gelöscht wurde, bleibt nur das Einsenden des Moduls ins Werk Elrest.

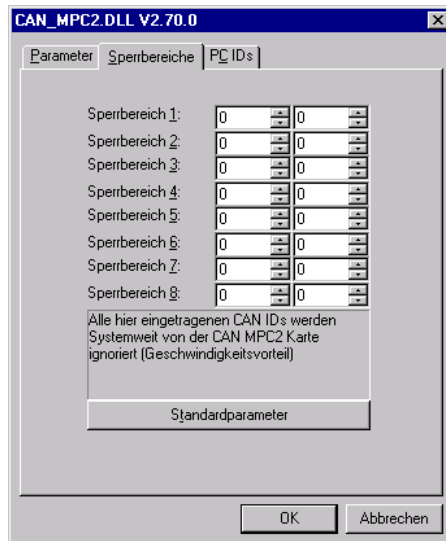
Soll ein ElaGraphII Projekt in eine Zielhardware gebracht werden, so wird ein sogenannter „Download“ auf das Zielsystem mit den entsprechenden Daten durchgeführt. Dies kann bei laufendem Betrieb an einer Anlage zu Problemen führen, weil in der Regel im Netzwerk noch andere Geräte kommunizieren. Dies hat zur Folge, daß die anderen Geräte den Download



stören. Um dies vermeiden zu können, gibt es eine Option, die bewirkt, daß der Treiber allen anderen Geräten bei einem Download eine Meldung schickt, die bewirkt, daß alle Geräte während dem Vorgang die Kommunikation unterbrechen.

Änderungen an einem Benutzerprofil des Modems können mittels Terminalprogramm (z.B. Hyperterminal)

Die „Sperrbereiche“ Seite Treiberdialogs



Es ist möglich die CAN-MPC2 Karte so zu parametrisieren, daß sie bestimmte Identifier Bereiche bei der CAN Kommunikation ignoriert. D.h. Wird ein CAN Netzwerk mit mehreren Busteilehmern betrieben, so ist es möglich einige Geräte „auszumaskieren“, damit die Karte performanter arbeitet. Sendet eine fremde Hardware z.B. von der CAN ID 200h bis 2FFh kann man diese Grenzen in der Liste eintragen, und ElaGraphII ignoriert jeglichen Telegrammverkehr zum/vom Modul.

Diese Einstellungen gelten global für den PC (für all seine 4 möglichen CAN-MPC2 Karten) und nicht für Module. Möchten Sie diese Einstellungen auf einen anderen Computer übernehmen, so kopieren Sie bitte die Datei CANMPC2.SET aus dem Verzeichnis \ElaSoft\Driver auf den Zielrechner.

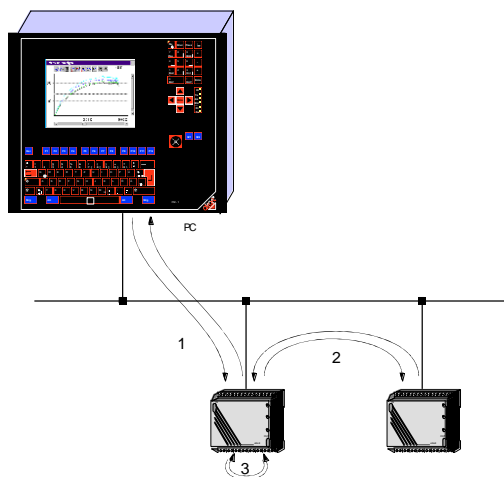
Hinweis:

Diese Einstellwerte werden erst nach einem Neustart von EOnline übernommen.

Übersicht

Der Datenaustausch findet immer zwischen mindestens zwei Geräten statt. Ein Gerät kann als PC, Feldbusmodul oder Bedienpult an einem Datenaustausch teilnehmen. In Abhängigkeit welches Gerätes mit welchem Kommuniziert, sind drei Arten der Geräteverbindung des zu unterscheiden:

1. Ein PC kommuniziert mit einem Gerät
2. Ein Gerät kommuniziert mit einem anderen Gerät
3. Ein Gerät kommuniziert mit sich selbst



Für jede Art der Geräteverbindung werden verschiedene CAN-Protokolle unterstützt. Die Protokolle unterscheiden sich in der Verwendung des CAN-Identifiers und dem Aufbau der Datenübertragung (Telegramm). Protokoll 1.0 ist ab Firmware Version 1.00, Protokoll 2.0 Firmware Version 1.30 erhältlich. Die Unterstützung der Betriebsarten Single-Master bzw. Multi-Master ist vom verwendeten Prozessor abhängig.

Der Vorteil bei Verwendung von Protokoll 2.0 ist, daß alle Geräte, auch der PC, als Module behandelt werden. Die gesamte Datenübertragung wird als Intermodulkommunikation interpretiert. Damit lassen sich mehrere PCs an eine CAN-Verbindung koppeln.

Für den Multi-Master Betrieb sind multi-master-fähige ElaCan-Module erforderlich. Das sind Module, die mit dem 82527-CAN-Chip ausgerüstet sind. Module mit den Prozessoren Intel 80960SA und SAB 80515A werden mit diesem CAN-Chip bestückt. Bei dem Siemens 80167 Prozessor sind die Funktionen des 82527-CAN-Chips integriert.

<i>Protokoll 1.0</i>		<i>Protokoll 2.0</i>	
Single- Master Betrieb	Multi- Master Betrieb	Single- Master Betrieb	Multi- Master Betrieb
Firmware ab V1.00	Firmware ab V1.20	Firmware ab V1.30	Firmware ab V1.30
SAB 80515	SAB 80515	SAB 80515	SAB 80515
SAB 80515A	SAB 80515A	SAB 80515A	SAB 80515A
Intel 80960SA	Intel 80960SA	Intel 80960SA	Intel 80960SA
Siemens 80167	Siemens 80167	Siemens 80167	Siemens 80167

Module mit dem Prozessor SAB 80515 werden ausschließlich mit dem alten 82526-CAN-Chip bestückt und sind in dieser Ausführung nicht multi- master- fähig.

Folgende Dienste sind im Protokoll eingebunden:

- Firmware- Variablen (verbindungsorientierter Datenaustausch)
- Speicherdirektzugriffe / Userwarevariablen (verbindungsorientierter Datenaustausch)
- Freie CAN ID's (verbindungsloser Datenaustausch)

Nähere Informationen erhalten Sie im Kapitel: Die Gerätetreiber.

Die Kommunikation kann entweder über ein Polling- oder über ein Interrupt-Verfahren aufgebaut werden.

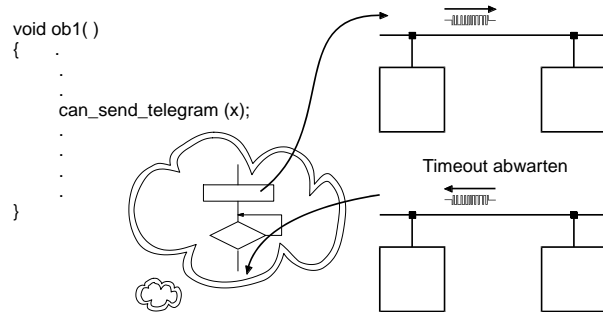


Polling-Verfahren

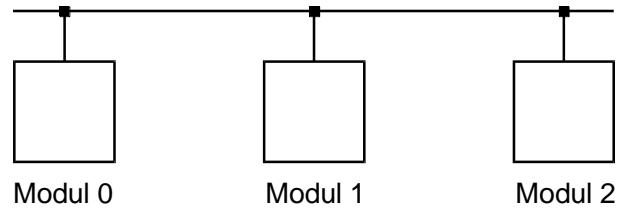
Es wird ein TIMEOUT eingestellt. Diese ist abhängig von der Übertragungsrate und beträgt standardmäßig 50 ms. Die TIMEOUT kann über CAN-Hex mit dem Datenpunkt „Intermodul Timeout“ in 10ms-Schritten eingestellt werden.

Anschließend wird ein Telegramm einmalig zum Kommunikationspartner gesendet. Dieser antwortet gewöhnlich innerhalb von 1-2ms (Baudrate @125kBaude) mit dem Quittierungstelegramm und die Funktion "can_send_telegram()" liefert eine "1" als Rückgabewert.

Wurde die TIMEOUT überschritten und es liegt noch kein Quittierungstelegramm vor, liefert die Funktion "can_send_telegram()" eine "0" als Rückgabewert.



Die HEX-Drehschalterstellung 0 entspricht dem niedrigsten CAN-Identifizier und somit der höchsten Priorität.



Die Standardeinstellung beträgt 50 ms TIMEOUT, bei der Kommunikation mit @123kBaude kann diese auf 20 ms reduziert werden.

Als Ursache für ein nicht beantwortetes Anfragetelegramm ist denkbar :

- Teilnehmer nicht vorhanden
- Temporäre Überlastung auf dem CAN-Segment, d.h. 100% Buslast von höher priorisierten Teilnehmern.
- Der Kommunikationspartner befindet sich in einer zeitkritischen Schleife (z.B. Interrupt mittels RS232 Schnittstelle oder einer Regelroutine) und kann nicht antworten. Deshalb sollte das Telegramm noch einmal verschickt werden.
- Ist das Quittierungstelegramm darauf erneut negativ, kann der Kommunikationspartner als „BUSOFF“ erklärt werden.
- Die Hardware eines Teilnehmers ist schadhaft. Nach dem Versenden mehrerer Error-Frames hat sich der Kommunikationsteilnehmer als „BUSOFF“ erklärt.

Beispiel für das Polling-Verfahren :

Mit folgenden Anweisungen würde ein Anfragetelegramm nach dem analogen Eingang 3 des Moduls 0 (M0_AA3) erfolgen :

```
void ob1()
{
    float istwert;
```



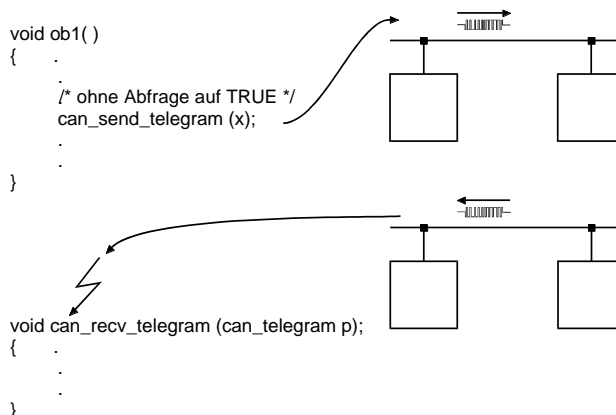
```

can_read_float(0,3,PARM_TYP_X,&istwert);
if (istwert > 200.0F)
    . . .
}
    
```

Interrupt-Verfahren

Die TIMEOUT wird zu Null gesetzt. Somit erfolgt kein Warten auf das Antworttelegramm. Nach dem einmaligen Versenden eines Telegramms läuft das Programm unverzögert weiter.

Falls der Kommunikationspartner antwortet wird ein Empfangsinterrupt ausgelöst. Die Interpretation des Telegramminhaltes muß in diesem Interrupt durchgeführt werden und obliegt der Firmware.



Bevor ein weiteres Telegramm gesendet wird muß das Antworttelegramm eingetroffen oder eine ausreichende Zeit verstrichen sein. Andernfalls kann es zu einer Adresskollision kommen.

Dieses Verfahren ist nur bei Automatisierungseinheiten möglich, die mit dem Zusatz "82527" ausgerüstet sind. Das umfaßt die 80515A, 80167 und 80960 CPU.

Beispiel für das Interrupt-Verfahren :

Nachdem ein Telegramm versendet wurde, antwortet der Kommunikationspartner und versendet ein Antworttelegramm mit dem aktuellen Istwert. In der Funktion can_recv_telegram(...) wird dieser empfangene Wert verarbeitet :

```

void can_recv_telegram(char modul,can_telegram *daten)
{
    switch( daten->parm_typ )
    {
        /* Empfangstelegramme */
        case PARM_TYP_MODULSTATUS:
            eprom_version[modul] = daten->d.c_data[1];
            modul_typ[modul]      = daten->d.c_data[2];
            break;
        case PARM_TYP_X:
            if( daten->parm_nr < MAX_ZONEN )
            {
                istwert[modul][daten->parm_nr] = daten->d.f_data;
                can_analog_eingang[modul][daten->parm_nr] = (short)daten->d.f_data;
            }
            break;
        case PARM_TYP_DIG_EIN_LESEN:
            can_eingang[modul].l_data = daten->d.l_data;
            break;
        case PARM_TYP_DIG_AUS_ZUSTAND:
            can_ausgang_zustand[modul].l_data = daten->d.l_data;
            break;
        case PARM_TYP_ANALOG_AUS_ZUSTAND:
            can_analog_ausgang_zustand[modul][daten->parm_nr] = (short)daten->d.f_data;
            break;
    }
}
    
```




```
    case default:  
        break;  
    }  
}
```



Analyse

Innerhalb dieser Empfangsinterruptroutine wird der empfangene Wert der Variablen "istwert" und "can_analog_eingang" zugewiesen. Im übrigen Programm kann ab sofort auf diese Variable mit dem neuen Wert zugegriffen werden.

Achtung bei Verwendung der CPU-Module mit den Prozessoren SAB 80515 und Siemens 80167:

Ein gleichzeitiger Zugriff auf 16- bzw. 32-Bit Variablen in `ob1()` und `can_rcv_telegram()` kann zu Datenüberschneidung führen. Dies kann hervorgerufen werden, indem während auf eine Variable in `ob1()` zugegriffen wird, der Empfangsinterrupt ausgelöst wird, und in `can_rcv_telegram()` ein zweiter Zugriff auf dieselbe Variable erfolgt.

Abhilfe: Bei Zugriff auf die entsprechenden Variablen in `ob1()`, können mit den Makros `ENABLE` und `DISABLE` alle Interrupts freigegeben bzw. gesperrt werden. Ein zweiter Zugriff auf eine Variable durch eine Interruptroutine wird somit unterbunden.

Einstellungen

Für die Kommunikation mit einem Zielsystem müssen Einstellungen der Netzwerk- und Kommunikationsparameter vorgenommen werden. Die Kommunikation beinhaltet beispielsweise den Datenaustausch zwischen einem PC und einem Modul. Ein Anfrage-Telegramm wird vom PC an ein Modul gesendet, wenn der PC von diesem Modul Daten anfordert. Das Modul empfängt dieses Telegramm und sendet ein Antwort-Telegramm, mit den geforderten Daten, an den PC zurück. Das Modul versucht sein Antwort-Telegramm zum nächst möglichen Zeitpunkt zu senden. Ist dies nicht sofort möglich, werden die Antwort-Telegramme in einen Datenspeicher geschrieben. Diese Daten werden bei der nächsten Möglichkeit gesendet. Besteht keine Möglichkeit die Daten zu senden, oder der Datenspeicher ist überfüllt, entstehen Echtzeitfehler in der Datenübertragung.

Die maximale Anzahl der zu übertragenden Telegramme pro Zeiteinheit sind physikalische Grenzen gesetzt durch:

1. Lesezykluszeit
2. Übertragungsrate
3. CAN-distance

Lesezykluszeit

Ist die Lesezykluszeit kleiner als die Zeit, welche benötigt wird um alle Telegramme innerhalb dieses Zeitabschnitts zu verarbeiten, können nicht alle Anfrage-Telegramme gesendet werden. Als Folge dessen können die entsprechenden Antwort-Telegramme ebensowenig empfangen werden. Es entstehen Echtzeitfehler.

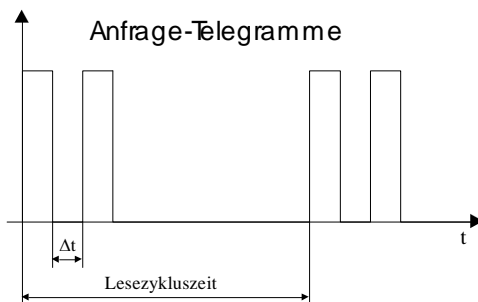
Tip: Die minimale Lesezykluszeit wie folgt einstellen

<i>Übertragungsrate</i>	<i>min. Lesezykluszeit</i>
≥ 100 kBaud	100 ms
50 kBaud	500 ms
≤ 20 kBaud	1000 ms

Übertragungsrate

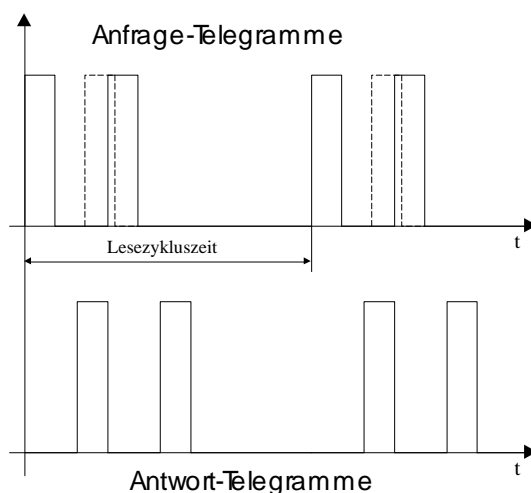
Der PC sendet innerhalb der Lesezykluszeit sein nächstes Anfrage-Telegramm mit der zeitlichen Verzögerung Δt . Die zeitliche Verzögerung Δt ist abhängig von der eingestellten Übertragungsrate. Je kleiner die Übertragungsrate ist, desto größer ist Δt .

