CAN / CANopen Erweiterung für IEC 61131-3

User Manual

Version 7.0

Ausgabe Mai 2011

Dokument-Nr.: L-1008d_07

SYSTEC electronic GmbH August-Bebel-Straße 29 D-07973 Greiz
Telefon: +49 (3661) 6279-0 Telefax: +49 (3661) 6279-99
Web: http://www.systec-electronic.com Mail: info@systec-electronic.com

Status/Änderungen

Status: Freigegeben

Datum/Version	Abschnitt	Änderung	Bearbeiter
2007/04/02 6.0	2, 3 und 5	Abschnitte 2, 3 und 5 komplett überarbeitet	R. Sieber
2011/05/25 7.0	4.4.5, 4.4.6, 6	Abschnitte neu: 4.4.5 (CAN_SDO_READ_BIN), 4.4.6 (CAN_SDO_WRITE_BIN) und 6 (CAN Layer 2 FB) neu	R. Sieber

Im Buch verwendete Bezeichnungen für Erzeugnisse, die zugleich ein eingetragenes Warenzeichen darstellen, wurden nicht besonders gekennzeichnet. Das Fehlen der © Markierung ist demzufolge nicht gleichbedeutend mit der Tatsache, dass die Bezeichnung als freier Warenname gilt. Ebenso wenig kann anhand der verwendeten Bezeichnung auf eventuell vorliegende Patente oder einen Gebrauchsmusterschutz geschlossen werden.

Die Informationen in diesem Handbuch wurden sorgfältig überprüft und können als zutreffend angenommen werden. Dennoch sei ausdrücklich darauf verwiesen, dass die Firma SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf den Gebrauch oder den Inhalt dieses Handbuches zurückzuführen sind. Die in diesem Handbuch enthaltenen Angaben können ohne vorherige Ankündigung geändert werden. Die Firma SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

Ferner sei ausdrücklich darauf verwiesen, dass SYS TEC electronic GmbH weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf falschen Gebrauch oder falschen Einsatz der Hard- bzw. Software zurückzuführen sind. Ebenso können ohne vorherige Ankündigung Layout oder Design der Hardware geändert werden. SYS TEC electronic GmbH geht damit keinerlei Verpflichtungen ein.

Informieren Sie sich:

Kontakt	**Direkt**	**Ihr Lokaler Distributor**
Adresse:	SYS TEC electronic GmbH August-Bebel-Str. 29 D-07973 Greiz GERMANY	Sie finden eine Liste unserer Distributoren unter http://www.systec-electronic.com/distributors
Angebots-Hotline:	+49 (0) 36 61 / 62 79-0 info@systec-electronic.com	
Technische Hotline:	+49 (0) 36 61 / 62 79-0 support@systec-electronic.com	
Fax:	+49 (0) 36 61 / 6 79 99	
Webseite:	http://www.systec-electronic.com	

7. Auflage Mai 2011

Inhalt

Tabellenverzeichnis

Bilderverzeichnis

1 Einleitung

Dieses Handbuch beschreibt im ersten Teil die Einbeziehung von CANopen-Diensten in SPS-Programme entsprechend der Norm IEC61131-3. Dies ermöglicht die Verwendung von Netzwerkvariablen sowie den Zugriff auf CANopen über spezifische Funktionsbausteine. Voraussetzung hierfür ist eine SPS mit CANopen-Interface.

Grundlage der Bereitstellung von CANopen-Diensten für SPS-Programme nach EC1131-3 bildet die vom CiA (CAN in Automation e.V.) im CiA Draft Standard 405 definierte Funktionalität. Diese wird von SYSTEC noch um zusätzliche, herstellerspezifische Funktionsbausteine (z.B. Senden und Empfangen von PDOs bzw. CAN Layer 2 Nachrichten, Erzeugen von SYNC-Objekten) ergänzt.

Der zweite Teil dieses Handbuches beschreibt die Anwendung von Funktionsbausteinen zur Verarbeitung protokoll-unabhängiger CAN-Nachrichten (CAN Layer 2 Nachrichten) in einem SPS-Programm. Mit Hilfe dieser CAN Layer 2 Funktionsbausteine kann die SPS auch mit nicht CANopen-fähigen CAN-Geräten kommunizieren.

Verwendung von CANopen-Netzwerkvariablen (→ Abschnitt 3)

Netzwerkvariablen stellen die einfachste Form des Datenaustausches mit anderen CANopen-Knoten dar. In einem SPS-Programm erfolgt der Zugriff auf Netzwerkvariablen in derselben Form wie auf interne, lokale Variablen der SPS. Aus Sicht des SPS-Programmierers ist es somit völlig unbedeutend, ob z.B. eine Input-Variable einem lokalen Eingang der Steuerung zugeordnet ist oder einen Eingang eines dezentralen Erweiterungsmoduls repräsentiert. Die Anwendung von Netzwerkvariablen erfordert nur allgemeine Grundkenntnisse über CANopen. **Im Allgemeinen sind ein CANopen-Konfigurator sowie die Verfügbarkeit von EDS-Dateien für die einzelnen CANopen-Geräte Voraussetzung zur Einbindung von Netzwerkvariablen (siehe Abschnitt 3.1).**

Mit Hilfe von Netzwerkvariablen ist es möglich,

- die Ein- und Ausgänge einer SPS durch den Einsatz von CANopen-Geräten zu erweitern,
- Prozessdaten zwischen verschiedenen Steuerungen auszutauschen, um so dezentrale Automatisierungsprojekt zu realisieren und
- neben SYSTEC Modulen auch beliebige andere, spezialisierte CANopen-Geräte in die Projektplanung einzubeziehen, um so mit Hilfe modularer Standardbaugruppen auch Steuerungen für Spezialaufgaben aufbauen zu können.

Verwendung von CANopen-Funktionsbausteinen (→ Abschnitt 4)

CANopen-Funktionsbausteine ermöglichen den direkten Zugriff auf spezifische CANopen-Dienste und bieten somit ein hohes Maß an Flexibilität in der Anwendung. Auch ist für deren Einsatz weder das Vorhandensein eines CANopen-Konfigurators noch von EDS-Dateien notwendig. **Dafür setzt die Verwendung von CANopen-Funktionsbausteinen detaillierte Kenntnisse über CANopen und seine verschiedenen Dienste voraus.**

Mit Hilfe von CANopen-Funktionsbausteinen ist es möglich,

- Daten direkt per SDO (Service Data Object) oder per PDO (Process Data Object) mit anderen CANopen-Knoten auszutauschen,
- den Status anderer CANopen-Knoten abzufragen und zu beeinflussen,
- Fehlermeldungen anderer CANopen-Knoten zu empfangen und
- die Generierung von SYNC-Nachrichten zu ermöglichen.

Verwendung von CAN Layer 2 Funktionsbausteinen (→ Abschnitt 6)

Die CAN Layer 2 Funktionsbausteine ermöglichen den protokoll-unabhängigen Austausch von Daten zwischen SPS und beliebigen CAN-Geräten. Hierbei verarbeitet das SPS-Programm unmittelbar die CAN-Nachrichten, die auf dem CAN-Bus übertragen werden.

Mit Hilfe von CAN Layer 2 Funktionsbausteinen ist es möglich,

- das CAN-Interface der Steuerung mit vom SPS-Programm vorgegebenen Parametern zu konfigurieren,
- CAN-Telegramme unmittelbar durch das SPS-Programm zu erstellen bzw. zu verarbeiten
- CAN-Nachrichten im Extended-Frame-Format (29 Bit CAN Identifier gemäß CAN 2.0B) zu senden bzw. zu empfangen und

Hinweis: CAN Layer 2 Funktionsbausteine können nicht simultan mit CANopen-Diensten auf demselben CAN-Interface verwendet werden. CAN Layer 2 Funktionsbausteine sind nur nutzbar, wenn zuvor die CANopen-Funktionalität für die betreffende CAN-Schnittstelle deaktiviert wurde.

Teil 1

CANopen

(High Level Protokoll)

2 Grundlagen der CANopen-Anbindung einer SPS

2.1 Unterschiede zwischen SPS mit und ohne CANopen-Master

Die CANopen-Erweiterung einer SPS, insbesondere aber die CANopen-Funktionsbausteine für IEC61131 basieren auf verschiedenen Diensten von CANopen, von denen einige gleichzeitig auf mehreren Knoten aktiv sein können (z.B. PDO- und SDO-Transfer), während andere nur exklusiv von einem einzigen Knoten ausgeführt werden dürfen (z.B. NMT-Master-Dienste). Diese Trennung widerspiegelt sich in der Aufteilung zwischen "SPS mit CANopen-Master" und "SPS ohne CANopen-Master". Auf einer SPS ohne CANopen-Master stehen nur die nicht exklusiven Dienste zur Verfügung, während auf einer SPS mit CANopen-Master zusätzlich auch die exklusiven Dienste nutzbar sind. Eine Übersicht hierzu gibt Tabelle 1. In Abhängigkeit von den jeweils verfügbaren Diensten steht auch dem SPS-Programm ein unterschiedlicher Umfang an CANopen-Funktionsbausteinen zur Verfügung. Eine detaillierte Auflistung hierzu enthält Tabelle 7 im Abschnitt 4.1.2.

Tabelle 1: Verfügbarkeit von Diensten auf SPS mit und ohne CANopen-Master

CANopen-Dienst	SPS ohne CANopen-Master	SPS mit CANopen-Master
PDO	X	X
SDO	X	X
Heartbeat	Producer / Consumer	Producer/Consumer
Lifeguarding	Slave	Master
NMT-Master	-	X
SYNC-Producer	-	X

Zeichenerklärung:
X = Funktionalität verfügbar
\- = Funktionalität nicht verfügbar

Bei SYSTEC-Steuerungen erfolgt die Unterscheidung zwischen "SPS mit CANopen-Master" und "SPS ohne CANopen-Master" modelabhängig entweder über eine Konfigurationsoberfläche, durch Bedienelemente (z.B. DIP-Schalter) oder anhand der Knotenadresse. Da innerhalb eines CANopen-Netzes jedem Knoten eine eindeutige, unikate Adresse zuzuordnen ist, wird auch bei der Kopplung an die Knotenadresse sichergestellt, dass netzwerkweit die Masterfunktionalität nur von einem einzigen Knoten ausgeführt werden kann.

Bei Steuerungstypen, bei denen die Aktivierung der Masterfunktionalität über die Auswahl der Knotenadresse erfolgt, ist die Masterfunktionalität einer SPS fest an die Knotenadresse 20H gebunden. Auf jeder anderen Knotenadresse ist bei solchen Steuerungstypen nur die Funktionalität "SPS ohne CANopen-Master" nutzbar.

Beinhaltet das Netzwerk nur eine einzige SPS, so muss diese auf jeden Fall im Master-Mode betrieben werden (z.B. Knotenadresse 20H). Dadurch wird sichergestellt, dass das SPS-Programm den Status anderer Knoten mit Hilfe des CANopen-Funktionsbausteines *CAN_GET_STATE* überwachen kann (siehe Abschnitt 2.2 für Grundlagen sowie Abschnitt 4.5.1 zur Beschreibung des Funktionsbausteines). Außerdem ist es nur einer SPS im Master-Mode möglich, SYNC-Objekte zu generieren (siehe Abschnitt 4.5.8). In Netzwerken mit mehreren Steuerungen muss genau eine SPS im Master-Mode aktiv sein.

2.2 Knotenkonfiguration durch SPS mit CANopen-Master

Eine SPS mit CANopen-Master ist in der Lage, andere Geräte im Netzwerk zu konfigurieren. Grundlage hierfür sind die beim Download des SPS-Programms ebenfalls mit auf die Steuerung übertragen DCF-Dateien der IO-Module. Außerdem übernimmt eine SPS mit CANopen-Master automatisch die Überwachung aller CANopen-Baugruppen im Netz mittels Heartbeat oder Lifeguarding. Dies ist auch gleichzeitig Grundlage für die Statusabfrage unter Verwendung des Funktionsbausteines *CAN_GET_STATE* (siehe Abschnitt 4.5.1). Primär wird die Knotenüberwachung durch die entsprechenden Einträge in der DCF-Datei des jeweiligen Knotens bestimmt. Die alternativ verwendete Default-Konfiguration von Heartbeat bzw. Lifeguarding, die dann benutzt wird, wenn für den betreffenden Knoten keine DCF-Datei vorhanden ist, beschreibt Abschnitt 5.5. Intern erfolgt die Knotenüberwachung durch die Master-SPS nach folgendem Prinzip:

Beim Zuschalten der Betriebsspannung (Reset) und nach dem Programmdownload überprüft die Master-SPS, ob in ihrem eigenen Object-Dictionary das Objekt 1F81H angelegt wurde (d.h., ob das Objekt 1F81H in der DCF-Datei der Master-SPS enthalten ist). Ist dies der Fall, werden die darin spezifizierten Knoten mittels Knoten-Scan gesucht. Ist das Objekt 1F81H nicht vorhanden, ermittelt die Steuerung alle im Netz verfügbaren CANopen-Geräte durch einen Netzwerk-Scan, der standardmäßig den gesamten Bereich für alle Knotenadressen von 1 bis 127 überdeckt (bei Bedarf kann der Knotenbereich über Index 3001H im Object-Dictionary der Master-SPS eingeschränkt werden, siehe Abschnitt 5.2). Dabei wird auf jeder Knotenadresse versucht, den Index 1000H im Object-Dictionary des Knotens auszulesen (Gerätetyp). Antwortet ein Gerät, wird geprüft, ob für diesen Knoten eine DCF-Datei auf der SPS vorhanden ist. Ist dies der Fall, konfiguriert die SPS das Gerätes mit den in der DCF-Datei spezifizierten Parametern. Ist keine DCF-Datei verfügbar, prüft die SPS, ob das betreffende Gerät mit Heartbeat überwacht werden kann. Unterstützt der Knoten kein Heartbeat, wird alternativ die Verfügbarkeit von Lifeguarding getestet. Geräte, die weder Heartbeat noch Lifeguarding unterstützen, können nicht überwacht werden. Somit ist für die betreffenden Knoten auch keine Statusabfrage durch das SPS-Programm möglich (siehe unten). Den Ablauf der Knotenkonfiguration durch CANopen-Master verdeutlicht Bild 1. Details zur Konfiguration von Heartbeat und Lifeguarding vermittelt Abschnitt 5.5.

Empfängt die Steuerung nach dem Netzwerk-Scan Bootup-Nachrichten von weiteren CANopen-Geräten (z.B. weil diese Geräte über eine Spannungsversorgung gespeist werden, die erst nachträglich zugeschalten wird), werden diese Knoten ebenfalls wie oben beschrieben konfiguriert.

Hinweis: Bei der Verwendung von DCF-Dateien zur Knotenkonfiguration ist der Anwender dafür verantwortlich, das Nodeguarding (Heartbeat bzw. Lifeguarding) entsprechend zu aktivieren. Andernfalls ist für den betreffenden Knoten keine Knotenüberwachung und damit auch keine Statusabfrage durch das SPS-Programm möglich (siehe unten).

Treten während der Konfigurationsphase Fehler auf, setzt die SPS einen entsprechenden Fehlerzustand und prüft diesen später beim Starten des SPS-Programms (siehe unten). Mögliche Ursachen für Konfigurationsfehler sind:

- ein im Objekt 1F81H als "Mandatory Slave" gekennzeichneter Knoten ist nicht vorhanden (das Fehlen eines nicht als "Mandatory Slave" gekennzeichneten Knotens ist **kein** Fehler) oder
- beim Konfigurieren eines Knotens mit den in der DCF-Datei spezifizierten Parametern trat ein Fehler auf (SDO-Abort wegen Zugriff auf nicht vorhandenes Objekt, Schreibzugriff auf nur lesbares Objekt, sonstige logische Konfigurationsfehler)

Beim Starten des SPS-Programms prüft die SPS zunächst, ob in der vorangegangenen Konfigurationsphase Fehler auftraten. War die Konfiguration erfolgreich und ist im Objekt 1F80H nichts gegenteiliges festgelegt, sendet die Master-SPS das NMT-Kommando "Enter Pre-Operational State", gefolgt von "Start Remote Node" für alle Knoten (Knotenadresse = 0). Dadurch werden die CANopen-Knoten veranlasst, einmalig ihre PDOs (**P**rocess **D**ata **O**bject) zu senden. Anschließend wartet die SPS bis die PDOs verarbeitet und die empfangenen Werte in Netzwerkabbild hinterlegt

wurden, bevor das SPS-Programm gestartet wird. Durch diesen Ablauf ist die Anfangsinitialisierung der Netzwerkvariablen sicher gestellt (siehe Abschnitt 2.3)

Hinweis: Die Wartezeiten können über Index 3003H im Object-Dictionary der Master-SPS konfiguriert werden, siehe Abschnitt 5.4.

Ist im Objekt 1F80H das Bit 3 ("Do not send NMT-Start Remote Node") gesetzt oder traten während der Konfigurationsphase Fehler auf, schaltet die SPS das Netz nicht selbständig in den Zustand Operational. Das SPS-Programm kann im Zustand Pre-Operational zunächst mit Hilfe der im Abschnitt 4.4 beschriebenen SDO-Funktionsbausteine weitere Konfigurationen vornehmen und schließlich mit Hilfe des im Abschnitt 4.5.2 beschriebenen NMT-Funktionsbausteines das Netzwerk (zwangsweise) in den Zustand Operational versetzten.

Beim Beenden des SPS-Programms sendet die Master-SPS das NMT-Kommando "Enter Pre-Operational State", um so allen CANopen-Knoten anzuzeigen, dass die PDO-Verarbeitung deaktiviert wurde.

Sowohl bei Heartbeat als auch bei Lifeguarding sendet ein Knoten in regelmäßigen Abständen (Überwachungszeit) seinen aktuellen Status an die SPS mit CANopen-Master. Dieser Status wird intern in der Netzwerkschicht ausgewertet und beim Aufruf des Funktionsbausteines *CAN_GET_STATE* an das SPS-Programm übergeben. Damit der Baustein *CAN_GET_STATE* auch auf Steuerungen ohne Master nutzbar ist, wird jeder erkannte Zustandswechsel eines Knotens per Broadcast-Nachricht (Default-COBID 50H, Konfiguration über Index 3002H, siehe Abschnitt 5.3) an alle anderen Steuerungen weiter gemeldet. Daher ist beim Deaktivieren der Master-SPS (z.B. durch Stoppen des SPS-Programms) auch auf den Steuerungen ohne Master keine Statusabfrage mehr möglich. Der Funktionsbaustein *CAN_GET_STATE* liefert dann den Status *UNKNOWN*. Dieser Status wird auch dann zurückgeliefert, wenn der betreffende Knoten weder Heartbeat noch Lifeguarding unterstützt. In diesem Fall ist aber prinzipiell keine Statusabfrage für den betreffenden Knoten möglich.

Die Master-SPS überwacht alle Knoten im Netzwerk mit dem Ziel, dem SPS-Programm (und damit dem Anwender) eine Statusabfrage der Knoten zu ermöglichen. Die Master-SPS reagiert jedoch nicht selbst auf auftretende Fehler, dies ist vielmehr Aufgabe des Anwenders.

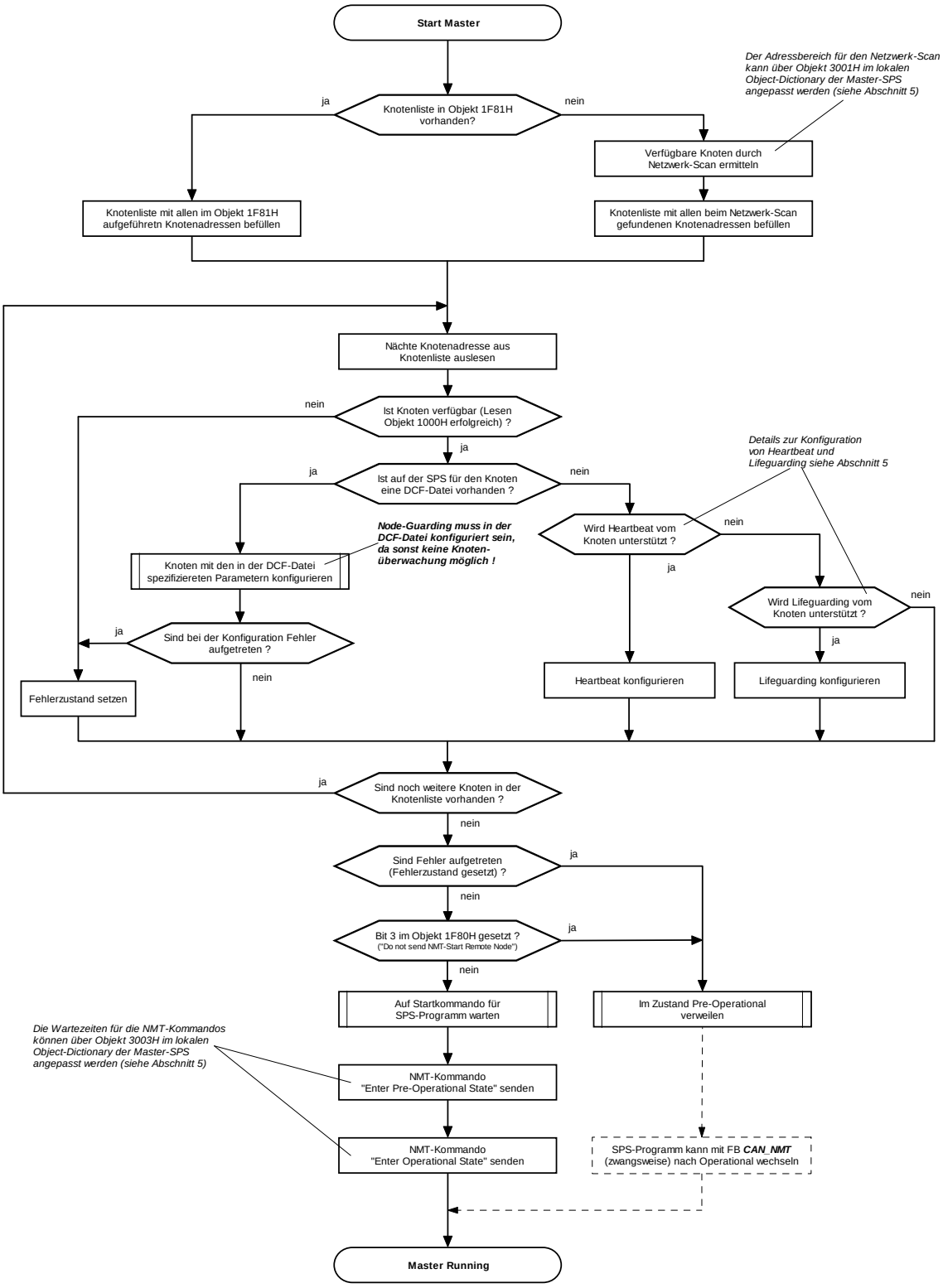


Bild 1: Ablauf der Knotenkonfiguration durch CANopen-Master

2.3 Anfangsinitialisierung der Netzwerkvariablen

Die primäre Anfangsinitialisierung von Netzwerkvariablen erfolgt mit den in der DCF-Datei der SPS festgelegten Initialwerten (Eintrag "ParameterValue=" bzw. "DefaultValue="). Diese werden beim Anlegen des dynamischen Object-Dictionaries (siehe Abschnitt 3.1) als Startwerte für die Netzwerkvariablen verwendet.

Zwischen den einzelnen Knoten wird der Inhalt von Netzwerkvariablen durch Versenden von PDOs (**P**rocess **D**ata **O**bject) ausgetauscht. In der Regel erfolgt die Übertragung der PDOs ereignisgesteuert, so dass die Kommunikationspartner lediglich über Änderungen der Variablen informiert werden (asynchrone Übertragung). Dieses Verfahren reduziert die Busbelastung auf das notwendige Minimum. Aus Sicht eines CANopen-Gerätes bedeutet das aber, dass es ohne weitere Maßnahmen vom Startzeitpunkt bis zur ersten Änderung den realen Wert einer Variable nicht kennt. Daher ist es zwingend notwendig, dass die Master-Steuerung beim Starten des SPS-Programms alle CANopen-Geräte veranlasst, einmalig ihre PDOs zu senden. Somit kennen alle CANopen-Knoten einschließlich der Master-SPS selbst die realen Anfangswerte der Eingangs-Netzwerkvariablen. In der Folge ist dann die Mitteilung von Änderungen ausreichend.

Die Anfangsinitialisierung von Netzwerkvariablen ist auf Steuerungen mit und ohne Masterfunktionalität unterschiedlich realisiert:

SPS mit CANopen-Master

Die einzige, von allen CANopen-Geräten unterstützte Möglichkeit, die Knoten zwangsweise zum Senden ihrer PDOs zu veranlassen, besteht im Wechsel des Zustandes von Pre-Operational nach Operational. Dazu sendet die Master-SPS beim Starten ihres eigenen SPS-Programms zunächst das NMT-Kommando "Enter Pre-Operational State", gefolgt von "Start Remote Node" für alle Knoten (Knotenadresse = 0). Dadurch werden alle zu diesem Zeitpunkt am Netzwerk aktiven CANopen-Knoten veranlasst, einmalig ihre PDOs zu senden. **Nach dem Starten eines SPS-Programms auf der Master-SPS befinden sich somit alle aktiven Knoten des Netzwerkes im Zustand Operational.**

Hinweis: Die Wartezeit zwischen den NMT-Kommandos "Enter Pre-Operational State" und "Start Remote Node" kann über Index 3003H / Subindex 1 im Object-Dictionary der Master-SPS konfiguriert werden (siehe Abschnitt 5.4).

Die SPS selbst verzögert das erstmalige Senden ihrer eigenen Ausgangs-Netzwerkvariablen bis zur Beendigung des ersten SPS-Zyklus. Dadurch besteht für das SPS-Programm die Möglichkeit, die Ausgangs-Netzwerkvariablen innerhalb des ersten SPS-Zyklus mit spezifischen Werten vorzubelegen, die von den in DCF-Datei festgelegten Initialwerten abweichen dürfen (siehe auch Abschnitt 7). Gleichzeitig wird so verhindert werden, dass die SPS Anfangs-PDOs mit ungültigen Daten (Anfangs-PDOs mit Null-Bytes) versendet.

CANopen-IO-Geräten, die sich erst nach dem Starten des SPS-Programms auf der Master-SPS am Netzwerk anmelden, sendet die Master-SPS das NMT-Kommandos "Enter Operational State" mit der spezifischen Knotenadresse des betreffenden Gerätes. Dadurch wird dieses Gerät ebenfalls veranlasst, seine PDOs zu mit Initialwerten übertragen. Gleichzeitig sendet auch die Master-SPS einmalig alle ihre PDOs, unabhängig davon, ob sich darin Werte geändert haben oder nicht. Hierdurch ist ebenfalls die korrekte Anfangsinitialisierung von Netzwerkvariablen für verspätet zugeschaltete Geräte gewährleistet.

SPS ohne CANopen-Master

Eine SPS ohne CANopen-Master verhält sich aus Sicht des Netzwerkes genau wie jedes andere CANopen-IO-Gerät. Beim Starten (Zuschalten der Betriebsspannung, Reset) sendet die SPS eine Bootup-Nachricht und meldet sich somit im Netzwerk an. Anschließend verweilt die SPS im Zustand Pre-Operational bis sie vom CANopen-Master das NMT-Kommandos "Enter Operational State" empfängt. Darauf reagiert die SPS genau wie jedes andere CANopen-IO-Gerät mit dem Senden ihrer Anfangs-PDOs.

Applikationsabhängig kann es für eine SPS notwendig sein, die Ausführung des eigenen SPS-Programms an den jeweiligen Netzwerkzustand (Pre-Operational / Operational) zu koppeln. Hierzu kann der Zustand des eigenen Knotens mit Hilfe des CANopen-Funktionsbausteines *CAN_GET_STATE* ermittelt werden (siehe Abschnitt 4.5.1). Das folgende Beispiel verdeutlicht die Abfrage und Auswertung des eigenen Knotenstatus:

```
VAR
    FB_CanGetState : CAN_GET_STATE;
END_VAR

CAL   FB_CanGetState (
      DEVICE := 0,                    (* own node *)
      ENABLE := TRUE)
LD    FB_CanGetState.STATE
EQ    16#0005                         (* Operational ? *)
JMPCN PreOperationalMode

OperationalMode:
(* ... *)
RET

PreOperationalMode:
(* ... *)
RET
```

3 IEC61131-Netzwerkvariablen für CANopen

3.1 Allgemeine Grundlagen für Netzwerkvariablen

Zahlreiche Steuerungsbaugruppen können mit Hilfe eines CANopen-Netzwerkes untereinander Daten austauschen bzw. durch CANopen-IO-Module um zusätzliche Ein- und Ausgänge erweitert werden. Auf Ebene des SPS-Programms erfolgt der Datenaustausch über Netzwerkvariablen, die entsprechend der Norm IEC61131-3 als "VAR_EXTERNAL" deklariert werden und somit als "außerhalb der Steuerung" gekennzeichnet sind. Die SPS selbst verwaltet lokal eine Kopie dieser Variablen, wobei der Netzwerkschicht die Aufgabe zukommt, diese Kopie mit dem realen Wert des CANopen-Gerätes zu synchronisieren. Von besonderer Bedeutung ist dabei auch die im Abschnitt 2.3 beschriebene Anfangsinitialisierung der Netzwerkvariablen.