Zwei Anfrage-Telegramme vom PC an ein Modul:



Wird für das Senden von mehreren Telegrammen eine längere Zeit benötigt als die eingestellte Lesezykluszeit, entstehen Echtzeitfehler.

Während der PC ein Antwort-Telegramm erhält, kann dieser kein neues Anfrage-Telegramm senden. Das nächste Anfrage-Telegramm wird dann unmittelbar nach dem vollständigen Erhalt des Antwort-Telegrammes gesendet.



Tip: Die maximale Anzahl der Telegramme / Sekunde wie folgt einstellen

Übertragungsrate	Firmware-Variablen	Ueware-Variablen
≥ 250 KBaud	120 Telegramme / Sek.	60 Telegramme / Sek.
125 KBaud	80 Telegramme / Sek.	40 Telegramme / Sek.
123 KBaud	80 Telegramme / Sek.	40 Telegramme / Sek.
100 KBaud	80 Telegramme / Sek.	40 Telegramme / Sek.
50 KBaud	40 Telegramme / Sek.	40 Telegramme / Sek.
20 KBaud	30 Telegramme / Sek.	30 Telegramme / Sek.
10 KBaud	30 Telegramme / Sek.	30 Telegramme / Sek.

Die Werte sind ausschließlich für die Schnittstellenkarte CAN-MPC2 gültig.

Beispiel:

Eingestellte Übertragungsrate : 50 kBaud
 Lesezykluszeit : 500 ms



Ergebnis:

Es können maximal 20 Anfrage-Telegramme / 500 ms gesendet werden. Dies bedeutet beispielsweise, daß der PC nicht mehr als 20 Datenpunkte / 500 ms von einem Gerät anfordern kann.

Wird die maximale Anzahl an Datenpunkten pro Zeiteinheit von einem Gerät angefordert, besteht für ein weiteres Gerät, bzw. für eine weitere Anfrage, keine Möglichkeit an einer Kommunikation teilzunehmen, da der CAN-Bus mit den 20 Datenpunkten / 500 ms ausgelastet ist.

Abhilfe:

1. Erhöhung der Übertragungsrate
2. Erhöhung der Lesezykluszeit

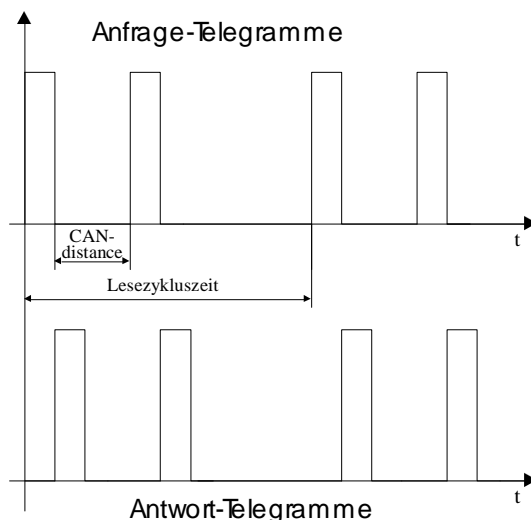
CAN-distance

Die CAN-distance ist definiert, als die Zeit in Millisekunden zwischen zwei Anfrage-Telegrammen. Sind mehrere Module über den CAN-Bus verbunden, können Module mit hoher Priorität dazu gezwungen werden, Module mit niedrigerer Priorität Zugriff auf den Bus zu gewähren.

Möchte ein weiteres Gerät mit niedrigerer Priorität ein Telegramm senden, so muß die CAN-distance so groß gewählt werden, daß dieses Gerät eine Zugriffsmöglichkeit auf den CAN-Bus erhält.

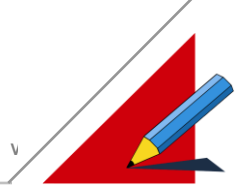
Bei Erhöhung der CAN-distance verringert sich die max. Anzahl der Telegramme/Lesezykluszeit. Dieser Parameter ist nur bei der Schnittstellenkarte CAN-MPC2 ab Version V2.08 verfügbar.

Beträgt die CAN-distance 0 ms, so sendet der PC das nächste Anfrage-Telegramm mit einer zeitlichen Verzögerung Δt (siehe Übertragungsrate). Für eine CAN-distance > 0 ms wird das folgende Anfrage-Telegramm mit der eingestellten Verzögerungszeit gesendet.



Bei Verwendung der Schnittstellenkarte CAN-MPC 2, ab Version 2.08, muß die CAN-distance für die Baudrate 50 kBaud mit mindestens 5 ms eingestellt werden !

Der Wertebereich beträgt: $0 \text{ ms} \leq \text{CAN-distance} \leq 255 \text{ ms}$



Kommunikationsroutinen

Die Kommunikationsroutinen können in drei Ebenen nach dem Schichtenmodell gegliedert werden:

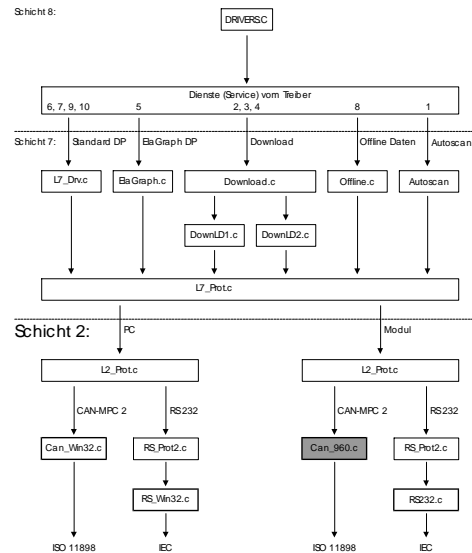
1. Low-Level Kommunikationsroutine
2. High-Level Kommunikationsroutine
3. Applikationsschicht

Low-Level Kommunikationsroutine

Die Low-Level Kommunikationsroutine wird durch die Schicht 2 im Schichtenmodell dargestellt. In dieser Schicht wird die Datenübertragung zwischen zwei Teilnehmern geregelt. Wichtigste Aufgaben der Schicht sind Fehlererkennung, Fehlerbehebung und Flußkontrolle. Die Fehlererkennung wird durch einen Fehlercode ermöglicht. Durch Wiederholung der Übertragung wird der Fehler behoben. Eine Flußkontrolle wird durch die Anpassung der Geschwindigkeit des Senders an die des Empfängers erreicht.

Als Test der Low-Level Kommunikationsroutine kann das Beispiel „DemoCAN“ (ab ElaGraph V2.70) eingesetzt werden. Der Test bezieht sich auf Intermodulkommunikation, d.h. auf die Kommunikation zwischen zwei CAN-Modulen.

In diesem Beispiel lassen sich über ein CAN-P200 alle relevanten Parameter der Schicht 2 überwachen.



Die Low-Level Kommunikationsroutinen bestehen aus folgenden C-Funktionen:

1. USIGN8 **can_req_unconfirmed**(USIGN8 card, USING32 send_id, USING8 lenght, USING8 _HUGE* send_dat)
2. USIGN8 **can_req_confirmed**(USIGN8 card, USIGN32 send_id, USIGN32 recv_id, USIGN8 length, USING8 _HUGE* dat)
3. USING8 **can_get_receivequeue**(USIGN8 card, USING32 _HUGE* recv_id, USING16 _HUGE* systemtick, USING8 _HUGE* len, USING8 _HUGE* recv_dat)
4. void **can_set_distance**(USIGN8 card, USIGN8 can_distance);
5. USIGN8 **can_reset**(USIGN8 card, USIGN16 io_address, USIGN16 baudrate, USIGN8 standard_id)

Beispiel DemoCAN:

Für dieses Beispiel ist ein CAN/P200 mit der Moduladresse 0 und ein beliebiges weiteres Modul mit der Adresse 1 notwendig.

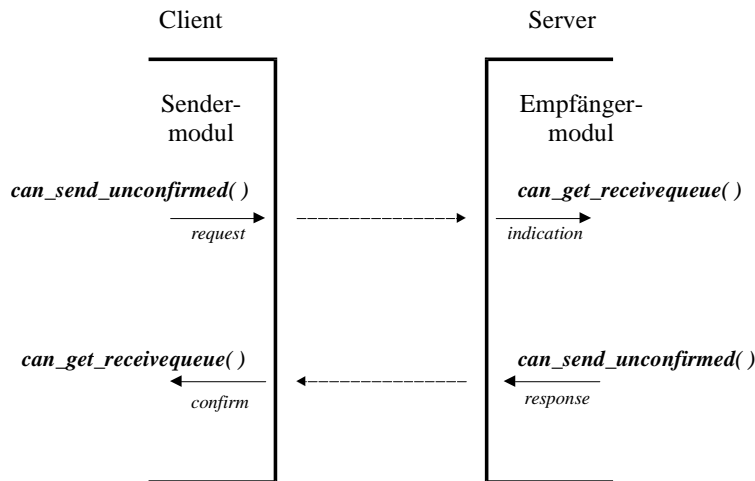
Das CAN/P200 sendet Telegramme mit der Funktion **can_send_unconfirmed**() an das zweite Modul. Das zweite Modul empfängt diese Telegramme und sendet ein Antworttelegramm an das CAN/P200 zurück. Das CAN/P200 muß alle Antworttelegramme, innerhalb einer eingestellten Zeit, mit der Funktion **can_get_receivequeue**() empfangen. Die Anzahl der nicht empfangenen Telegramme wird am Display vom CAN/P200 angezeigt.

Der Anwender hat die Möglichkeit folgende Parameter am CAN/P200 zu modifizieren:

- Anzahl der Telegramme pro Zeiteinheit (Taste „Pfeil auf“ und „Pfeil ab“)
- CAN-distance in Millisekunden (Taste „memo“ und „?“)
- Wartezeit auf ein Antworttelegramm (Taste „+“ und „-“)
- Aktualisierung der Werte nach Tastenbetätigung (Taste „S1“)
- stetige Aktualisierung der Werte (Taste „S2“)
- Zurücksetzen der Werte (Taste „S4“)



Folgende Graphik soll dem Leser den nötigen Überblick zu obiger Theorie vermitteln:



Dem Anwender ist die Anzahl der Telegramme meist durch die Anwendung gegeben. Treten hier Schwierigkeiten in der Kommunikation auf, so kann entweder die Wartezeit auf ein Antworttelegramm erhöht, oder die CAN-distance dem System angepaßt werden.

Störungen in der Kommunikation können unter folgenden Bedingungen auftreten:

- Das Sendermodul sendet sein zweites Telegramm, während das Empfängermodul noch das erste Telegramm bearbeitet. **Abhilfe:** CAN-distance erhöhen
- Es werden zu viele Telegramm gesendet, daß keine Möglichkeit besteht, innerhalb der Wartezeit die Antworttelegramme zu empfangen (Buslast zu hoch). **Abhilfe:** Wartezeit erhöhen

Protokoll 1.x:

Bei Verwendung der Elrest Protokolle ist zu Unterscheiden in:

1. Single-Master Betrieb
2. Multi-Master Betrieb

Bis zur Firmware-Version 1.19 wird nur Single-Master Betrieb, ab Firmware-Version 1.20 zusätzlich Multi-Master Betrieb unterstützt.

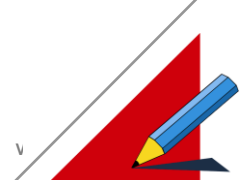
Single-Master Betrieb bedeutet, daß nur ein Modul aktiv Telegrammverkehr mit einem anderen Modul betreiben darf. Die Kommunikation kann nur von einem Modul ausgehen. In der Intermodulkommunikation kann es zu Kollisionen kommen, falls zwei Module gleichzeitig an ein Zielsystem Telegramme verschicken.

Beim Multi-Master Betrieb kann die Kommunikation von mehreren Modulen ausgehen. Die zur Kommunikation notwendigen CAN-ID's werden aus dem Parameter Intermodul-Master berechnet. Kollisionen können hier nicht auftreten .

Die Länge der CAN-ID kann wie folgt eingesetzt werden:

Prozessor	11-Bit ID	29-Bit ID
SAB 80515	✓	–
SAB 80515A	✓	✓
Intel 80960SA	✓	✓
Siemens 80167	✓	✓

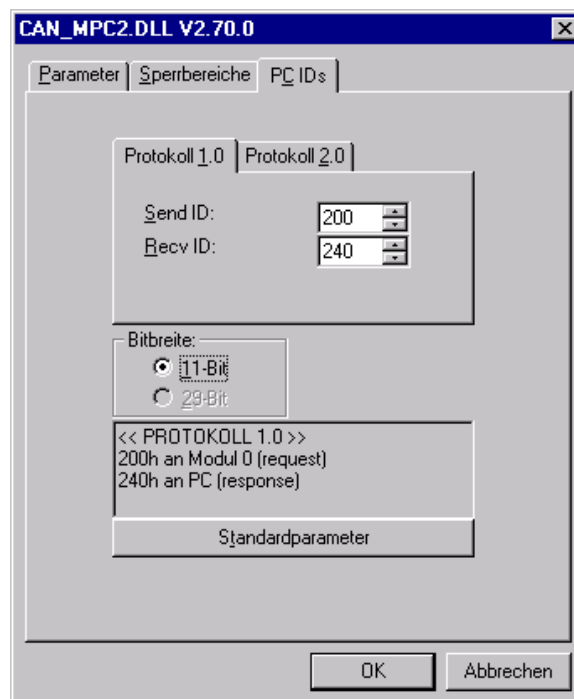
Achtung: Zur Verwendung der 29-Bit ID ist eine geeignete Schnittstellenkarte notwendig!



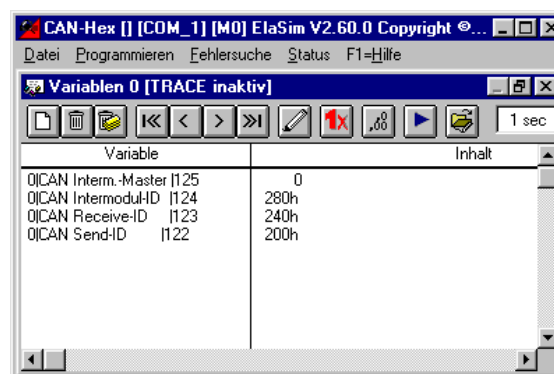
Bei der Auswahl der Netzwerktreiber können folgende Einstellungen vorgenommen werden:

1. **Send ID:** Der PC sendet alle Daten auf dieser Basisadresse
2. **Recv ID:** Der PC empfängt alle Daten auf dieser CAN Basisadresse

Diese Konfiguration kann mit allen Prozessoren verwendet werden.



Die Einstellungen der Module müssen identisch mit den Einstellungen der Schnittstellenkarte sein. Diese können über die serielle Schnittstelle mit dem Anwenderprogramm CAN-Hex vorgegeben werden.



Nähere Informationen erhalten Sie im Kapitel: Die Gerätetreiber.

Hinweis:

Diese Einstellwerte werden erst nach einem Neustart von EOnline übernommen.

Bitte beachten Sie, daß bei Verwendung von C-basierden Datenpunkte auch bei Anwahl von Protokoll 1.0 automatisch auf Protokoll 2.0 umgeschaltet wird, sofern es sich um ein Modul mit Firmware >= 1.30 handelt.

Dies hat zur Folge, daß falls in diesen Modulen Multi-Master Betrieb vorgewählt wurde es zwingend notwendig ist auch auf der PC-Seite das Protokoll 2.0 mit Multi-Master Betrieb vorzuwählen.

Master/Slave Protokoll

Ein Telegramm besteht aus einem 6 Byte langem Block, der folgendermaßen aufgebaut ist:

Das erste Byte beinhaltet die Nummer <nr>, mit der die Zonen durchnummeriert sind, wobei Zone 1...16 der <nr> 0...15 entspricht.

Das zweite Byte ist der Parametertyp <typ>. Die Parameter 0...49 und 128...177 bezeichnen von Elrest definierte Datenpunkte. Die Parameter 0...49 sind Schreib-Vorgänge in das Modul (das MSB <c> ist 0).



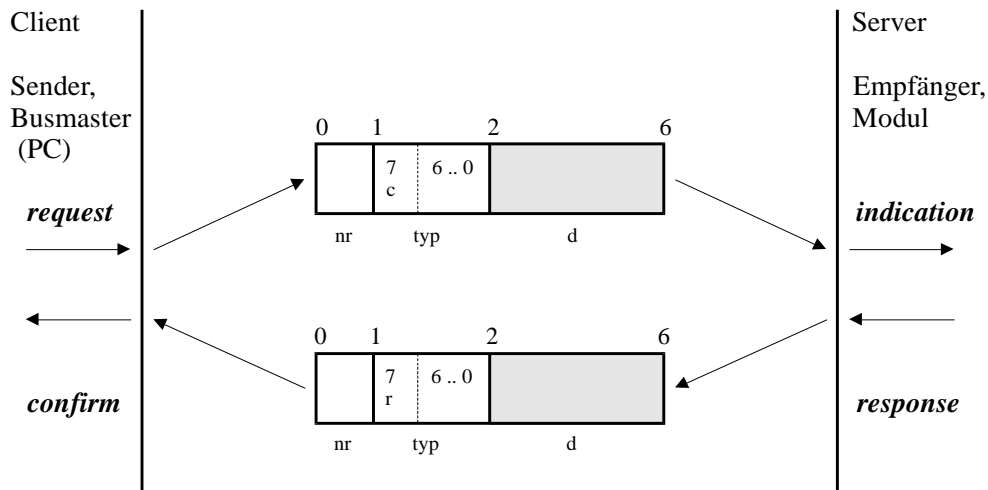
Die Parametertypen 128...177 sind die entsprechenden Lese-Vorgänge aus dem Modul (das MSB <c> ist 1). So würde z.B. mit <typ> = 4 die Schaltfrequenz geschrieben werden, während mit <typ> = 132 die Schaltfrequenz gelesen wird. In beiden Fällen wird ein Anfrage- und Antwort-Telegramm gesendet.

Die letzten 4 Byte sind für die zu sendende Daten <d> vorgesehen, falls es sich um einen Schreibvorgang handelt.

Die Reihenfolge der Datenbytes <d> muß beachtet werden. Sie ist abhängig von der Notation des verwendeten Prozessors.

Nähere Informationen erhalten Sie im Kapitel: ElaSim, Bibliotheksfunktionen (FIRMWARE).

Auf diese Weise schickt der PC Daten an ein Modul im CAN-Netzwerk:

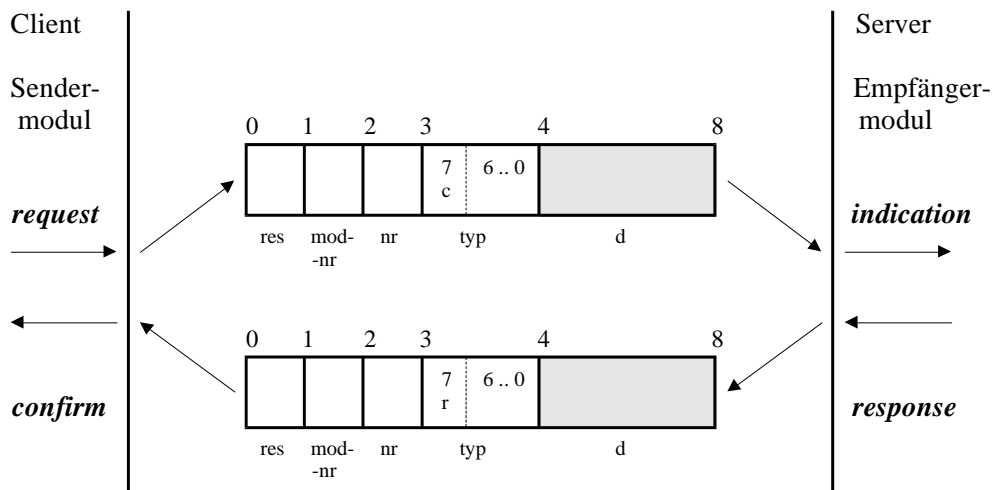


Die Basisadressen sind werkseitig eingestellt auf:

- Frame request → indication : Send ID = 0x200h
- Frame response → confirm : Recv ID = 0x240h

Intermodul Protokoll

Für die Intermodulkommunikation wird ein um zwei Byte erweiterter Datenblock verwendet. Sie enthalten die Modulnummer (im Byte < mod-nr >), die bisher nur 6 Bit in Anspruch nimmt. Dadurch ist die maximal mögliche Modulanzahl auf 64 begrenzt. Es wurden im Telegramm bereits 16 Bit vorgesehen um in der nächsten Phase auf 16000 Module zu erweitern.





Single- Master Betrieb bis zur Firmware-Version 1.19

Möglich mit allen verwendeten Prozessoren

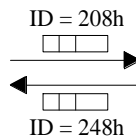
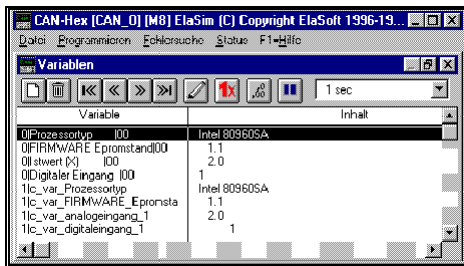
1. Fall :

Der PC schreibt bzw. liest Daten in das bzw. aus dem Modul. Das Modul kann von sich aus nicht aktiv werden. Dabei sendet der PC auf den Identifiern 200h bis 23Fh und das Modul quittiert auf den Identifiern 240h bis 27Fh. Die 200h ist eine Basisadresse, auf die Nummer des Empfängermoduls addiert wird. Es können bis zu 64 Module am CAN-Bus angeschlossen werden, dann ist der Adressraum ausgeschöpft. Genauso verhält es sich mit der Basisadresse 240h. Auf diese Adresse wird allerdings die Nummer des Absendermoduls aufaddiert. Der Empfänger ist immer der PC.

Wenn man mit dem CAN-Hex Monitorprogramm verschiedene CAN-Variablen aus Modul 8 abfragen will, wird zwischen PC und dem Modul eine Kommunikation aufgebaut. Auf der PC-Seite öffnet der Benutzer den „Variable Editor“. Der PC schickt ein Telegramm an Modul 8, in dem er nach Daten nachfragt. Modul 8 ist in diesem Beispiel so programmiert, daß das Telegramm im Programmteil „can_rcv_telegram“ einen Speichervorgang auslöst. Die Datenpunkte „PARM_TYP_X_LESEN“ und PARM_TYP_DIG_EIN_LESEN“ nach denen gefragt wird, speichert Modul 8 in C-Variablen ab. Die gewünschten Daten sendet Modul 8 in einem Antworttelegramm an den PC. Hier werden sie im „Variable Editor“ ausgegeben.

Client, PC

request



Server, Modul 8

```

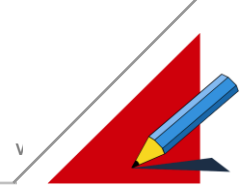
/* indication: */
/* Empfangsinterrupt wird ausgelöst */
void can_rcv_telegram(char modul, \
    can_telegram *daten)
{
    if (modul == 8)
    {
        switch (daten -> parm_typ)
        {
            case PARM_TYP_X_LESEN:
                /* Istwert (X) */
                c_var_analogeingang_1 = \
                    daten -> d.f_data;
                break;
            case PARM_TYP_DIG_EIN_LESEN:
                /* Digitaler Eingang */
                c_var_digitaleingang_1 = \
                    daten -> d.l_data;
                break;
        } /* switch */
    } /* if */
    . . .
    /* response */
}
    
```

confirm

2 Fall :

Intermodulkommunikation, d. h. Module tauschen untereinander Daten aus. Es muß kein Busmaster (PC) vorhanden sein. Dabei sendet ein Modul auf den Identifiern 280h bis 2BFh und das Modul quittiert auf den Identifiern 280h bis 2BFh. Der Adressraum für Modulanfragen und Quittungen ist nicht mehr getrennt, wie im 1. Fall. Der Identifier 280h ist hier die Basisadresse auf die die Modulnummer des Empfängermoduls addiert wird.

Modul 1 sendet Daten an Modul 3 mit dem Identifier 283h. In Modul 3 wird der CAN-Empfangsinterrupt „can_rcv_telegram“ ausgelöst, d. h. die Daten haben ihr Ziel erreicht. Anschließend sendet Modul 3 an Modul 1 das Antworttelegramm mit dem Identifier 281h. Die Kommunikation kann mit den Funktionen „can_send_telegram“ und „can_rcv_telegram“ so durchgeführt werden, wie in den beiden Ausschnitten aus den „ob.c“-Dateien der Module 1 und 3 zu sehen ist:

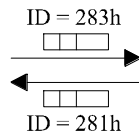


Client, Modul 1

```

void obl()
{
    . . .
    /* Sollwert: */
    telegram.parm_typ = PARM_TYP_W;
    /* Zone 5: */
    telegram.parm_nr = 4;
    /* Sollwerteintrag: */
    telegram.d.f_data = 98.5F;
    /* request von Modul 1 an 3 */
    can_send_telegram(3,&telegram);
    /* confirm von Modul 3: */
    /* can_send_telegram == true */
    . . .
    void van_rcv_telegram(char modul,\
        can_telegram *daten)
    {
        if (modul == 3)
        {
            switch (daten -> parm_typ)
            {
                case PARM_TYP_W:
                    /* in „daten -> d.f_data“ */
                    /* steht in diesem Fall */
                    /* kein Sollwert, das */
                    /* Telegramm von Modul 3 */
                    /* ist eine Quittung! */
                case PARM_TYP_W_LESEN:
                    sollwert[modul]= \
                        daten -> d.f_data;
                case . . .
            }
        }
    }
}

```



Server, Modul 3

```

. . .
/* indication */
void can_rcv_telegram(char modul, \
    can_telegram *daten)
/* response */
{
    if (modul == 1)
    {
        switch (daten ->parm_typ)
        {
            case PARM_TYP_W:
                sollwert[modul] = \
                    daten -> d.f_data;
                break;
            case PARM_TYP_W_LESEN:
                /* in „daten -> d.f_data“ */
                /* steht in diesem Fall */
                /* kein Sollwert, das */
                /* Telegramm von Modul 1 */
                /* ist nur eine Nachfrage */
                /* nach Daten! */
            case . . .
        }
    }
}

```

Umgekehrt kann es erforderlich sein, daß von Modul 3 die Kommunikation ausgeht. Es kann an Modul 1 Daten senden oder um Daten nachfragen.

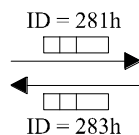
**Da es sich hierbei um Single-Master Betrieb handelt, darf nur ein Modul aktiv Telegrammverkehr mit einem Anderen betreiben.
Die Kommunikation darf nur entweder von Modul 1 (oben) oder von Modul 3 (unten) ausgehen, nicht von Beiden!**

Client, Modul 3

```

. . .
/* Sollwert: */
telegram.parm_typ = PARM_TYP_W;
/* Zone 10: */
telegram.parm_nr = 9;
/* Sollwerteintrag: */
telegram.d.f_data = 97.5F;
/* request von Modul 3 an 1 */
can_send_telegram(1,&telegram);
/* confirm von Modul 1: */
/* can_send_telegram == true */
. . .
void van_rcv_telegram(char modul,\
    can_telegram *daten)
{
    if (modul == 1)
    {
        switch (daten -> parm_typ)
        {
            case PARM_TYP_W:
                sollwert[modul] = \
                    daten -> d.f_data;
                /* ist eine Quittung */
        }
    }
}

```



Server, Modul 1

```

. . .
/* indication */
void can_rcv_telegram(char modul, \
    can_telegram *daten)
/* response */
{
    if (modul == 3)
    {
        switch (daten ->parm_typ)
        {
            case PARM_TYP_W:
                sollwert[modul] = \
                    daten -> d.f_data;
                /* hier steht in „daten ->“ */
                /* d.f_data“ der Sollwert */
        }
    }
}

```

Es konnte bisher in den „can_rcv_telegram“-Funktionen der beiden Module nicht erkannt werden, von wem die Kommunikation ausgelöst wurde. Das kann ab der Firmware-Version 1.20 mit der erweiterten Struktur „can_telegram“ (siehe Anhang) zusätzlich überwacht werden. Soll Modul 1 alle Daten eines Parametertyps, die es von Modul 3 übermittelt



bekommt speichern, so kann jetzt zwischen echten Schreib-Vorgängen von Modul 3 auf Modul 1 und Quittungen für Schreib-Vorgänge von Modul 1 auf Modul 3 unterschieden werden.



3. Fall :

Ein Modul sendet Daten an sich selbst, um einen Datenaustausch mit den Regelroutinen und/oder Zugriff auf die digitalen/analoge Ein- und Ausgänge zu ermöglichen.

Dieser Fall wird in der USERWARE genau wie die Kommunikation zwischen zwei verschiedenen Modulen (2. Fall) behandelt. Im Programmteil „ob1()“ der Datei „ob.c“ kann die Funktion „can_send_telegram“, mit der eigenen Modulnummer als Empfängeradresse, verwendet werden. Nur wird das Telegramm nicht auf den CAN- Bus geschickt, sondern intern verarbeitet. Der Programmteil „can_rcv_telegram“ wird bei diesem Fall der Kommunikation nicht ausgelöst.

Modul 5, (Ausschnitt aus „ob.c“):

```

/* Modulglobale Variable */
static u_long can_eingang[1];

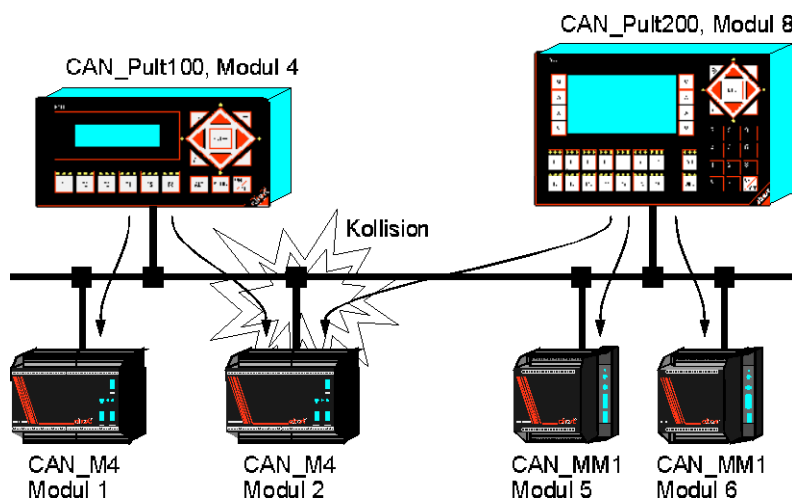
void ob1()
{
    . . .
    /* Digitalen Eingang abfragen: */
    telegram.parm_typ = PARM_TYP_DIG_EIN;
    /* Zone 1: */
    telegram.parm_nr = 0;
    /* request von Modul 5 an Modul 5 */
    can_eingang[0].l_data = can_send_telegram(5,&telegram);
    /* confirm von Modul 5 */
    . . .
}
    
```

Kollisionen:

Der Busmaster kommuniziert mit den Modulen des CAN in einem separaten Adressraum (Master → Modul:

0x200h bis 0x23Fh, Modul → PC: 0x240h bis 0x27Fh). Für alle Module im CAN steht nur ein Adressraum zur Verfügung (Modul → Modul: 0x280h bis 0x2BFh). In der Intermodulkommunikation kann es also zu Kollisionen kommen, falls zwei Module gleichzeitig an ein Zielmodul Telegramme senden.

Als Beispiel für das Auftreten von Kommunikationsfehlern ist im nächsten Bild ein CAN-Netzwerk aufgebaut. Ein CAN/P100 mit zwei CAN/M4 Modulen und ein CAN/P200 mit zwei CAN/MM1 Modulen, aber alle auf einem CAN-Bus. Das CAN/P100 kommuniziert mit beiden CAN/M4 Modulen. Vom CAN/P200 und vom CAN/P100 aus wird gleichzeitig versucht auf das Modul 2 zuzugreifen:

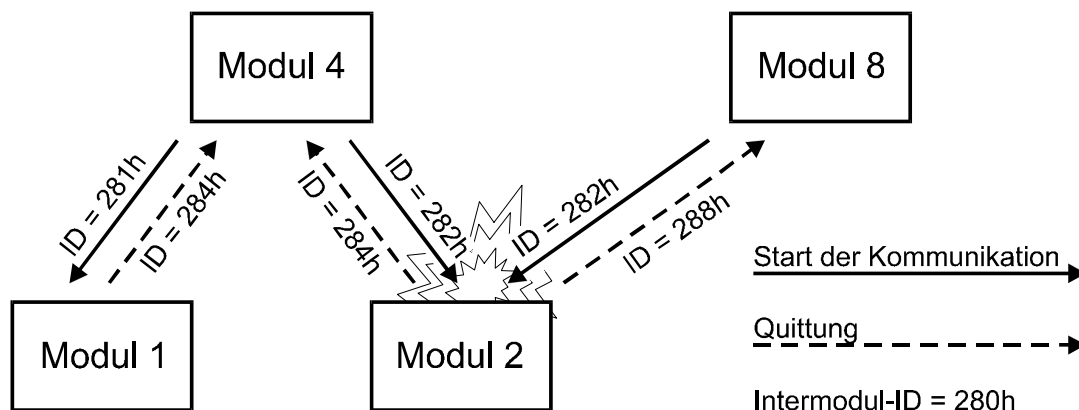


Quittierungs-Telegramme kommen zwar von den verschiedenen Empfängern auf dem gleichen CAN-Identifizier an den Absender zurück, das kann aber mit dem Polling-Verfahren (siehe Kapitel Polling-Verfahren) nicht gleichzeitig passieren. Die Funktion „can_send_telegram“ wartet auf eine Quittierung. Es ist in diesem Fall nicht möglich vor dem Eintreffen der Quittierung ein zweites Telegramm abzuschicken. Zum Beispiel kann Modul 4 an Modul 1 (auf CAN-ID = 0x281h) oder



an Modul 2 (auf CAN-ID = 0x282h) senden, ohne daß es bei den Quittierungen der beiden Module (beide auf CAN-ID = 284h) zu einer Kollision kommt, da die Quittierungen nicht gleichzeitig gesendet werden.