Aus Sicht von CANopen stellt die SPS ein "gewöhnliches" IO-Modul dar, dessen Ein- und Ausgänge für den Anwender jedoch nicht auf Klemmen herausgeführt sind, sondern als Netzwerkvariablen in das Prozessabbild gemappt werden. Je nach Anzahl und Umfang der im SPS-Programm verwendeten Netzwerkvariablen ändert sich das Erscheinungsbild der SPS in Bezug auf netzwerkseitige Ein- und Ausgänge, so dass sich ein und dieselbe SPS bei der Ausführung unterschiedlicher Programme gegenüber dem CANopen-Netzwerk verschieden darstellen kann. Die SPS verwendet hierzu ein dynamisches Object-Dictionary (datenbank-ähnliche Struktur zur Verwaltung von Variablen sowie Kommunikations- und Mappingparametern), CANopen-IO-Module besitzen dagegen in de Regel ein statisches Object-Dictionary.

Entsprechend den Festlegungen im CiA Draft Standard 405 werden die Netzwerkvariablen einer SPS innerhalb des Object-Dictionary im Bereich von Index A000h - AFFFh angelegt.

Für die weiteren Erläuterungen zur Verknüpfung von Steuerungen mit dezentralen Erweiterungs-baugruppen spielen folgende Begriffe eine zentrale Rolle:

CANopen-IO-Modul: Das CANopen-IO-Modul ist eine Baugruppe, die dem Netzwerk bestimmte Ressourcen wie Ein- und Ausgänge zur Verfügung stellt. Für das Netzwerk Management (NMT) ist ein solches Modul eine Slave-Baugruppe.

Mapping: Als Mapping wird die Zuordnung von Variablen sowie Ein- und Ausgängen auf die Bytes bzw. Bitpositionen innerhalb einer CAN-Nachricht bezeichnet.

CANopen-Konfigurator: Der CANopen-Konfigurator ist ein spezielles Software-Tool, mit dessen Hilfe es möglich ist, CANopen-Netzwerke zu planen und zu verwalten, Ein- und Ausgänge verschiedener Baugruppen untereinander logisch zu verbinden sowie Einstellungen von Netzwerkparameter vorzunehmen. Weiterhin dient der CANopen-Konfigurator zur Verknüpfung von Netzwerkvariablen in SPS-Programmen mit den Ein- und Ausgängen des jeweiligen CANopen-IO-Moduls. **Ein CANopen-Konfigurator ist grundsätzlich ein externes Software-Tool, das nicht im Lieferumfang des IEC61131-Programmier-systems *OpenPCS* enthalten ist.** Hierfür geeignet ist beispielsweise das Programm "ProCANopen" der Firma Vector Informatik.

EDS-Datei: Die EDS-Datei (Electronic Data Sheet) wird vom Hersteller des Gerätes mitgeliefert und beschreibt die verschiedenen Eigenschaften der Baugruppe wie z.B. nutzbare IOs, werksseitige Standardeinstellungen für Mapping und Netzwerkkommunikation sowie die vom Anwender modifizierbaren Parameter.

DCF-Datei: Die DCF-Datei (Device Configuration File) wird vom CANopen-Konfigurator als Resultat des Konfigurationsprozesses erzeugt. Dabei benutzt der Konfigurator die EDS-Datei als "Vorlage" und ergänzt diese um die vom Anwender festgelegten Parameter wie z.B. Identifier und Mapping.

Um einer SPS dezentrale IOs zuzuordnen, werden die im SPS-Programm verwendeten Netzwerkvariablen mit Ein- und Ausgängen von CANopen-IO-Baugruppen verknüpft. In der Regel ist hierzu ein CANopen-Konfigurator notwendig. Im Gegensatz zu Standard-IO-Baugruppen existiert für eine SPS jedoch kein EDS-File mit Angabe der zur Verfügung stehenden Ein- und Ausgänge. Für eine SPS legt erst der Endanwender durch ein konkretes SPS-Programm Anzahl und Typ der über CANopen erreichbaren Ein- und Ausgänge fest, indem er hierfür die entsprechenden Netzwerkvariablen deklariert. Daher existiert für eine SPS nur ein allgemeines EDS-File, aus dem lediglich hervorgeht, dass die Steuerung dynamische Objekte unterstützt.

Voraussetzung für die Verwendung von Netzwerkvariablen ist in der Regel das Vorhandensein von EDS-Dateien für die jeweiligen CANopen-IO-Module sowie eines CANopen-Konfigurators. Für einfache Netzwerktopologien (wenige dezentrale Standard-IO-Baugruppen) können auch die im Lieferumfang der Programmierumgebung *OpenPCS* enthaltenen vordefinierte DCF-Dateien verwendet werden (siehe Abschnitt 3.2.3). Sind diese Voraussetzungen nicht erfüllt, können keine Netzwerkvariablen eingesetzt werden. Eine Netzwerkkommunikation ist dann nur mit Hilfe der im Abschnitt 4 beschriebenen CANopen-Funktionsbausteine möglich.

3.2 Konfigurationsprozess

3.2.1 Netzwerk-Konfiguration

Von großer Bedeutung ist das EDS-File für den Konfigurationsprozess von IO-Baugruppen. Dazu wird das EDS-File vom CANopen-Konfigurator eingelesen, um dem Anwender Zugriff auf die vom CANopen-IO-Modul bereitgestellten Ressourcen zu ermöglichen. Durch die Konfiguration legt der Anwender u.a. fest, welche Ein- bzw. Ausgänge verwendet werden, in welchem Bit oder Byte einer CAN-Nachricht ein entsprechender Wert auf dem Bus übertragen wird (Mapping) und welcher Identifier hierfür zu benutzen ist. Als Resultat dieses Konfigurationsprozesses erzeugt der CANopen-Konfigurator eine DCF-Datei für den entsprechenden Knoten (siehe Bild 2). Eine manuelle Konfiguration von CANopen-Baugruppen ist aber nur dann erforderlich, wenn der Anwender die vom Hersteller fest vorgegebenen Standardparameter (Identifier, Mapping) ändern möchte oder muss. Die Standardparameter ergeben sich in der Regel nach einem fest definierten Algorithmus aus der einstellbaren Knotennummer (Node-ID) eines Gerätes und sind dem jeweiligen Handbuch zu entnehmen.

Im Gegensatz zu IO-Baugruppen mit statischen Ein- und Ausgängen benutzt eine SPS dynamische Objekte - also die im jeweiligen SPS-Programm definierten Netzwerkvariablen. Da aber dynamisch zur Laufzeit angelegte Objekte dem Hersteller eines Gerätes nicht bekannt sein können, existieren hierfür auch keine Angaben im EDS-File. Darum ist für die Verknüpfung dynamischer Objekte stets ein Konfigurator notwendig. Das Ergebnis des Konfigurationsprozesses wird dann wiederum in der DCF-Datei abgelegt. Das IEC61131-Programmiersystem verwendet die vom Konfigurator generierte DCF-Datei für die SPS zum Auflösen der als VAR_EXTERNAL deklarierten Netzwerkvariablen und generiert daraus die notwendigen Steuerinformationen für die Netzwerkschicht. **Die DCF-Datei der Steuerung ist damit das zentrale Bindeglied zwischen CANopen und dem IEC61131-SPS-Programm.** Einen Überblick über die Konfiguration von CANopen-Knoten vermittelt Bild 2.

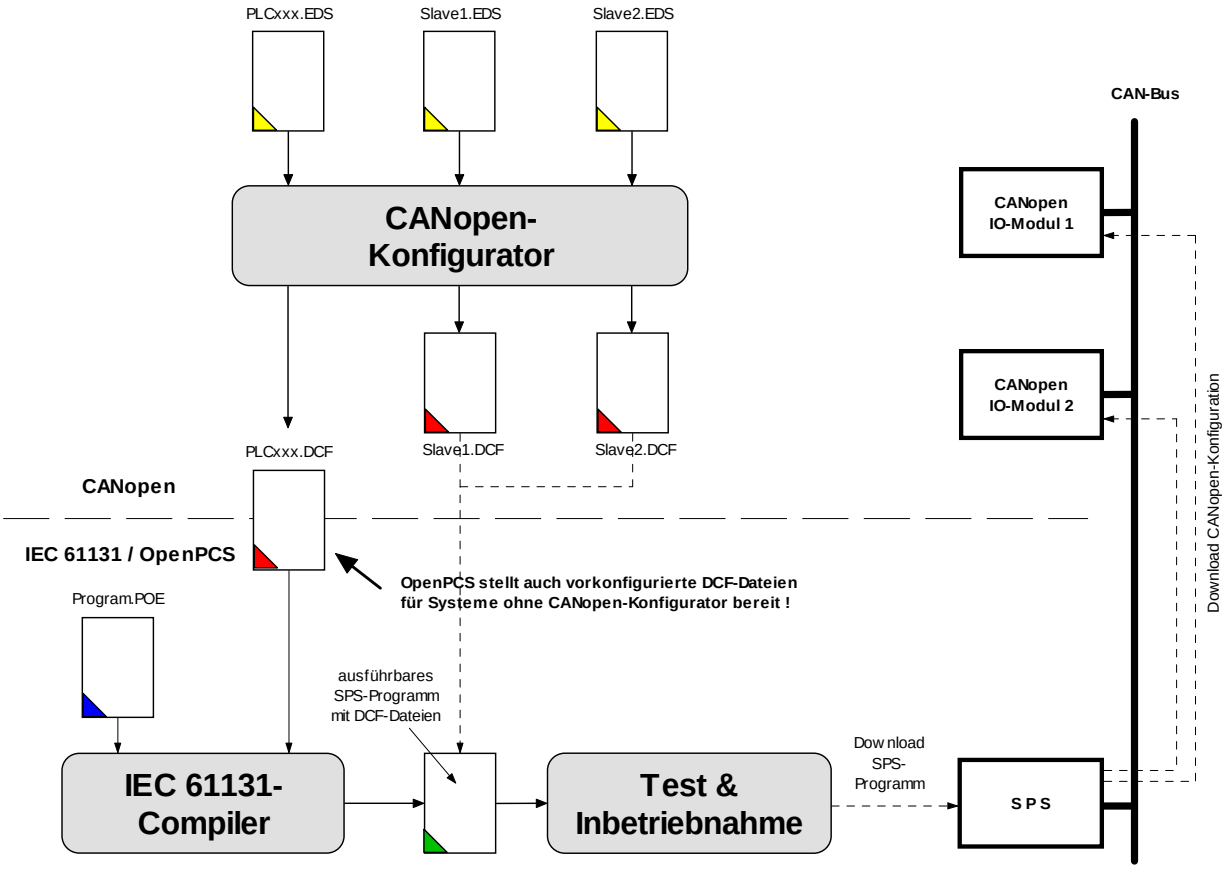


Bild 2: Konfiguration der CANopen-Knoten

Die als Resultat der Netzwerkkonfiguration entstandenen DCF-Dateien werden im IEC61131-Programmiersystem der Steuerungs-Ressource zugeordnet. Die DCF-Datei für die SPS wird sowohl zum Auflösen der als VAR_EXTERNAL deklarierten Netzwerkvariablen verwendet, als auch zur Konfiguration des Object-Dictionaries der SPS selbst. Die DCF-Dateien der übrigen IO-Module werden beim Download des SPS-Programms ebenfalls mit auf die Steuerung übertragen und dort gespeichert. Der in der SPS enthaltene CANopen-Master verwendet diese DCF-Dateien, um damit beim Systemstart die IO-Module in der Feldebene entsprechend zu konfigurieren. Voraussetzung hierfür ist jedoch, dass die Masterfunktionalität der SPS-Baugruppe aktiviert wurde (z.B. per DIP-Schalter oder Konfigurationsdatei, Details hierzu sind dem jeweiligen Steuerungs-Handbuch zu entnehmen).

Alternativ besteht auch die Möglichkeit, CANopen-IO-Module durch das SPS-Programm zu konfigurieren. Hierzu können beim Programmstart die notwendigen Parameter mit Hilfe der im Abschnitt 4.4 beschriebenen SDO-Funktionsbausteine in das Object-Dictionary der IO-Baugruppen geschrieben werden.

Ablage der EDS-Dateien für SYSTEC-Geräte:

Die EDS-Dateien für verschiedene SYSTEC-Geräte sind im Verzeichnis *EDS-DCF* innerhalb des *OpenPCS*-Pfades (z.B. *C:\OpenPCS\EDS-DCF*\) abgelegt. Die EDS-Dateien für CANopen-IO-Module sind ebenfalls im Lieferumfang für dieser Geräte enthalten. Für die SPS stehen verschiedene EDS-Dateien zur Auswahl:

Tabelle 2: Übersicht der EDS-Dateien für SYSTEC-Steuerungen

Dateiname	Verwendung
PLC02PDO.EDS	Unspezifische/allgemeingültige EDS-Datei für geringe Anzahl von Knoten und Netzwerkvariablen ermöglicht max. 2 Sende-PDOs und 2 Empfangs-PDOs
PLC64PDO.EDS	Unspezifische/allgemeingültige EDS-Datei für große Anzahl von Knoten und Netzwerkvariablen ermöglicht max. 64 Sende-PDOs und 64 Empfangs-PDOs
{DeviceName}_CAN0.EDS	Gerätespezifische EDS-Datei für CAN-Instanz 0 (in der Regel primäre CAN-Instanz mit dynamischem Object-Dictionary) Bsp.: PmC14_CAN0.EDS, PLCcoreCF54_CAN0.EDS
{DeviceName}_CAN1.EDS	Gerätespezifische EDS-Datei für CAN-Instanz 1 (in der Regel sekundäre CAN-Instanz mit statischem Object-Dictionary) Bsp.: PmC14_CAN1.EDS, PLCcoreCF54_CAN1.EDS

Die EDS-Dateien unterscheiden sich unter anderem in der Anzahl der möglichen PDOs zum Datenaustausch zwischen SPS und CANopen-IO-Modul. Für Steuerungen mit einem dynamischen Object-Dictionary gilt jedoch, dass stets nur so viele PDOs angelegt werden, wie die konkrete Konfiguration erfordert.

3.2.2 CANopen-Konfigurator

Mit Hilfe eines **CANopen-Konfigurators** werden die Netzwerkparameter zum Austausch von Prozessdaten, wie z.B. den zu verwendenden Sendemodus (synchron, asynchron), die benutzten Identifier oder das Mapping entsprechend den Anforderungen des Anwenders festgelegt. Weiterhin ermöglicht der Konfigurator die für Netzwerkvariablen erforderlichen **Verknüpfungen zwischen SPS und CANopen-IO-Modulen** zu erstellen. Für die Prozessdaten (im allgemeinen die Ein- und Ausgänge) der IO-Baugruppen werden dabei symbolische Namen vergeben, um diese als Netzwerkvariablen im SPS-Programm später referenzieren zu können.

Die Schnittstelle zwischen dem CANopen-Konfigurator und der *OpenPCS*-Programmierumgebung bildet die DCF-Datei der SPS. Diese ist als Konfigurationsfile der jeweiligen Hardware zuzuordnen. Damit erhält die Steuerung alle für den Austausch von Prozessdaten mit CANopen-IO-Modulen notwendigen Netzwerk-Informationen.

3.2.3 Vordefinierte DCF-Dateien

Um SPS-Programme für einfache Netzwerktopologien (wenige dezentrale IO-Baugruppen) auch ohne CANopen-Konfigurator schreiben zu können, liefert die Firma SYSTEC zusammen mit der Programmierumgebung *OpenPCS* vordefinierte DCF-Dateien aus, die eine Einbindung von bis zu 5 dezentralen SYSTEC-IO-Baugruppen ermöglichen. Eine pauschale Unterstützung von Fremdgeräten kann auf diese Weise nicht garantiert werden, da die Baugruppen zum Teil individuell spezifische Merkmale besitzen. Der SPS-Programmierer ist bei der Verwendung von vorkonfigurierten DCF-Dateien auf die Standardkonfiguration (Identifier, Mapping) der IO-Baugruppen sowie die in der DCF-Datei definierten Variablennamen für Ein- und Ausgänge festgelegt, es entfallen aber der Aufwand für Anschaffung und Einarbeitung in ein entsprechendes Konfigurations-Tool.

Die Verwendung von vordefinierten DCF-Dateien ermöglicht es, **ohne CANopen-Konfigurator** eine SPS um die Ein- und Ausgänge von bis zu 5 CANopen-IO-Modulen zu erweitern.

Grundvoraussetzung für die Verwendung von vordefinierten DCF-Dateien ist, dass die eingesetzten CANopen-IO-Baugruppen die vom CiA (CAN in Automation e.V.) in den CiA Draft Standards 301 und 401 festgelegte Standard-Konfiguration unterstützen.

Geeignet für den Einsatz mit vordefinierten DCF-Dateien sind beispielsweise folgende CANopen-Geräte:
- SYSTEC CANopen IO-C12 (phyPS-409-Y)
- SYSTEC CANopen IO-X1
- SYSTEC CANopen IO-X2
- SYSTEC CANopen IO-X3
- SYSTEC CANopen IO-X4
- SYSTEC CANopen-Chip164 (MM-215-Y)
- SYSTEC CANopen-ChipF40 (MM-217-Y)
- CANopen-IO-Baugruppen von Fremdherstellern, die eine Standard-Konfiguration gemäß den CiA Draft Standards 301 und 401 verwenden

Ablage der vordefinierten DCF-Dateien:

Die vordefinierten DCF-Dateien befinden sich unter der Bezeichnung ***DxSALVE.DCF*** im Verzeichnis *EDS-DCF* innerhalb des *OpenPCS*-Pfades (z.B. *C:\OpenPCS\EDS-DCF*\). Die Zahl, die das symbolische Zeichen "x" im Dateinamen ersetzt, bezeichnet die Anzahl der unterstützten CANopen-IO-Module:

Tabelle 3: Übersicht der vordefinierten DCF-Dateien

Dateiname	Unterstützte Save-Adressen
D1SLAVE.DCF	40H
D2SLAVE.DCF	40H, 41H
D3SLAVE.DCF	40H, 41H, 42H
D4SLAVE.DCF	40H, 41H, 42H, 43H
D5SLAVE.DCF	40H, 41H, 42H, 43H, 44H

Es wird empfohlen, stets die DCF-Datei als Konfigurationsfile für die Ressource einzubinden, mit der gerade alle benötigen IO-Baugruppen abgedeckt sind. Bei Verwendung einer Konfigurationsdatei, die mehr als die benötigte Anzahl von IO-Baugruppen unterstützt, hat dies zwar keinen negativen Einfluss auf die Funktionalität des SPS-Programms. Es wird aber mehr Speicher als notwendig auf der SPS belegt sowie gleichzeitig der Verwaltungsaufwand innerhalb der Netzwerkschicht erhöht, was zu größeren Programmlaufzeiten führt.

Variablen der vordefinierten DCF-Dateien:

Bei der Deklaration der Netzwerkvariablen im SPS-Programm sind die in der DCF-Datei festgelegten symbolischen Namen für die Prozessdatenobjekte zu verwenden. In den vordefinierten DCF-Dateien sind für die CANopen-Geräte mit den Knotenadressen 40H...44H die in Tabelle 4 angegebenen Netzwerkvariablen definiert.

Tabelle 4: Netzwerkvariablen der vordefinierten DCF-Dateien

Variablenname	Variablentypen	Zugriffsart
IN0_IN7_xxH	BYTE, USINT, SINT	lesen
IN8_IN15_xxH	BYTE, USINT, SINT	lesen
IN16_IN23_xxH	BYTE, USINT, SINT	lesen
OUT0_OUT7_xxH	BYTE, USINT, SINT	schreiben
OUT8_OUT15_xxH	BYTE, USINT, SINT	schreiben
OUT16_OUT23_xxH	BYTE, USINT, SINT	schreiben
AIN0_xxH	WORD, UINT, INT	lesen
AIN1_xxH	WORD, UINT, INT	lesen
AIN2_xxH	WORD, UINT, INT	lesen
AIN3_xxH	WORD, UINT, INT	lesen
AOUT0_xxH	WORD, UINT, INT	schreiben
AOUT1_xxH	WORD, UINT, INT	schreiben

Die symbolische Zeichenfolge "xx" im Variablennamen wird durch die jeweilige Knotennummer des unterstützten CANopen-Gerätes ersetzt. So enthält die Datei *D1SLAVE.DCF* beispielsweise die Definition für die Variable *IN0_IN7_40H*, die Datei *D3SLAVE.DCF* definiert insgesamt die Variablen *IN0_IN7_40H*, *IN0_IN7_41H*, und *IN0_IN7_42H*.

Die für jedes CANopen-IO-Modul konkret nutzbare Anzahl von Ein- und Ausgängen ist der jeweiligen Gerätedokumentation zu entnehmen.

Besonderheiten bei der Verwendung vordefinierten DCF-Dateien:

- Während digitale Ein- und Ausgänge nach der Einbindung von vordefinierten DCF-Dateien ohne zusätzlichen Konfigurationsaufwand sofort funktionsfähig sind, müssen analoge Eingänge in der Regel erst freigeschalten werden. Informationen hierzu sind dem Manual des jeweiligen CANopen-Gerätes zu entnehmen. Möglich ist das Freischalten beispielsweise beim Programmstart mit Hilfe der im Abschnitt 4.4 beschriebenen SDO-Funktionsbausteine.

- In den vordefinierten DCF-Dateien sind für die PDOs jeweils Maximalkonfigurationen angelegt (z.B- AIN0 … AIN3 für RPDO1). Die einzelnen Geräte unterstützen in der Regel jedoch weniger Objekte in einem PDO (z.B. nur AIN0 und AIN1). Dadurch kann es zum Auftreten von Emergency-Nachrichten mit dem Errorcode 16#8210 ("PDO not processed due to length error") oder Errorcode 16#8220 ("PDO length exceeded") kommen.

3.3 Einbindung von DCF-Dateien in das SPS-Projekt

DCF-Dateien können auf verschiedene Weise in das SPS-Projekt eingebunden werden. Da in der Regel zum Erstellen der DCF-Dateien ein CANopen-Konfigurator benutzt wird, der sämtliche DCF-Dateien in einem gemeinsamen Projekt zusammen fasst, sollte nach Möglichkeit dieses Netzwerk auch als konsistente Einheit in die *OpenPCS*-Programmierumgebung importiert werden. Das Vorgehen hierzu beschreibt Abschnitt 3.3.1. Alternativ besteht aber auch die Option, Netzwerkknoten in *OpenPCS* anzulegen und diesen dann bereits existierende DCF-Dateien manuell zuzuweisen. Das ist beispielsweise dann notwendig, wenn entweder mit vordefinierten DCF-Dateien gearbeitet wird (siehe Abschnitt 3.2.3) oder wenn das Format der vom CANopen-Konfigurator erzeugten Projektdatei von *OpenPCS* nicht erkannt wird. Die dafür notwendigen Schritte erläutert Abschnitt 3.5.

Beim Einbinden von DCF-Dateien ist zu beachten, dass hierbei jeweils der momentane Inhalt der DCF-Datei als Kopie in die OpenPCS-Programmierumgebung importiert wird. Um Änderungen, die nachträglich an der DCF-Datei vorgenommen wurden, ebenfalls in das SPS-Projekt zu übernehmen, ist das bereits importierte Netzwerk aus der OpenPCS-Programmierumgebung zu löschen und der Importvorgang erneut zu wiederholen.

3.3.1 Einbinden von kompletten Netzwerkprojekten

Beim Einbinden von kompletten Netzwerkprojekten werden sämtliche zu einem CANopen-Netzwerk gehörigen DCF-Dateien als konsistente Einheit in die *OpenPCS*-Programmierumgebung übernommen. Dies ist der einfachste und zugleich sicherste Weg für den Import von DCF-Dateien in das IEC61131-Programmiersystem. Um ein Netzwerkprojekt in *OpenPCS* einzubinden, ist im Projektbrowser in die Ansicht "Network" zu wechseln (siehe Bild 3).

Zum Einbinden eines kompletten Netzwerkprojektes ist wie folgt vorzugehen:

1. Anklicken des "PowerMap" Icons mit der rechten Maustaste (siehe Bild 3)

2. Im Kontextmenü "Load File -> Import ProCANopen network" auswählen (siehe Bild 3)

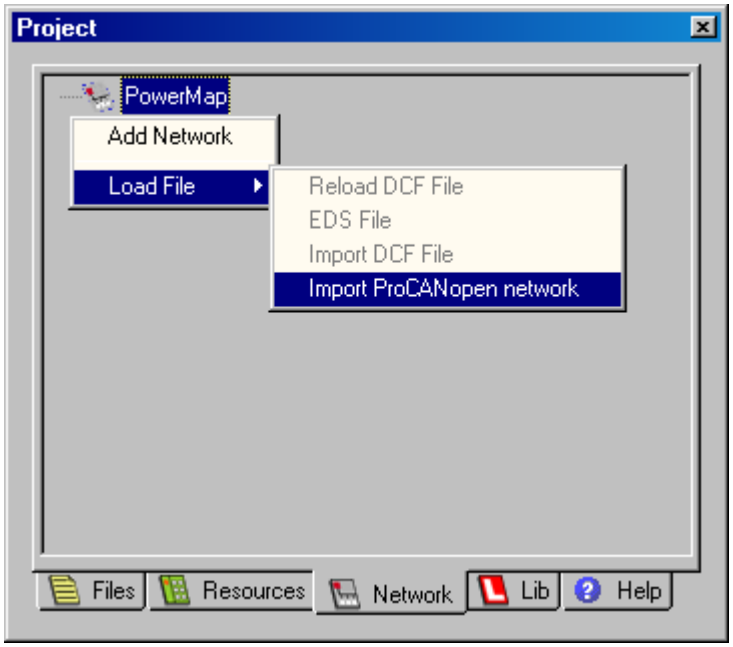


Bild 3: Importieren von Netzwerken in OpenPCS

3. Im sich öffnenden File-Dialog das Verzeichnis mit dem zu importierenden Netzwerkprojekt auswählen und mit der Schaltfläche "*OK*" bestätigen

   Nachdem der Import des Netzwerkes abgeschlossen ist, werden die entsprechenden CANopen-Knoten in der Ansicht "Network" des Projektbrowsers angezeigt (siehe Bild 4).

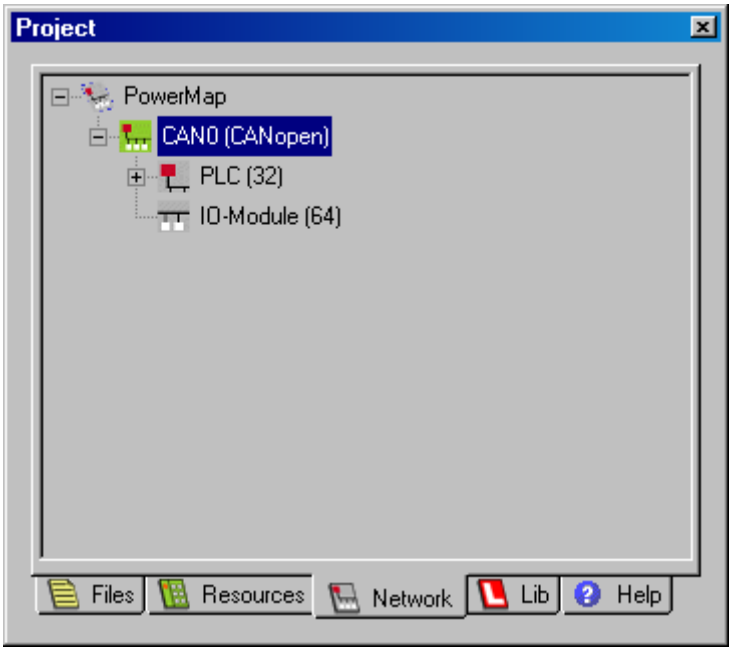


Bild 4: Darstellung des importierten Netzwerkes im OpenPCS-Projektbrowser

4. Bei Bedarf kann der Name des importierten Netzwerkprojektes beliebig geändert werden. Hierfür ist im Kontextmenü des Netzwerkknotens der Menüpunkt "Rename Network" zuständig.

5. Um das importierte Netzwerkprojekt mit der aktiven SPS-Ressource zu verknüpfen, ist im Kontextmenü des Netzwerkes der Menüpunkt "Link to Active Resource" zu wählen (siehe Bild 5).

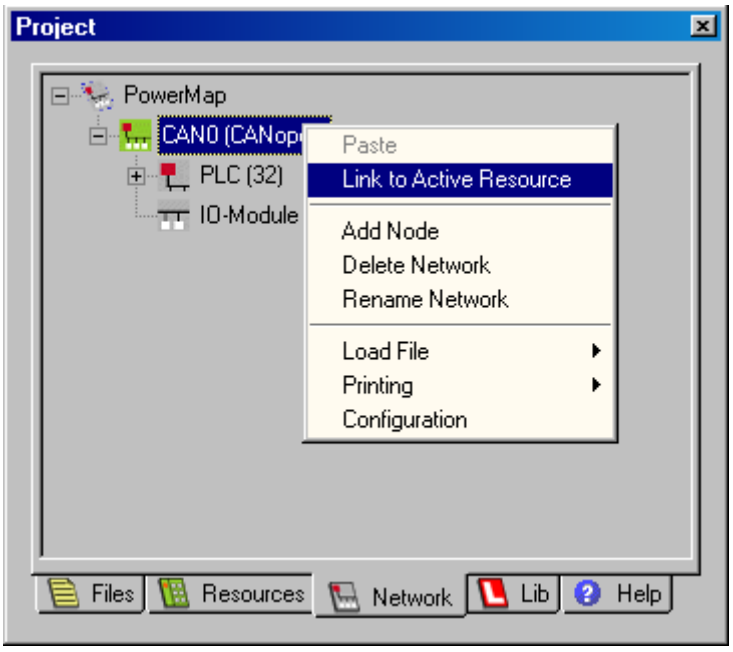


Bild 5: Verknüpfen des importierten Netzwerkprojektes mit der aktiven SPS-Ressource

Die weiteren Schritte die notwendig sind, um die im Netzwerkprojekt definierten Netzwerkvariablen im SPS-Programm benutzen zu können, beschreibt Abschnitt 3.4.

3.3.2 Manuelles einbinden einzelner DCF-Dateien

Die manuelle Einbindung einzelner DCF-Dateien stellt eine Alternative zu der im Abschnitt 3.3.1 beschriebenen Einbindung von kompletten Netzwerkprojekten dar. Das manuelle Einbinden ermöglicht beispielsweise die Verwendung vordefinierten DCF-Dateien (siehe Abschnitt 3.2.3) sowie den Import von DCF-Dateien aus Netzwerkprojekten, bei denen das Format der vom CANopen-Konfigurator erzeugten Projektdatei von *OpenPCS* nicht erkannt wird. Um DCF-Dateien in *OpenPCS* einzubinden, ist im Projektbrowser in die Ansicht "Network" zu wechseln (siehe Bild 6).

Zum manuellen Einbinden von DCF-Dateien ist wie folgt vorzugehen:

1. Anklicken des "PowerMap" Icons mit der rechten Maustaste (siehe Bild 6)

2. Im Kontextmenü "Add Network" auswählen (siehe Bild 6)

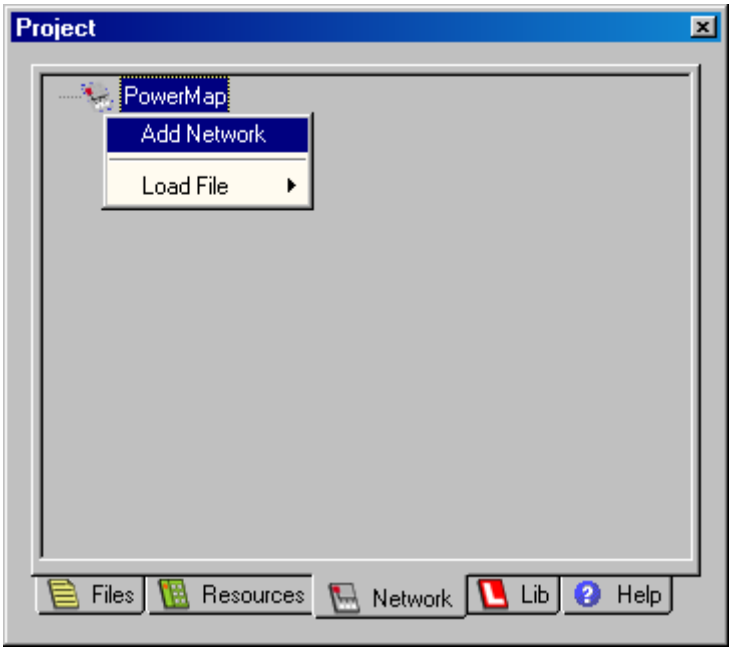


Bild 6: Anlegen eines Netzwerkes in OpenPCS

3. Im Dialog "Create a new file" (siehe Bild 7) sind als "File Type" die Kategorie "PowerMap Network" und als "Template" der Eintrag "CANopen Network" auszuwählen. Im Feld "Name" ist eine beliebige Bezeichnung für das neu anzulegende Netzwerk anzugeben. Schließlich ist der Dialog mit der Schaltfläche "*OK*" zu beenden.

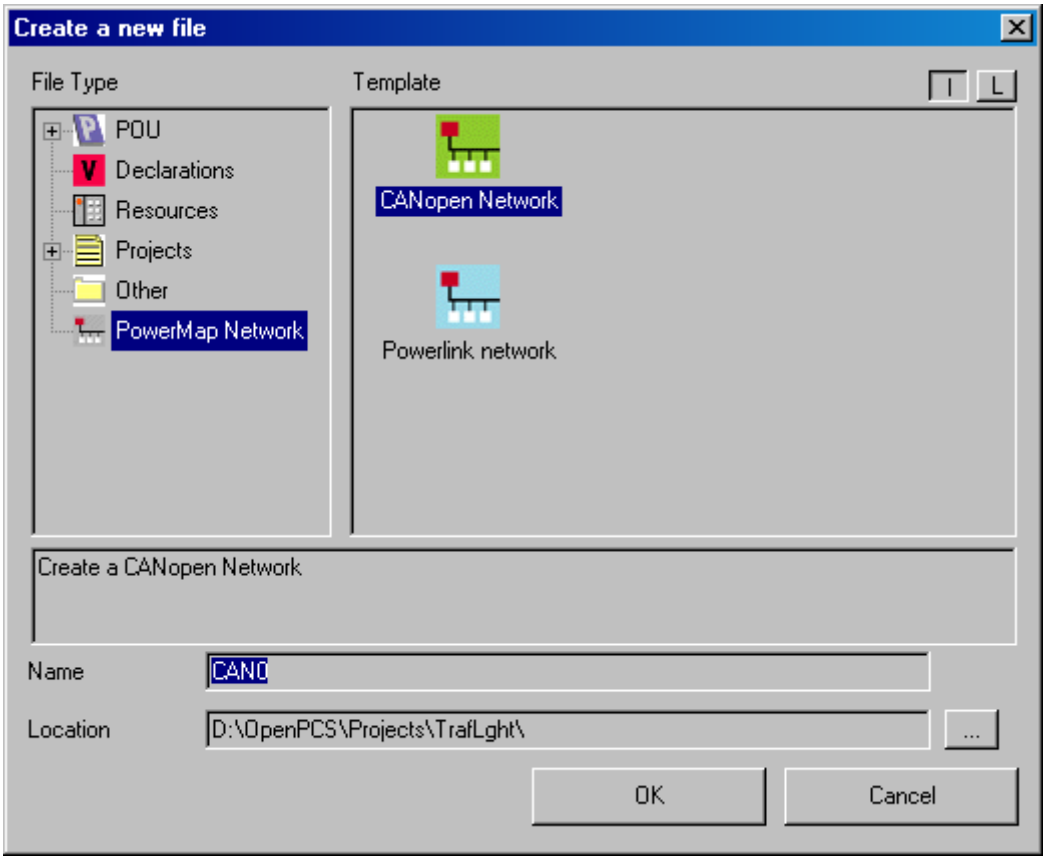


Bild 7: Erzeugen eines neuen Netzwerkeintrages

4. Anklicken des Icons für das neu angelegte Netzwerk und Auswahl des Menüpunktes "Load File -> Import DCF File" im Kontextmenü des Netzwerkes (siehe Bild 8)

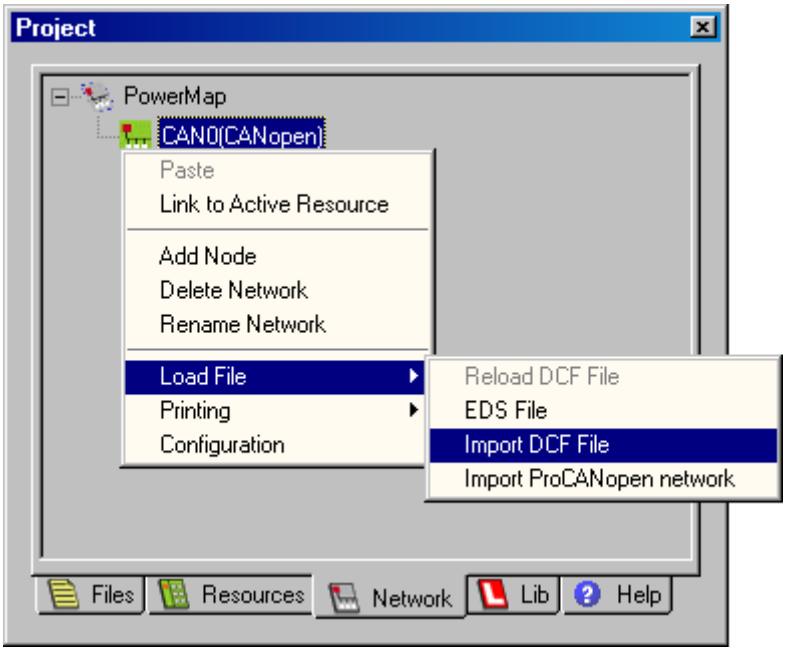


Bild 8: Hinzufügen eines neuen Netzwerkknotens

5. Im sich öffnenden File-Dialog die zu importierende DCF-Datei auswählen und mit der Schaltfläche "*OK*" bestätigen. Dadurch wird ein neuer Netzwerkknoten angelegt und diesem die entsprechende

DCF-Datei zugewiesen. Dieser Vorgang ist für jede zu importierende DCF-Datei zu wiederholen.

Nachdem der manuelle Import aller DCF-Dateien abgeschlossen ist, werden die entsprechenden CANopen-Knoten in der Ansicht "Network" des Projektbrowsers angezeigt (siehe Bild 9).

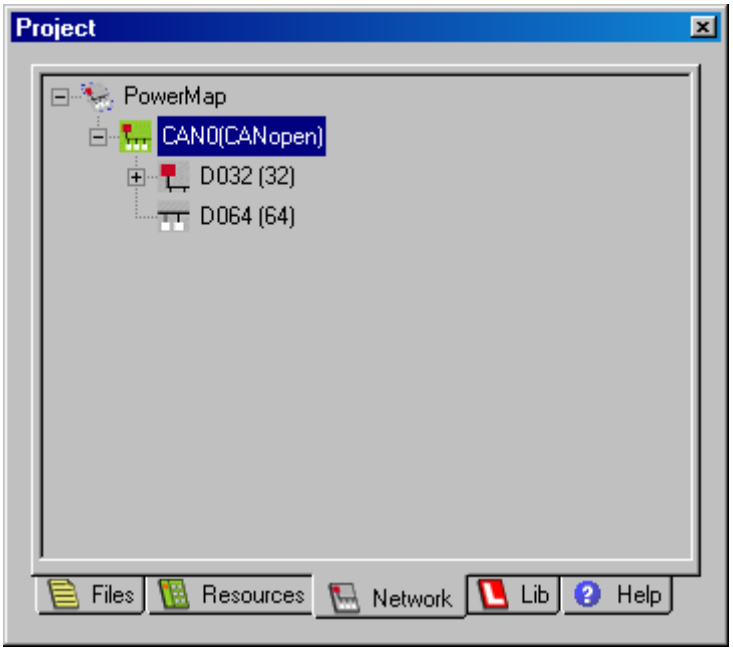


Bild 9: Darstellung der importierten Netzwerkknoten im OpenPCS-Projektbrowser

6. Bei Bedarf können die Name das Netzwerk und die importierten Knoten beliebig geändert werden. Hierfür sind im Kontextmenü die Menüpunkt "Rename Network" bzw. "Rename Node" zuständig (siehe Bild 10)

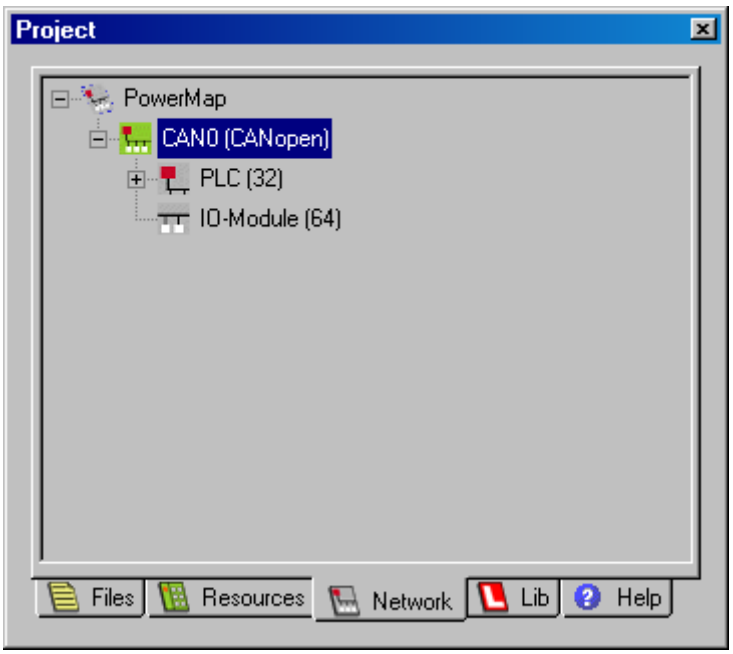


Bild 10: Darstellung des Netzwerkes nach Anpassung der Namen

7. Um das aus den manuell importierten DCF-Dateien aufgebaute Netzwerk mit der aktiven SPS-Ressource zu verknüpfen, ist im Kontextmenü des Netzwerkes der Menüpunkt "Link to Active Resource" zu wählen (siehe Bild 11).

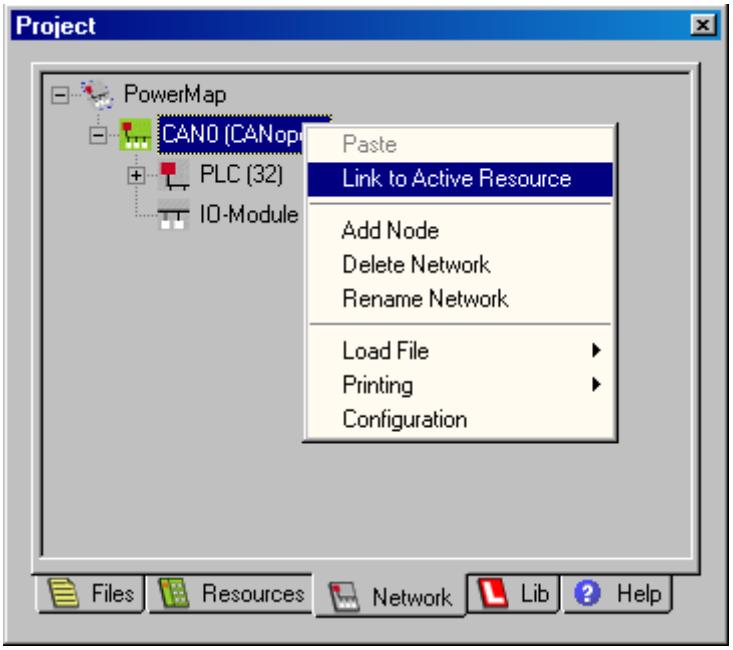


Bild 11: Verknüpfen des manuell erstellten Netzwerkprojektes mit der aktiven SPS-Ressource

Die weiteren Schritte die notwendig sind, um die im Netzwerkprojekt definierten Netzwerkvariablen im SPS-Programm benutzen zu können, beschreibt Abschnitt 3.4.

3.4 Verwendung von Netzwerkvariablen im SPS-Programm

Netzwerkvariablen werden in einem SPS-Programm innerhalb der Schlüsselworte **VAR_EXTERNAL ... END_VAR** deklariert. Dadurch sind sie als außerhalb des Programms und somit letztlich auch als "außerhalb der SPS" gekennzeichnet. Die Deklaration der Netzwerkvariablen selbst unterscheidet sich dabei nicht von der Deklaration der lokalen Variablen:

```
VAR_EXTERNAL
    NetVar1 : BYTE ;
    NetVar2 : UINT ;
END_VAR
```

Neben CANopen kann es aber noch beliebige andere Quellen für externe Variablen geben, so z.B. EPL (Ethernet Power Link) oder OPC. Daher benötigt die *OpenPCS*-Programmierumgebung eine Zuordnungstabelle, in der die Quelle einer externen Variable spezifiziert ist.

Für die Deklaration von Netzwerkvariablen im SPS-Programm sind daher insgesamt folgende Schritte notwendig:

1. Anlegen eines Netzwerkes und Einbinden der erforderlichen DCF-Dateien in das SPS-Projekt wie im Abschnitt 3.3 beschrieben

2. Aufruf des Menüpunktes "File -> New…", um mit Hilfe des Dialoges "Create a new file" eine neue Zuordnungstabelle anzulegen. Im Dialog "Create a new file" (siehe Bild 12) sind als "File Type" die Kategorie "Declarations" und als "Template" der Eintrag "I/O Mapping" auszuwählen. Im Feld "Name" ist eine beliebige Bezeichnung für die neu anzulegende Zuordnungstabelle anzugeben. Schließlich ist der Dialog mit der Schaltfläche "*OK*" zu beenden.

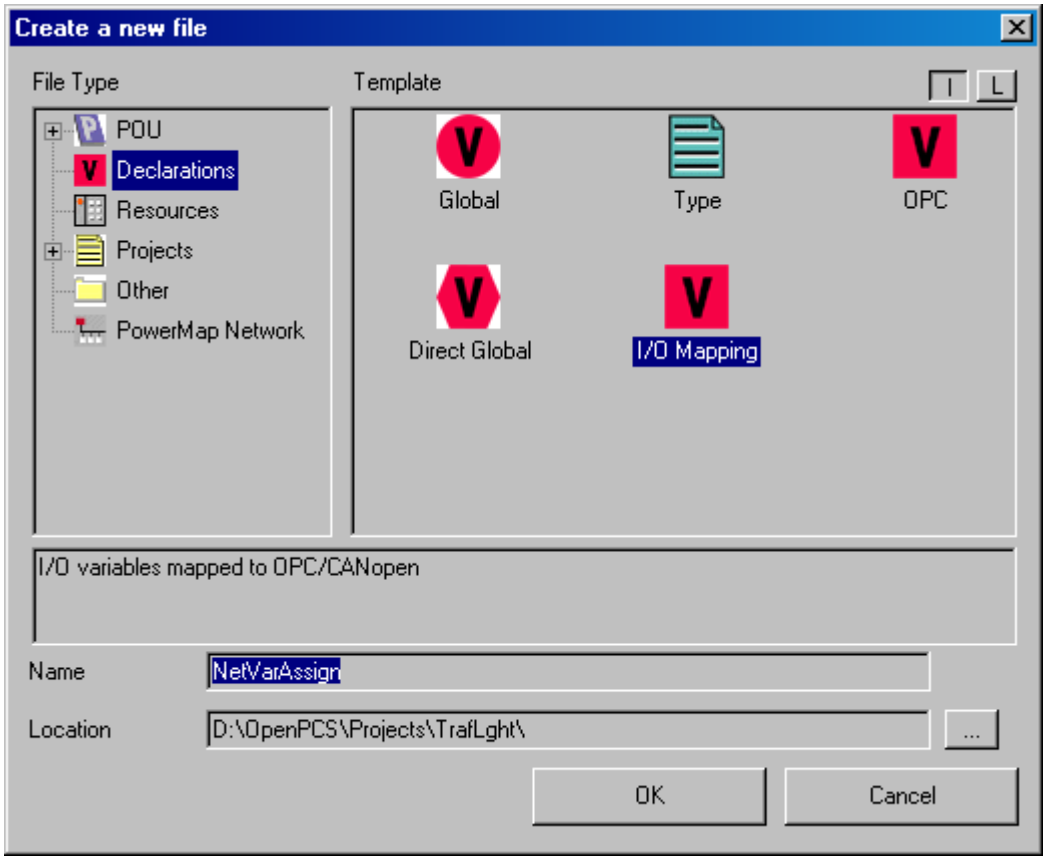


Bild 12: Anlegen einer neuen Zuordnungstabelle

3. Nach dem Anlegen der Zuordnungstabelle sind die Einträge für die Netzwerkvariablen aus dem Netzwerkknoten der SPS (hier im Beispiel Knoten "PLC (32)") durch Doppelklick in die Zuordnungstabelle einzufügen (siehe Bild 13). Nach dem Einfügen können Name (Spalte "Name") und Typ (Spalte "IEC Type") der Netzwerkvariablen in der Zuordnungstabelle für die Verwendung im SPS-Programm angepasst werden (siehe dazu auch Bemerkungen weiter unten im Text).

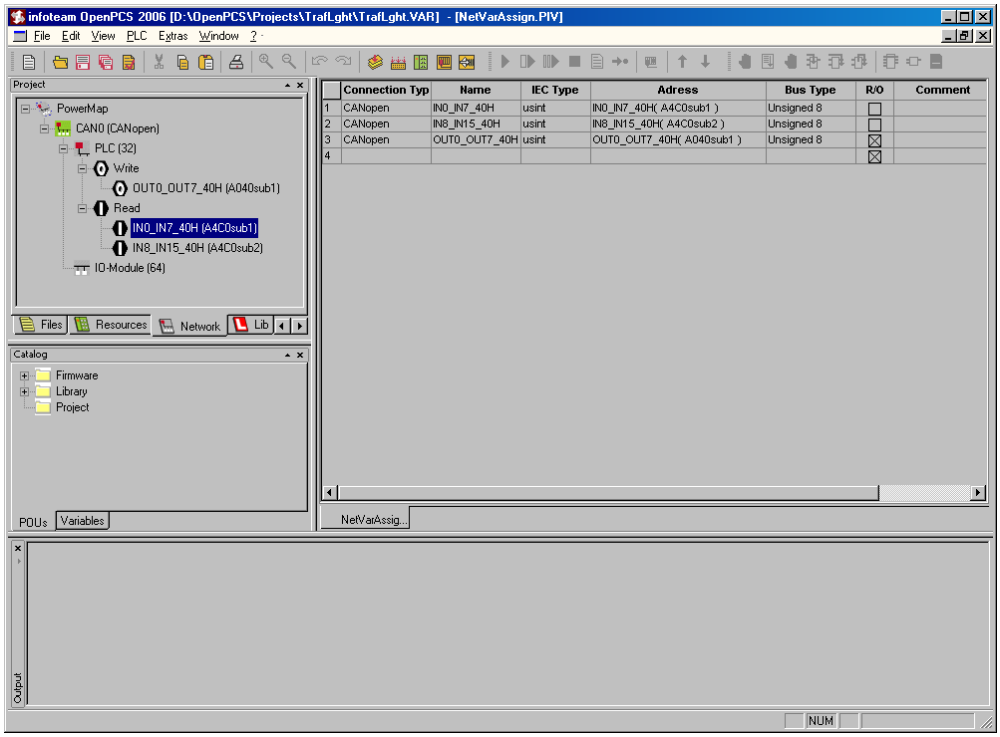


Bild 13: Einfügen der Netzwerkvariablen in die Zuordnungstabelle

4. Um die neu angelegte Zuordnungstabelle mit der aktiven SPS-Ressource zu verknüpfen, ist im Projektbrowser in die Ansicht "Files" zu wechseln und anschließend im Kontextmenü der Zuordnungstabelle der Menüpunkt "Link to Active Resource" zu wählen (siehe Bild 14)

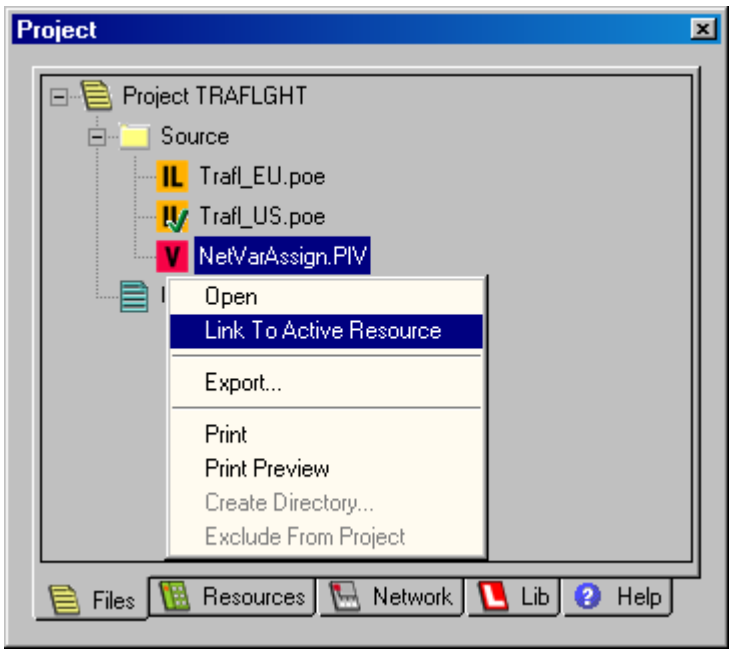


Bild 14: Verknüpfen der Zuordnungstabelle mit der aktiven SPS-Ressource

5. In der Ansicht "Files" im Projektbrowser kann kontrolliert werden, dass die für ein SPS-Programm mit Netzwerkvariablen notwendigen Komponenten mit der aktiven SPS-Ressource verknüpft sind (siehe Bild 15):

   - SPS-Programm (hier im Beispiel "TrafL_US")
   - Netzwerk (hier im Beispiel "CAN0", zum Anlegen siehe Abschnitt 3.3)
   - Zuordnungstabelle (hier im Beispiel "NetVarAssign")

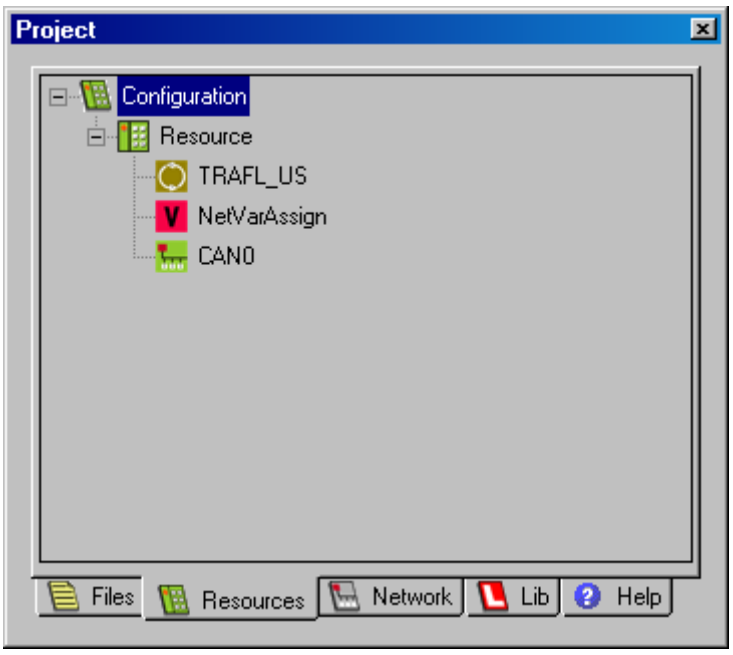


Bild 15: Darstellung der Ressourcen-Komponenten im Projektbrowser

6. Deklaration der erforderlichen Netzwerkvariablen im SPS-Programm innerhalb der Sektion **VAR_EXTERNAL ... END_VAR**, dabei müssen jeweils Name (Spalte "Name") und Typ (Spalte "IEC Type") mit der Definition der Netzwerkvariablen in der Zuordnungstabelle exakt übereinstimmen. Beispiel:

```
VAR_EXTERNAL
    NetVar1 : BYTE ;
    NetVar2 : UINT ;
END_VAR
```

Bei der Deklaration von Netzwerkvariablen sind folgende Punkte zu beachten:

- Die **Namen für die Netzwerkvariablen** müssen zwischen SPS-Programm und Zuordnungstabelle exakt übereinstimmen. Der Variablenname ist der gemeinsame Bezugspunkt zwischen IEC61131 und CANopen.

- Als **Typ für die Netzwerkvariablen** ist jeweils ein zwischen IEC61131 und CANopen kompatibler Datentyp zu wählen (siehe Tabelle 5).

- Als **Anfangswerte für die Netzwerkvariablen** werden die in der DCF-Datei der SPS festgelegten Initialwerte verwendet (Eintrag "ParameterValue=" bzw. "DefaultValue="). Die erneute Angabe eines expliziten Anfangswertes bei der Deklaration der Netzwerkvariablen im SPS-Programm ist daher nicht zulässig, um so Widersprüchen mit den Festlegungen in der DCF-Datei zu vermeiden. Jedoch kann das SPS-Programm innerhalb des ersten SPS-Zyklus Ausgangs-Netzwerkvariablen noch vor ihrem erstmaligen Senden spezifische Werten zuweisen, die von den in DCF-Datei festgelegten Initialwerten abweichen dürfen (zur Anfangsinitialisierung von Netzwerkvariablen siehe auch Abschnitt 2.3).

Zur Deklaration von Netzwerkvariablen im SPS-Programm ist gemäß der IEC61131 ein Datentyp entsprechend dem Verwendungszweck (Logik, Arithmetik) auszuwählen. Hier besteht keine eindeutige Zuordnung zwischen IEC61131 und CANopen. Tabelle 5 enthält die Zuordnung der von IEC61131 und CANopen verwendeten Datentypen, die als Netzwerkvariablen eingesetzt werden können.