Wird das Interrupt-Verfahren benutzt (siehe Kapitel Interrupt-Verfahren), so muß der Anwender sicherstellen, daß die Kommunikation abgeschlossen ist, bevor eine neue begonnen wird. Einerseits steht das Modul sofort nach dem Versenden eines Telegramms für andere Berechnungen zur Verfügung und ist nicht durch das Warten auf die Quittierung für die TIMEOUT lahmgelegt, wie das beim Polling-Verfahren der Fall ist. Andererseits muß, bevor das Modul ein weiteres Telegramm abschickt, in der Userware überprüft werden, ob die begonnene Kommunikation beendet wurde. Das ist der Fall, wenn bereits eine Quittierung eingetroffen ist oder wenn lange genug auf die Quittierung gewartet wurde. Andernfalls ist es denkbar, daß die Quittierung auf das erste Telegramm und die Quittierung auf das zweite Telegramm kollidieren, da alle Quittierungen an ein Modul mit dem selben Identifier (z.B. ID = 0x284h bei Modul 4) verschickt werden.



Die im Bild dargestellte Kollision findet dann statt, wenn es zwei Modulen erlaubt ist mit ein und dem selben Modul eine Kommunikation aufzubauen. So ist es denkbar, daß Modul 4 und Modul 8 zur selben Zeit ein Telegramm mit dem Identifier 0x282h (= Empfängeradresse des Moduls 2) auf den Bus schicken.

Um eine fehlerfreie Kommunikation zu gewährleisten muß man solche Fälle in der Userware abfangen.

Eine Möglichkeit ist es eine „Modul-Hierarchie“ einzuhalten, in der nur die höhergestellten Module berechtigt sind mit Ihren unterstellten Modulen eine Kommunikation zu beginnen.

Multi-Master Betrieb mit der Firmware-Version ab 1.20

Möglich mit den Prozessoren: SAB 80515A-18, Intel 80960SA, Siemens 80167
Nicht möglich mit dem Prozessor: SAB 80515-12

Multi- Master Betrieb bedeutet: jeder darf mit jedem kommunizieren. Was die Rechte betrifft, mit Anderen im Netz Kontakt aufzunehmen, sind alle Module gleich. Eine Hierarchie ist nicht mehr nötig. Damit es trotzdem nicht zu Kollisionen kommt, empfangen die Module ihre Telegramme nicht mehr auf nur einer Adresse, sondern abhängig vom Absender und von der Art des Telegramms auf einem Adressraum. Zur Berechnung des passenden Identifiers müssen zuerst folgende Datenpunkte bekannt sein:

Das Netz:

- CAN-Intermodul-ID → Basisadresse, 0x280h
- CAN-Intermodul-Master → Anzahl der Master, für CAN-Intermodul-Master ≥ Anzahl der am CAN angeschlossenen Module: Multi-Master Betrieb
für CAN-Intermodul-Master = 0: Intermodulkommunikation wie Kapitel zuvor

Die Kommunikation:

- Request → Nummer des Moduls, von dem die Kommunikation ausgeht; Nachfrage



Indication	→	Nummer des Moduls, das dieses erste Telegramm erhält; Anzeige
Response	→	Nummer des Moduls, das auf das erhaltene Telegramm hin eine Quittung an den Absender verschickt; Antwort
Confirm	→	Nummer des Moduls, das diese Quittung erhält; Bestätigung
Meine-Modul-Nr.	→	Numerierung der Module von 0 ... n

Der Empfangs-Adressraum eines Moduls ist für eingehende Anfrage-Telegramme und Quittungs-Telegramme unterteilt:

Berechnung der Request-Identifizier: $ID = \text{CAN- Intermodul-ID} + 2 \times \text{CAN- Intermodul-Master} \times \text{Indication} + \text{CAN- Intermodul-Master} + \text{Request}$

Berechnung der Response- Identifizier: $ID = \text{CAN- Intermodul-ID} + 2 \times \text{CAN- Intermodul-Master} \times \text{Confirm} + \text{Response}$

Der Empfangs-Adressraum für ein Modul ist doppelt so breit wie die Anzahl der Master ($2 \times \text{CAN- Intermodul- Master}$). Die erste Hälfte für Anfrage- Telegramme die zweite Hälfte für Quittungs- Telegramme. Mit dieser Verteilung haben die Anfrage- Telegramme an ein Modul die höhere Priorität als eintreffende Quittungs- Telegramme.

Empfangsbereich für ein Modul = von ... bis;

von = $\text{CAN_Intermodul_ID} + 2 \times \text{CAN- Intermodul- Master} \times \text{meine-Modul-Nr.}$

bis = von + $2 \times \text{CAN- Intermodul- Master} - 1$

Eine Kollision, wie sie im Beispiel aus vorigen Kapitel gezeigt wurde, kommen im Multi- Master Betrieb nicht vor:

CAN-Intermodul-ID = 0x280h

CAN-Intermodul-Master = 0x008h;

(CAN-Intermodul-Master muß für einen Multi- Master Betrieb ≥ 8 gewählt werden. Es sind zwar nur 6 Module am CAN angeschlossen, die höchste Modulnummer ist aber Nr. 8)

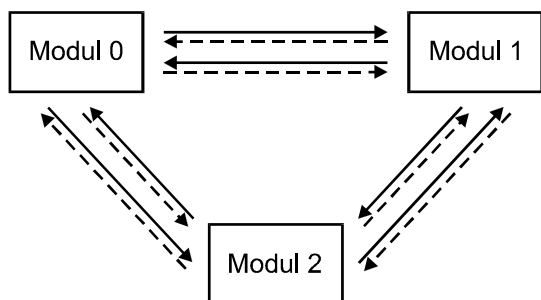
Die Kollision trat auf, weil Modul 4 und Modul 8 auf dem selben Identifizier (0x282h) und zur selben Zeit an Modul 2 ein Telegramm verschickt haben.

Im Multi- Master Betrieb sendet Modul 4 an Modul 2 auf: $ID = 280h + 2 \times 8 \times 2h + 8h + 4h = 2ACh$

Und Modul 8 sendet an Modul 2 auf: $ID = 280h + 2 \times 8 \times 2h + 8h + 8h = 2B0h$

Es werden also verschiedene Adressen benutzt. Aus diesem Grund können beide Telegramme zu Modul 2 durchkommen.

Am Beispiel eines CAN mit drei angeschlossenen Modulen wird die Berechnung und Verteilung der Adressen deutlich.



CAN-Intermodul-ID = 280h
CAN-Intermodul-Master = 3h

Empfangsbereich für Modul 0 = $280h + 2 \times 3 \times 0h \dots (280h + 2 \times 3 \times 0h) + 2 \times 3h - 1 = 280h \dots 285h$



Empfangsbereich für Modul 1 = 286h ... 28Bh
 Empfangsbereich für Modul 2 = 28Ch ... 291h

Anfrage von Modul 0 an Modul 1: $ID = 280h + 2 \times 3 \times 1h + 3h + 0h = 289h$

Antwort von Modul 1 an Modul 0: $ID = 280h + 2 \times 3 \times 0h + 1h = 281h$

Anfrage von Modul 0 an Modul 2: $ID = 280h + 2 \times 3 \times 2h + 3h + 0h = 28Fh$

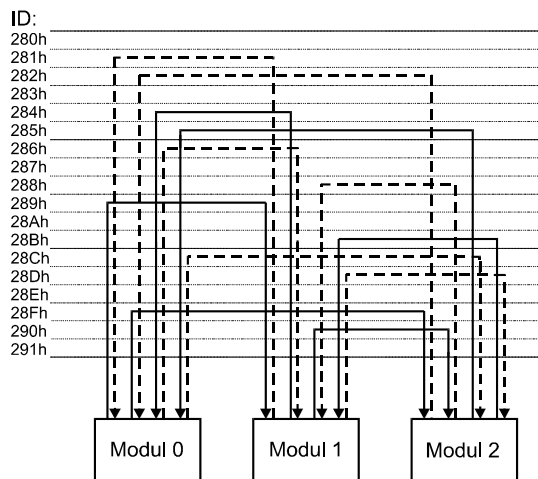
Antwort von Modul 2 an Modul 0: $ID = 280h + 2 \times 3 \times 0h + 2h = 282h$

Anfrage von Modul 1 an Modul 0: $ID = 280h + 2 \times 3 \times 0h + 3h + 1h = 284h$

Antwort von Modul 0 an Modul 1: $ID = 280h + 2 \times 3 \times 1h + 0h = 286h$

Anfrage von Modul 1 an Modul 2: $ID = 280h + 2 \times 3 \times 2h + 3h + 1h = 290h$

Antwort von Modul 2 an Modul 1: $ID = 280h + 2 \times 3 \times 1h + 2h = 288h$



Im Adressraum von Modul 0 bleiben die Identifier 0x280h und 0x283h unbenutzt. Rein rechnerisch würde Modul 0 auf 0x283h an sich selbst Telegramme verschicken und auf 0x280h Quittungen von sich selbst empfangen. Die Telegramme die ein Modul an sich selbst verschickt werden aber intern verarbeitet und gar nicht erst auf den Bus geschickt. Quittungen auf solche Telegramme gibt es nicht.

Dasselbe gilt für die Adressen 0x287h und 0x28Ah aus dem Adressraum von Modul 1 und für die Adressen 0x28Eh und 0x291h aus dem Adressraum von Modul 2.

Es ist möglich bis zu 4.000h Master in einem 29-Bit CAN zu verküpfen, da ein 20.000.000h breiter Adressraum zur Verfügung steht.

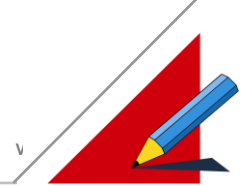
Die beschriebene Zuordnung der Identifier ist in der Firmware realisiert. Mit der Angabe der Konstanten „CAN-Intermodul-Master“ und der Modulnummer des gewünschten Empfängers über die bereits bekannten Kommunikationsfunktionen erfolgt die Berechnung des der betroffenen Identifier automatisch. Die Handhabung der Kommunikationsfunktionen bleibt weitgehend gleich.

1. Fall :

Ein PC, der im CAN angeschlossen ist, ist im Multi- Master Betrieb immer noch eine Ausnahme, damit das System mit alten Systemen kompatibel bleibt (→ siehe nächstes Kapitel). Er ist ein „Master“, wie auch alle anderen angeschlossenen Module, d. h. er kann die Kommunikation mit allen Modulen im Netz aufnehmen. Die Module selbst können zwar alle anderen Module im Netz ansprechen, nicht aber den PC. Weiterhin verschickt der PC Telegramme auf der Basisadresse 200h und empfängt Quittungen auf der Basisadresse 240h.

Beim Verschicken von Telegrammen benutzt der PC als „Modulnummer“ eine „virtuelle_modulnummer“, die als -1 gewählt wurde. Da der Wertebereich der Modulnummern momentan von 0 ... 63 verläuft, treten keine Überschreitungen auf.

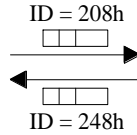
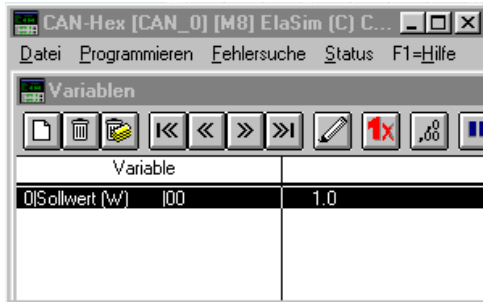
In dem Beispiel für Fall 1 aus dem vorigen Kapitel. ergeben sich folgende Änderungen:



Client, PC, „Modul -1“

Server, Modul 8

request



```

/* indication: Empfangsinterrupt wird
ausgelöst */
void can_rcv_telegram(char modul, \
    can_telegram *daten)
{
    /* modul = -1 */
    /* daten->CAN_TELEGRAM_TYP=INDICATIO*/
    /* daten->client_modul_no = -1 */
    /* daten->server_modul_no = 8 */
    if (modul == -1)
    {
        if (daten->type == \
            CAN_TELEGRAM_INDICATION)
        {
            if (daten->parm_typ == \
                PARM_TYP_W)
            {
                /* Beispielsweise kann hier*/
                /* der vom PC aus */
                gelesene */
                /* Sollwert in einer C- */
                /* -Variablen gespeichert */
                /* werden. */
                c_variable_sollwert = \
                    daten->d.f_data;
            }
        }
    }
}
/* response */

```

confirm

2. Fall:

Bisher war es bei der Intermodulkommunikation nicht direkt möglich zu unterscheiden, ob ein eintreffendes Telegramm eine Anfrage eines anderen Moduls oder die Quittung für ein zuvor selbst abgeschicktes Telegramm war (siehe vorheriges Kapitel, Fall 2).

Dieses Problem ist gelöst. Um das Modul identifizieren zu können, das die Kommunikation ausgelöst hat, wurde die Struktur „can_telegram“ („daten“ ist vom Typ „can_telegram“) um die Komponenten „CAN_TELEGRAM_TYP“, „client_modul_no“ und „server_modul_no“ erweitert.

Das bereits bekannte Beispiel aus dem Kapitel zuvor, wurde ausgebaut, um die neuen Möglichkeiten herauszustellen. Auf Modul 1 und Modul 3 läuft beide male das folgende Programm:



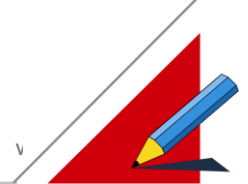
```

static  short  my;                /* die Nummer dieses Modules          */
static  short  indication_error=0; /* es ist ein weiteres Telegramm vom „type“ */
                                        /* „CAN_TELEGRAM_INDICATION“ eingetroffen. */
                                        /* → Fehler! Das kann nicht sein.        */
static  short  indication=0;      /* Anzahl der Indication's           */
static  short  indication_client=0; /* Nummer des Client-Modules         */
static  short  indication_server=0; /* Nummer des Server-Modules         */
static  short  confirm_error=0;   /* es ist ein weiteres Telegramm vom „type“ */
                                        /* „CAN_TELEGRAM_CONFIRM“ eingetroffen.  */
                                        /* → Fehler! Das kann nicht sein.        */
static  short  confirm=0;        /* Anzahl der Confirme's            */
static  short  confirm_client=0; /* Nummer des Client-Modules         */
static  short  confirm_server=0; /* Nummer des Server-Modules         */
static  char   trigger_send = FALSE; /* Auslöser zum verschicken eines Telegrammes */
static  char   trigger_send_cnt = 0; /* Anzahl der verschickten Telegramme */
static  ushort obl_loops =0;      /* Anzahl der „obl“-Durchläufe       */
. . .
void      ob22()
{
    my = can_set(CANMODUS_MODULADRESSE,0); /* lesen der Modulnummer dieses Moduls */
}

void      obl()
{
    if ( trigger_send ) /* wenn „trigger_send“ von CAN-Hex aus zu 1 gesetzt wird: ... */
    {
        if ( my == 1 ) /* . . . und wenn die Nummer dieses Moduls die 1 ist, dann ... */
            can_write_long(3,0,50,2000); /* . . . dann wird ein Sondertelegramm */
                                        /* (parm_typ = 50) an Modul 3 verschickt. */
        if ( my == 3 ) /* ... und wenn die Nummer dieses Moduls die 3 ist, dann . . . */
            can_write_long(1,0,50,2000); /* ... dann wird ein Sondertelegramm */
                                        /* (parm_typ = 50) an Modul 1 verschickt. */
        trigger_send = FALSE; /* ... dann wird der Telegrammauslöser wieder */
                                /* zurückgesetzt. */
        trigger_send_cnt++; /* ... dann wird der Telegramme-Zähler inkrementiert. */
    }
    obl_loops++; /* „obl_loops“ wird bei jedem „obl“-Durchlauf inkrementiert. */
}
. . .
void can_rcv_telegram(char modul,can_telegram *daten)
{
    if ( modul == -1 ) /* Anfrage per PC wird abgefangen */
        return;
    if ( daten->type == CAN_TELEGRAM_INDICATION ) /* Anfrage per Intermodulkommunikation */
    {
        /* Anfrage von einem anderen Modul an mich */
        if ( daten->server_modul_no != my )
        {
            indication_error++; /* Fehler! Das kann nicht sein. */
        }

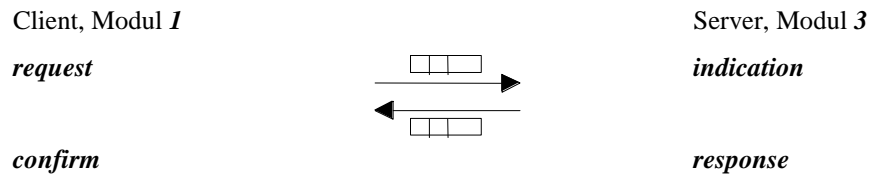
        indication++; /* es ist eine weitere Indication in diesem Modul aufgetreten */
        indication_client = daten->client_modul_no; /* Von dem Modul mit der Nummer */
                                                    /* „client_modul_no“ ist die */
                                                    /* Kommunikation */
        indication_server = daten->server_modul_no; /* die Nummer dieses Moduls */
    }
    if ( daten->type == CAN_TELEGRAM_CONFIRM )
    {
        /* Antwort eines anderen Modules von meiner Anfrage */
        if ( daten->client_modul_no != my )
        {
            confirm_error++; /* Fehler! Das kann nicht sein. */
        }
        confirm++; /* es ist eine weitere Confirm in diesem Modul aufgetreten */
        confirm_client = daten->client_modul_no; /* die Nummer dieses Modules */
        confirm_server = daten->server_modul_no; /* Dieses modul hat mit dem Modul mit */
                                                    /* der Nummer „server_modul_no“ */
                                                    /* eine */
        Kommunikation begonnen.
    }
}

```



```
}  
. . .
```

Wenn dieses Programm auf den Modulen 1 und 3 läuft, kann einerseits von Modul 1 aus ein Sondertelegramm an Modul 3 gesendet werden. In diesem Fall sieht die Konstellation der Module folgendermaßen aus:



Ein Request wird von Modul 1 gesendet, indem von CAN-Hex (M1) aus die C-Variable „trigger_send“ zu 1 gesetzt wird. Modul 3 erhält das Telegramm (die C-Variable „indication“ von Modul 3 wird von 0 zu 1). Modul 3 schickt wiederum eine Quittung an Modul 1 zurück (die C-Variable „confirm“ von Modul 1 wird von 0 zu 1).