Es ist zu beachten, dass keine eindeutige Zuordnung der Datentypen zwischen IEC61131 und CANopen besteht. Daher ist der IEC61131-Typ für das SPS-Programm entsprechend dem Verwendungszweck der Variable auszuwählen.

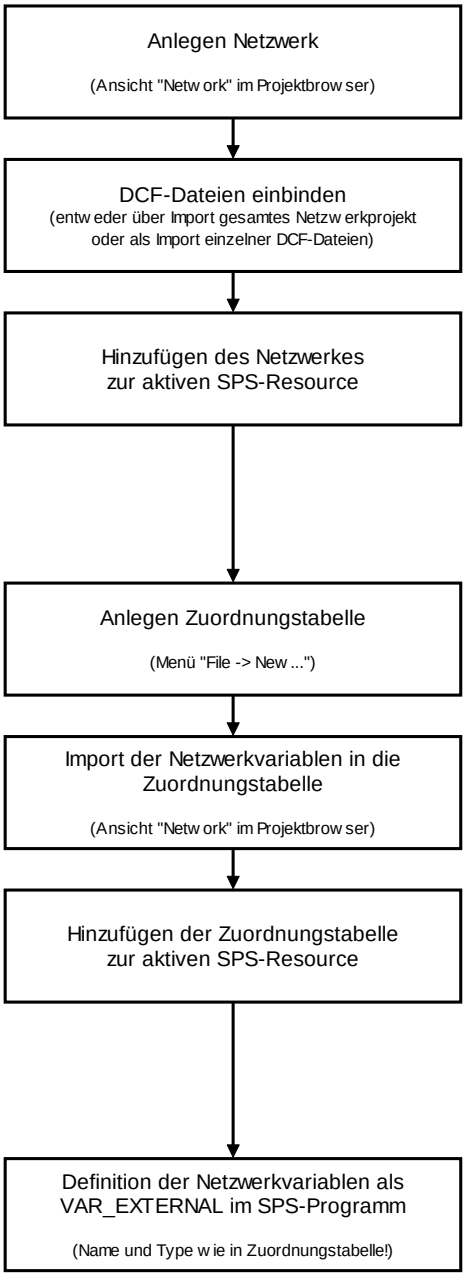
Tabelle 5: Zuordnung der Datentypen zwischen IEC61131 und CANopen

IEC61131	CANopen	Verwendung	Datengröße (Bit)
BOOL	Boolean (*)	Logik	1
BYTE	Unsigned8	Logik	8
USINT	Unsigned8	Arithmetik (ohne VZ)	8
SINT	Integer8	Arithmetik (mit VZ)	8
WORD	Unsigned16	Logik	16
UINT	Unsigned16	Arithmetik (ohne VZ)	16
INT	Integer16	Arithmetik (mit VZ)	16
DWORD	Unsigned32	Logik	32
UDINT	Unsigned32	Arithmetik (ohne VZ)	32
DINT	Integer32	Arithmetik (ohne VZ)	32
REAL	Float	Arithmetik	32

Die in Tabelle 5 mit (*) gekennzeichneten Datentypen sind nicht für alle Steuerungstypen verfügbar.

3.5 Zusammenfassung notwendiger Schritte

Zur Einbindung von Netzwerkvariablen in ein SPS-Programm sind die im folgenden beschriebenen Schritte notwendig. Eine praktische Umsetzung zeigt das Beispielprojekt im Abschnitt 3.6. .



Das schrittweise Vorgehen zum Anlegen eines Netzwerkes und zum Einbinden der DCF-Dateien beschreibt Abschnitt 3.3

Das schrittweise Vorgehen zum Anlegen von Netzwerkvariablen beschreibt Abschnitt 3.4

3.6 Beispielprojekt für Netzwerkvariablen

Das in der *Softwarepaket " SYSTEC-OpenPCS-Extension"* enthaltenen Beispielprojekt *"TrafLghtl*" (Traffic Light, Ampel) zeigt die Einbindung von Netzwerkvariablen anhand der Verwendung einer vordefinierte DCF-Dateien. Benutzt wird dafür das File *"D1SALVE.DCF"*, das entsprechend für die Projekt-Konfiguration der Ressource eingetragen ist.

Das Beispielprojekt *"TrafLght"* realisiert die Funktion einer Fußgänger-Bedarfsampel. Die SPS steuert dabei die Lampen für die Fahrzeuge (rot, gelb, grün) und ein CANopen-IO-Modul (Knotenadresse 40H) schaltet die Signale für die Fußgänger (rot und grün). Die SPS und das dezentrale IO-Modul stellen jeweils einen Taster zum Starten der Ampel bereit. Bei einem 0/1-Übergang an einem dieser Eingänge wird der Ampelzyklus gestartet.

Für den Zugriff auf die Ein- und Ausgänge des CANopen-IO-Moduls sind im SPS-Programm (*TRAFLGHT.POE*) die in *D1SLAVE.DCF* definierten Netzwerkvariablen wie folgt angelegt:

```
VAR_EXTERNAL
    IN0_IN7_40H : BYTE ;    (* declared in D1SLAVE.DCF *)
    OUT0_OUT7_40H : BYTE ;  (* declared in D1SLAVE.DCF *)
END_VAR
```

Die logische Verknüpfung der Taster am lokalen Eingang der SPS und an dem über CANopen zugänglichen Eingang des IO-Moduls erfolgt durch die Befehlssequenz

```
WaitStartButton:
(* Link buttons of PLC and I/O module *)
LD  StartButton                    (* button on PLC        *)
OR  IN0_IN7_40H.0                  (* button on I/O module *)
ST  FB_StartCondition.CLK
```

Für das Setzen der beiden Ausgänge am CANopen-IO-Modul ist die nachstehende Befehlsfolge zuständig:

```
ProgExit:
(* copy pedestrian data into network variables *)
(* for I/O module                              *)
LD  PedGreen
ST  OUT0_OUT7_40H.0
LD  PedRed
ST  OUT0_OUT7_40H.1
```

Hierdurch werden die beiden lokalen Bitvariablen in die Ausgabevariable des CANopen-IO-Moduls vom Typ *BYTE* kopiert.

4 IEC61131-Funktionsbausteine für CANopen

4.1 Allgemeine Grundlagen für CANopen-Funktionsbausteine

Innerhalb der IEC61131-3 stehen verschiedene herstellerspezifische Funktionsbausteine für den Zugriff auf CANopen zur Verfügung. Diese beinhalten als Untermenge die vom CiA (CAN in Automation e.V.) im CiA Draft Standard 405 definierte Funktionalität. Darüber hinaus existieren zusätzliche Funktionsbausteine zum Senden und Empfangen von PDOs bzw. CAN Layer 2 Nachrichten sowie zum Erzeugen von SYNC-Objekten.

4.1.1 Übersicht der CANopen-Funktionsbausteine

Tabelle 6 zeigt eine Übersicht der CANopen-Funktionsbausteine für die IEC61131-3. Alle Bausteine sind als herstellerspezifische Funktionsbausteine realisiert und somit Bestandteil der Firmware einer SPS. Abhängig von der Betriebsart einer Steuerung sind nicht in jedem Fall alle Bausteine verfügbar. Detaillierte Informationen hierzu finden sich im Abschnitt 4.1.2.

Tabelle 6: Übersicht der CANopen-Funktionsbausteine für IEC61131-3

Funktionsblock	DS 405	Bedeutung	Abschn.
CAN_GET_LOCAL_NODE_ID	ja	Abfrage der eigenen Knotenadresse	4.2.1
CAN_GET_CANOPEN_KERNEL_STATE	ja	Abfrage Status des CANopen-Kernels der eigenen SPS	4.2.2
CAN_REGISTER_COBID	nein	COBID für Empfang von PDOs registrieren	4.3.1
CAN_PDO_READ8	nein	empfangenes PDO auslesen	4.3.2
CAN_PDO_WRITE8	nein	PDO senden	4.3.3
CAN_SDO_READ8	ja	Lesen von Objekteinträgen eines Knotens mittels SDO-Transfer	4.4.1
CAN_SDO_WRITE8	ja	Schreiben von Objekteinträgen eines Knotens mittels SDO-Transfer	4.4.2
CAN_SDO_READ_STR	ja	Lesen von Zeichenfolgen aus dem Objektverzeichnis eines Knotens mittels SDO-Transfer	4.4.3
CAN_SDO_WRITE_STR	ja	Schreiben von Zeichenfolgen in das Objektverzeichnis eines Knotens mittels SDO-Transfer	4.4.4
CAN_SDO_READ_BIN	ja	Lesen von Binärdaten aus dem Objektverzeichnis eines Knotens mittels SDO-Transfer	4.4.5
CAN_SDO_WRITE_BIN	ja	Schreiben von Binärdaten in das Objektverzeichnis eines Knotens mittels SDO-Transfer	4.4.6
CAN_GET_STATE	ja	Abfrage Status eines Knoten	4.5.1
CAN_NMT	ja	Senden von NMT-Nachrichten	4.5.2
CAN_RECV_EMCY_DEV	ja	empfangene Emergency-Nachricht auslesen (spezifischer Knoten)	4.5.3
CAN_RECV_EMCY	ja	empfangene Emergency-Nachricht auslesen (beliebiger Knoten)	4.5.4
CAN_WRITE_EMCY	nein	Emergency-Nachricht senden	4.5.5
CAN_RECV_BOOTUP_DEV	nein	empfangene Bootup-Nachricht auslesen (spezifischer Knoten)	4.5.6
CAN_RECV_BOOTUP	nein	empfangene Bootup-Nachricht	4.5.7

		auslesen (beliebiger Knoten)	
CAN_ENABLE_CYCLIC_SYNC	nein	Freigeben oder Sperren von zyklischen SYNC-Nachrichten	4.5.8
CAN_SEND_SYNC	nein	Senden einer einzelnen SYNC-Nachricht	4.5.9

Die Spalte "CiA 405" in Tabelle 6 gibt an, ob die Funktionalität des jeweiligen Bausteins durch den CiA Draft Standard 405 definiert ist ("ja"), oder ob dieser Baustein eine herstellerspezifische Ergänzung dieses Standards darstellt ("nein"). Es ist zu beachten, dass aufgrund spezifischer Eigenheiten des IEC61131-Programmiersystems auch die Implementierung der durch den CiA definierten Bausteine teilweise vom Draft Standard 405 abweicht. Allgemeine Informationen hierzu enthält der Abschnitt 4.1.4, detaillierte Angaben sind im Abschnitt zum jeweiligen Funktionsbaustein aufgeführt.

4.1.2 Verfügbarkeit der Funktionsbausteine auf Steuerungen mit und ohne CANopen-Master

Die CANopen-Funktionsbausteine für IEC61131 basieren auf verschiedenen Diensten von CANopen, von denen einige gleichzeitig auf mehreren Knoten aktiv sein können (z.B. PDO- und SDO-Transfer), während andere nur exklusiv von einem einzigen Knoten ausgeführt werden dürfen (z.B. NMT-Dienste). Auf einer SPS ohne CANopen-Master stehen nur die nicht exklusiven Dienste zur Verfügung, während auf einer SPS mit CANopen-Master zusätzlich auch die exklusiven Dienste nutzbar sind. Die Unterscheidung zwischen beiden SPS-Varianten erfolgt bei SYSTEC-Steuerungen im Allgemeinen anhand der Knotenadresse (siehe Abschnitt 2.1).

Tabelle 7 zeigt die Nutzbarkeit der einzelnen CANopen-Funktionsbausteine auf Steuerungen mit und ohne Master. Ein SPS-Programm kann die Knotenadresse - und damit indirekt auch den Umfang der nutzbaren Funktionsbausteine - mit Hilfe des Funktionsbausteines CAN_GET_LOCAL_NODE_ID bestimmen (siehe Abschnitt 4.2.1).

Tabelle 7: Nutzbarkeit von CANopen-FBs auf Steuerungen mit und ohne Master

Funktionsblock	SPS ohne CANopen-Master	SPS mit CANopen-Master
CAN_GET_LOCAL_NODE_ID	X	X
CAN_GET_CANOPEN_KERNEL_STATE	X	X
CAN_REGISTER_COBID	X	X
CAN_PDO_READ8	X	X
CAN_PDO_WRITE8	X	X
CAN_SDO_READ8	X	X
CAN_SDO_WRITE8	X	X
CAN_SDO_READ_STR	X	X
CAN_SDO_WRITE_STR	X	X
CAN_SDO_READ_BIN	X	X
CAN_SDO_WRITE_BIN	X	X
CAN_GET_STATE	(x)	X
CAN_NMT	-	X
CAN_RECV_EMCY_DEV	-	X
CAN_RECV_EMCY	-	X
CAN_WRITE_EMCY	X	X
CAN_RECV_BOOTUP_DEV	-	X
CAN_RECV_BOOTUP	-	X
CAN_ENABLE_CYCLIC_SYNC	-	X
CAN_SEND_SYNC	-	X

Zeichenerklärung:
X = Funktionalität verfügbar
(x) = Funktionalität eingeschränkt verfügbar, siehe Text
- = Funktionalität nicht verfügbar

Der Funktionsbaustein *CAN_GET_STATE* kann auf einer SPS ohne CANopen-Master nur indirekt realisiert werden. Hierzu ist die Unterstützung einer SPS mit CANopen-Master notwendig. Detaillierte Informationen hierzu enthalten die Abschnitte 2.2 (Beschreibung der Knotenüberwachung) und 4.5.1 (Beschreibung des Funktionsbausteines *CAN_GET_STATE)*.

4.1.3 Synchronisation zwischen CANopen-Funktionsbaustein und SPS-Programm

Die überwiegende Zahl der CANopen-Funktionsbausteine für IEC113-3 wird asynchron zum eigentlichen SPS-Programm ausgeführt. Die Prozesssynchronisation zwischen CANopen und SPS-Programm erfolgt mit Hilfe der Signale ENABLE und CONFIRM der Funktionsbausteine. Das Zusammenspiel der beiden Signale verdeutlicht Bild 16.

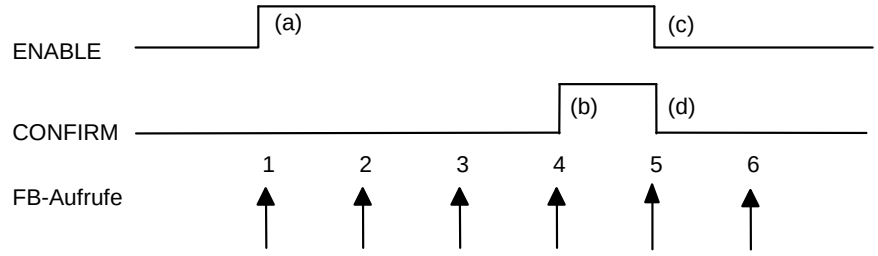


Bild 16: Prozesssynchronisation zwischen CANopen und SPS-Programm

Die vollständige und erfolgreiche Abarbeitung eines asynchron zum SPS-Programm ausgeführten CANopen-Dienstes vollzieht sich dabei in folgenden Schritten:

1. Nachdem das SPS-Programm alle Input-Variablen mit gültigen Werten initialisiert hat, setzt es den Eingang ENABLE auf TRUE und veranlasst den Aufruf des CANopen-Funktionsbausteines (Aufruf 1). Der Funktionsbaustein erkennt eine steigende Flanke am Eingang ENABLE, übernimmt darauf hin alle Eingangswerte und startet den entsprechenden CANopen-Dienst (Schritt (a)). Anschließend kehrt der Funktionsbaustein zum SPS-Programm zurück, wobei der initiierte CANopen-Dienst weiterhin im Hintergrund abgearbeitet wird.

2. Bis zur vollständigen Abarbeitung des CANopen-Dienstes wird der Funktionsbaustein mehrfach durch das SPS-Programm aufgerufen. Dabei muss der Eingang ENABLE auch weiterhin auf TRUE gesetzt bleiben, um dem Baustein die weitere Ausführung des CANopen-Dienstes im Hintergrund zu erlauben (Aufrufe 2 und 3).

3. Nach dem erfolgreichen Abschluss des CANopen-Dienstes setzt der Funktionsbaustein seinen Ausgang CONFIRM auf TRUE. Dies signalisiert dem SPS-Programm die Beendigung des Dienstes durch CANopen und zeigt weiterhin an, dass ggf. weitere Output-Variablen nun mit gültigen Werten besetzt sind (z.B. mit den von einem Knoten gelesenen Daten, Schritt (b), Aufruf 4).

4. Das SPS-Programm seinerseits quittiert dem Funktionsbaustein den Abschluss des CANopen-Dienstes, indem es den Eingang ENABLE auf FALSE setzt. Dabei signalisiert das SPS-Programm gleichzeitig, dass es die ggf. vom Funktionsbaustein gelieferten Output-Variablen übernommen hat (Schritt (c), Aufruf 5). Im letzten Schritt setzt der Funktionsbaustein seinen Ausgang CONFIRM wieder auf FALSE, so dass sich das System nun wieder im Ausgangszustand befindet (Schritt (d)).

Die Netzwerkschicht gestattet in der Regel zu jedem Zeitpunkt immer nur die Ausführung einer begrenzten Anzahl asynchron zum SPS-Programm ausgeführter CANopen-Dienste. Mit dem Start eines Dienstes durch Setzen des Eingangs ENABEL auf TRUE (Schritt 1) wird die betreffende Ressource für die Nutzung durch andere Funktionsbausteine gesperrt. Dieser Lock-Zustand bleibt so lange erhalten, bis der betreffende Funktionsbaustein nach Abschluss des Dienstes (FB setzt seinen Ausgang CONFIRM auf TRUE, Schritt 3) nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (Schritt 4). Der zwischenzeitliche Aufruf eines anderen CANopen-Funktionsbausteines führt zur Fehlermeldung TRANSFER_BUSY am Ausgang ERROR.

Der Ausgang CONFIRM wechselt nur bei erfolgreichem Abschluss des jeweiligen CANopen-Dienstes von FALSE nach TRUE. Eventuell aufgetretene Fehler werden an den Ausgängen ERROR respektive ERRORINFO angezeigt. Es ist daher erforderlich, dass ein SPS-Programm neben dem Ausgang CONFIRM auch den Wert von ERROR überwacht, um so auch aufgetretene Fehler auswerten zu können.

Ein Aufruf des Funktionsbausteines mit auf FALSE gesetztem Eingang ENABLE führt in jedem Fall zum Abbruch eines im Hintergrund aktiven CANopen-Dienstes und veranlasst das interne Rücksetzen des Funktionsbausteines. Gleichzeitig werden der Ausgang CONFIRM auf FALSE sowie die Ausgänge ERROR und ERRORINFO auf den Wert NO_ERROR gesetzt.

4.1.4 Input-/Output-Parameter der CANopen-Funktionsbausteine

Die CANopen-Funktionsbausteine wurden so realisiert, dass sie die durch den CiA Draft Standard 405 definierte Funktionalität erfüllen. Teilweise weichen jedoch die verwendeten Input- und Output-Parameter von der Spezifikation des Standards ab.

- Es werden keine Arrays oder Strukturen als Input- oder Output-Parameter eines Funktionsbausteines benutzt, sondern die jeweils im Array oder in der Struktur enthaltenen Memberelemente als direkte Variablen angesprochen ("flache" Input-/Output-Struktur). Dies reduziert den internen Overhead bei der Parameterübergabe.

- Statt einer spezifischen Typdefinition (z.B. "CIA405_DEVICE") wird der jeweils zugrunde liegende Elementartyp (z.B. USINT) für die Input- und Output-Parameter des Funktionsbausteines verwendet.

- Anstelle von Aufzählungstypen (z.B. "CIA405_STATE") werden numerische Konstanten verwendet. Diese sind im Abschnitt 4.1.5 aufgelistet.

Die jeweils konkret benutzten Input-/Output-Parameter der CANopen-Funktionsbausteine sind detailliert im Abschnitt zum entsprechenden Funktionsbaustein aufgeführt.

4.1.5 CANopen-spezifische Konstanten

Zur Kennzeichnung von internen Fehlerzuständen der Netzwerkschicht definiert der CiA Draft Standard 405 den spezifischen Datentyp "CIA405_CANOPEN_KERNEL_ERROR". Hier sind die Fehlerzustände zusammengefasst, die innerhalb der lokalen Netzwerkschicht einer SPS auftreten können. Diese Errorcodes werden von verschiedenen Funktionsbausteinen als Ausgabeparameter ERROR verwendet. Tabelle 8 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Errorcodes auf.

Tabelle 8: Konstanten für Datentyp "CIA405_CANOPEN_KERNEL_ERROR"

Konstante		Errorcode
16#0000	(= 00 dez)	NO_ERROR
16#0001	(= 01 dez)	OTHER_ERROR
16#0002	(= 02 dez)	DATA_OVERFLOW
16#0003	(= 03 dez)	TIME_OUT
16#0010	(= 16 dez)	CAN_BUS_OFF
16#0011	(= 17 dez)	CAN_ERROR_PASSIVE
16#0021	(= 33 dez)	GENERIC_ERROR (SYSTEC spezifisch)
16#0022	(= 34 dez)	FUNCTION_NOT_AVAIALBE
16#0023	(= 35 dez)	NO_MASTER_MODE
16#0024	(= 36 dez)	INVALID_DEVICE
16#0025	(= 37 dez)	TRANSFER_BUSY
16#0030	(= 48 dez)	NO_SDO_CHANNEL_AVAIALBE
16#0031	(= 49 dez)	SDO_BUSY
16#0032	(= 50 dez)	SDO_INITIALIZE_ERROR
16#0033	(= 51 dez)	SDO_LENGTH_ERROR
16#0034	(= 52 dez)	SDO_ERROR (SDO Abort Code an ERRORINFO)
16#0040	(= 64 dez)	NO_VALID_DATA_AVAIALBE
16#0041	(= 65 dez)	COBID_ALREADY_REGISTERED
16#0042	(= 66 dez)	NO_FREE_COBID_TABLE_ENTRY
16#0043	(= 67 dez)	NO_SUCH_COBID_REGISTERED
16#0044	(= 68 dez)	NO_FREE_RECEIVE_CHANNEL
16#0045	(= 69 dez)	DATA_LENGTH_ZERO_NOT_ALLOWED

Zur Kennzeichnung des Status, in dem sich ein CANopen-Gerät befindet, definiert der CiA Draft Standard 405 den spezifischen Datentyp "CIA405_STATE". Tabelle 9 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Statuswerten auf. Die Statuswerte UNKNOWN und NOT_AVAIL stellen eine Erweiterung der entsprechenden Definitionen des CiA Draft Standard 301 dar, alle anderen Konstantenwerte korrespondieren mit diesem Standard.

Tabelle 9: Konstanten für Datentyp "CIA405_STATE"

Konstante	Statuswert
16#0000	INIT
16#0001	RESET_COMM
16#0002	RESET_APP
16#0003	PRE_OPERATIONAL
16#0004	STOPPED
16#0005	OPERATIONAL
16#0006	UNKNOWN
16#0007	NOT_AVAIL

Zur Kennzeichnung des Status, in den ein CANopen-Gerät wechseln soll, definiert der CiA Draft Standard 405 den spezifischen Datentyp "CIA405_TRANSITION_STATE". Tabelle 10 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Statuswerten auf. Die Konstantenwerte korrespondieren mit den entsprechenden Definitionen des CiA Draft Standard 301.

Tabelle 10: Konstanten für Datentyp "CIA405_TRANSITION_STATE"

Konstante	Statuswert
16#0000	START_REMOTE_NODE
16#0001	STOP_REMOTE_NODE
16#0002	ENTER_PRE_OPERATIONAL
16#0003	RESET_NODE
16#0004	RESET_COMMUNICATION

Ergänzend definiert der CiA Draft Standard 405 die spezifischen Datentypen "CIA405_SDO_ERROR" sowie "EMCY_ERR_CODE" und "EMCY_ERR_REGISTER". Diese Datentypen repräsentieren die von einem anderen Knoten generierten Fehlermeldungen.

Zur Auswahl des SDO-Transfermodus bei einigen SDO-Funktionsbausteinen wird der SYSTEC-spezifische Datentyp "CAN_SDO_TYPE" verwendet. Tabelle 11 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden SDO-Typen auf.

Tabelle 11: Konstanten für Datentyp "CAN_SDO_TYPE"

Konstante	SDO-Typ
0	SDO_TYPE_AUTO_BEST_CASE
1	SDO_TYPE_SEGMENTED_TRANSFER
2	SDO_TYPE_BLOCK_TRANSFER

Der Datentyp "CIA405_SDO_ERROR" wird für den Parameter ERRORINFO der SDO-Funktionsbausteine verwendet und liefert den SDO Abort Code des Kommunikationspartners. Die allgemeinen Abort Codes sind im CiA Draft Standard 301 definiert, diese können aber durch den Hersteller der jeweiligen CANopen-Baugruppe erweitert werden. Tabelle 12 listet die Zuordnung der verwendeten numerischen Konstanten zu den typischen SDO Abort Codes auf.

Tabelle 12: Konstanten für Datentyp "CIA405_SDO_ERROR"

Konstante	SDO Abort Code
16#05030000	TOGGEL_BIT_ERROR
16#05040000	TIME_OUT
16#05040001	UNKNOWN_COMMAND_SPECIFIER
16#05040002	INVALID_BLOCK_SIZE
16#05040003	INVALID_SEQUENCE_NUMBER
16#05040004	CRC_ERROR
16#05040005	OUT_OF_MEMORY
16#06010000	UNSUPPORTED_ACCESS
16#06010001	READ_TO_WRITE_ONLY_OBJ
16#06010002	WRITE_TO_READ_ONLY_OBJ
16#06020000	OBJECT_NOT_EXIST
16#06040041	OBJECT_NOT_MAPPABLE
16#06040042	PDO_LENGTH_EXCEEDED
16#06040043	GEN_PARAM_INCOMPATIBILITY
16#06040047	GEN_INTERNAL_INCOMPATIBILITY
16#06060000	ACCESS_FAILED_DUE_HW_ERROR
16#06070010	DATA_TYPE_LENGTH_NOT_MATCH
16#06070012	DATA_TYPE_LENGTH_TOO_HIGH
16#06070013	DATA_TYPE_LENGTH_TOO_LOW
16#06090011	SUB_INDEX_NOT_EXIST
16#06090030	VALUE_RANGE_EXCEEDED
16#06090031	VALUE_RANGE_TOO_HIGH
16#06090032	VALUE_RANGE_TOO_LOW
16#06090036	MAX_VALUE_LESS_MIN_VALUE
16#08000000	GENERAL_ERROR
16#08000020	DATA_NOT_TRANSF_OR_STORED
16#08000021	DATA_NOT_TRANSF_DUE_LOCAL_CONTROL
16#08000022	DATA_NOT_TRANSF_DUE_DEVICE_STATE
16#08000023	OBJECT_DICTIONARY_NOT_EXIST

Die Datentypen "EMCY_ERR_CODE" und "EMCY_ERR_REGISTER" werden für die entsprechenden Parameter der Funktionsbausteine CAN_RECV_EMCY und CAN_RECV_EMCY_DEV verwendet. Sie beinhalten die Emergency Error Information desjenigen Knotens, der die entsprechende Emergency-Nachricht generiert hat. Die allgemeinen Emergency Errors sind im CiA Draft Standard 301 definiert, diese können aber durch den Hersteller der jeweiligen CANopen-Baugruppe erweitert werden.

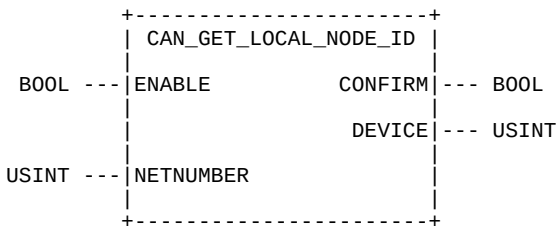
4.2 Funktionsbausteine für Zugriff auf lokalen CANopen-Kernel

Die Funktionsbausteine für den Zugriff auf den lokalen CANopen-Kernel der eigenen SPS ermöglichen die Abfrage der Knotenadresse sowie des Status der Netzwerkschicht. Diese Funktionsbausteine erfordern keine Kommunikation mit anderen Knoten.

4.2.1 Funktionsbaustein CAN_GET_LOCAL_NODE_ID

FB zum Abfragen der lokalen Knotenadresse.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	lokale Knotenadresse der SPS
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

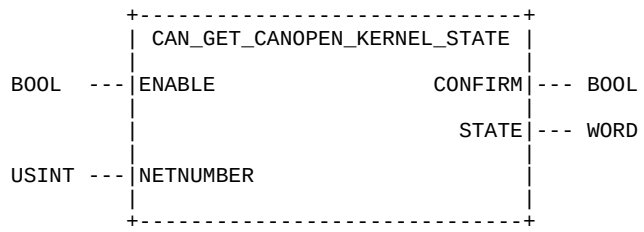
Beschreibung

Der Funktionsbaustein CAN_GET_LOCAL_NODE_ID dient zur Abfrage der lokalen Knotenadresse der SPS. Die Knotenadresse einer Steuerung hat Einfluss auf die Verfügbarkeit der verschiedenen Funktionsbausteine (SPS mit und ohne CANopen-Master, siehe Abschnitt 4.1.2)

4.2.2 Funktionsbaustein CAN_GET_CANOPEN_KERNEL_STATE

FB zur Statusabfrage des CANopen-Kernels der eigenen SPS.