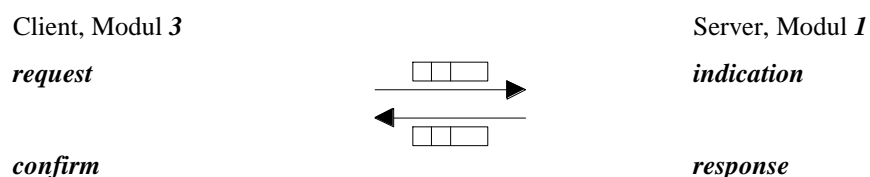
Die Werte für die Variablen „confirm_client“ (M1) und „indication_client“ (M3) sind übereinstimmend 1. „confirm_server“ (M1) und „indication_server“ (M3) nehmen einheitlich den Wert 3 an. Die Information wer bei der Kommunikation der Server und wer der Client ist steht also in beiden Modulen zur Verfügung.

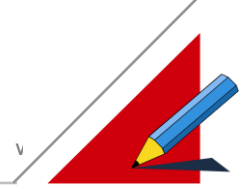
Variable	Value
1 CONFIRM	1
1 CONFIRM_CLIENT	1
1 CONFIRM_ERROR	0
1 CONFIRM_SERVER	3
1 INDICATION	0
1 INDICATION_CLIENT	0
1 INDICATION_ERROR	0
1 INDICATION_SERVER	0
1 JOB1_LOOPS	32520
1 TRIGGER_SEND	0
1 TRIGGER_SEND_CNT	1

Variable	Value
1 CONFIRM	0
1 CONFIRM_CLIENT	0
1 CONFIRM_ERROR	0
1 CONFIRM_SERVER	0
1 INDICATION	1
1 INDICATION_CLIENT	1
1 INDICATION_ERROR	0
1 INDICATION_SERVER	3
1 JOB1_LOOPS	32633
1 TRIGGER_SEND	0
1 TRIGGER_SEND_CNT	0

Da es sich hierbei um ein Multi-Master-System handelt, dürfen alle Module aktiv Telegrammverkehr mit einem Anderen Betreiben. Die Kommunikation darf von Modul 1 (oben) oder von Modul 3 (unten) ausgehen, auch von Beiden gleichzeitig!

Der umgekehrte Fall (die Kommunikation geht von Modul 3 aus) sieht folgendermaßen aus:





Diesmal wird ein Request von Modul 3 aus abgeschickt: über CAN-Hex (M3) wird die C-Variable „trigger_send“ zu 1 gesetzt. Modul 1 erhält das Telegramm (die C-Variable „indication“ von Modul 1 wird von 0 zu 1). Modul 1 schickt eine Quittung an Modul 3 zurück (die C-Variable „confirm“ von Modul 3 wird von 0 zu 1).

Die Werte sind für die Variablen „confirm_client“ (M3) und „indication_client“ (M1) übereinstimmend 3. „confirm_server“ (M3) und „indication_server“ (M1) nehmen einheitlich den Wert 1 an. Also: Bei dieser Kommunikation ist Modul 3 der Client und Modul 1 ist der Server.

Variable	Wert
1ICONFIRM	1
1ICONFIRM_CLIENT	1
1ICONFIRM_ERROR	0
1ICONFIRM_SERVER	3
1INDICATION	1
1INDICATION_CLIENT	3
1INDICATION_ERROR	0
1INDICATION_SERVER	1
1IOB1_LOOPS	-21454
1ITRIGGER_SEND	0
1ITRIGGER_SEND_CNT	1

Variable	Wert
1ICONFIRM	1
1ICONFIRM_CLIENT	3
1ICONFIRM_ERROR	0
1ICONFIRM_SERVER	1
1INDICATION	1
1INDICATION_CLIENT	1
1INDICATION_ERROR	0
1INDICATION_SERVER	3
1IOB1_LOOPS	-21300
1ITRIGGER_SEND	0
1ITRIGGER_SEND_CNT	1

3. Fall:

Keine Änderung zum vorherigen Kapitel.

Single- Master Betrieb mit der Firmware-Version ab 1.20

Möglich mit allen verwendeten Prozessoren

Natürlich ist es weiterhin möglich mit der Firmware-Version 1.20 das CAN- Netzwerk nur mit einem Master zu betreiben, indem der Datenpunkt „CAN-Intermodul-Master“ als 0 angegeben wird. Somit können Firmware- Versionen ≤ 1.19 zusammen mit der Firmware-Version 1.20 als Single- Master- System betrieben werden.

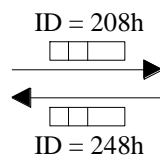
Die einzige Änderung, die in der Userware durchgeführt werden muß um sie mit der Firmware-Version 1.20 ebenfalls nutzen zu können, betrifft die Kommunikation der Module mit dem PC:

Bisher in Firmware-Version 1.19:

Bei einer Anfrage vom PC beinhaltet die Variable „modul“ aus der Funktion „can_rcv_telegram“ die Modulnummer des Empfänger- Moduls

Bsp.: Client, PC

request



Server, Modul 8

```

/* indication:
/* Empfangsinterrupt wird ausgelöst */
void can_rcv_telegram(char modul, \
can_telegram *daten)
/* response */
{
    if (modul == 8)
    {
        . . .
    }
}
    
```

confirm

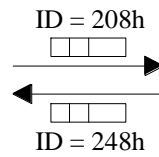
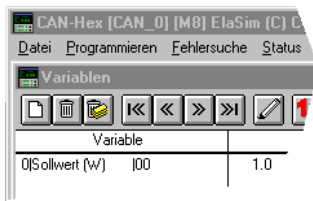
Jetzt in Firmware-Version 1.20:

Bei einer Anfrage vom PC beinhaltet die Variable „modul“ die Nummer „virtuelle_modulnummer“, die -1.



Bsp.: Client, PC

request



Server, Modul 8

```

/* indication:
/* Empfangsinterrupt wird ausgelöst */
void can_rcv_telegram(char modul, \
    can_telegram *daten)
/* response */
{
    if (modul == -1)
    {
        . . .
    }
}
    
```

confirm

Ab der Firmware-Version 1.20 steht eine erweiterte Struktur für „can_telegram“ zur Verfügung (siehe Anhang).

Protokoll 2.0

Das Protokoll 2.0 stellt ein gänzlich neues Protokollverfahren dar, welches primär auf Geschwindigkeit optimiert wurde. Die vorigen Protokolle 1.x werden selbstverständlich weiter unterstützt.

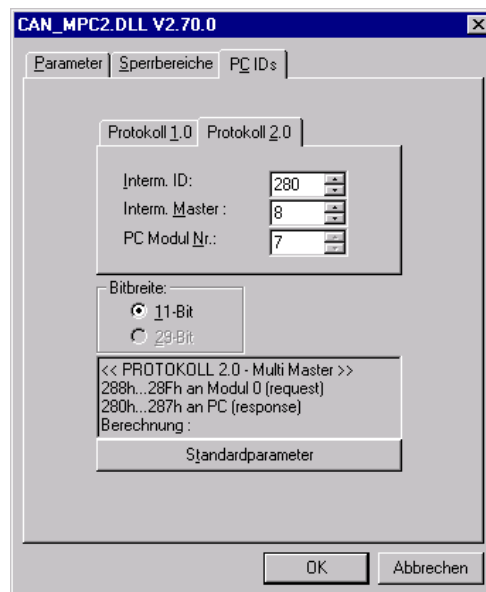
Der wesentliche Unterschied zu Protokoll 1.x ist dadurch gegeben, daß sich sämtliche Geräte wie in der Intermodulkommunikation verhalten. Es existiert nur eine Basisadresse. Der PC erhält eine vorgegebene Adresse als Offset zur Basisadresse. Diese muß am PC eingestellt werden. Alle Module erhalten eine Moduladresse als Offset zur Basisadresse. Diese muß am Modul eingestellt werden.

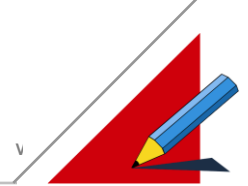
Es ist darauf zu achten, das es keine Adressüberschneidungen entstehen. Das Protokoll 2.0 unterstützt wie das Protokoll 1.x zwei Betriebsarten:

- 1. Single- Master Betrieb → Intermodul Master = 0
- 2. Multi- Master Betrieb → Intermodul Master > 0

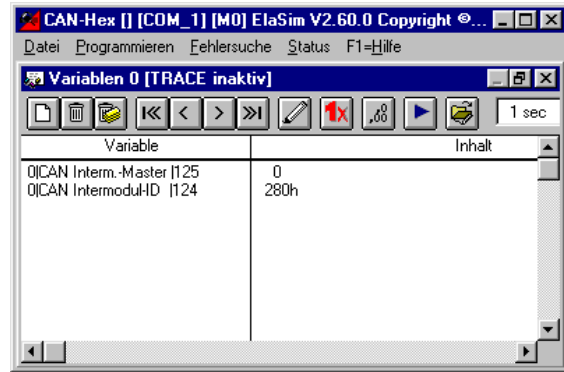
Bei der Auswahl der Netzwerktreiber können folgende Einstellungen der Schnittstellenkarte CAN-MPC 2 vorgenommen werden:

1. **Interm. ID:** Basisadresse für die Intermodulkommunikation
2. **Interm. Master:** Gibt die Betriebsart des projektierten CAN Netzes an.
0 : Single Master Betrieb
> 0 : Multi-Master Betrieb
3. **PC Modul Nr.:** Gilt nur bei Multi-Master Betrieb. Gibt die PC Modulnummer an.





Die Einstellungen der Module müssen identisch mit den Einstellungen der Schnittstellenkarte sein. Diese können über die serielle Schnittstelle mit dem Anwenderprogramm CAN-Hex vorgegeben werden.



Nähere Informationen erhalten Sie im Kapitel: Die Gerätetreiber.

Intermodul Protokoll

Für die Intermodulkommunikation wird ein Datenblock aus 8 Byte verwendet.

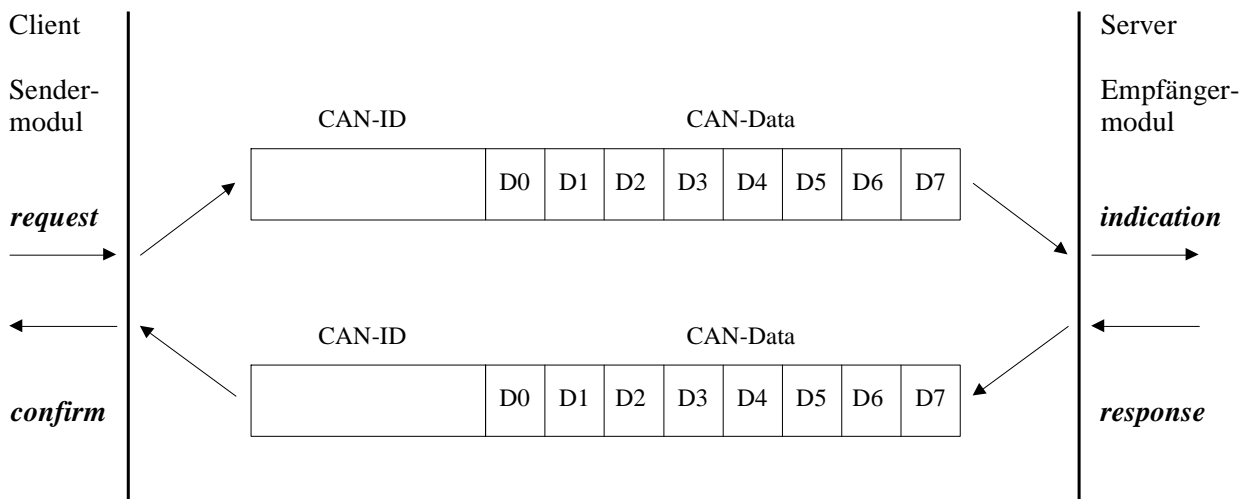


Bild: Intermodulkommunikation

Telegrammaufbau:

Die Information der „request“ und „response“ logischen Adresse wird ausschließlich in der CAN-ID übermittelt. Die Datenbytes D0 bis D3 dienen zur Übertragung von Dienstkennzeichnung und Adressen, die Datenbytes D4 bis D7 zur Übertragung der Nutzdaten des Dienstes.

Die Basisadressen der Intern. ID ist werkseitig eingestellt auf: 0x280h. Diese Basisadresse kann beliebig vorgewählt werden.

Single- Master Betrieb mit der Firmware-Version ab 1.30

Single- Master Betrieb → Intermodul Master = 0

In dieser Betriebsart erhält der Master automatisch die PC Modul Nr. 0x3Fh = Moduladresse 63. Dabei sendet der Master auf den Identifiern 0x280h + Moduladresse und empfängt auf den Identifiern 0x2BFh. Für alle Module steht nur ein Adressraum zur Verfügung. Deshalb kann es hier zu Adresskollisionen führen, wenn zwei Module gleichzeitig an ein Zielmodul Telegramme senden.

Abhilfe kann durch die Betriebsart Multi- Master geschaffen werden.

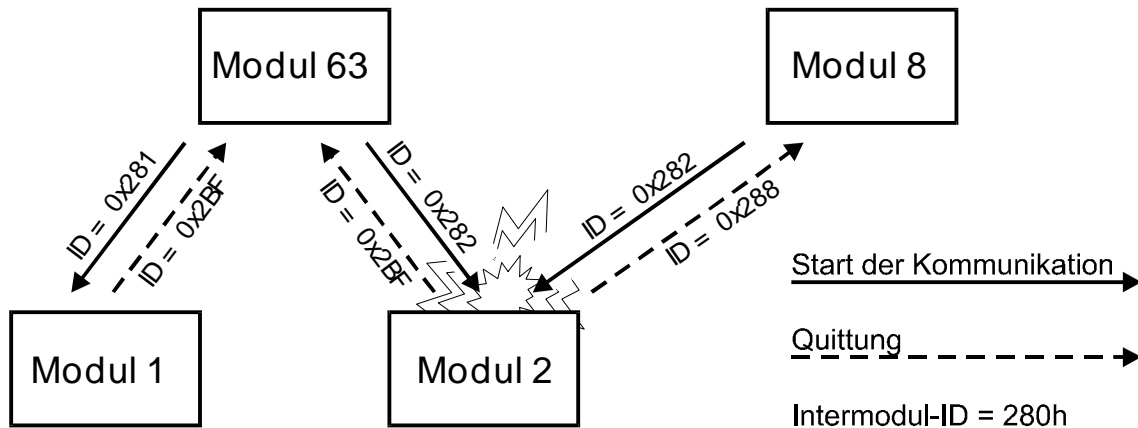


Abbildung: Adresskollision

Multi- Master Betrieb mit der Firmware-Version ab 1.30

Multi- Master Betrieb → Intermodul Master > 0

Nur mit den Prozessoren: SAB 80515A, Intel 80960 und Siemens 80167

Ist der Datenpunkt Intermodul-Master auf **größer Null** eingestellt , kann es aufgrund des ID-Berechnungsalgorithmus zu **keinen** Adresskollisionen kommen. Für jedes Telegramm wird eine eigene ID vergeben.

Die Schnittstellenkarten CAN-MPC 1 und CAN-MPC 2 unterstützen CAN- ID's mit einer Länge von 11 Bit.

Bei Einsatz der 11Bit CAN ID können maximal 16 Master (Datenpunkt Interm. Master max. 16) zum Einsatz kommen.

Bei Einsatz der 29 Bit CAN ID können aus softwaretechnischer Sicht maximal 16.000 Master (Datenpunkt Interm. Master max. 16000) zum Einsatz kommen => Die Schnittstellenkarte CAN-MPC 3 (in Vorbereitung) kann Telegramme mit 29 Bit ID's verarbeiten.

Die Berechnung der Request- und Response- Identifier erfolgt analog zu Kapitel Protokoll 1.x, Multi- Master Betrieb mit der FIRMWARE- Version ab 1.20, mit dem Unterschied, daß eingesetzte PCs mit der eingestellten PC Modul Nr. an einer Kommunikation teilnehmen.

Werden mehrerer PC's eingesetzt, dürfen sich deren PC Modul Nr. nicht überschneiden !

Beispiel:

In einem CAN-Netzwerk sollen zwei PC's und zwei CAN-Module mit folgenden Einstellungen untereinander kommunizieren:

IntermodulID : 0x400h
 IntermodulMaster : 0x005h

Die Identifier berechnen sich nach den bekannten Formeln:

$ID_{request}$ (Anforderung) = Intermodul-ID + 2 x Intermodul Master x Idication x Intermodul Master + Response



ID_{response} (Quittierung) = Intermodul-ID + 2 x Intermodul Master x Confirm x Response

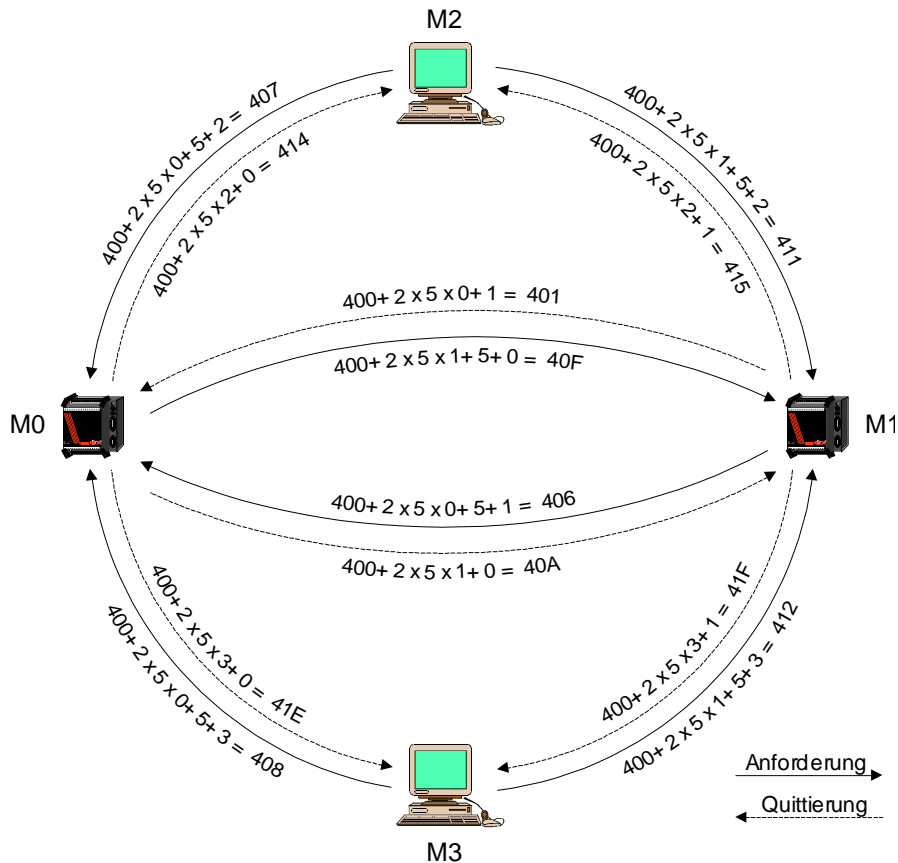


Abbildung: CAN_IDs_Berechnung

Das Senden von Telegrammen zwischen zwei PC's kann über die Nutzung der freien CAN ID's realisiert werden. Die freien CAN ID's dürfen sich nicht mit den übrigen CAN-Identifizier überschneiden.

Die Berechnung der an einer Kommunikation teilnehmenden Geräte, bei bekannter CAN ID, ist in folgendem Beispiel aufgeführt:

Anforderung-Telegramm von M3 an M1 mit der CAN ID: 0x412

$$M_{\text{Server}} = \frac{ID_{\text{Request}} - \text{IntermodulID}}{2 \cdot \text{IntermodulMaster}} = \frac{412 - 400}{2 \cdot 5} = 1$$

$$M_{\text{Client}} = ID_{\text{Request}} - \text{IntermodulID} - (2 \cdot \text{IntermodulMaster} \cdot M_{\text{Server}}) - \text{IntermodulMaster} = 412 - 400 - (2 \cdot 5 \cdot 1) - 5 = 3$$



Sonstige Protokolle

- CAL
Die erste Bestrebung der CiA eines genormten Protokolls. Die Teile der Multiplexed Variables wurden in dem Elrest Protokoll 1.0 umgesetzt.
- CANopen
Im europäischen Raum setzt sich dieses Protokoll als defacto Standard durch.
- DeviceNet
Im asiatischen und amerikanischen Raum setzt sich dieses Protokoll als defacto Standard durch.
- Mannesmann / Demag Protokoll für Spritzgußmaschinen.
Hierzu wurde von Elrest bereits eine Portierung vorgenommen.
- Fernheizungsregler Protokoll
Hierbei handelt es sich um ein kundenspezifisches Protokoll, daß von Anbieter zu Anbieter geringfügig variiert.

Download

Das Programmieren der CAN-Module wird als Download bezeichnet. Unterschieden wird zwischen „FIRMWARE programmieren“ und „USERWARE programmieren“.

Um Firmware programmieren zu können muß sich das CAN-Modul in der HEX-Schalterstellung „F“ bzw. in Servicestellung befinden. In dieser Einstellung ist die Ausführung der Userware deaktiviert. Dies kann vorteilhaft sein, wenn sich zum Beispiel eine nicht funktionsfähige Userware auf dem CAN-Modul befindet, und es dadurch nicht möglich ist eine Verbindung zwischen PC und CAN-Modul herzustellen.

Beispiel:

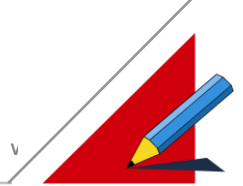
Ein Userware Download wurde mit der Taste „Abbruch“ unterbrochen. Das Modul ist nicht mehr kommunikationsfähig.

Abhilfe: Die nicht vollständig programmierte Userware muß in der HEX-Schalterstellung „F“ bzw. in Servicestellung über „Modul löschen“ gelöscht, oder mit „FIRMWARE programmieren“ neu programmiert werden.

Die Funktionalität von „USERWARE programmieren“ ist identisch mit „FIRMWARE programmieren“. Jedoch erfolgt aus der Firmware ein Zugriff auf die Userware.

Der Download wird auf folgenden CAN-ID's ausgeführt:

<i>Protokoll</i>	<i>Betriebsart</i>	<i>CAN-ID's</i>	<i>Abschaltung der Module</i>
1.0	Single-Master	0x200	0...63
	Multi-Master	0x280	
2.0	Single-Master	0x280...0x2BF	0...63
	Multi-Master	nach Formel	0...CAN-Intermodul-ID + 2 * CAN-Intermodul-Master * Indication + CAN-Intermodul-Master + Request



Die Gerätetreiber

So, wie die Netzwerktreiber über die Eigenschaften des Netzwerkes Bescheid wissen, muß der Gerätetreiber über die Eigenschaften der angeschlossenen Geräte Auskunft geben können.

Wie nicht anders zu erwarten, benötigen unterschiedliche Hardwarekomponenten auch unterschiedliche Treibersoftware. Möchte man wissen, welcher Treiber das Gerät benötigt, so besteht der einfachste Weg darin unter CAN-Hex einen sog. „Autoscan“ mit der angeschlossenen Hardware durchzuführen, der passende Treiber wird automatisch voreingestellt. Diese Methode ist die einfachste.

Ist es aus irgend einem Grunde nicht möglich, dieses durchzuführen, besteht eine weitere Möglichkeit darin, den Namen, der im Modul eingebauten CPU aus folgender Tabelle herauszusuchen :

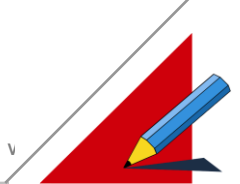
<i>Hersteller</i>	<i>CPU</i>	<i>Treiber</i>	<i>Alias Name</i>	<i>Typ</i>
<i>Siemens</i>	SAB 80535/N-MOS	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SAB 80C515A-N18	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SAB80C535-LB	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SABC515-LN	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SAB80C515NCB	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SABC515C	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SABC515C-LM	CAN_M515.DLL	CPU515	8 Bit CPU
<i>Siemens</i>	SAB-C167CR-LM	CAN_M167.DLL	CPU167	16 Bit CPU
<i>Intel</i>	I960SB-16	CAN_M960.DLL	CPU960	32 Bit CPU

Da sich die Treiber im Aussehen und Handling nicht wesentlich unterscheiden, wird im folgenden exemplarisch nur auf die CAN_M515.DLL eingegangen.

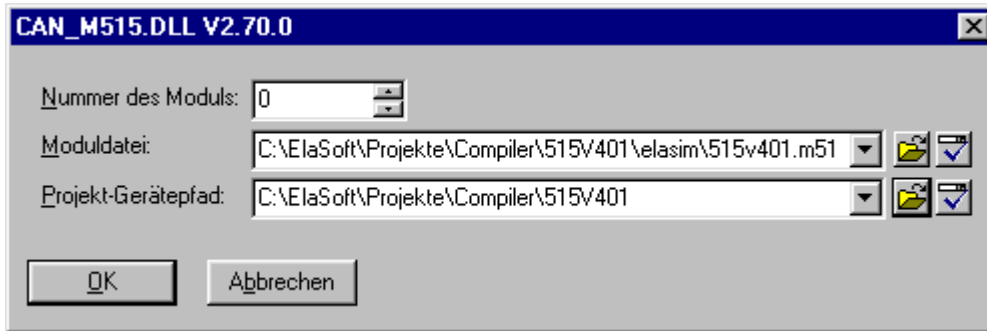
Allgemeiner Aufbau der Gerätetreiber

ElaGraphII ruft aus dem Treiber grundsätzlich 3 verschiedene Dialoge auf:

1. Geräteauswahl Dialog
2. Datenpunktauswahl für Multi Select
3. Datenpunktauswahl für Single Select



Geräteauswahl Dialog



Dieser Dialog ermöglicht es dem Anwender sein Modul zu Adressieren. Zusätzlich kann eine Parameterdatei und der Projekt-Gerätepfad angegeben werden.

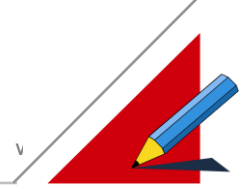
- Nummer des Moduls Nummer die über den Hex Schalter des Moduls eingestellt wurde. (wird auch als „Node“ bezeichnet)
- Moduldatei Parameterdatei für das Modul. Diese kann folgendes Kürzel haben :
 .M51 (CPU515) bzw. MAP (CPU960+CPU167) Datei: Wurde ein ElaSim Projekt erzeugt und compiliert, so wird automatisch eine *.M?? Datei angelegt. In Ihr werden die Adressen der im Programm erzeugten Variablen abgelegt. Diese wird benötigt, wenn man die Werte der Variablen vom PC aus auslesen will.
- Projekt-Gerätepfad Datenpunkte aus den Seiten „Selbstdefinierte Datenpunkte“, „Speicher-Direktzugriff (ElaSim/ElaGraph)“ und „Sondertelegramme ElaSim/ElaGraph“ werden in einer Tabelle gehalten, die in diesem Pfad abgelegt wird. Für jedes Gerät kann somit eine getrennte Tabelle erstellt werden. Der Pfad wird bei Aufruf durch die Projektverwaltung bereits voreingestellt.
 Wird dieser Pfad nicht angegeben, wird systemglobal eine Tabelle erstellt. Auf diese Tabelle können dann allerdings auch andere Geräte aus dem eigenen Projekt oder aus anderen Projekten zugreifen
Geben Sie bei Benutzung der besonderen Datenpunkte diesen Pfad immer an!



Auswahl der Moduldatei

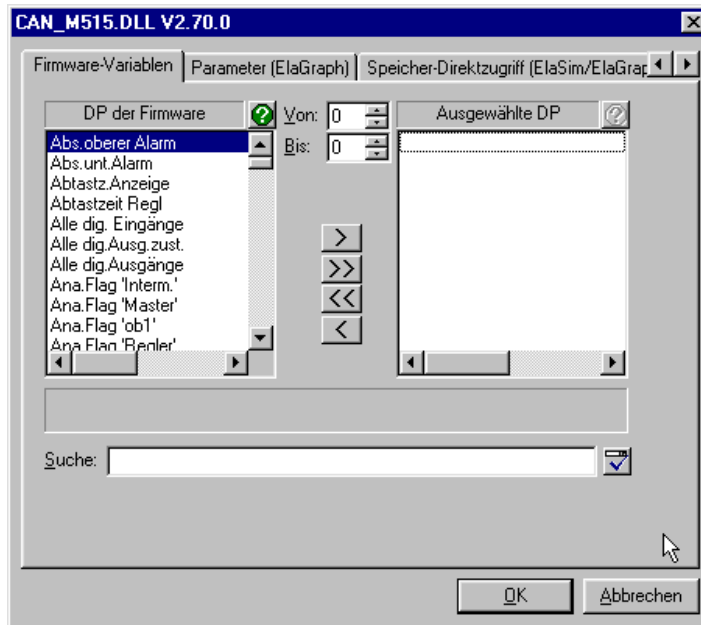


Die ausgewählte Datei wird in die Liste der zuletzt benutzten Dateien eingefügt.



Datenpunktauswahl für Single/Multi Select

Da die Funktionalität der Single und Multi select (= Einzel- und Mehrfachselektion) Dialoge für Datenpunkte identisch ist, wird der Übersicht halber nur auf den Multi Select Dialog eingegangen :



Der Dialog hat mehrere Seiten, die Datenpunkte, je nach Anwendungsfall, schematisch gruppieren. Alle Seiten jedoch verfügen über eine „Suche Funktion“ :

Suche Der hier eingegebene Begriff wird gesucht. Falls eine Liste diesen Begriff beinhaltet, wird er markiert. Als Suchbegriff können auch „Wildcars“ eingesetzt werden :

„?“ Steht für ein einzelnes Zeichen

„*“ Steht für n Zeichen

So findet „??gabe*“ z.B. Eingabe und Ausgabe

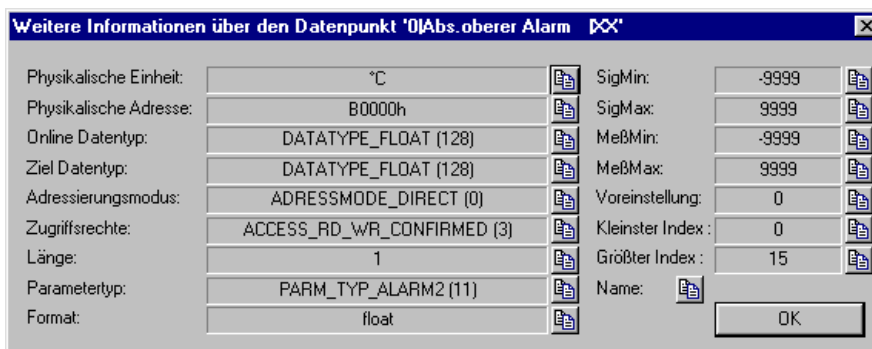


Die Suche wird gestartet



Ebenfalls sehen die Zusatzinformationen über den Datenpunkt (bei Drücken von)

bei allen Seiten (falls vorhanden) gleich aus :



Physikalische Einheit


Physikalische Einheit der gemessenen Größe

Physikalische Adresse

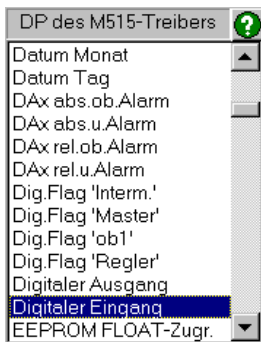
Die Speicheradresse des Datenpunktes im Zielsystem (kann durch den



Treiber modifiziert werden !)