Prototyp des Funktionsbausteines



Operandenbedeutung

STATE	Status bzw. Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_GET_CANOPEN_KERNEL_STATE dient zur Statusabfrage des CANopen-Kernels der eigenen SPS.

4.3 Funktionsbausteine für PDOs und CAN Layer 2

PDOs und CAN Layer 2 Nachrichten werden aus Sicht der SPS zu beliebigen Zeitpunkten von einem Sender Generiert. Ihr Empfang steht in keinem festen Bezug zur Ausführung des SPS-Programms, sonder erfolgt asynchron. Folglich kann der Empfang solcher Nachrichten auch nicht durch den Aufruf eines Funktionsbausteines gesteuert bzw. beeinflusst werden. Die Verarbeitung von PDOs und CAN Layer 2 Nachrichten durch ein SPS-Programm setzt daher die Zwischenspeicherung der asynchron empfangenen Nachrichten in der Netzwerkschicht bis zum Aufruf des Funktionsbausteines CAN_PDO_READ8 (siehe Abschnitt 4.3.2) voraus. Aufgrund der Anzahl möglicher CAN-Messages ist hier jedoch eine Filterung der für das SPS-Programm relevanten Nachrichten notwendig. Dazu werden mit Hilfe des Funktionsbausteines CAN_REGISTER_COBID (siehe Abschnitt 4.3.1) zunächst die CAN-Identifier (COBIDs) der relevanten Nachrichten registriert. Der Empfang, und damit ein Zugriff durch den Funktionsbaustein CAN_PDO_READ8, ist grundsätzlich nur für registrierte Nachrichten möglich.

Die Funktionsbausteine für PDOs und CAN Layer 2 Nachrichten erweitern herstellerspezifisch den vom CiA Draft Standard 405 definierte Funktionsumfang verfügbarer CANopen-Dienste.

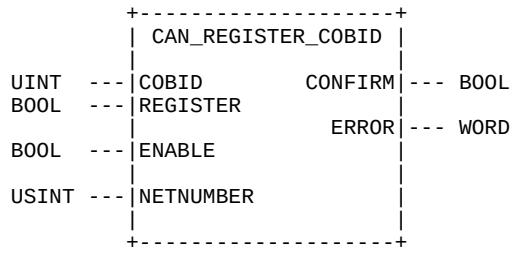
Hinweis: Die hier beschriebenen Funktionsbausteine CAN_PDO_READ8 (siehe Abschnitt 4.3.2) und CAN_PDO_WRITE8 (siehe Abschnitt 4.3.3) ermöglichen die Einbindung von nicht CANopen-fähigen Geräten in bestehende CANopen-Netze. Sie sind als Ergänzung zu den im CiA Draft Standard 405 definierten Funktionsbausteinen für CANopen konzipiert und unterliegen somit den durch CANopen vorgegebenen Restriktionen (z.B. Auswahl der nutzbaren CAN-Identifier).

Einen uneingeschränkten Datenaustausch auf CAN Layer 2 Ebene ohne CANopen-Restriktionen ermöglichen die im Teil 2 diese Manuals beschriebenen CAN Layer 2 Funktionsbausteine (siehe Abschnitt 6).

4.3.1 Funktionsbaustein CAN_REGISTER_COBID

FB zum Registrieren oder Löschen des Empfangs von PDOs und CAN Layer 2 Nachrichten durch die Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

COBID	COBID (CAN-Identifier) der neu in die Registrierung einzutragenden bzw. aus der Registrierung zu löschenden Nachricht (0 zum Löschen aller Nachrichten)
REGISTER	TRUE = COBID in Registrierung eintragen FALSE = COBID aus Registrierung löschen
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

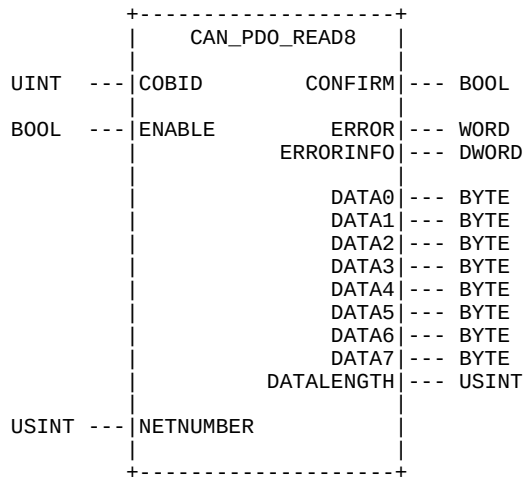
Der Funktionsbaustein CAN_REGISTER_COBID dient zum Registrieren eines PDOs bzw. einer CAN Layer 2 Nachricht für den Empfang durch die Netzwerkschicht oder zum Löschen einer solchen Registrierung. Beim Aufruf des Bausteines mit auf TRUE gesetztem Eingang REGISTER wird die angegebene COBID (CAN-Identifier) zum Empfang von Nachrichten in der Netzwerkschicht registriert. Beim Aufruf mit REGISTER = FALSE wird Registrierung der betreffenden COBID wieder gelöscht. Der Aufruf des Bausteines mit REGISTER = FALSE und COBID = 0 führt zum Löschen aller Registrierungen sowie zum Verwerfen der im Zwischenpuffer der Netzwerkschicht abgelegten Nachrichten.

Die Netzwerkschicht unterstützt einen Zugriff auf PDOs und CAN Layer 2 Nachrichten durch den Aufruf des Funktionsbausteines CAN_PDO_READ8 (siehe Abschnitt 4.3.2) grundsätzlich nur für registrierte Nachrichten.

4.3.2 Funktionsbaustein CAN_PDO_READ8

FB zum Lesen von PDOs und CAN Layer 2 Nachrichten aus dem Empfangspuffer der Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

COBID	COBID (CAN-Identifier) der zu lesenden Nachricht (PDO bzw. CAN Layer 2)
DATA0 - DATA7	Datenbytes der empfangenen CAN-Nachricht
DATALENGTH	Länge der empfangenen CAN-Nachricht
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	reserviert für zusätzliche Fehlerinformationen
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

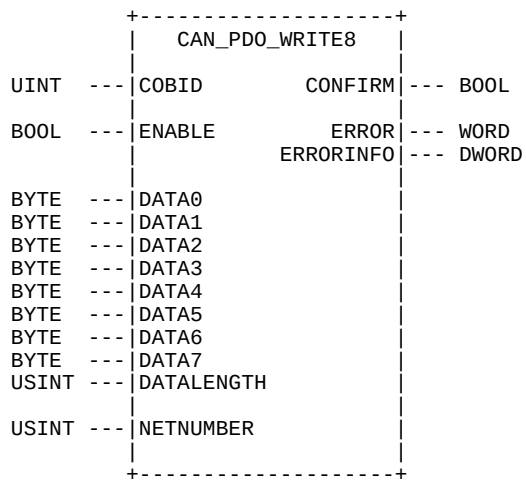
Der Funktionsbaustein CAN_PDO_READ8 dient zum Lesen von PDOs und CAN Layer 2 Nachrichten aus dem Empfangspuffer der Netzwerkschicht. Es wird nur ein Zugriff auf die zuvor mit Hilfe des Funktionsbausteines CAN_REGISTER_COBID (siehe Abschnitt 4.3.1) registrierten Nachrichten unterstützt. Werden zwischen zwei aufeinander folgenden Aufrufen von CAN_PDO_READ8 mehrerer Nachrichten mit derselben COBID empfangen, so überschreibt die zuletzt empfangene Nachricht die jeweils vorangegangene. Somit bleibt also stets nur die aktuellste Nachricht im Empfangspuffer erhalten. Nach dem Auslesen durch den Funktionsbaustein CAN_PDO_READ8 wird die entsprechende Nachricht aus dem Empfangspuffer der Netzwerkschicht gelöscht. Das verhindert ein mehrfaches Lesen einer Nachricht durch das SPS-Programm.

Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthalten die Elemente DATA0 bis DATA7 die einzelnen Bytes der empfangenen Nachricht. Der Ausgang DATALENGTH gibt dabei die Anzahl der gültigen Datenbytes (ab DATA0) an. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer in der Netzwerkschicht keine Nachrichten mit der angegebenen COBID. Anhand von CONFIRM kann somit unterschieden werden, ob eine gültige Nachricht mit 0 Bytes Länge oder keine Nachricht empfangen wurde. Ist keine Nachricht verfügbar, wird dies ebenfalls durch den Fehlercode NO_VALID_DATA_AVAILABLE am Ausgang ERROR angezeigt (siehe Tabelle 8 im Abschnitt 4.1.5).

4.3.3 Funktionsbaustein CAN_PDO_WRITE8

FB zum Senden von PDOs und CAN Layer 2 Nachrichten durch die Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

COBID	COBID (CAN-Identifier) der zu sendenden Nachricht (PDO bzw. CAN Layer 2)
DATA0 - DATA7	Datenbytes der zu sendenden CAN-Nachricht
DATALENGTH	Länge der zu sendenden CAN-Nachricht
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	reserviert für zusätzliche Fehlerinformationen
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_PDO_WRITE8 dient zum Senden von PDOs und CAN Layer 2 Nachrichten durch die Netzwerkschicht. An die Elemente DATA0 bis DATA7 sind die einzelnen Bytes der zu sendenden Nachricht zu übergeben. Der Eingang DATALENGTH spezifiziert dabei die Anzahl der gültigen Datenbytes (ab DATA0).

Beim Aufruf des Funktionsbausteines CAN_PDO_WRITE8 wird die zu sendende Nachricht im Sendepuffer des CANopen-Kernel abgelegt. Tritt hierbei kein Fehler auf (Nachricht konnte korrekt im Sendepuffer hinterlegt werden), kehrt der Baustein mit auf TRUE gesetztem Ausgang CONFIRM zurück. Es erfolgt jedoch keine Rückmeldung zum SPS-Programm darüber, ob die Nachricht tatsächlich erfolgreich gesendet werden konnte.

4.4 Funktionsbausteine für SDOs

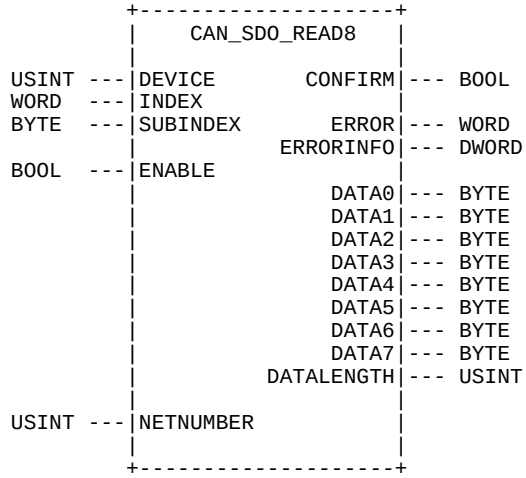
Mit Hilfe von SDOs kann eine SPS Einträge im Object Dictionary eines Knotens lesen oder schreiben. Der Datentransfer läuft dabei im Quittungsbetrieb, so dass Kommunikationsfehler sicher erkannt werden. Da der vollständige SDO-Transfer stets aus mehreren CAN-Nachrichten besteht, muss dieser asynchron im Hintergrund abgewickelt werden, um das SPS-Programm selbst nicht zu blockieren. Die Funktionsbausteine für SDOs kehren nach der Initiierung des Transfers sofort wieder zurück. Zur Synchronisation zwischen CANopen-Funktionsbaustein und SPS-Programm dient das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM.

Die Netzwerkschicht stellt einen SDO-Kanal zur Nutzung durch das SPS-Programm bereit. Dadurch kann zu jedem Zeitpunkt immer nur ein einziger SDO-Funktionsbaustein aktiv sein. Mit der erfolgreichen Initiierung des SDO-Transfers wird der SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Dieser Lock-Zustand bleibt so lange erhalten, bis der aktive SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen werden (siehe Abschnitt 4.1.3).

4.4.1 Funktionsbaustein CAN_SDO_READ8

FB zum Lesen von Objekteinträgen eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu lesenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu lesenden Indexeintrages
SUBINDEX	Nummer des zu lesenden Subindexeintrages

DATA0 - DATA7	Datenbytes des gelesenen Eintrages
DATALENGTH	Länge des gelesenen Eintrages
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_READ8 dient zum Lesen von Objekteinträgen eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthalten die Elemente DATA0 bis DATA7 die einzelnen Bytes des gelesenen Objekteintrages. Der Ausgang DATALENGTH gibt dabei die Anzahl der gültigen Datenbytes (ab DATA0) an.

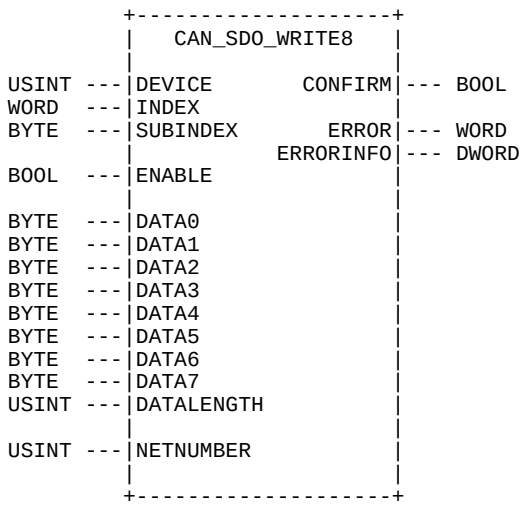
Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte aus dem eigenen OD gelesen werden.

4.4.2 Funktionsbaustein CAN_SDO_WRITE8

FB zum Schreiben von Objekteinträgen eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu schreibenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu schreibenden Indexeintrages
SUBINDEX	Nummer des zu schreibenden Subindexeintrages
DATA0 - DATA7	Datenbytes des zu schreibenden Eintrages
DATALENGTH	Länge des zu schreibenden Eintrages
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_WRITE8 dient zum Schreiben von Objekteinträgen eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

An die Elemente DATA0 bis DATA7 sind die einzelnen Bytes des zu schreibenden Objekteintrages zu übergeben. Der Eingang DATALENGTH spezifiziert dabei die Anzahl der gültigen Datenbytes (ab DATA0).

Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der

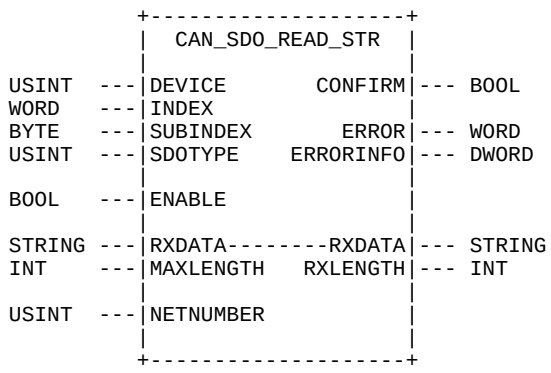
betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte im eigenen OD geschrieben werden.

4.4.3 Funktionsbaustein CAN_SDO_READ_STR

FB zum Lesen von Zeichenfolgen aus dem Objektverzeichnis eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu lesenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu lesenden Indexeintrages
SUBINDEX	Nummer des zu lesenden Subindexeintrages
SDOTYPE	Typ des zu verwendenden SDO-Transfermodus entsprechend dem Datentyp "CAN_SDO_TYPE" (siehe Tabelle 11 im Abschnitt 4.1.5) Hinweis: Wird diesem Eingang nicht explizit ein anderer Wert zugewiesen (Eingang offen bzw. nicht benutzt), verwendet der Funktionsbaustein den Modus "SDO_TYPE_AUTO_BEST_CASE". Hierbei wählt die Netzwerkschicht selbständig den am besten geeigneten SDO-Transfermodus anhand der zu übertragenden Datenmenge.
RXDATA	Stringvariable zur Aufnahme der gelesenen Zeichen
MAXLENGTH	Begrenzung der Anzahl zu lesender Zeichen, bei 0 wird intern die Pufferlänge des übergebenen Strings ermittelt und als Begrenzung der Anzahl zu lesender Zeichen verwendet (Hinweis: die Standard-Puffergröße eines Strings in OpenPCS beträgt 32 Zeichen).
RXLENGTH	Länge der gelesenen Zeichenfolge
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)

ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_READ_STR dient zum Lesen von Zeichenfolgen aus dem Objektverzeichnis eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthält der als Element RXDATA übergebene String die Zeichenfolge des gelesenen Objekteintrages. Der Ausgang RXLENGTH gibt dabei die Anzahl der gelesenen Zeichen an (entspricht LEN(RXDATA);).

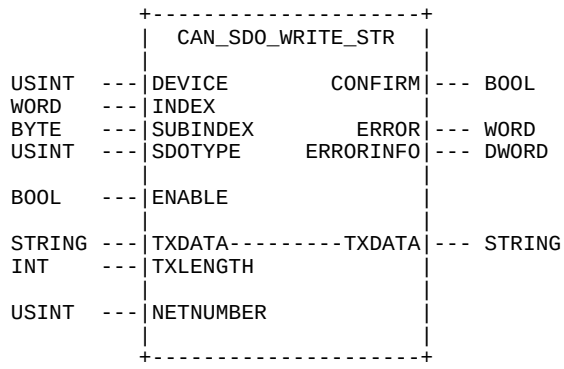
Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte aus dem eigenen OD gelesen werden.

4.4.4 Funktionsbaustein CAN_SDO_WRITE_STR

FB zum Schreiben von Zeichenfolgen in das Objektverzeichnis eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu schreibenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu schreibenden Indexeintrages
SUBINDEX	Nummer des zu schreibenden Subindexeintrages
SDOTYPE	Typ des zu verwendenden SDO-Transfermodus entsprechend dem Datentyp "CAN_SDO_TYPE" (siehe Tabelle 11 im Abschnitt 4.1.5) Hinweis: Wird diesem Eingang nicht explizit ein anderer Wert zugewiesen (Eingang offen bzw. nicht benutzt), verwendet der Funktionsbaustein den Modus "SDO_TYPE_AUTO_BEST_CASE". Hierbei wählt die Netzwerkschicht selbständig den am besten geeigneten SDO-Transfermodus anhand der zu übertragenden Datenmenge.
TXDATA	Stringvariable mit der zu schreibenden Zeichenfolge
TXLENGTH	Anzahl der zu schreibenden Zeichen, bei 0 wird intern die Länge der im String TXDATA enthaltenen Zeichenfolge ermittelt (entspricht LEN(TXDATA);) und als Anzahl der zu schreibenden Zeichen verwendet.
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_WRITE_STR dient zum Schreiben von Zeichenfolgen in das Objektverzeichnis eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

Am Element TXDATA ist die in das Objektverzeichnis zu schreibende Zeichenfolge zu übergeben. Der Eingang TXLENGTH spezifiziert dabei die Anzahl der gültigen Zeichen. Ist dieser Wert 0, wird intern die Länge der im String TXDATA enthaltenen Zeichenfolge ermittelt (entspricht LEN(TXDATA);) und als Anzahl der zu schreibenden Zeichen verwendet. In diesem Fall wird also der gesamte belegte Stringinhalt geschrieben.

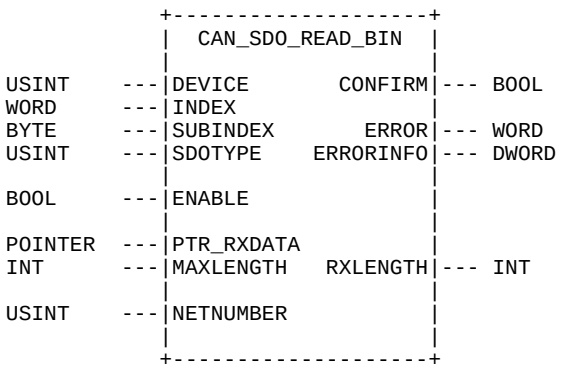
Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte im eigenen OD geschrieben werden.

4.4.5 Funktionsbaustein CAN_SDO_READ_BIN

FB zum Lesen von Binärdaten aus dem Objektverzeichnis eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu lesenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu lesenden Indexeintrages
SUBINDEX	Nummer des zu lesenden Subindexeintrages
SDOTYPE	Typ des zu verwendenden SDO-Transfermodus entsprechend dem Datentyp "CAN_SDO_TYPE" (siehe Tabelle 11 im Abschnitt 4.1.5) Hinweis: Wird diesem Eingang nicht explizit ein anderer Wert zugewiesen (Eingang offen bzw. nicht benutzt), verwendet der Funktionsbaustein den Modus "SDO_TYPE_AUTO_BEST_CASE". Hierbei wählt die Netzwerkschicht selbständig den am besten geeigneten SDO-Transfermodus anhand der zu übertragenden Datenmenge.
PTR_RXDATA	Adresse eines Objektes, in dem die gelesenen Daten abgelegt werden
MAXLENGTH	Begrenzung der Anzahl zu lesender Bytes, bei 0 wird intern die Größe des über PTR_RXDATA adressierten Objektes ermittelt und als Begrenzung der Anzahl zu lesender Bytes verwendet (es werden max. so viele Bytes gelesen, wie das Objekt aufnehmen kann).
RXLENGTH	Anzahl der gelesenen Datenbytes
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_READ_BIN dient zum Lesen von Binärdaten aus dem Objektverzeichnis eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthält das über PTR_RXDATA adressierte Objekt die Daten des gelesenen Objekteintrages. Der Ausgang RXLENGTH gibt dabei die Anzahl der gelesenen Bytes an.

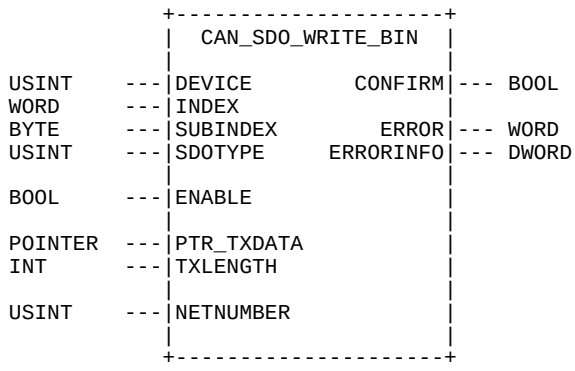
Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte aus dem eigenen OD gelesen werden.

4.4.6 Funktionsbaustein CAN_SDO_WRITE_BIN

FB zum Schreiben von Binärdaten in das Objektverzeichnis eines Knotens mittels SDO-Transfer.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu schreibenden Knotens (1-127 oder 0 für lokales OD)
INDEX	Nummer des zu schreibenden Indexeintrages
SUBINDEX	Nummer des zu schreibenden Subindexeintrages
SDOTYPE	Typ des zu verwendenden SDO-Transfermodus entsprechend dem Datentyp "CAN_SDO_TYPE" (siehe Tabelle 11 im Abschnitt 4.1.5) Hinweis: Wird diesem Eingang nicht explizit ein anderer Wert zugewiesen (Eingang offen bzw. nicht benutzt), verwendet der Funktionsbaustein den Modus "SDO_TYPE_AUTO_BEST_CASE". Hierbei wählt die Netzwerkschicht selbständig den am besten geeigneten SDO-Transfermodus anhand der zu übertragenden Datenmenge.

PTR_TXDATA	Adresse eines Objektes, das die zu schreibenden Daten beinhaltet
TXLENGTH	Anzahl der zu schreibenden Zeichen, bei 0 wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu schreibender Bytes verwendet
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
ERRORINFO	SDO Abort Code des Kommunikationspartners entsprechend dem Datentyp "CIA405_SDO_ERROR" (siehe Tabelle 12 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SDO_WRITE_STR dient zum Schreiben von Binärdaten in das Objektverzeichnis eines Knotens unter Verwendung des SDO-Transfers. Der SDO-Transfer wird dabei stets im Hintergrund ausgeführt, so dass zur Synchronisation zwischen Funktionsbaustein und SPS-Programm das im Abschnitt 4.1.3 beschriebene Verfahren unter Verwendung der beiden Parameter ENABLE und CONFIRM anzuwenden ist.

Am Element PTR_TXDATA ist die Adresse eines Objektes anzugeben, das die zu schreibenden Daten beinhaltet. Der Eingang TXLENGTH spezifiziert dabei die Anzahl der gültigen Zeichen. Ist dieser Wert 0, wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu schreibender Bytes verwendet.

Die Netzwerkschicht unterstützt zu jedem Zeitpunkt immer nur eine begrenzte Anzahl von SDO-Transfers durch das SPS-Programm (Standard: max. 5 Transfers, falls keine abweichende Angabe im Manual der SPS). Nach dem Start des SDO-Transfers durch Setzen von ENABLE auf TRUE wird der betreffende SDO-Kanal für die Nutzung durch andere Bausteine gesperrt. Der Lock-Zustand bleibt erhalten, bis der SDO-Funktionsbaustein nach Abschluss des Datentransfers nochmals mit auf FALSE gesetztem ENABLE Eingang aufgerufen wurde (siehe Abschnitt 4.1.3). Entfällt der Aufruf mit ENABLE = FALSE, bleibt die Ressource dauerhaft blockiert und steht dem SPS-Programm nicht mehr zur Verfügung.

Der Aufruf des Funktionsbausteines mit DEVICE = 0 führt zum Zugriff auf das lokale Object-Dictionary der SPS. Somit können auch Werte im eigenen OD geschrieben werden.

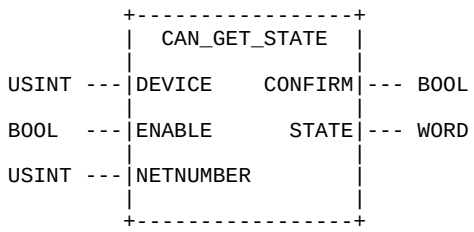
4.5 Funktionsbausteine für Master-Dienste

Die Funktionsbausteine für Master-Dienste ermöglichen Statusabfragen, Überwachen und Verändern des Betriebsmodus beliebiger Knoten sowie das Generieren von SYNC-Nachrichten. Dabei kommen CANopen-Dienste zum Einsatz, deren Verwendung stets nur exklusiv einem einzigen Knoten innerhalb des Netzwerkes gestattet ist (z.B. NMT-Dienste). Daher wird die Funktionalität "SPS mit CANopen-Master" fest an eine bestimmte Knotenadresse gekoppelt, so dass diese Funktion immer nur von einer einzigen Steuerung im Netzwerk übernommen werden kann (siehe Abschnitt 4.1.2).

4.5.1 Funktionsbaustein CAN_GET_STATE

FB zum Abfragen des Knotenstatus von Geräten.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des abzufragenden Knotens (1-127 oder 0 für eigenen Knoten)
STATE	Knotenstatus entsprechend dem Datentyp "CIA405_STATE" (siehe Tabelle 9 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_GET_STATE dient zur Abfrage des Knotenstatus eines Gerätes. Die Statusabfrage basiert auf der Überwachung mittels Heartbeat oder Lifeguarding. Detaillierte Informationen hierzu enthält der Abschnitt 2.2. Die Rückgabewerte am Ausgang STATE besitzen folgende Bedeutung:

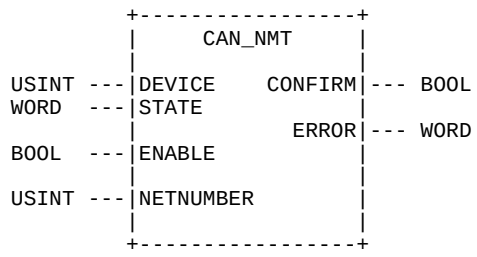
UNKNOWN:	Das CANopen-Gerät auf der angegebenen Adresse unterstützt weder Heartbeat noch Lifeguarding, so dass sein Status nicht überwacht werden kann. Auf einer SPS ohne CANopen-Master wird dieser Status auch dann gemeldet, wenn entweder keine SPS mit CANopen-Master im Netz verfügbar ist, die eine Statusweiterleitung gemäß Abschnitt 2.2 unterstützt oder wenn sich diese Master-SPS im Stop-Zustand befindet (SPS-Programm angehalten).
NOT_AVAIL:	Das CANopen-Gerät auf der angegebenen Adresse antwortet nicht mehr auf Heartbeat- bzw. Lifeguarding-Anfragen und ist somit nicht mehr für das System verfügbar.
andere:	Mit Ausnahme der Statuswerte UNKNOWN und NOT_AVAIL korrespondieren die Rückgabewerte mit den entsprechenden Definitionen des CiA Draft Standard 301 (siehe Tabelle 9 im Abschnitt 4.1.5).

Der Aufruf des Funktionsbausteines mit DEVICE = 0 liefert den lokalen Knotenstatus der eigenen SPS.