Online Datentyp	Der Datentyp, der an der Oberfläche Zielsystem den Datenpunkt repräsentiert. So wird z.B. der Prozessortyp im Zielsystem als 2 Byte gehalten. An der PC Oberfläche wird aber ein Klartext dargestellt.
Ziel Datentyp	Der Datentyp, der an im Zielsystem den Datenpunkt repräsentiert. So wird z.B. der Prozessortyp im Zielsystem als 2 Byte gehalten. An der PC Oberfläche wird aber ein Klartext dargestellt.
Adressierungsmodus	Verschiedene Zugriffsverfahren, die der Treiber unterstützt
Zugriffsrechte	Wie auf den Datenpunkt zugegriffen werden kann. (lesend, schreibend, beides)
Länge	Datenpunkte können auch als Vektoren realisiert werden. Diese Länge gibt an Wie viele Bytes des Zieldatentyps vom Datenpunkt realisiert werden.
Parametertyp	ElaSim/ElaGraph interne Beschreibung des Datenpunktes
Format	Formatierung an der PC Oberfläche
SigMin	Kleinster Wert des Signals (wird bei 0 ignoriert)
SigMax	Größter Wert des Signal (wird bei 0 ignoriert)
MeßMin	Kleinster Wert des abgebildeten Meßbereichs (wird bei 0 ignoriert)
MeßMax	Größter Wert des Meßbereichs (wird bei 0 ignoriert)
Voreinstellung	Standardwert des Datenpunktes
Kleinster Index	Kleinster Index des Datenpunktes (z.B. Eingang 7). Dieser Wert kann als kleinster Index bei „Von“ gewählt werden
Größter Index	Größter Index des Datenpunktes (z.B. Ausgang 15). Dieser Wert kann als kleinster Index bei „Bis“ gewählt werden
Name	Name des Datenpunktes (in Zwischenablage kopieren)
	Der links nebenstehende Wert wird in die Zwischenablage kopiert.

In allen Dialogen werden 2 Listen gezeigt. Dabei zeigt die linke Liste die Datenpunkte, die das Modul physikalisch besitzt, die rechte Liste zeigt die ausgewählten Datenpunkte, die beim Drücken der OK Taste übernommen werden:



(Liste linke Seite) In dieser Liste werden alle Typen von Datenpunkten angezeigt, die das Gerät physikalisch kennt

Durch einen Doppelklick der linken Maustaste wird ein Datenpunkt in die linke Liste übernommen.

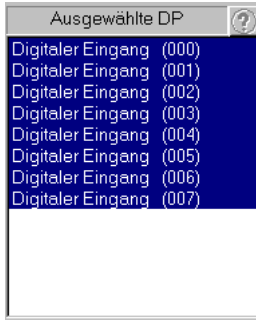


> Die links blau unterlegten Datenpunkte mit dem Index Von..Bis in die rechte Liste einfügen

>> Alle links dargestellten Datenpunkte in die linke Liste übernehmen

<< Alle rechts dargestellten Datenpunkt löschen

< Alle rechts blau unterlegten Datenpunkte löschen



Hier erscheinen die ausgewählten Datenpunkte. Um sie zu übernehmen, drücken Sie bitte die OK Taste

Von:
Bis:

Bitte geben Sie die Indizes ein. Wollen sie z.B. alle digitalen Eingänge von 0 bis 7 ansehen, so geben Sie wie abgebildet bei Von 0 und bei Bis 7 ein. Übernehmen Sie nun einen Datenpunkt von der linken Liste, werden auf der rechten Seite alle 8 Datenpunkte erzeugt

Werden sehr viele Datenpunkte von der linken Auswahlbox in die rechte übertragen erscheint folgender Fortschrittsbalken:

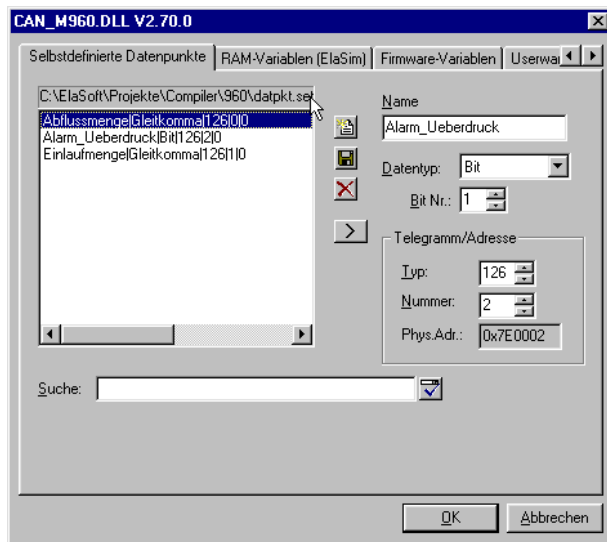


Wollen Sie den Vorgang abbrechen, so drücken Sie bitte diese Taste:





Selbstdefinierte Datenpunkte



Über die selbstdefinierten Datenpunkte können Sie sich eine Tabelle x-beliebiger Datenpunkte anlegen, auf die Sie dann einfach zugreifen können. Unter Angabe eines für Sie sinnvollen Namens bestimmen Sie den Datentyp für den Zugriff und den Telegrammtyp, unter dem der Datenpunkt versendet oder angefordert wird.

Auf diese Weise können Sie einfach einen Datenaustausch für die einzelnen Programmierarten ElaGraph/ElaDesign/ElaSim realisieren.

Tabelle Die Liste der projektierten Datenpunkte. Die Liste wird in einer Datei abgespeichert, die sich im Projekt-Gerätepfad befindet, falls dieser im Gerätedialog angegeben wurde. Ansonsten werden die Daten PC global von allen Geräten zugänglich gespeichert.

Name Kennzeichner des Datenpunktes. Hier können Sie bereits einen sinnvollen logischen Namen für den Datenpunkt vergeben. Für die Syntax bestehen geringe Einschränkungen.



Bereitet die Eingabefelder vor zur Eingabe eines neuen Datenpunkts. Das Feld Name wird freigegeben.



Überträgt den eingestellten Datenpunkt in die Tabelle. Eine Speicherung in die Datei erfolgt erst bei Bestätigung des Dialogs durch OK.



Löscht die dunkel unterlegten Einträge aus Liste.



Bereits vorhandene Datenpunkte können die Übertragung in die Eingabefelder editiert und verändert werden.

Datentyp Definiert, wie der Speicherbereich zu interpretieren ist. Folgende Datentypen stehen zur Verfügung :

Bezeichnung	C- Datentyp
Bit	BOOL
Byte	Char (Beschränkung auf 1 Byte pro Nummer)
Doppelwort	Long
Gleitkomma	Float
Text	Char[], fest 20 Zeichen (damit 5 Telegramme)
Wort	Short (Beschränkung auf 2 Byte pro Nummer)

Bit-Nr Auswahl der Bit-Nr 0-31 bei Verwendung des Datentyps Bit. Max. 32 Bits können unter demselben Typ und Nummer gespeichert werden.

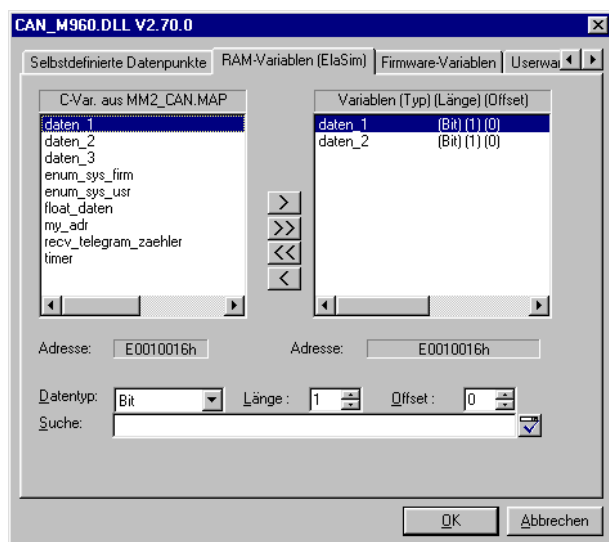


Typ / Nummer	<p>Pro CAN-Typ stehen 256 Nummern zur Verfügung die jeweils als Telegramm verwendet werden können. Als Datenbereich werden 4Bytes genutzt, die im Datentyp spezifiziert werden (Byte und Wort sind allerdings eingeschränkt und können nur einmal pro Telegramm spezifiziert werden).</p> <p>Für die selbstdefinierten Datenpunkte steht der Typ-Bereich 123-126 (CPU960), bzw. 119-126(CPU167) zur Verfügung. Die Bereiche 50-118 werden vom ElaGraph-Kernel benutzt. Wenn Sie ElaGraph nicht verwenden, stehen Ihnen dies Bereiche zusätzlich zur Verfügung.</p>
Phys. Adresse	<p>Errechnete hexadezimale Adresse aus Typ und Nummer als Anzeige. Adresse = Typ * 0x10000 + Nummer = Typ << 16 + Nummer</p>

Bitte beachten Sie, daß Sie zur Nutzung dieser Datenpunkte bestimmte Userware-Bibliotheken zum Zielgerät laden müssen!
Siehe auch: [Voraussetzungen Datenpunkte](#)



RAM-Variablen (ElaSim)



Ein Speicherdirektzugriff umfaßt das Lesen und Schreiben von Speicherblöcken (Memory Access) bis 1024 Bytes. Die Speicherdirektzugriffe werden bereits seit dem Protokoll 1.0 unterstützt, dort aber mit der Einschränkung, daß diese lediglich vom PC (Master- Slave) bearbeitet werden können. Ab Protokoll 2.0 kann ein Speicherzugriff auch von Modul zu Modul (Intermodulkommunikation) erfolgen.

Wird ein ElaSim Programm compiliert, so erzeugt der Compiler eine sogenannte „MAP“ Datei (für CPU515 hat es die Endung .M51, für die CPU960 und CPU167 .MAP).

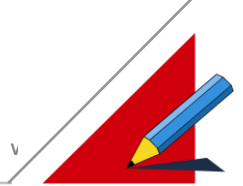
In dieser Datei wird festgelegt, in welcher Adresse welche symbolische Variable abgelegt wird. Somit ist es möglich, während dem laufenden Programm Einfluß von außen auf den Programmablauf zu bekommen.

Die Seite ist nur aktiv geschaltet, wenn im Geräteauswahl Dialog eine Map Datei ausgewählt wurde. Auf der linken Seite erscheine die symbolischen Bezeichner, auf der rechten die ausgewählten Variablen.

Hinweis:

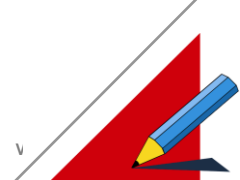
Erfolgt ein neuer Compilervorgang, so muss die Ressource-Datei neu erzeugt werden!

Adresse	Bezeichnet die Adresse der Variablen im Zielsystem																								
Datentyp	Da beim Compilerlauf die Information des definierten Datentyps verloren geht, muß dieser von Hand ausgewählt werden. So kann ein Text auch als char[] interpretiert werden usw. Es stehen folgende Datentypen zur Verfügung :																								
	<table border="0"> <thead> <tr> <th>Bezeichnung</th> <th>C- Datentyp</th> </tr> </thead> <tbody> <tr> <td>Bit</td> <td>BOOL</td> </tr> <tr> <td>Byte</td> <td>Char</td> </tr> <tr> <td>Byte pos.</td> <td>unsigned char</td> </tr> <tr> <td>Datum</td> <td>STRUCT_DATE</td> </tr> <tr> <td>Doppelw.p.</td> <td>unsigned long</td> </tr> <tr> <td>Doppelwort</td> <td>long</td> </tr> <tr> <td>Gleitkomma</td> <td>float</td> </tr> <tr> <td>Text</td> <td>char[]</td> </tr> <tr> <td>Wort</td> <td>short</td> </tr> <tr> <td>Wort pos</td> <td>unsigned short</td> </tr> <tr> <td>Zeit</td> <td>STRUCT_TIME</td> </tr> </tbody> </table>	Bezeichnung	C- Datentyp	Bit	BOOL	Byte	Char	Byte pos.	unsigned char	Datum	STRUCT_DATE	Doppelw.p.	unsigned long	Doppelwort	long	Gleitkomma	float	Text	char[]	Wort	short	Wort pos	unsigned short	Zeit	STRUCT_TIME
Bezeichnung	C- Datentyp																								
Bit	BOOL																								
Byte	Char																								
Byte pos.	unsigned char																								
Datum	STRUCT_DATE																								
Doppelw.p.	unsigned long																								
Doppelwort	long																								
Gleitkomma	float																								
Text	char[]																								
Wort	short																								
Wort pos	unsigned short																								
Zeit	STRUCT_TIME																								

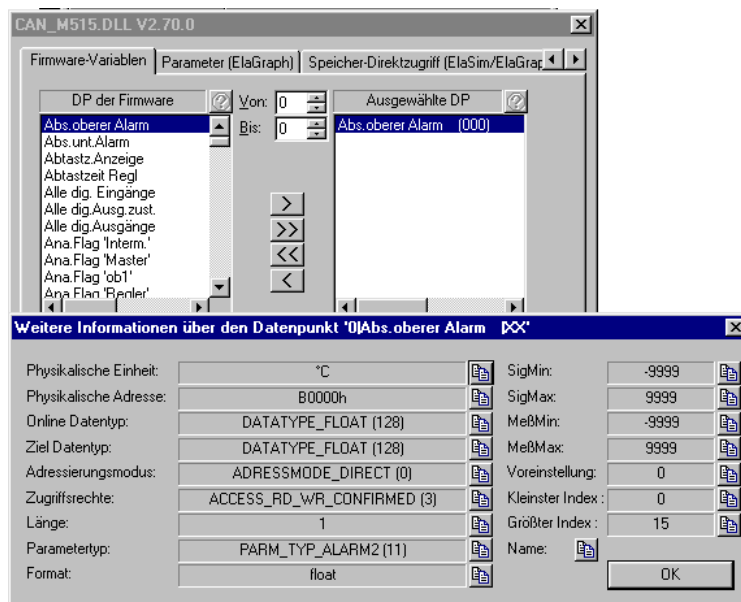


Länge	Wird hier eine Zahl größer 1 eingetragen, so wird der Datenpunkt als Vektor interpretiert.
Offset	Eine Zahl größer 0 bewirkt daß nicht die Adresse der Variablen ausgelesen wird, sondern ein n - Bytes Offset auf die Basisadresse addiert wird. Diese Option wird verwendet, um Elemente eines Feldes auslesen zu können. (Vorsicht ! bei n>1 dimensionalen Feldern kommt es darauf an, wie der Compiler seine Werte ablebt. Bitte sehen sie bei Bedarf in den entsprechenden Handbüchern nach)

Bitte beachten Sie, daß der Zugriff auf diese Datenpunkte eines anderen Gerätes von einem Zielgerät aus nicht möglich ist!
Siehe auch: [Voraussetzungen Datenpunkte](#)



Firmware-Variablen



In dieser Seite werden alle Datenpunkte aufgelistet, die in der Elrest Firmware zur Verfügung stehen.

Firmware : „... ist der Teil der Software, der ohne Quellcode dem Kunden als ausführbares Programm (Download einer .HEX Datei) auf der gewünschten Hardware mitgeliefert wird. Dieses ausführbare Programm ist eine gerätespezifische Software, die dem Kunden ermöglicht, die Hardwarekomponenten über bestimmte Funktionalitäten, der Firmware-Variablen, anzusprechen.“

Die Firmware bildet sozusagen das BIOS (Basic Input Output System), das als Minimum vorhanden sein muß, um einen Einfluß auf die Hardwarekomponenten des Gerätes zu bekommen.

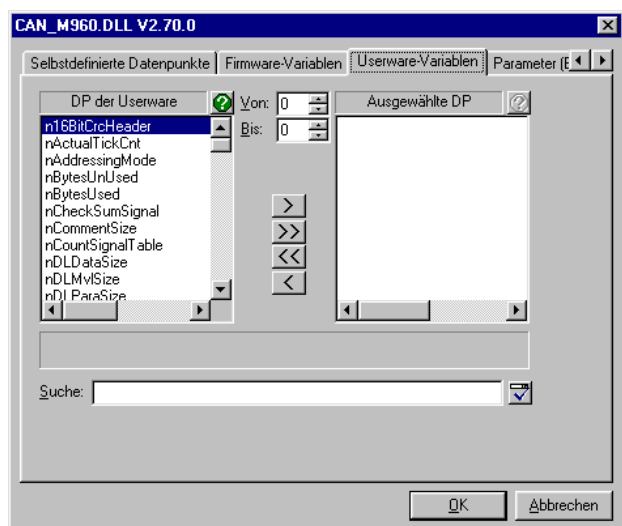
Firmware-Variablen : Diese Datenpunkte bilden die Schnittstelle zur Firmware und ermöglichen dem Benutzer Zugriff auf die vorhandene Hardware. Der Datenaustausch der Firmware- Variablen wird bereits seit dem Protokoll 1.0 unterstützt.