4.5.2 Funktionsbaustein CAN_NMT

FB zum Senden von NMT-Nachrichten.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des zu steuernden Knotens (1-127 oder 0 für alle Knoten)
STATE	Knotenstatus entsprechend dem Datentyp "CIA405_TRANSITION_STATE" (siehe Tabelle 10 im Abschnitt 4.1.5)
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

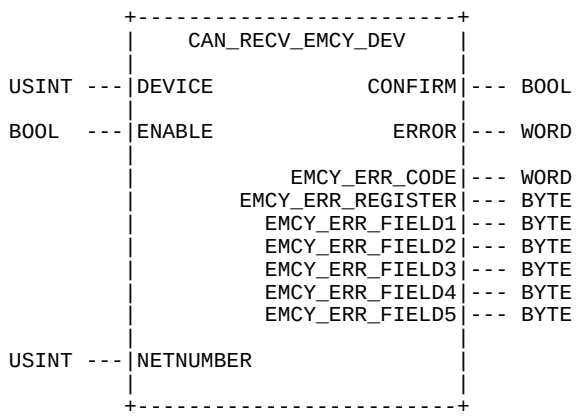
Der Funktionsbaustein CAN_NMT dient zum Steuern des Status eines Knotens (DEVICE = 1...127) bzw. bei DEVICE = 0 aller Knoten im Netz. Mit Hilfe dieses Bausteins kann ein SPS-Programm das Netzwerk auch dann manuell in den Zustand Operational setzen, wenn der CANopen-Master den automatischen Netzwerkstart unterdrückt hat. Dies ist beispielsweise dann der Fall, wenn im Objekt 1F80H das Bit 3 ("Do not send NMT-Start Remote Node") gesetzt ist oder wenn während der Konfigurationsphase Fehler auftraten (siehe Abschnitt 2.2). Tabelle 8 enthält die möglichen Stati (siehe Abschnitt 4.1.5).

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2).

4.5.3 Funktionsbaustein CAN_RECV_EMCY_DEV

FB zum Lesen von Emergency-Nachrichten eines spezifischen Knotens aus dem Empfangspuffer der Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE — Adresse des Knotens (1-127), für den der Empfang von Emergency-Nachrichten zu prüfen ist

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
Emergency Error Informationen entsprechend dem CiA Draft Standard 301

ERROR — Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)

NETNUMBER — Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)

ENABLE — Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM — Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_RECV_EMCY_DEV dient zum Lesen von Emergency-Nachrichten eines spezifischen Knotens aus dem Empfangspuffer der Netzwerkschicht. Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthalten die Elemente EMCY_ERR die Emergency Error Informationen des Knotens entsprechend dem CiA Draft Standard 301. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der Netzwerkschicht keine Emergency-Nachricht für den betreffenden Knoten.

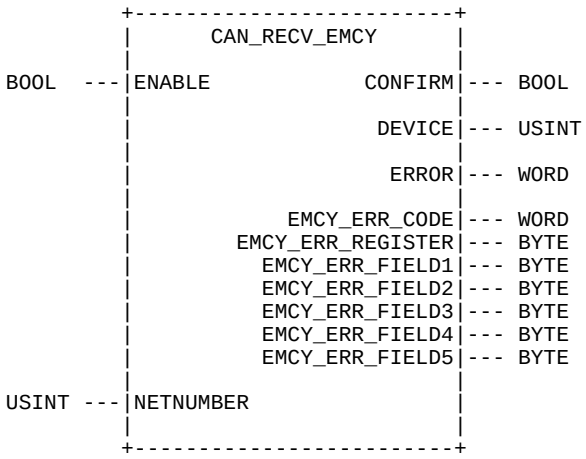
Der Funktionsbaustein gibt stets die zuerst im Empfangspuffer eingetragene Emergency-Nachricht (= älteste Nachricht) des jeweiligen Knotens zurück, anschließend wird die Nachricht aus dem Empfangspuffer gelöscht. Somit kann jede Emergency-Nachricht nur einmal vom SPS-Programm gelesen werden. Die Funktionsbausteine CAN_RECV_EMCY_DEV und CAN_RECV_EMCY (siehe Abschnitt 4.5.4) greifen beide auf denselben Empfangspuffer zurück.

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2).

4.5.4 Funktionsbaustein CAN_RECV_EMCY

FB zum Lesen von Emergency-Nachrichten eines beliebigen Knotens aus dem Empfangspuffer der Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE — Adresse des Knotens (1-127), von dem eine Emergency-Nachrichten empfangen wurde

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
Emergency Error Informationen entsprechend dem CiA Draft Standard 301

ERROR — Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)

NETNUMBER — Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)

ENABLE — Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM — Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_RECV_EMCY dient zum Lesen von Emergency-Nachrichten eines beliebigen Knotens aus dem Empfangspuffer der Netzwerkschicht. Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, gibt der Ausgang DEVICE die Knotenadresse an, von der eine Nachricht empfangen wurde. Die Elemente EMCY_ERR enthalten die Emergency Error Informationen des Knotens entsprechend dem CiA Draft Standard 301. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der Netzwerkschicht keine Emergency-Nachrichten.

Der Funktionsbaustein gibt stets die zuerst im Empfangspuffer eingetragene Emergency-Nachricht (= älteste Nachricht) zurück, anschließend wird die Nachricht aus dem Empfangspuffer gelöscht. Somit kann jede Emergency-Nachricht nur einmal vom SPS-Programm gelesen werden. Die

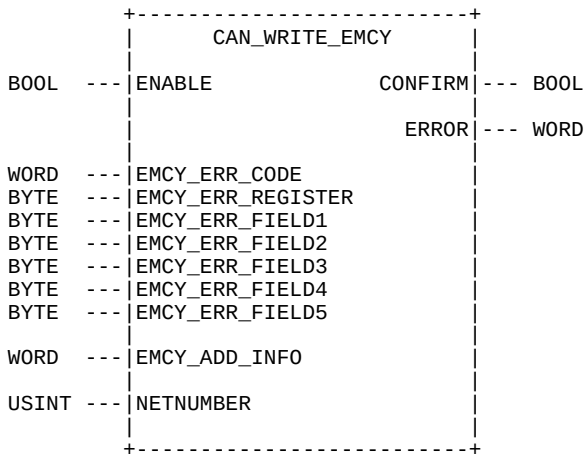
Funktionsbausteine CAN_RECV_EMCY_DEV (siehe Abschnitt 4.5.3) und CAN_RECV_EMCY greifen beide auf denselben Empfangspuffer zurück.

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2).

4.5.5 Funktionsbaustein CAN_WRITE_EMCY

FB zum Senden von applikationsspezifischen Emergency-Nachrichten durch die Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
: Emergency Error Informationen entsprechend dem CiA Draft Standard 301 für die zu sendende Emergency-Nachricht

EMCY_ADD_INFO
: Zusätzliche, anwenderspezifische Emergency Error Information, wird im Index 1003H des Object-Dictionary (Error Field, siehe CiA Draft Standard 301) eingetragen, ist kein Bestandteil der zu sendenden Emergency-Nachricht, sondern dient nur zu Diagnosezwecken und kann somit auch Null sein

ERROR
: Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)

NETNUMBER
: Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)

ENABLE
: Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)

CONFIRM
: Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

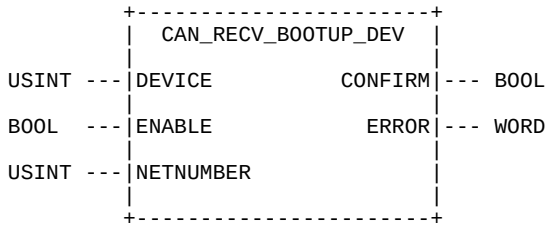
Der Funktionsbaustein CAN_WRITE_EMCY dient zum Senden applikationsspezifischen Emergency-Nachrichten durch die Netzwerkschicht. Die Elemente EMCY_ERR enthalten die zu sendenden Emergency Error Informationen der Anwenderapplikation entsprechend dem CiA Draft Standard 301. Mit dem Element EMCY_ADD_INFO kann die Anwenderapplikation eine weitere Fehlerinformation übergeben, die jedoch nicht Bestandteil der zu sendenden Emergency-Nachricht ist, sonder im Index 1003H des Object-Dictionary (Error Field, siehe CiA Draft Standard 301) eingetragen wird. Diese Fehlerinformation dient lediglich zu Diagnosezwecken und kann somit auch Null sein. Der Index 1003H des Object-Dictionary kann mit Hilfe eines Konfigurations- und Diagnosetools ausgelesen werden.

Beim Aufruf des Funktionsbausteines CAN_WRITE_EMCY wird die zu sendende Nachricht im Sendepuffer des CANopen-Kernel abgelegt. Tritt hierbei kein Fehler auf (Nachricht konnte korrekt im Sendepuffer hinterlegt werden), kehrt der Baustein mit auf TRUE gesetztem Ausgang CONFIRM zurück. Es erfolgt jedoch keine Rückmeldung zum SPS-Programm darüber, ob die Nachricht tatsächlich erfolgreich gesendet werden konnte.

4.5.6 Funktionsbaustein CAN_RECV_BOOTUP_DEV

FB zum Lesen von Bootup-Nachrichten eines spezifischen Knotens aus dem Empfangspuffer der Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des Knotens (1-127), für den der Empfang von Bootup-Nachrichten zu prüfen ist
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_RECV_BOOTUP_DEV dient zum Lesen von Bootup-Nachrichten eines spezifischen Knotens aus dem Empfangspuffer der Netzwerkschicht. Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, wurde ein Bootup-Nachricht für den angegebenen Knoten empfangen. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der Netzwerkschicht keine Bootup-Nachricht für den betreffenden Knoten.

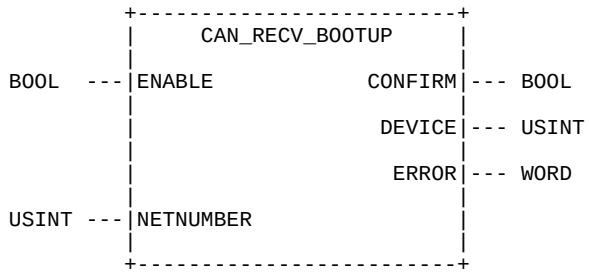
Nach dem Lesen einer Bootup-Nachricht wird diese aus dem Empfangspuffer gelöscht und somit nur einmal an das SPS-Programm gemeldet. Die Funktionsbausteine CAN_RECV_BOOTUP_DEV und CAN_RECV_BOOTUP (siehe Abschnitt 4.5.7) greifen beide auf denselben Empfangspuffer zurück.

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2).

4.5.7 Funktionsbaustein CAN_RECV_BOOTUP

FB zum Lesen von Bootup-Nachrichten eines beliebigen Knotens aus dem Empfangspuffer der Netzwerkschicht.

Prototyp des Funktionsbausteines



Operandenbedeutung

DEVICE	Adresse des Knotens (1-127), von dem eine Bootup-Nachrichten empfangen wurde
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_RECV_BOOTUP dient zum Lesen von Bootup-Nachrichten eines beliebigen Knotens aus dem Empfangspuffer der Netzwerkschicht. Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, gibt der Ausgang DEVICE die Knotenadresse an, von der eine Nachricht empfangen wurde. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der Netzwerkschicht keine Bootup-Nachrichten.

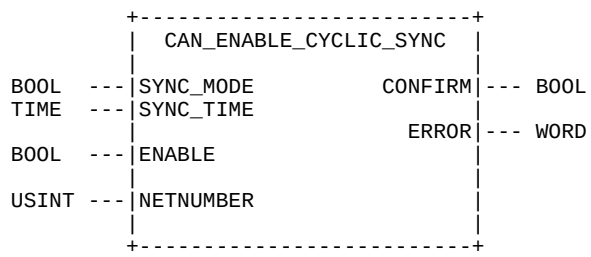
Der Funktionsbaustein gibt stets die zuerst im Empfangspuffer eingetragenen Bootup-Nachricht (= älteste Nachricht) zurück, anschließend wird die Nachricht aus dem Empfangspuffer gelöscht. Somit kann jede Bootup-Nachricht nur einmal vom SPS-Programm gelesen werden. Die Funktionsbausteine CAN_RECV_BOOTUP_DEV (siehe Abschnitt 4.5.6) und CAN_RECV_BOOTUP greifen beide auf denselben Empfangspuffer zurück.

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2).

4.5.8 Funktionsbaustein CAN_ENABLE_CYCLIC_SYNC

FB zum Freigeben oder Sperren von zyklischen SYNC-Nachrichten.

Prototyp des Funktionsbausteines



Operandenbedeutung

SYNC_MODE	TRUE = Generierung von zyklischen SYNC-Nachrichten freigeben FALSE = Generierung von zyklischen SYNC-Nachrichten sperren
SYNC_TIME	Zeit zwischen zwei aufeinanderfolgenden SYNC-Nachrichten, oder 0 für SYNC-Nachricht nach jedem SPS-Zyklus
ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_ENABLE_CYCLIC_SYNC dient zum Freigeben bzw. Sperren der Generierung von zyklischen SYNC-Nachrichten durch die SPS. Bei Freigabe generiert die Steuerung zwischen zwei aufeinander folgenden SPS-Zyklen eine SYNC-Nachricht, falls das letzte SYNC mindestens um die am Eingang SYNC_TIME angegebene Zeit zurück liegt. Ist die Zeit seit dem letzen SYNC kleiner als SYNC_TIME, wird keine neue SYNC-Nachricht erzeugt. Der Eingangswert SYNC_TIEM = 0 veranlasst die Steuerung, jeden SPS-Zyklus mit einer SYNC-Nachricht abzuschließen. In diesem Fall erfolgt der Austausch des Netzwerk-Prozessabbildes synchron zum Austausch des Prozessabbildes für die lokalen Ein- und Ausgänge der SPS.

Die Steuerung überprüft stets am Ende des SPS-Zyklus die Zeitbedingung zum Senden einer SYNC-Nachricht. Die am Eingang SYNC_TIME angegebene Zeit ist daher die Mindestzeit zwischen zwei aufeinander folgenden SYNC-Nachrichten. Das reale Zeitintervall zwischen zwei SYNC-Nachrichten kann daher im worst case um die Länge eines SPS-Zyklus schwanken:

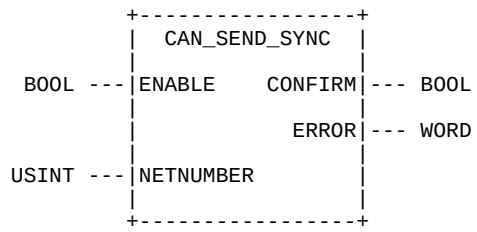
$T_{sync\ [worst\ case]} = SYNC\_TIME + T_{SPS\text{-}Zyklus}$

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2) und kann nur alternativ zum Funktionsbaustein CAN_SEND_SYNC (siehe Abschnitt 4.5.9) verwendet werden.

4.5.9 Funktionsbaustein CAN_SEND_SYNC

FB zum Senden einer einzelnen SYNC-Nachricht.

Prototyp des Funktionsbausteines



Operandenbedeutung

ERROR	Errorcode entsprechend dem Datentyp "CIA405_CANOPEN_KERNEL_ERROR" (siehe Tabelle 8 im Abschnitt 4.1.5)
NETNUMBER	Netzwerknummer (Hinweis: falls die SPS nur ein CANopen-Interface unterstützt, kann das Setzen dieses Eingangs entfallen, da numerische Variablen gemäß IEC61131 bereits mit dem Initialwert 0 vorbelegt sind)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 4.1.3)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 4.1.3)

Beschreibung

Der Funktionsbaustein CAN_SEND_SYNC dient zum Generieren einzelner SYNC-Nachrichten unter vollständiger Steuerung des SPS-Programms. Bei jedem Aufruf des Bausteins mit auf TRUE gesetztem Eingang ENABLE wird eine SYNC-Nachricht gesendet. Durch diese gezielte Einflussnahme ist es möglich, SYNC-Nachrichten nur dann zu generieren, wenn sich wirklich relevante Daten geändert haben (z.B. in Verbindung mit den Funktionsbausteinen für PDOs und CAN Layer 2 Nachrichten CAN_PDO_READ8 bzw. CAN_PDO_WRITE8, siehe Abschnitt 4.3).

Dieser Funktionsbaustein steht nur auf einer Steuerung im Modus "SPS mit CANopen-Master" zur Verfügung (siehe Abschnitt 4.1.2) und kann nur alternativ zum Funktionsbaustein CAN_ENABLE_CYCLIC_SYNC (siehe Abschnitt 4.5.8) verwendet werden.

4.6 Beispielprojekt für CANopen-Funktionsbausteine

Das in der *OpenPCS*-Standardinstallation enthaltene Beispielprojekt *"CopDemo"* zeigt die Anwendung von verschiedenen CANopen-Funktionsbausteinen. Es beinhaltet zwei Programme mit identischer Funktionalität, wobei aber ein Programm den Datenaustausch mittels PDO-Funktionsbausteinen (*"PDODEMO.POE"*) und das andere mit Hilfe von SDO-Funktionsbausteinen (*"SDODEMO.POE"*) realisiert.

Beide Demoprogramme des Beispielprojekts *"CopDemo"* lesen Prozessdaten von einem CANopen-IO-Modul (Knotenadresse 40H) ein und schreiben sie anschließend wieder auf das IO-Modul zurück. Damit folgen die Ausgänge des IO-Moduls dessen Eingangswerten. Das Demoprogramm für die PDO-Funktionsbausteine liest dazu das vom IO-Modul als Reaktion auf Änderungen an dessen Eingängen selbst generierte PDO ein und sendet die darin enthaltenen Prozessdaten wiederum per PDO an das IO-Modul zurück. Im Demoprogramm für die SDO-Funktionsbausteine liest die SPS zyklisch den aktuellen Wert der Eingänge aus dem entsprechenden Eintrag im Object-Dictionary des

Knotens aus und schreibt ihn anschließend ebenfalls per SDO wieder auf den OD-Eintrag für die Ausgänge zurück.

Den folgenden Betrachtungen liegt das Demoprogramm für SDO-Funktionsbausteine zugrunde (*"SDODEMO.POE"*). Die interne Ablaufsteuerung basiert auf der Variable *ProcStep*, die den jeweils aktuellen Prozessschritt angibt und zur Programmverzweigung dient. Ihr Wert wird jeweils nach dem erfolgreichen Abschluss eines Prozessschrittes incrementiert:

```
(* === SDO processing === *)
RunCycle:
LD    ProcStep
EQ    1
JMPC  StartSDORead
LD    ProcStep
EQ    2
JMPC  RunSDORead
```

Wie im Abschnitt 4.1.3 (Synchronisation zwischen CANopen-Funktionsbaustein und SPS-Programm) beschrieben, ist zum Starten eines SDO-Transfers eine steigende Flanke am Eingang ENABLE notwendig. Dazu wird die Instanz *FB_SDO_Read8* zunächst mit *ENABLE := FALSE* aufgerufen. Im anschließenden zweiten Aufruf ist dann *ENABLE := TRUE* gesetzt, zusätzlich werden die notwendigen CANopen-Kommunikationsparameter übergeben (*DEVICE*, *INDEX*, *SUBINDEX*):

```
(* Init SDO read *)
StartSDORead:
CAL   FB_SDO_Read8 (
      ENABLE := FALSE)

CAL   FB_SDO_Read8 (
      DEVICE := RX_DEVICE,
      INDEX := RxIndex,
      SUBINDEX := RxSubIndex,
      ENABLE := TRUE)

LD    ProcStep
ADD   1
ST    ProcStep
RET
```

Nach dem Starten des SDO-Transfers wird die zur internen Ablaufsteuerung verwendete Variable *ProcStep* incrementiert, so dass ab dem folgenden SPS-Zyklus der Prozessschritt zum Warten auf den Abschluss des SDO-Transfers gerufen wird. Diesen Prozessschritt wiederholt die SPS solange zyklisch, wie der Ausgangswert von *FB_SDO_Read8.CONFIRM* auf FALSE verweilt und der Ausgang *FB_SDO_Read8.ERROR* keine Fehler anzeigt (gleich Null ist):

```
(* Processing SDO read *)
RunSDORead:
CAL   FB_SDO_Read8 (
      ENABLE := TRUE)

LD    FB_SDO_Read8.CONFIRM
JMPC  RunSDOReadEnd
LD    FB_SDO_Read8.ERROR
EQ    0
RETC

RunSDOReadError:
LD    ProcStep  (* restart SDO transfer after error *)
SUB   1
ST    ProcStep
RET
```

Kehrt der Funktionsbaustein mit einem Ausgangswert ungleich Null an *FB_SDO_Read8.ERROR* zurück, so wurde der SDO-Transfer aufgrund eines Fehlers abgebrochen. Der Errorcode von *FB_SDO_Read8.ERROR* gibt dann dessen Ursache an (entsprechend Tabelle 8 im Abschnitt 4.1.5). Als Folge wird der Wert der Variable *ProcStep* wieder decrementiert, so dass im darauf folgenden SPS-Zyklus der Prozessschritt zum Starten des SDO-Transfers erneut ausgeführt wird.

Enthält der Ausgang *FB_SDO_Read8.CONFIRM* den Wert TRUE, zeigt dies den erfolgreichen Abschluss des SDO-Transfers an. In diesem Fall wird die Programmausführung am Label *RunSDOReadEnd* fortgesetzt (siehe unten). Wie im Abschnitt 4.1.3 (Synchronisation zwischen CANopen-Funktionsbaustein und SPS-Programm) beschrieben, muss nun das SPS-Programm seinerseits den Abschluss des SDO-Transfers gegenüber der Netzwerkschicht quittieren. Dazu wird die Instanz *FB_SDO_Read8* nochmals mit *ENABLE := FALSE* aufgerufen. Dies bewirkt gleichzeitig, dass die Netzwerkschicht den SDO-Kanal wieder zur Nutzung durch andere Funktionsbausteine freigibt. Anschließend werden die gelesenen Daten lokal übernommen sowie die Variable *ProcStep* erneut incrementiert. Der SDO-Transfer ist damit endgültig beendet, ab dem nächsten SPS-Zyklus setzt die Steuerung ihre Programmausführung mit dem folgenden Prozessschritt fort.

```
RunSDOReadEnd:
CAL   FB_SDO_Read8 (
      ENABLE := FALSE)

      (* copy received data *)
LD    FB_SDO_Read8.DATALENGTH
ST    DataLength
LD    FB_SDO_Read8.DATA0
ST    Data[0]

LD    ProcStep
ADD   1
ST    ProcStep
RET
```

Ohne den letzten Aufruf mit *ENABLE := FALSE* bleibt der SDO-Kanal im Lock-Zustand und kann nur noch von der sperrenden Instanz - in diesem Fall *FB_SDO_Read8* - benutzt werden. Für das Demoprogramm *"SDODEMO.POE"* würde dies bedeuten, dass die nachfolgenden Aufruf der Instanz *FB_SDO_Write8* zum zurücksenden der Daten (hier nicht mehr aufgeführt) mit der Fehlermeldung TRANSFER_BUSY am Ausgang ERROR abbricht. Ein Senden der Daten wäre also nicht mehr möglich.

5 Konfiguration einer SPS mit CANopen-Master

5.1 Allgemeine Grundlagen zur Master-Konfiguration

Grundlage für die Konfiguration des CANopen-Masters selbst ist die DCF-Datei der Master-SPS. Das Einbinden der DCF-Datei in die SPS-Konfiguration beschreibt Abschnitt 3.3. In dieser DCF-Datei sind die nachfolgend beschriebenen Konfigurationsobjekte entsprechend zu parametrieren. Hierfür ist vorzugsweise der "SYSTEC CANopen Master Configurator" zu verwenden, der in der *OpenPCS*-Programmierumgebung über den Menüpunkt "Extras -> Tools -> CANopen Master Configurator" aufgerufen wird (siehe Bild 17). Der Master-Konfigurator selbst ist im Manual L-1109 "Software Manual CANopen Master Configurator" ausführlich beschrieben.

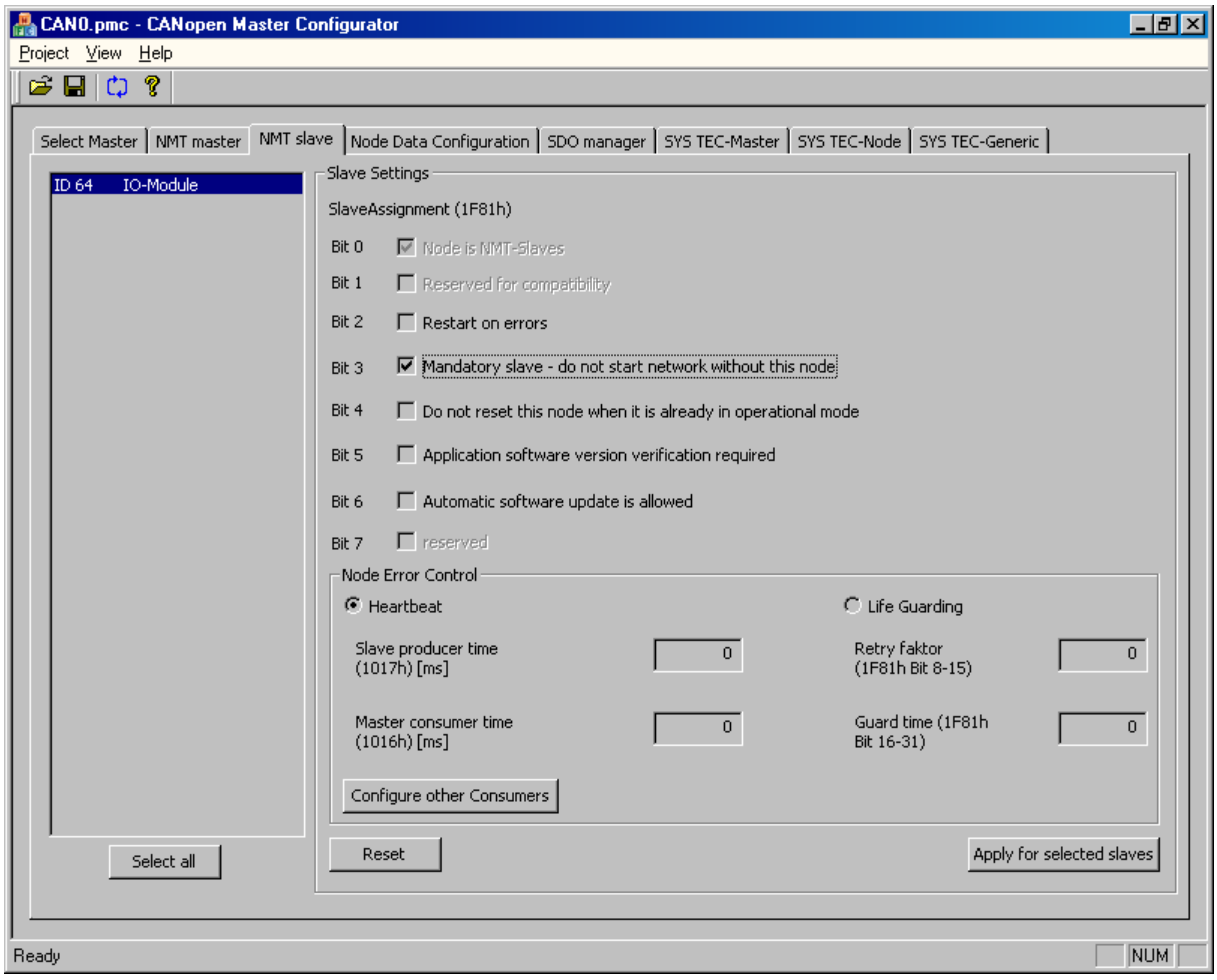


Bild 17: SYSTEC CANopen Master Configurator

5.2 Definition der Knotenliste

Die Definition der vom CANopen-Master zu konfigurierenden und zu überwachenden Knoten erfolgt anhand der Knotenliste im Objekteintrag 1F81H (NMT Slave Assignmet, siehe CiA DS302 V3). Die Konfiguration dieses Objektes sollte vorzugsweise in der Anchicht "NMT Slave" des "SYSTEC CANopen Master Configurator" erfolgen.

Ist im Object-Dictionary der Master-SPS das Objekt 1F81H nicht angelegt (d.h., ob das Objekt 1F81H ist nicht in der DCF-Datei der Master-SPS enthalten), dann ermittelt die Steuerung alle im Netz verfügbaren CANopen-Geräte durch einen Netzwerk-Scan, der standardmäßig den gesamten Bereich für alle Knotenadressen von 1 bis 127 überdeckt. Aufgrund applikationsspezifischer Erfordernisse kann es jedoch notwendig sein, den Bereich der zu scannenden Knotenadressen einzuschränken, so dass nicht alle verfügbaren Knotennummern zwischen 1 und 127 in den Netzwerkscan einbezogen werden. Zur Einstellung der Intervallgrenzen wird ein herstellerspezifisches Objekt mit dem Index 3001H angelegt. Dieses Objekt beinhaltet zwei Subeinträge zur Kennzeichnung der Untergrenze (Subindex 1) und der Obergrenze (Subindex 2). Den Aufbau des Objektes 3001H beschreibt Tabelle 13. Der Netzwerkscan erfolgt dann nur innerhalb dieses Intervalls (inkl. Unter- und Obergrenze). Eine Knotenüberwachung ist für Knoten mit einer Knotennummer außerhalb des Intervalls nicht möglich.

Tabelle 13: Objektverzeichnis-Eintrag für Intervallgrenzen des Netzwerkscan

Index	Sub-index	Name	Default-Wert	Typ	Attr.	Bedeutung
3001H	0	Anzahl der Einträge	2	u8	ro	
	1	Netzwerkscan Untergrenze	1	u8	rw	erste gültige Knotennummer für Netzwerkscan
	2	Netzwerkscan Untergrenze	127	u8	rw	letzte gültige Knotennummer für Netzwerkscan

Ist das Objekt 3001H im Objektverzeichnis nicht vorhanden, gelten die angegebenen Standardwerte.