Der Adressbereich der Firmware- Variablen ist fixiert in der physikalischen Adresse von 0...49 (0x0000xxxx...0x0031xxxx) und 127 (0x007Fxxxx).

Unter Verwendung von EOnline / ZOnline muß der Bediener keine Notationsberücksichtigungen bei der Intermodulkommunikation vornehmen.



Userware-Variablen



Userware : „... ist der Teil der Software, der die Applikationsaufgaben definiert. Letztendlich ist es der Teil der Software über den der Anwender frei bestimmen kann, egal mit welchem Werkzeug diese erstellt worden ist. Als Beispiel : Reglervorgänge, Menüführung, Ablaufsteuerungen etc.“

Werkzeuge zur Erstellung von Userware sind :

- ElaSim (C- Programme)
- ElaGraph (grafische Programmierung)
- ElaGraphII (Ressourcen orientiertes Programmierwerkzeug)

Userware- Variablen : Ein Zugriff auf Useware-Variablen kann erfolgen, wenn eine CPU960 oder CPU167 verwendet wird, und wenn die Userware auf das Zielsystem geladen wurde.

Die Datenpunkte spiegeln die Struktur

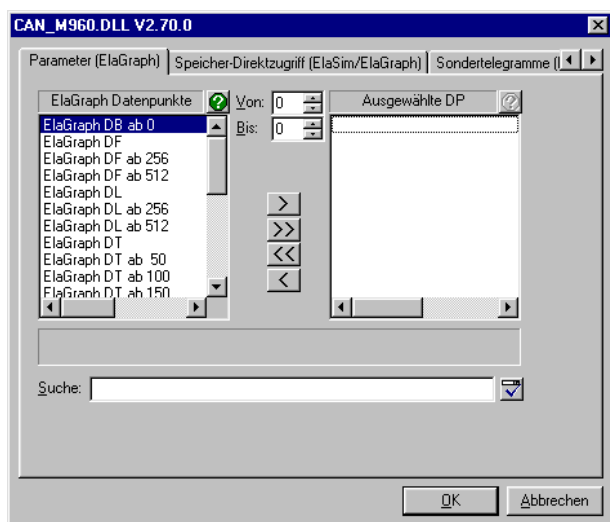
```
STRUCT_USERWARE_MODUL    _DATA* psUserwareModul;
```

in Uwmodul.h wieder.

Bitte beachten Sie, daß der Zugriff auf diese Datenpunkte eines anderen Gerätes von einem Zielgerät aus nicht möglich ist!
 Siehe auch: [Voraussetzungen_Datenpunkte](#)



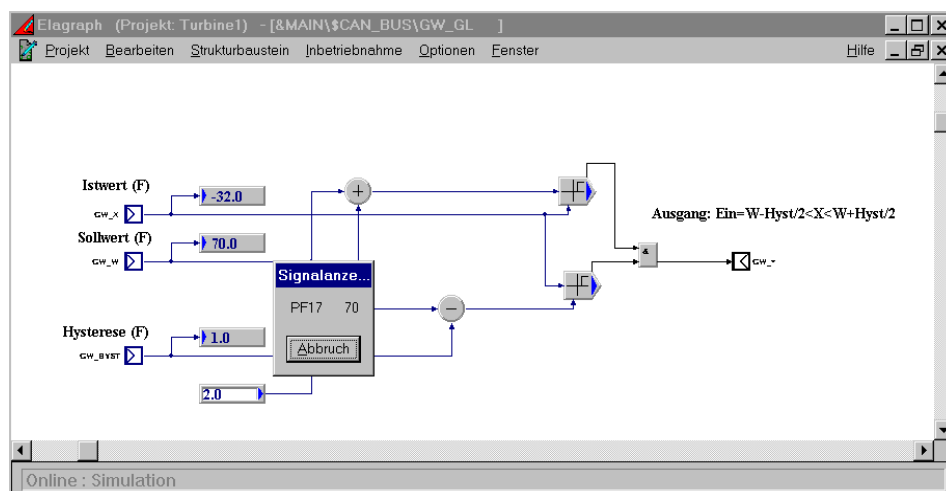
Parameter (ElaGraph)



In dieser Seite können Datenpunkte aus ElaGraph projiziert werden.

ElaGraph Datenpunkte

Wenn sie ein ElaGraph Programm geschrieben haben, ist es möglich jede Verbindungslinie mit dem Treiber aus dem Zielsystem zu lesen. Wird ein Projekt simuliert, und Sie klicken auf eine Verbindungslinie :



Erscheint eine Kennung der Verbindungslinie und der passende Wert. Dieser Mechanismus greift auch während dem Laufenden Programm in Zielsystem.

Hinweis:

Diese Datenpunkte können durch Editiervorgänge in ElaGraph verschoben werden.

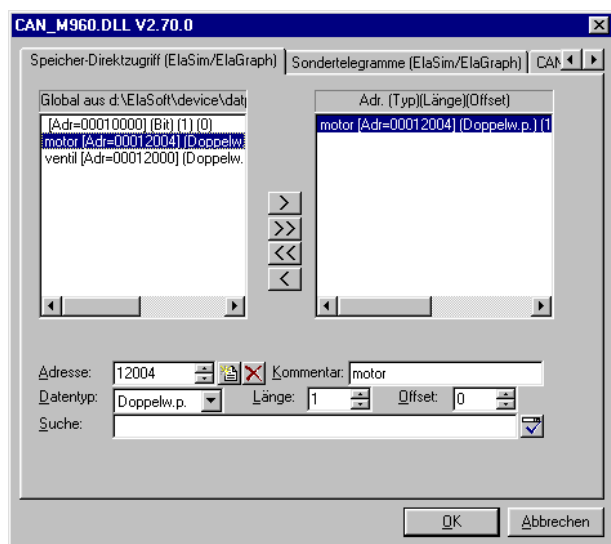
Somit ist von einer Projektierung mit diesen Datenpunkten abzusehen, sie dienen lediglich zu Debug Zwecken.

Bitte beachten Sie, daß Sie zur Nutzung dieser Datenpunkte bestimmte Userware-Bibliotheken zum Zielgerät laden müssen!

Siehe auch: [Voraussetzungen Datenpunkte](#)



Speicher Direktzugriff (ElaSim/ElaGraph)



Eine weitere Zugriffsmöglichkeit auf beliebige Variablen ist der Speicher- Direktzugriff. Hier können Daten direkt unter Angabe der physikalischen Adresse im Zielsystem bearbeitet werden. Man definiert in der linken Liste seine benötigten Adressen, denen man auch einen Klartext geben kann.

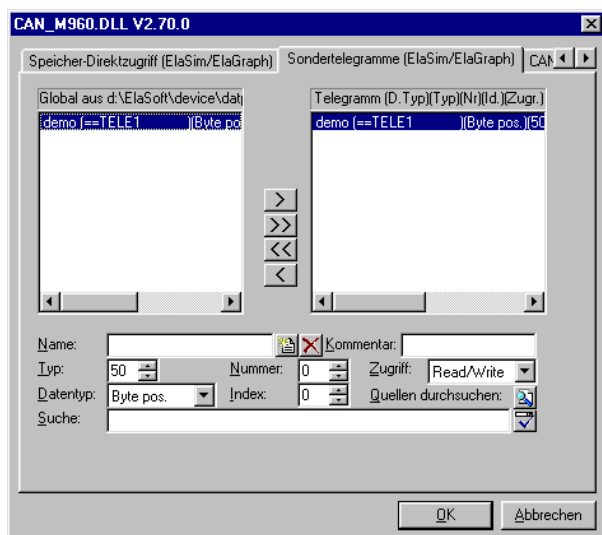
Tabelle	Die Liste der projektierten Adressdatenpunkte. Die Liste wird in einer Datei abgespeichert, die sich im Projekt-Gerätepfad befindet, falls dieser im Gerätedialog angegeben wurde. Ansonsten werden die Daten PC global von allen Geräten zugänglich gespeichert.																								
Adresse	Hexadezimaler Wert der auszulesenden Adresse																								
Kommentar	Klartext der Adresse (ist nur an der Treiberoberfläche bekannt)																								
Datentyp	Definiert, wie der Speicherbereich zu interpretieren ist. Folgende Datentypen stehen zur Verfügung :																								
	<table border="0"> <tr> <td style="padding-left: 20px;">Bezeichnung</td> <td>C- Datentyp</td> </tr> <tr> <td style="padding-left: 20px;">Bit</td> <td>BOOL</td> </tr> <tr> <td style="padding-left: 20px;">Byte</td> <td>Char</td> </tr> <tr> <td style="padding-left: 20px;">Byte pos.</td> <td>unsigned char</td> </tr> <tr> <td style="padding-left: 20px;">Datum</td> <td>STRUCT_DATE</td> </tr> <tr> <td style="padding-left: 20px;">Doppelw.p.</td> <td>unsigned long</td> </tr> <tr> <td style="padding-left: 20px;">Doppelwort</td> <td>long</td> </tr> <tr> <td style="padding-left: 20px;">Gleitkomma</td> <td>float</td> </tr> <tr> <td style="padding-left: 20px;">Text</td> <td>char[]</td> </tr> <tr> <td style="padding-left: 20px;">Wort</td> <td>short</td> </tr> <tr> <td style="padding-left: 20px;">Wort pos</td> <td>unsigned short</td> </tr> <tr> <td style="padding-left: 20px;">Zeit</td> <td>STRUCT_TIME</td> </tr> </table>	Bezeichnung	C- Datentyp	Bit	BOOL	Byte	Char	Byte pos.	unsigned char	Datum	STRUCT_DATE	Doppelw.p.	unsigned long	Doppelwort	long	Gleitkomma	float	Text	char[]	Wort	short	Wort pos	unsigned short	Zeit	STRUCT_TIME
Bezeichnung	C- Datentyp																								
Bit	BOOL																								
Byte	Char																								
Byte pos.	unsigned char																								
Datum	STRUCT_DATE																								
Doppelw.p.	unsigned long																								
Doppelwort	long																								
Gleitkomma	float																								
Text	char[]																								
Wort	short																								
Wort pos	unsigned short																								
Zeit	STRUCT_TIME																								
Länge	Wird hier eine Zahl größer 1 eingetragen, so wird die Adresse als Vektor interpretiert.																								
Offset	Eine Zahl größer 0 bewirkt, daß nicht die Adresse ausgelesen wird, sondern ein mal vielfaches des eingetragenen Datentyps auf die eingetragene Adresse addiert wird (Offset).																								



Bitte beachten Sie, daß der Zugriff auf diese Datenpunkte eines anderen Gerätes von einem Zielgerät aus nicht möglich ist!
Siehe auch: [Voraussetzungen_Datenpunkte](#)






Sondertelegramme (ElaSim/ElaGraph)



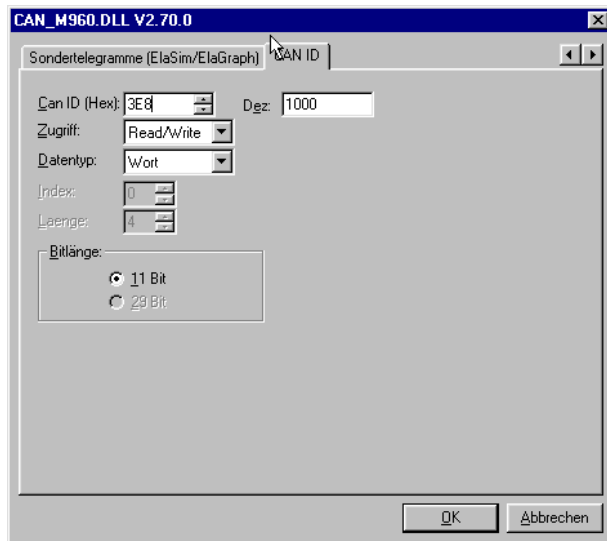
In dieser Seite ist es möglich den Mechanismus von Firmware-Datenpunkten zu nutzen, die nicht in der Seite „Firmware-Variablen“ definiert sind. Dies ist dann besonders hilfreich, wenn man nicht über den aktuellsten Gerätetreiber verfügt, aber eine neue Hardware hat, die den neuen Datenpunkt kennt, oder man hat ein ElaSim Programm geschrieben, in dem ein Datenpunkt definiert wurde (Siehe ElaSim Beschreibung) und man möchte dem Programm vom PC aus eine Nachricht schicken.

Für remanente Datenpunkte sind einige Adressbereich reserviert.

Name	Kennzeichner des Datenpunktes.
	Übernimmt die Einstellungen des Dialogs, und erzeugt einen Eintrag in der „Sondertelegramme“ Liste. Die Liste wird in einer Datei abgespeichert, die sich im Projekt-Gerätepfad befindet, falls dieser im Gerätedialog angegeben wurde. Ansonsten werden die Daten PC global von allen Geräten zugänglich gespeichert.
	Löscht die dunkel unterlegten Einträge aus der „Sondertelegramme“ Liste
Kommentar	Eine kurze Beschreibung für den Datenpunkt.
Typ	Der PARAM_TYP des Telegramms (Siehe ElaSim Beschreibung von 0 bis 49 sind diese Typen von der Firmware belegt – dargestellt wird es im Dialog durch eine rot unterlegte Nummer). Die Bereiche ab 119 sollten Sie ebenfalls nicht benutzen, das sie für selbstdefinierte Datenpunkte zur Verwendung kommen. Verwenden Sie keine selbstdefinierten Datenpunkte, können Sie auch diesen Bereich verwenden.
Nummer	Der Untertyp von PARAM_TYP
Zugriff	Wie darf auf den Datenpunkt zugegriffen werden ? lesend / schreibend / lesend u. schreibend
Datentyp	Welchen Datentyp hat das Telegramm ?
Index	Da ein Telegramm immer 4 Byte lang ist, ist es notwendig z.B. beim Datentyp Byte anzugeben auf welchen „Byteindex“ im Telegramm man zugreifen will. So gibt es bei 4 Byte langen Datentypen keinen Index, bei 2 Byte langen 2 Werte auf die zugegriffen werden kann und bei einem Byte lange 4 Möglichkeiten.
Quellen durchsuchen	Diese Option bietet die Möglichkeit die C Dateien eines ElaSim Projektes nach einem „#define“ zu durchsuchen und aufzulisten. Dies erlaubt Ihnen die Möglichkeit (falls sie sich in ElaSim an eine Namenskonvention für Sondertelegramme halten), alle definierten Telegramme auflisten zu lassen.
	



CAN ID



Ab Protokoll 2.0 sind freie ID's verfügbar (nicht CPU515). Hierbei handelt es sich im eigentlichen Sinn um kein Protokoll, sondern um den direkten Zugriff auf die Daten beliebiger CAN-Identifiers. Dies wird auch als Layer 2 Kommunikation gemäß ISO/ISI Referenzmodell bezeichnet.

CAN ID ermöglicht Ihnen eine Kommunikation mit einem definierten CAN-Identifizier.

CAN ID

CAN-Identifizier, der überwacht werden soll (linkes Eingabefeld ermöglicht hexadezimale Eingabe, rechtes Eingabefeld die dezimale Eingabe).

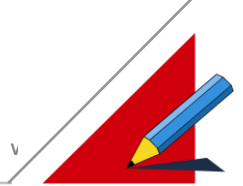
Zugriff

Lesend / schreibend / beides ?

Datentyp

Kennzeichnet den Datentyp des Telegramms

Bezeichnung	C- Datentyp
Bit	BOOL
Byte	Char
Byte pos.	unsigned char
Datum	STRUCT_DATE
Doppelw.p.	unsigned long
Doppelwort	long
Gleitkomma	float
Text	char[]
Wort	short
Wort pos	unsigned short
Zeit	STRUCT_TIME



Profibus Schnittstelle

Die physikalischen Profibus Schnittstelle

- Profibus DP normkonform.

Die Software Protokolle der Profibus Schnittstelle

- Profibus DP Slave Anschaltung

Performance Test

RS232 Schnittstelle direkt

	Nutzdaten [Bytes/sec] Baud=9600 Bits/s
Protokoll 1.0 Download	55 – 60
Protokoll 2.0 Download	720 – 780

RS232 Schnittstelle via Modem

	Nutzdaten [Bytes/sec] Baud=9600 Bits/s
Protokoll 1.0 Download	q.e.d.
Protokoll 2.0 Download	480-520 (167er Prozessor)

CAN Schnittstelle

	Nutzdaten [Bytes/sec] Baud=123000 Bits/s
Protokoll 1.0 Download	380 – 400
Protokoll 2.0 Download	2500 – 2800