5.3 Konfiguration der COBID für Nodestate-Nachrichten

Grundlage für den CANopen-Funktionsbaustein *CAN_GET_STATE*, mit dessen Hilfe ein SPS-Programm den aktuellen Status eins Netzwerkknotens bestimmen kann, ist die Knotenüberwachung durch Heartbeat bzw. Lifeguarding. Da diese Knotenüberwachung jedoch nur auf der Master-SPS erfolgt, sind die aktuellen Stati der Netzwerkknoten allen anderen Steuerungen zunächst unbekannt. Daher sendet eine Master-SPS beim Erkennen der Änderung eines Knotenstatus eine Broadcast-Nachricht mit dem aktuellen Knotenstatus an alle Steuerungen ohne CANopen-Master. Damit ist der neue Status auch auf diesen Steuerungen bekannt und kann dort ebenfalls durch Aufruf des CANopen-Funktionsbausteines *CAN_GET_STATE* abgefragt werden. Zur Vermeidung von Konflikten bei der Einbindung von fremden CANopen-Geräten ist die Default-COBID über das Objektverzeichnis konfigurierbar (siehe Tabelle 14). Bei einer Änderung der Default-COBID ist zu beachten, dass diese konsistent für alle betreffenden Knoten erfolgen muss.

Tabelle 14: Objektverzeichnis-Eintrag für COBID der Nodestate-Nachrichten

Index	Sub-index	Name	Default-Wert	Typ	Attr.	Bedeutung
3002H	0	Anzahl der Einträge	3	u8	ro	
	1	COBID Nodestate-Message	50H	u32	rw	COBID der Nodestate-Nachricht, die von der Master-SPS beim Statuswechsel eines überwachten Knotens gesendet wird

Ist das Objekt 3002H im Objektverzeichnis nicht vorhanden, gelten die angegebenen Standardwerte.

5.4 Definition von Wartezeiten

Beim Starten des SPS-Programms sendet die Master-SPS das NMT-Kommando "Enter Pre-Operational State", gefolgt von "Start Remote Node" für alle Knoten (Knotenadresse = 0). Dadurch werden die CANopen-Knoten veranlasst, einmalig ihre PDOs zu senden. Anschließend wartet die SPS bis die PDOs verarbeitet und die empfangenen Werte in Netzwerkabbild hinterlegt wurden, bevor das SPS-Programm gestartet wird. Dadurch ist die Anfangsinitialisierung der Netzwerkvariablen sicher gestellt. Die entsprechenden Wartezeiten können über das Objekt 3003H im Object-Dictionary der Master-SPS konfiguriert werden (siehe Tabelle 15)

Tabelle 15: Objektverzeichniseinträge zur Definition von Wartezeiten

Index	Sub-index	Name	Default-Wert	Typ	Attr.	Bedeutung
3003H	0	Anzahl der Einträge	1	u8	ro	
	1	Boot Network Delay	50	u16	rw	Wartezeit in [ms] zwischen dem Senden der NMT-Kommandos *EnterPreOperational* und *EnterOperational* beim Starten des SPS-Programms
	2	PLC Start Delay	100	U16	rw	Wartezeit in [ms] zwischen dem NMT-Kommando *EnterOperational* und dem Starten des SPS-Programms, um die Initial-PDOs zu verarbeiten und die empfangenen Werte im Netzwerkabbild zu hinterlegen

Ist das Objekt 3003H im Objektverzeichnis nicht vorhanden, gelten die angegebenen Standardwerte.

5.5 Konfiguration von Heartbeat/Lifeguarding der CANopen Geräte

Der CANopen-Master verwendet Heartbeat oder Lifeguarding zur Überwachung der CANopen-Geräte. Primär wird die Knotenüberwachung durch die entsprechenden Einträge in der DCF-Datei des jeweiligen Knotens bestimmt. Ist für den betreffenden Knoten jedoch keine DCF-Datei verfügbar, erfolgt die Auswahl der Überwachungsmethode sowie die Konfiguration der Überwachungszeit über die DCF-Datei der SPS (die im Master-Modus betrieben werden muss). Ist in der DCF-Datei weder Heartbeat noch Lifeguarding als Überwachungsmethoden konfiguriert, so verwendet der Master die optimale Methode (d.h. Heartbeat). Steht Heartbeat für das Gerät nicht zur Verfügung, dann wird Lifeguarding verwendet.

Konfiguration von Heartbeat:

Die Konfiguration von Heartbeat erfolgt mit dem Objekteintrag 1016H (Heartbeat-Consumer, siehe auch CiA DS301 V4.x). Für jedes zu überwachende Gerät muss ein Eintrag in einem freien Subindex erfolgen. Der Subindex kann frei gewählt werden. Der Eintrag ist folgendermaßen aufgebaut.

Datentyp: UNSIGNED32

Tabelle 16: Konfiguration von Heartbeat

Bits	**31-24**	**23-16**	**15-0**
Value	Reserved (00H)	Node-ID	Heartbeat Time
Encoded as	-	UNSIGNED8	UNSIGNED16
Example, Subindex 1	00h	40H	0BB8H

Im angeführten Beispiel wird für das Gerät mit der Knotenadresse 40H der Heartbeat-Consumer auf 3000ms eingestellt. Der Heartbeat-Producer des Gerätes selbst wird mit einem Drittel des Wertes (im Beispiel 1000ms) konfiguriert. Das bedeutet, dass die Master-SPS für das Gerät den Wert 1000 im Objekteintrag 1017H hinterlegt.

Den Ablauf bei der Konfiguration von Heartbeat verdeutlicht Bild 18.

Konfiguration von Lifeguarding:

Die Konfiguration von Lifeguarding erfolgt mit dem Objekteintrag 1F81H (NMT Slave Assignmet, siehe CiA DS302 V3). Für jedes zu überwachende Gerät muss der Eintrag in den Subindex erfolgen, der der Knotenadresse des Gerätes entspricht. Dieser Eintrag ist folgendermaßen aufgebaut:

Datentyp: UNSIGNED32

Tabelle 17: Konfiguration von Lifeguarding

Bits	**31-16**	**15-8**	**7-0**
Value	Guard Time	Retry Factor	Not used
Encoded as	UNSIGNED16	UNSIGNED8	UNSIGNED8
Example, Subindex 41H	05DCH	41H	0H

Im angeführten Beispiel wird für das Gerät mit der Knotenadresse 41H Lifeguarding mit einer Zeit von 1500ms eingestellt. Der Wiederholfaktor für das Gerät wird mit 5 festgelegt. Die Master-SPS stellt im Gerät folgende Werte ein:

Guard Time	Objekteintrag 100CH:	1500ms
Lifetime Factor	Objekteintrag 100DH:	5

Der Abstand zwischen zwei Anforderungen des Masters entspricht der Guard Time multipliziert mit einem Sicherheitsfaktor von 0.8. Für das angegebene Beispiel bedeutet dies eine Anforderung ca. alle 1200ms.

Den Ablauf bei der Konfiguration von Lifeguarding verdeutlicht Bild 19.

Konfiguration mit Defaultwerten:

Wurde für ein CANopen-Gerät in der DCF-Datei weder Heartbeat noch Lifeguarding konfiguriert, dann verwendet die Master-SPS Standardwerte für die Knotenüberwachung. Die Konfiguration läuft in diesem Fall nach folgender Strategie ab:

1. Unterstützt das Gerät Heartbeat?
   Zugriff auf den Objekteintrag 1017H des Gerätes: Wenn der Objekteintrag vorhanden ist, dann werden der Heartbeat-Producer des Gerätes (Objekt 1017H) auf 1000ms und der Heartbeat-Consumer der Master-SPS (Objekt 1016H) auf 3000ms eingestellt.

2. Unterstützt das Gerät kein Heartbeat aber Lifeguarding, dann wird im Gerät eine Lifetime (Objekt 100CH) von 1000ms und eine Lifetimefaktor (Objekt 100DH) von 3 eingestellt. Diese Parameter werden in der Master-SPS im Objekteintrag 1F81H für das Gerät hinterlegt.

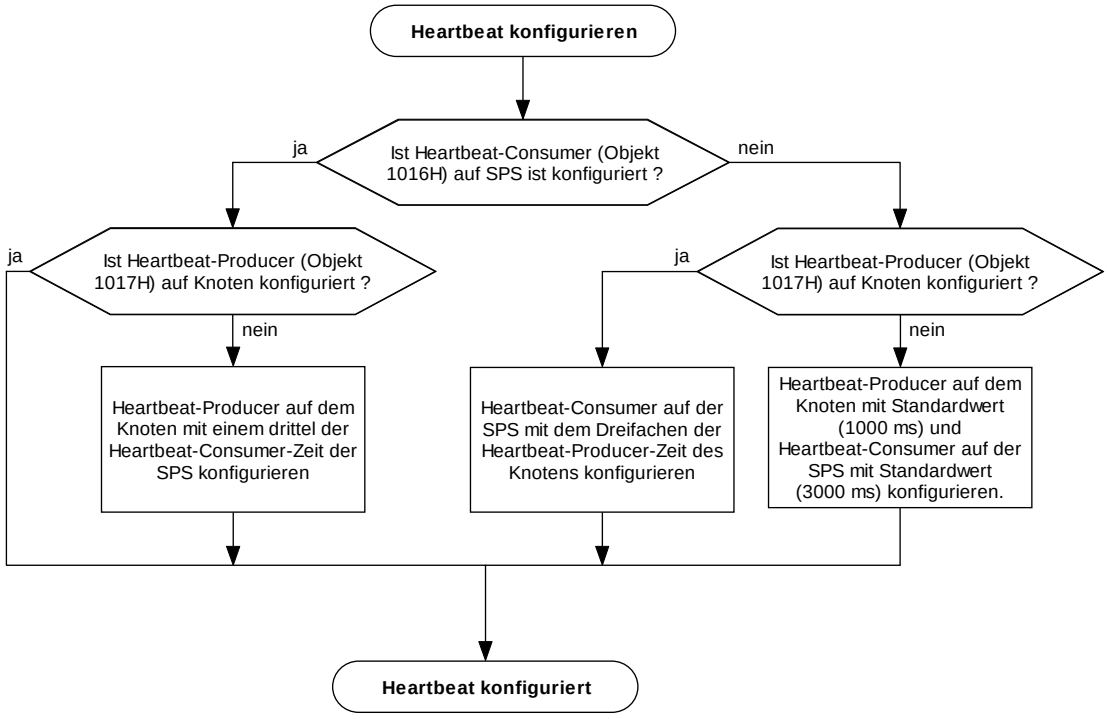


Bild 18: Ablauf der Konfiguration von Heartbeat

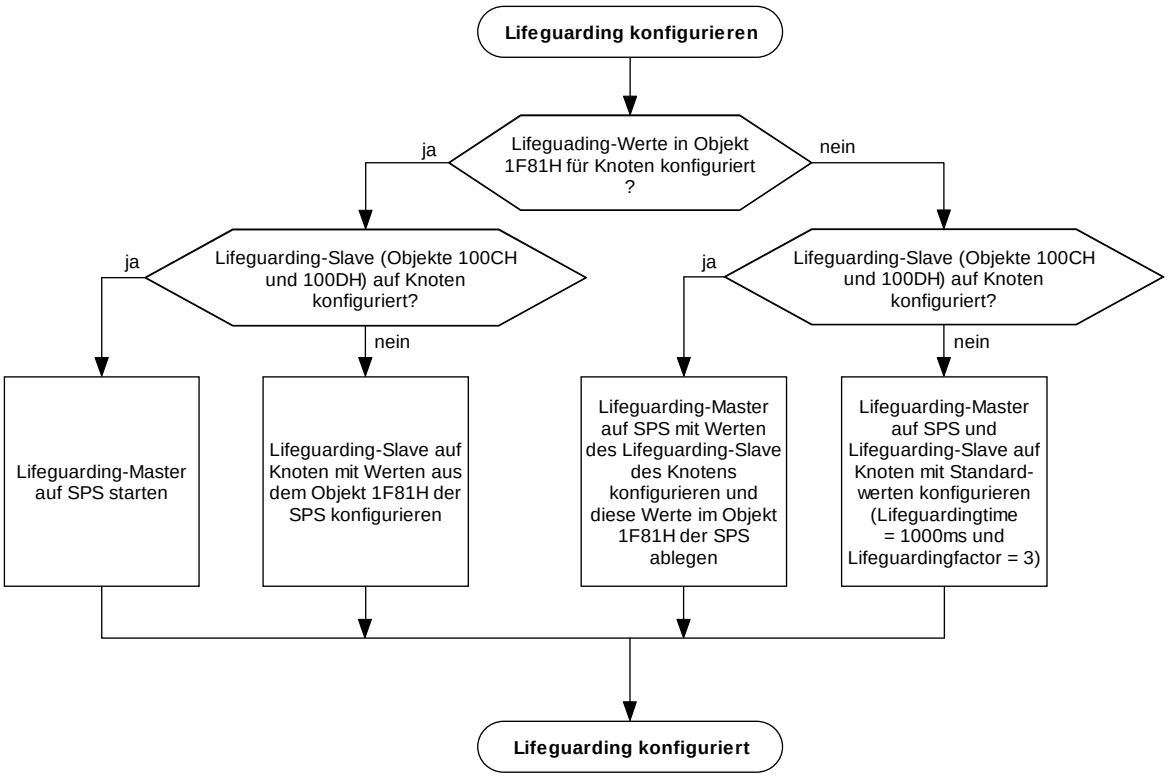


Bild 19: Ablauf der Konfiguration von Lifeguarding

Teil 2

CAN Layer 2

(Low Level Protokoll)

6 IEC61131-Funktionsbausteine für CAN Layer 2

6.1 Allgemeine Grundlagen für CAN Layer 2 Funktionsbausteine

Innerhalb der IEC61131-3 stehen verschiedene herstellerspezifische Funktionsbausteine für den direkten Zugriff auf die CAN-Schnittstelle Verfügung. Dadurch können CAN-Telegramme unmittelbar durch das SPS-Programm erstellt bzw. verarbeitet werden. Weiterhin unterstützen diese Funktionsbausteine CAN-Nachrichten im Extended-Frame-Format (29 Bit CAN Identifier gemäß CAN 2.0B). Die CAN Layer 2 Funktionsbausteine ermöglichen der SPS einen flexiblen Datenaustausch mit beliebigen, nicht CANopen-fähigen Geräten.

Hinweis: CAN Layer 2 Funktionsbausteine können nicht simultan mit CANopen-Diensten auf demselben CAN-Interface verwendet werden. CAN Layer 2 Funktionsbausteine sind nur nutzbar, wenn zuvor die CANopen-Funktionalität für die betreffende CAN-Schnittstelle deaktiviert wurde (siehe Abschnitt 6.2.1).

6.1.1 Übersicht der CAN Layer 2 Funktionsbausteine

Tabelle 18 zeigt eine Übersicht der CAN Layer 2 Funktionsbausteine für die IEC61131-3. Alle Bausteine sind als herstellerspezifische Funktionsbausteine realisiert und somit Bestandteil der Firmware einer SPS. Abhängig von dem in der Steuerung eingesetzten CAN-Controller sind nicht in jedem Fall alle Funktionalitäten voll verfügbar. Dies betrifft insbesondere die Filterung von CAN-Nachrichten (REGISTER_CANID / UNREGISTER_CANID) sowie die Verwendung von RTR-Frames. Verfügbarkeit bzw. Einschränkungen bestimmter Funktionalitäten sind im System Manual der jeweiligen Steuerung aufgeführt.

Tabelle 18: Übersicht der CAN Layer 2 Funktionsbausteine für IEC61131-3

Funktionsblock	Bedeutung	Abschn.
CANL2_INIT	Initialisierung der CAN-Schnittstelle (Setzt Deaktivierung der CANopen-Funktionalität für diese Schnittstelle voraus)	6.2.1
CANL2_SHUTDOWN	Deaktivierung der CAN-Schnittstelle	6.2.2
CANL2_RESET	Rücksetzen der CAN-Schnittstelle	6.2.3
CANL2_GET_STATUS	Abfrage Status der CAN-Schnittstelle	6.2.4
CANL2_DEFINE_CANID	Definieren eines CAN-Objektes für den Datenaustausch	6.2.5
CANL2_DEFINE_CANID_RANGE	Definieren eines Bereiches von CAN-Objekten für den Datenaustausch	6.2.6
CANL2_UNDEFINE_CANID	Verwerfen eines zuvor definierten CAN-Objektes	6.2.7
CANL2_UNDEFINE_CANID_RANGE	Verwerfen eines zuvor definierten Bereiches von CAN-Objekten	6.2.8
CANL2_MESSAGE_READ8	empfangene CAN-Nachricht auslesen, Daten werden an FB-Ausgängen übergeben	6.2.9
CANL2_MESSAGE_READ_BIN	empfangene CAN-Nachricht auslesen, Daten werden in das über Pointer adressierte Objekt geschrieben	6.2.10
CANL2_MESSAGE_WRITE8	CAN-Nachricht senden, Daten werden an FB-Eingängen übergeben	6.2.11

CANL2_MESSAGE_WRITE_BIN	CAN-Nachricht senden, Daten werden aus dem über Pointer adressierten Objekt gelesen	6.2.12
CANL2_MESSAGE_UPDATE8	Daten einer RTR-Nachricht aktualisieren, Daten werden an FB-Eingängen übergeben	6.2.13
CANL2_MESSAGE_UPDATE_BIN	Daten einer RTR-Nachricht aktualisieren, Daten werden aus dem über Pointer adressierten Objekt gelesen	6.2.14

6.1.2 Synchronisation zwischen CAN Layer 2 Funktionsbaustein und SPS-Programm

Die Prozesssynchronisation zwischen CAN-Interface und SPS-Programm erfolgt mit Hilfe der Signale ENABLE und CONFIRM der Funktionsbausteine. Die Bedeutung der Signale ENABLE und CONFIRM sowie der Ablauf der Synchronisation sind identisch zu der im Abschnitt 4.1.3 beschriebenen Synchronisation für CANopen-Funktionsbausteine (Details siehe Abschnitt 4.1.3).

6.1.3 CAN Layer 2 spezifische Konstanten

Zur Kennzeichnung von Fehlerzuständen wird der Datentyp "CANL2_ERROR" verwendet. Hier sind die Fehlerzustände zusammengefasst, die innerhalb des CAN-Interfaces einer SPS auftreten können. Diese Errorcodes werden von verschiedenen Funktionsbausteinen als Ausgabeparameter ERROR verwendet. Tabelle 19 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Errorcodes auf.

Tabelle 19: Konstanten für Datentyp "CANL2_ERROR"

Konstante		Errorcode
16#00	(= 00 dez)	NO_ERROR
16#01	(= 01 dez)	OTHER_ERROR
16#02	(= 02 dez)	INVALID_NETNUMBER
16#03	(= 03 dez)	INVALID_PARAMETER
16#04	(= 04 dez)	NO_MESSAGE
16#05	(= 05 dez)	UNSUPPORTED_BITRATE
16#06	(= 06 dez)	INIT_FAILED
16#07	(= 07 dez)	DEVICE_BUSY
16#08	(= 08 dez)	TX_BUFFER_OVERRUN
16#09	(= 09 dez)	NO_FREE_CHANNEL
16#0A	(= 10 dez)	COBID_ALREADY_REGISTERED
16#0B	(= 11 dez)	POINTER_TYPE_NOT_SUPPORTED

Der Datentyp "CANL2_BUS_STATUS" dient zur Kennzeichnung des Status, in dem sich eine CAN-Schnittstelle befindet. Tabelle 20 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Statuswerten auf.

Tabelle 20: Konstanten für Datentyp "CANL2_BUS_STATUS"

Konstante		Statuswert
16#00	(= 00 dez)	BUS_STATE_OK
16#01	(= 01 dez)	BUS_STATE_WARNING_LIMIT
16#02	(= 02 dez)	BUS_STATE_ERROR_PASSIVE
16#03	(= 03 dez)	BUS_STATE_BUS_OFF

Der Datentyp "CANL2_CDRV_STATUS" dient zur Signalisierung von Fehler- und Statuszuständen innerhalb des CAN-Treibers. Jedem Ereignis ist dabei ein eigenes Bit innerhalb der Statusmaske zugeordnet. Zu einem Zeitpunkt können mehrere Ereignisse parallel auftreten, in diesem Fall sind mehrere Bits innerhalb der Statusmaske gleichzeitig gesetzt (ODER-Verknüpfung). Tabelle 21 listet die Zuordnung der verwendeten numerischen Konstanten zu den entsprechenden Statuswerten auf.

Tabelle 21: Konstanten für Datentyp "CANL2_CDRV_STATUS"

Konstante		Statuswert
16#0000		CDRV_STATE_OK
16#0001	(Bit 0)	CDRV_STATE_WARNING_LIMIT_SET
16#0002	(Bit 1)	CDRV_STATE_WARNING_LIMIT_RESET
16#0004	(Bit 2)	CDRV_STATE_ERROR_PASSIVE_SET
16#0008	(Bit 3)	CDRV_STATE_ERROR_PASSIVE_RESET
16#0010	(Bit 4)	CDRV_STATE_BUS_OFF
16#0020	(Bit 5)	CDRV_STATE_OVERRUN (=Hardware Overrun)
16#0040	(Bit 6)	CDRV_STATE_STUFF_ERROR
16#0080	(Bit 7)	CDRV_STATE_FORM_ERROR
16#0100	(Bit 8)	CDRV_STATE_ACK_ERROR
16#0200	(Bit 9)	CDRV_STATE_CRC_ERROR
16#0400	(Bit 10)	CDRV_STATE_RX_BUFF_HIGH_OVERRUN
16#0800	(Bit 11)	CDRV_STATE_RESERVE1
16#1000	(Bit 12)	CDRV_STATE_RX_BUFF_LOW_OVERRUN
16#2000	(Bit 13)	CDRV_STATE_RESERVE2
16#4000	(Bit 14)	CDRV_STATE_BUS_OFF_RESET
16#8000	(Bit 15)	CDRV_STATE_FIRST_BUS_CONTACT

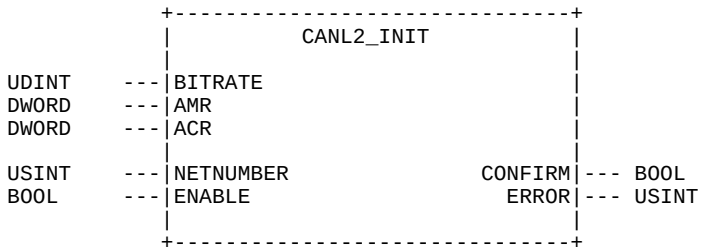
6.2 Funktionsbausteine für CAN Layer 2

Die Funktionsbausteine für den Zugriff auf den lokalen CANopen-Kernel der eigenen SPS ermöglichen die Abfrage der Knotenadresse sowie des Status der Netzwerkschicht. Diese Funktionsbausteine erfordern keine Kommunikation mit anderen Knoten.

6.2.1 Funktionsbaustein CANL2_INIT

FB zum Initialisieren der CAN-Schnittstelle.

Prototyp des Funktionsbausteines



Operandenbedeutung

BITRATE	Bitrate in Bit/s, z.B.: BITRATE := 125000 125 kBit/s BITRATE := 250000 250 kBit/s BITRATE := 1000000 1 MBit/s
AMR	Konfigurationswert für Acceptance Mask Register (AMR) des CAN-Controllers, ermöglicht hardwareseitige Filterung von CAN-Identifiern im CAN-Controller (Standardwert zur Verarbeitung aller CAN-Nachrichten: AMR := 16#FFFFFFFF)
ACR	Konfigurationswert für Acceptance Code Register (ACR) des CAN-Controllers, ermöglicht hardwareseitige Filterung von CAN-Identifiern im CAN-Controller (Standardwert zur Verarbeitung aller CAN-Nachrichten: ACR := 16#00000000)
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_INIT dient zur Initialisierung der CAN-Schnittstelle. **Dies setzt die Deaktivierung der CANopen-Funktionalität für diese Schnittstelle voraus.** Das Vorgehen hierzu ist im System Manual der jeweiligen Steuerung beschrieben. Sofern die SPS die Konfiguration über ein WEB-Frontend unterstützt, kann hier in der Regel der *"Enable State"* der betreffenden Schnittstelle auf *"Disabled"* gesetzt werden. Dadurch wird die Schnittstelle nicht mehr automatisch für CANopen initialisiert und steht zur freien Verwendung durch das SPS-Programm zur Verfügung.

Am Eingang BITRATE ist der Wert für Bitrate in Bit/s anzugeben, also beispielsweise *125000* für 125 kBit/s. Die Werte der beiden Eingänge AMR (Acceptance Mask Register) und ACR (Acceptance Code Register) werden unmittelbar in das AMR bzw. ACR Register des CAN-Controllers

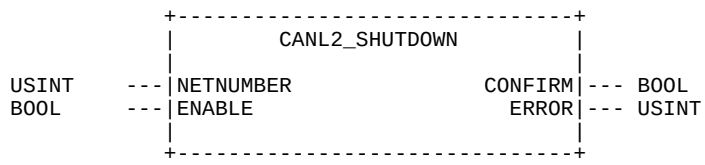
übernommen. Aus der Kombination beider Werte resultiert einen Empfangsfilter, das nur von CAN-Nachrichten passiert werden kann, deren CAN-Identifier die Filterkriterien erfüllen. Durch entsprechende Festlegung von AMR und ACR lassen sich alle nicht relevanten CAN-Nachrichten bereits hardwaremäßig im CAN-Controller selbst vom Empfang ausschließen. Dies reduziert zum einen die CPU-Last der Steuerung und kann einem Überlaufen des Empfangspuffers vorbeugen. Die genaue Bedeutung von AMR und ACR sowie die damit realisierbaren Filterwerte sind dem Datenblatt des jeweiligen CAN-Controllers zu entnehmen. Im Regelfall werden die Filter jedoch so gesetzt, dass der CAN-Controller alle CAN-Nachrichten empfängt. Dazu sind AMR und ACR wie folgt zu belegen:

```
AMR := 16#FFFFFFFF;
ACR := 16#00000000;
```

6.2.2 Funktionsbaustein CANL2_SHUTDOWN

FB zum Deaktivieren der CAN-Schnittstelle.

Prototyp des Funktionsbausteines



Operandenbedeutung

NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

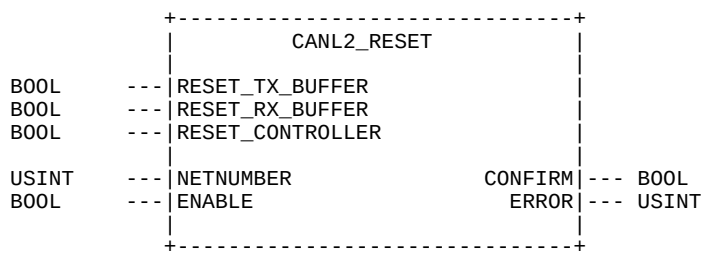
Beschreibung

Der Funktionsbaustein CANL2_SHUTDOWN dient zur Deaktivierung der CAN-Schnittstelle.

6.2.3 Funktionsbaustein CANL2_RESET

FB zum Rücksetzen der CAN-Schnittstelle.

Prototyp des Funktionsbausteines



Operandenbedeutung

RESET_TX_BUFFER	TRUE = Sendepuffer rücksetzen (alle Nachrichten verwerfen) FALSE = Sendepuffer nicht verändern
RESET_RX_BUFFER	TRUE = Empfangspuffer rücksetzen (alle Nachrichten verwerfen) FALSE = Empfangspuffer nicht verändern
RESET_CONTROLLER	TRUE = CAN-Controller rücksetzen und neu initialisieren FALSE = CAN-Controller nicht verändern
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

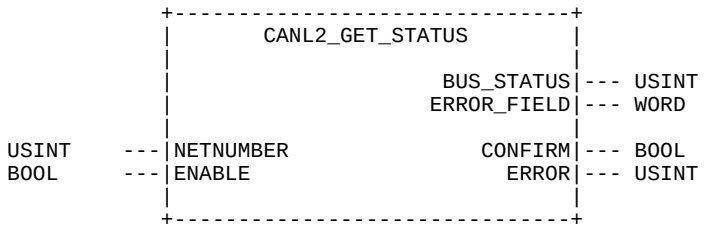
Beschreibung

Der Funktionsbaustein CANL2_RESET dient zur Rücksetzen der CAN-Schnittstelle. Über die Eingänge RESET_TX_BUFFER, RESET_RX_BUFFER und RESET_CONTROLLER kann das SPS-Programm steuern, welche Teile der Schnittstelle zurück zu setzen sind. Sind die Eingänge RESET_TX_BUFFER, RESET_RX_BUFFER auf TRUE gesetzt, werden die entsprechenden Puffer zurück gesetzt und somit alle noch darin enthaltenen Nachrichten verworfen. Ist der Eingang RESET_CONTROLLER auf TRUE gesetzt, führt dies zum Rücksetzen des CAN-Controllers mit anschließender Neuinitialisierung.

6.2.4 Funktionsbaustein CANL2_GET_STATUS

FB zum Abfragen des Status der CAN-Schnittstelle.

Prototyp des Funktionsbausteines



Operandenbedeutung

BUS_STATUS	Bus-Status entsprechend dem Datentyp "CANL2_BUS_STATUS" (siehe Tabelle 20 im Abschnitt 6.1.3)
ERROR_FIELD	CAN-Treiber-Status entsprechend dem Datentyp "CANL2_CDRV_STATUS" (siehe Tabelle 21 im Abschnitt 6.1.3)
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)

ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

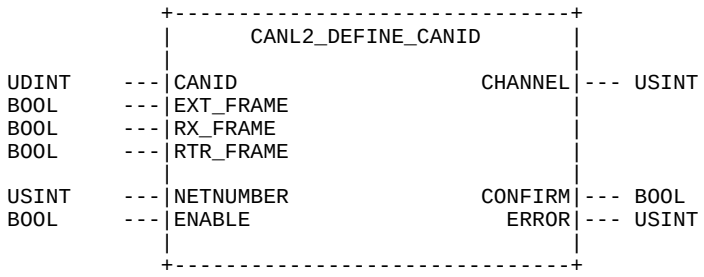
Beschreibung

Der Funktionsbaustein CANL2_GET_STATUS dient zum Abfragen des Status der CAN-Schnittstelle. Der Ausgang BUS_STATUS gibt Auskunft über den Bus-Status der Steuerung. Der Ausgang ERROR_FIELD informiert über aktuelle Status-Ereignisse innerhalb des CAN-Treibers. Dabei ist jedem Ereignis ein eigenes Bit innerhalb der Statusmaske am Ausgang ERROR_FIELD zugeordnet. Zu einem Zeitpunkt können mehrere Ereignisse parallel auftreten, in diesem Fall sind mehrere Bits innerhalb der Statusmaske gleichzeitig gesetzt (ODER-Verknüpfung).

6.2.5 Funktionsbaustein CANL2_DEFINE_CANID

FB zum Definieren eines CAN-Objektes für den Datenaustausch.

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier, für den ein CAN-Objekt zu definieren ist
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RX_FRAME	TRUE = das CAN-Objekt wird für den Empfang verwendet FALSE = das CAN-Objekt wird zum Senden verwendet
RTR_FRAME	TRUE = CAN-Objekt wird für RTR verwendet FALSE = CAN-Objekt wird nicht für RTR verwendet
CHANNEL	liefert die vom CAN-Treiber zugewiesene Kanal-Nummer für das CAN-Objekt zurück
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_DEFINE_CANID dient zum Definieren eines CAN-Objektes für den Datenaustausch. Der Eingang EXT_FRAME gibt an, ob der am Eingang CANID übergebene CAN-Identifier als 11-Bit Identifier einer Standard-Nachricht (CAN 2.0A) oder als 29-Bit Identifier einer Extended-Nachricht (CAN 2.0B) zu interpretieren ist. Der Eingang RX_FRAME bestimmt, ob das Objekt zum Senden oder Empfangen von Daten verwendet wird.

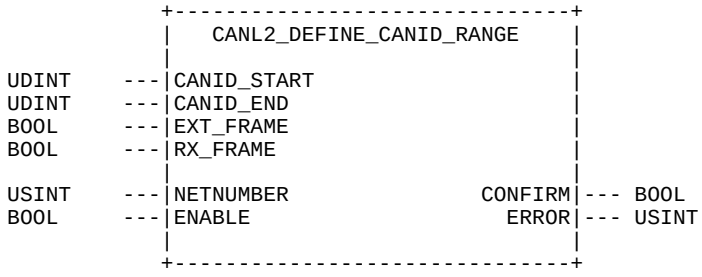
Über den Eingang RTR_FRAME kann ein CAN-Objekt als RTR-Objekt (Remote Transmission Request) kennzeichnet werden. Remote-Frames werden von einem Knoten nur dann gesendet, wenn er von einem Knoten dazu aufgefordert wurde.

Der Ausgang CHANNEL liefert die vom CAN-Treiber für das CAN-Objekt zugewiesene Kanal-Nummer. Die Kanal-Nummer ist insbesondere für RTR-Objekte von Bedeutung, da hier prinzipbedingt jede Nachricht in einem eigenen Kanal des CAN-Controllers verwaltet werden muss.

6.2.6 Funktionsbaustein CANL2_DEFINE_CANID_RANGE

FB zum Definieren eines Bereiches von CAN-Objekten für den Datenaustausch.

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID_START	erster (kleinster) CAN-Identifier des zu definierenden Bereiches von CAN-Objekten
CANID_END	letzter (größter) CAN-Identifier des zu definierenden Bereiches von CAN-Objekten
EXT_FRAME	TRUE = der CAN-Identifier-Bereich kennzeichnet Extended-Frames (29 Bit) FALSE = der CAN-Identifier-Bereich kennzeichnet Standard-Frames (11 Bit)
RX_FRAME	TRUE = die CAN-Objekte werden für den Empfang verwendet FALSE = die CAN-Objekte werden zum Senden verwendet
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

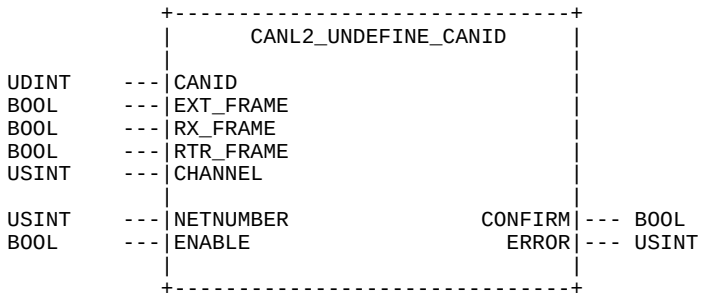
Der Funktionsbaustein CANL2_DEFINE_CANID_RANGE dient zum Definieren eines Bereiches von CAN-Objekten für den Datenaustausch. Der Eingang EXT_FRAME gibt an, ob der durch die Eingänge CANID_START und CANID_END definierte CAN-Identifier-Bereich als 11-Bit Identifier für Standard-Nachrichten (CAN 2.0A) oder als 29-Bit Identifier für Extended-Nachrichten (CAN 2.0B) zu interpretieren ist. Der Eingang RX_FRAME bestimmt, ob die Objekte zum Senden oder Empfangen von Daten verwendet werden.

Mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID_RANGE können keine RTR-Objekte (Remote Transmission Request) angelegt werden. Hierfür ist bei Bedarf der Funktionsbaustein CANL2_DEFINE_CANID zu verwendet und ggf. mehrfach aufzurufen.

6.2.7 Funktionsbaustein CANL2_UNDEFINE_CANID

FB zum Verwerfen eines zuvor definierten CAN-Objektes.

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID EXT_FRAME RX_FRAME RTR_FRAME	Hier sind dieselben Parameter anzugeben, die auch zuvor beim Definieren des CAN-Objektes mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID verwendet wurden.
CHANNEL	die vom Funktionsbaustein CANL2_DEFINE_CANID zurück gelieferte Kanal-Nummer für das CAN-Objekt zurück
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE CONFIRM	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2) Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

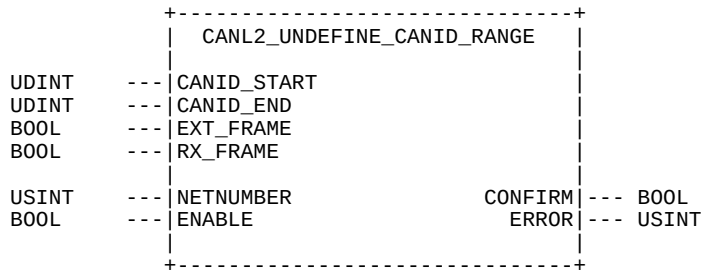
Der Funktionsbaustein CANL2_UNDEFINE_CANID dient zum Verwerfen eines zuvor mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID definierten CAN-Objektes.

Um das zu verwerfende CAN-Objekt eindeutig identifizieren zu können, sind an den Eingängen CANID, EXT_FRAME, RX_FRAME und RTR_FRAME dieselben Parameter anzugeben, die auch zuvor beim Definieren des CAN-Objektes mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID verwendet wurden.

6.2.8 Funktionsbaustein CANL2_UNDEFINE_CANID_RANGE

FB zum Verwerfen eines zuvor definierten Bereiches von CAN-Objekten.

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID_START CANID_END EXT_FRAME RX_FRAME	Hier sind dieselben Parameter anzugeben, die auch zuvor beim Definieren des Bereiches von CAN-Objekten mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID_RANGE verwendet wurden.
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

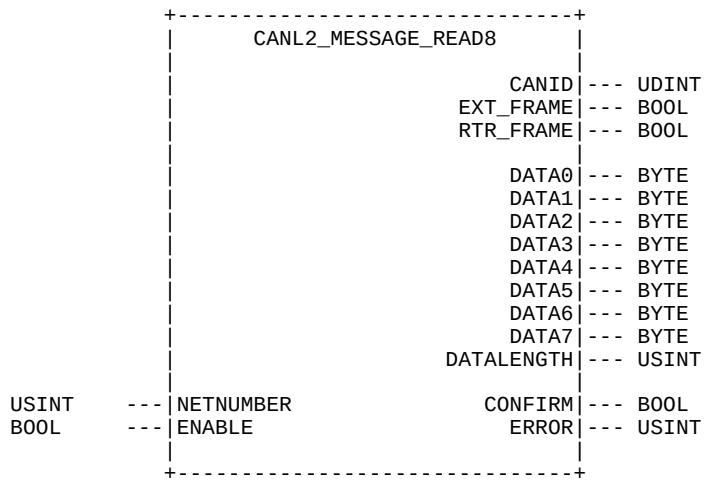
Der Funktionsbaustein CANL2_UNDEFINE_CANID_RANGE dient zum Verwerfen eines zuvor mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID RANGE definierten Bereiches von CAN-Objekten.

Um den Bereich zu verwerfender CAN-Objekte eindeutig identifizieren zu können, sind an den Eingängen CANID_START, CANID_END, EXT_FRAME und RX_FRAME dieselben Parameter anzugeben, die auch zuvor beim Definieren des CAN-Objektes mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID_RANGE verwendet wurden.

6.2.9 Funktionsbaustein CANL2_MESSAGE_READ8

FB zum Auslesen einer empfangenen CAN-Nachricht (Übergabe der Daten an FB-Ausgängen).

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier der empfangenen CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wurde als RTR-Frame übertragen FALSE = CAN-Objekt wurde nicht RTR-Frame übertragen
DATA0 - DATA7	Datenbytes der empfangenen CAN-Nachricht
DATALENGTH	Länge der empfangenen CAN-Nachricht
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_READ8 dient zum Auslesen einer empfangenen CAN-Nachricht aus dem Empfangspuffer der CAN-Schnittstelle. Die Übergabe der Daten erfolgt byteweise an den Ausgängen des Funktionsbausteins (Alternative: FB CANL2_MESSAGE_READ_BIN, siehe Abschnitt 6.2.10). Es können nur CAN-Nachrichten empfangen werden, für die zuvor mit Hilfe der Funktionsbausteine CANL2_DEFINE_CANID bzw. CANL2_DEFINE_CANID_RANGE entsprechende CAN-Objekte angelegt wurden.

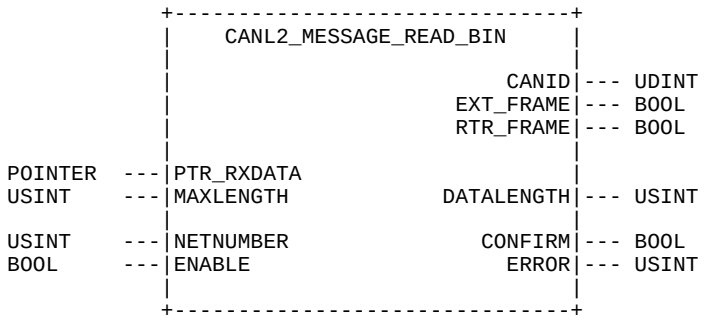
Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthalten die Elemente DATA0 bis DATA7 die einzelnen Bytes der empfangenen Nachricht. Der Ausgang DATALENGTH gibt dabei die Anzahl der gültigen Datenbytes (ab DATA0) an. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der CAN-Schnittstelle keine Nachrichten. Anhand von CONFIRM kann somit unterschieden werden, ob eine gültige Nachricht mit 0 Bytes Länge oder keine Nachricht empfangen wurde. Ist keine Nachricht verfügbar, wird dies

ebenfalls durch den Fehlercode NO_MESSAGE am Ausgang ERROR angezeigt (siehe Tabelle 19 im Abschnitt 6.1.3).

6.2.10 Funktionsbaustein CANL2_MESSAGE_READ_BIN

FB zum Auslesen einer empfangenen CAN-Nachricht (Übergabe der Daten in dem durch Pointer adressierten Objekt).

Prototyp des Funktionsbausteines



Operandenbedeutung

PTR_RXDATA	Adresse eines Objektes, in dem die Daten der empfangene CAN-Nachricht abgelegt werden
MAX_LENGTH	Begrenzung der Anzahl abzulegender Bytes, bei 0 wird intern die Größe des über PTR_RXDATA adressierten Objektes ermittelt und als Begrenzung der Anzahl abzulegender Bytes verwendet (es werden max. so viele Bytes abgelegt, wie das Objekt aufnehmen kann).
DATALENGTH	Anzahl der abgelegten Bytes
CANID	CAN-Identifier der empfangenen CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wurde als RTR-Frame übertragen FALSE = CAN-Objekt wurde nicht RTR-Frame übertragen
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_READ_BIN dient zum Auslesen einer empfangenen CAN-Nachricht aus dem Empfangspuffer der CAN-Schnittstelle. Die Daten der empfangenen Nachricht werden in dem über PTR_RXDATA adressierten Objekt abgelegt (Alternative: FB CANL2_MESSAGE_READ8, siehe Abschnitt 6.2.9). Es können nur CAN-Nachrichten empfangen werden, für die zuvor mit Hilfe der Funktionsbausteine CANL2_DEFINE_CANID bzw. CANL2_DEFINE_CANID_RANGE entsprechende CAN-Objekte angelegt wurden.

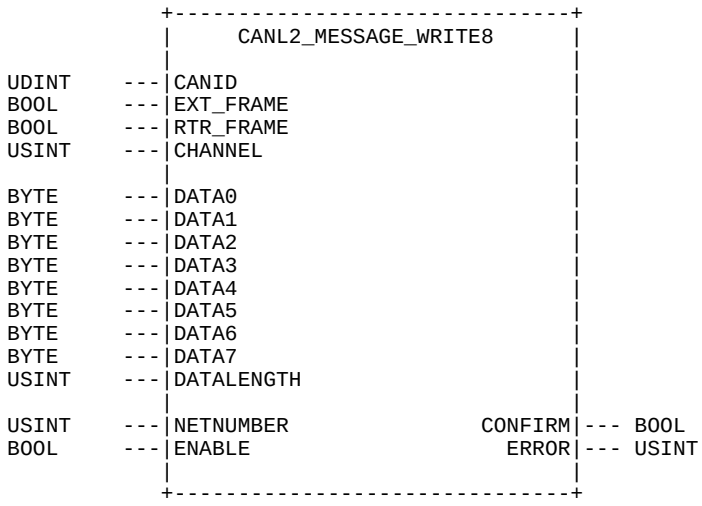
Mit Hilfe des Eingangs MAX_LENGTH lässt sich die Anzahl abzulegender Bytes begrenzen. Ist MAX_LENGTH auf 0 gesetzt, wird intern die Größe des über PTR_RXDATA adressierten Objektes ermittelt und als Begrenzung der Anzahl abzulegender Bytes verwendet. Es werden aber in jedem Fall stets nur maximal so viele Bytes abgelegt, wie das Objekt aufnehmen kann. Damit wird ein Überschreiben von Speicherbereichen anderer Objekte oder Variablen verhindert.

Ist bei der Rückkehr des Funktionsbausteines der Ausgang CONFIRM auf TRUE gesetzt, enthält das über PTR_RXDATA adressierte Objekt die Daten der empfangenen Nachricht. Der Ausgang DATALENGTH gibt dabei die Anzahl der gültigen Datenbytes an. Ist der Ausgang CONFIRM dagegen auf FALSE gesetzt, so enthält der Empfangspuffer der CAN-Schnittstelle keine Nachrichten. Anhand von CONFIRM kann somit unterschieden werden, ob eine gültige Nachricht mit 0 Bytes Länge oder keine Nachricht empfangen wurde. Ist keine Nachricht verfügbar, wird dies ebenfalls durch den Fehlercode NO_MESSAGE am Ausgang ERROR angezeigt (siehe Tabelle 19 im Abschnitt 6.1.3).

6.2.11 Funktionsbaustein CANL2_MESSAGE_WRITE8

FB zum Senden einer CAN-Nachricht (Übergabe der Daten an FB-Eingängen).

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier der zu sendenden CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wird als RTR-Frame übertragen FALSE = CAN-Objekt wird nicht RTR-Frame übertragen
CHANNEL	Wurde das CAN-Objekt mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID definiert, dann ist hier die von diesem FB zurück gelieferte Kanal-Nummer für das CAN-Objekt zu übergeben. Wurde das CAN-Objekt mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID_RANGE definiert, dann ist hier 0 zu übergeben (Hinweis: ein mit CANL2_DEFINE_CANID_RANGE definiertes CAN-Objekt kann nicht für RTR-Nachrichten verwendet werden).
DATA0 - DATA7	Datenbytes der zu sendenden CAN-Nachricht
DATALENGTH	Länge der zu sendenden CAN-Nachricht

NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_WRITE8 dient zum Senden einer CAN-Nachricht. Die Übergabe der Daten erfolgt byteweise an den Eingängen des Funktionsbausteins (Alternative: FB CANL2_MESSAGE_WRITE_BIN, siehe Abschnitt 6.2.12). Es können nur CAN-Nachrichten gesendet werden, für die zuvor mit Hilfe der Funktionsbausteine CANL2_DEFINE_CANID bzw. CANL2_DEFINE_CANID_RANGE entsprechende CAN-Objekte angelegt wurden.

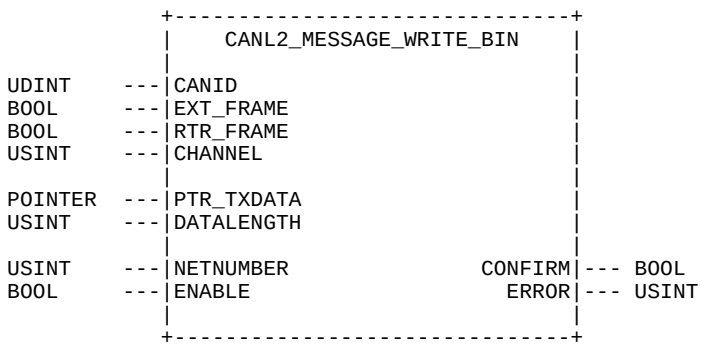
An den Elementen DATA0 bis DATA7 sind die einzelnen Bytes der zu sendenden Nachricht zu übergeben. Der Eingang DATALENGTH spezifiziert dabei die Anzahl der gültigen Datenbytes (ab DATA0).

Beim Aufruf des Funktionsbausteines CANL2_MESSAGE_WRITE8 wird die zu sendende Nachricht im Sendepuffer der CAN-Schnittstelle abgelegt. Tritt hierbei kein Fehler auf (Nachricht konnte korrekt im Sendepuffer hinterlegt werden), kehrt der Baustein mit auf TRUE gesetztem Ausgang CONFIRM zurück. Es erfolgt jedoch keine Rückmeldung zum SPS-Programm darüber, ob die Nachricht tatsächlich erfolgreich gesendet werden konnte.

6.2.12 Funktionsbaustein CANL2_MESSAGE_WRITE_BIN

FB zum Senden einer CAN-Nachricht (Übergabe der Daten in dem durch Pointer adressierten Objekt).

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier der zu sendenden CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wird als RTR-Frame übertragen FALSE = CAN-Objekt wird nicht RTR-Frame übertragen

CHANNEL	Wurde das CAN-Objekt mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID definiert, dann ist hier die von diesem FB zurück gelieferte Kanal-Nummer für das CAN-Objekt zu übergeben. Wurde das CAN-Objekt mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID_RANGE definiert, dann ist hier 0 zu übergeben (Hinweis: ein mit CANL2_DEFINE_CANID_RANGE definiertes CAN-Objekt kann nicht für RTR-Nachrichten verwendet werden).
PTR_TXDATA	Adresse eines Objektes, in dem die mit der CAN-Nachricht zu sendenden Daten enthalten sind
DATALENGTH	Anzahl der mit der CAN-Nachricht zu sendenden Bytes, bei 0 wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu sendender Bytes verwendet
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_WRITE_BIN dient zum Senden einer CAN-Nachricht. Die Daten der zu sendenden Nachricht werden in dem über PTR_TXDATA adressierten Objekt erwartet (Alternative: FB CANL2_MESSAGE_WRITE8, siehe Abschnitt 6.2.11). Es können nur CAN-Nachrichten gesendet werden, für die zuvor mit Hilfe der Funktionsbausteine CANL2_DEFINE_CANID bzw. CANL2_DEFINE_CANID_RANGE entsprechende CAN-Objekte angelegt wurden.

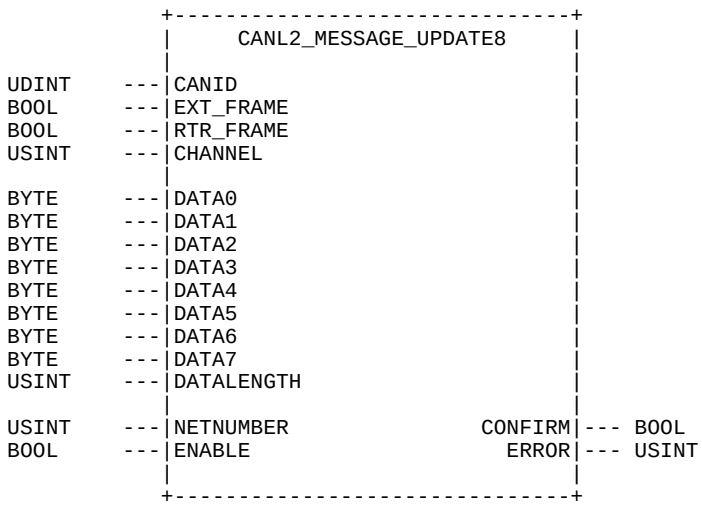
Die Daten der zu sendenden Nachricht werden dem über PTR_TXDATA adressierten Objekt entnommen. Mit Hilfe des Eingangs DATALENGTH lässt sich die Anzahl der zu sendenden Bytes begrenzen. Ist DATALENGTH auf 0 gesetzt, wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu sendender Bytes verwendet.

Beim Aufruf des Funktionsbausteines CANL2_MESSAGE_WRITE_BIN wird die zu sendende Nachricht im Sendepuffer der CAN-Schnittstelle abgelegt. Tritt hierbei kein Fehler auf (Nachricht konnte korrekt im Sendepuffer hinterlegt werden), kehrt der Baustein mit auf TRUE gesetztem Ausgang CONFIRM zurück. Es erfolgt jedoch keine Rückmeldung zum SPS-Programm darüber, ob die Nachricht tatsächlich erfolgreich gesendet werden konnte.

6.2.13 Funktionsbaustein CANL2_MESSAGE_UPDATE8

FB zum Aktualisieren von Daten einer RTR-Nachricht (Übergabe der Daten an FB-Eingängen).

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier der zu aktualisierenden CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wird als RTR-Frame übertragen FALSE = CAN-Objekt wird nicht RTR-Frame übertragen
CHANNEL	Kanalnummer, die bei der Definition des CAN-Objekts vom Funktionsbaustein CANL2_DEFINE_CANID zurück geliefert wurde (Hinweis: mit CANL2_DEFINE_CANID_RANGE definierte CAN-Objekte können nicht für RTR-Nachrichten verwendet werden)
DATA0 - DATA7	Datenbytes der zu aktualisierenden CAN-Nachricht
DATALENGTH	Länge der zu aktualisierenden CAN-Nachricht
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_UPDATE8 dient zum Aktualisieren von Daten einer RTR-Nachricht. Die Übergabe der Daten erfolgt byteweise an den Eingängen des Funktionsbausteins (Alternative: FB CANL2_MESSAGE_UPDATE_BIN, siehe Abschnitt 6.2.14). Es können nur CAN-Objekte aktualisiert werden, die zuvor mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID angelegt wurden.

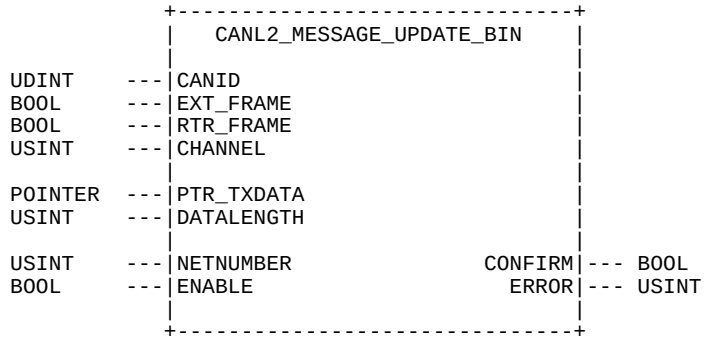
An den Elementen DATA0 bis DATA7 sind die einzelnen Bytes der zu aktualisierenden Nachricht zu übergeben. Der Eingang DATALENGTH spezifiziert dabei die Anzahl der gültigen Datenbytes (ab DATA0).

Beim Aufruf des Funktionsbausteines CANL2_MESSAGE_UPDATE8 werden nur die Daten der Nachricht im Sendepuffer des CAN-Controllers aktualisiert. Um die Nachricht selbst auf dem CAN-Bus zu übertragen, muss diese jedoch explizit von einem anderen Knoten per RTR-Frame angefordert werden.

6.2.14 Funktionsbaustein CANL2_MESSAGE_UPDATE_BIN

FB zum Aktualisieren von Daten einer RTR-Nachricht (Übergabe der Daten in dem durch Pointer adressierten Objekt).

Prototyp des Funktionsbausteines



Operandenbedeutung

CANID	CAN-Identifier der zu aktualisierenden CAN-Nachricht
EXT_FRAME	TRUE = der CAN-Identifier kennzeichnet einen Extended-Frame (29 Bit) FALSE = der CAN-Identifier kennzeichnet einen Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Objekt wird als RTR-Frame übertragen FALSE = CAN-Objekt wird nicht RTR-Frame übertragen
CHANNEL	Kanalnummer, die bei der Definition des CAN-Objekts vom Funktionsbaustein CANL2_DEFINE_CANID zurück geliefert wurde (Hinweis: mit CANL2_DEFINE_CANID_RANGE definierte CAN-Objekte können nicht für RTR-Nachrichten verwendet werden)
PTR_TXDATA	Adresse eines Objektes, in dem die zu aktualisierenden Daten der CAN-Nachricht enthalten sind
DATALENGTH	Anzahl der in der CAN-Nachricht zu aktualisierenden Bytes, bei 0 wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu aktualisierender Bytes verwendet
NETNUMBER	Netzwerknummer
ERROR	Errorcode entsprechend dem Datentyp "CANL2_ERROR" (siehe Tabelle 19 im Abschnitt 6.1.3)
ENABLE	Eingang zur Freigabe bzw. Sperrung des FB (siehe Abschnitt 6.1.2)
CONFIRM	Ausgang für Fertigmeldung durch den FB (siehe Abschnitt 6.1.2)

Beschreibung

Der Funktionsbaustein CANL2_MESSAGE_UPDATE_BIN dient zum Aktualisieren von Daten einer RTR-Nachricht. Die zu aktualisierenden Daten werden in dem über PTR_TXDATA adressierten Objekt erwartet (Alternative: FB CANL2_MESSAGE_UPDATE8, siehe Abschnitt 6.2.13). Es können nur CAN-Objekte aktualisiert werden, für die zuvor mit Hilfe des Funktionsbausteins CANL2_DEFINE_CANID angelegt wurden.

Die Daten der zu aktualisierenden Nachricht werden dem über PTR_TXDATA adressierten Objekt entnommen. Mit Hilfe des Eingangs DATALENGTH lässt sich die Anzahl der zu aktualisierenden Bytes begrenzen. Ist DATALENGTH auf 0 gesetzt, wird intern die Größe des über PTR_TXDATA adressierten Objektes ermittelt und als Anzahl zu aktualisierender Bytes verwendet.

Beim Aufruf des Funktionsbausteines CANL2_MESSAGE_UPDATE_BIN werden nur die Daten der Nachricht im Sendepuffer des CAN-Controllers aktualisiert. Um die Nachricht selbst auf dem CAN-Bus zu übertragen, muss diese jedoch explizit von einem anderen Knoten per RTR-Frame angefordert werden.

7 Index

Dokument:	**CAN / CANopen-Erweiterung für IEC 61131**
Dokumentnummer:	**L-1008-07, Mai 2011**

Wie würden Sie dieses Handbuch verbessern?

Haben Sie in diesem Handbuch Fehler entdeckt? Seite

Eingesandt von:

Kundennummer:

Name:

Firma:

Adresse:

Einsenden an:

SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz, Germany
Fax : +49 (3661) 6279-99