Turmgasse 4
89073 Ulm

Tel. 0731 / 97057 - 0
Fax 0731 / 97057-39

email info@frenzel-berg.de

frenzel + berg electronic

CANopen guideline

1 Einleitung

CAN steht für ‚Controller Area Network' und wurde von den Firmen Bosch und Intel ursprünglich als Bussystem für Fahrzeuge (Autobus) entwickelt. Inzwischen hat sich CAN aber auch im Bereich der Automatisierungstechnik als Feldbus bewährt. Wie praktisch alle Feldbusse setzt auch CAN auf dem OSI 7-Schichtmodel auf. CAN ist nur für die OSI Schichten 1 und 2 in der ISO 11898 genormt. Die "fehlende" Anwenderschicht wird durch die auf CAN aufsetzenden Schicht 7 Protokolle DeviceNet, SDS, CAL und die CANopen Profile definiert. Der CAN-Bus ist ein serielles Bussystem, bei dem alle angeschlossenen Stationen gleichberechtigt sind, d. h. jedes Steuergerät (CAN-Knoten) kann senden und empfangen. Einfach ausgedrückt können sich die angeschlossenen Steuergeräte über die Leitungen „unterhalten" und gegenseitig Informationen austauschen. Durch die lineare Struktur des Netzwerks ist das Bussystem bei Ausfall einer Station für alle anderen Stationen weiterhin voll verfügbar.

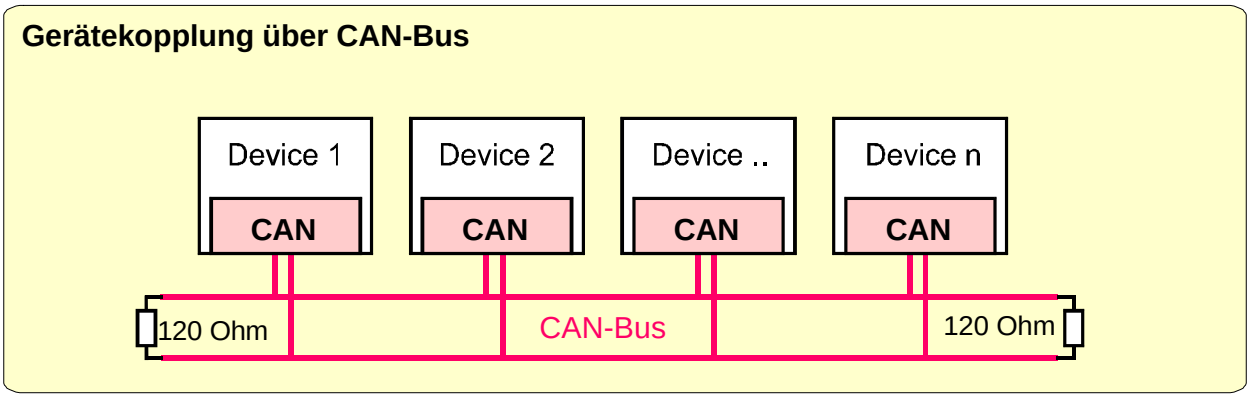


Abbildung 1: Gerätekopplung über CAN-Bus

1.1 Prinzip des Datenaustausches

Bei der CAN-Bus-Datenübertragung werden keine Stationen adressiert, sondern der Inhalt einer Nachricht wird durch einen eindeutigen Nummerncode (Identifier) gekennzeichnet. Neben dieser Inhaltskennzeichnung legt der Identifier auch die Priorität der Nachricht fest.

Möchte eine beliebige Station eine Nachricht versenden, übergibt sie Nachricht und Identifier an den CAN-Controller. Dieser übernimmt die Übertragung der Nachricht. Sendet er derzeit als einziger, oder hat seine Nachricht die höchste Priorität, dann empfangen alle anderen CAN-Controller im Netz diese Nachricht. Noch im empfangenden CAN-Controller wird selektiert, ob diese Nachricht für die eigene Station erforderlich ist. Um die Selektion durchführen zu können, wird dem CAN-Controller bei der Initialisierung mitgeteilt, welche Nachrichten der Station zugeordnet werden müssen. Ist die empfangene Nachricht für eine Station uninteressant, wird Sie vom dessen CAN-Controller ignoriert.

Die Nachrichtenübertragung erfolgt bitweise auf einem differenziellen Leitungspaar (CAN-high und CAN-low-Leitung). Dabei werden für die Bitinformation zwei Zustände (dominant = 0 und rezessiv = 1) unterschieden.

1.2 Aufbau des CAN-Telegramms

Die Anzahl der Bits eines Telegramms hängt von der Größe des Datenfelds ab. Die Information einer Nachricht ist im Identifier und in den eventuell zusätzlich mitgesendeten Datenbytes enthalten. Der Identifier ist Teil des Statusfelds im CAN-Telegramm und liefert die Information in Signalform (vom Anwender festgelegte Bedeutung), wobei im Datenfeld bis zu 8 Bytes als Variablen mitgesendet werden können.

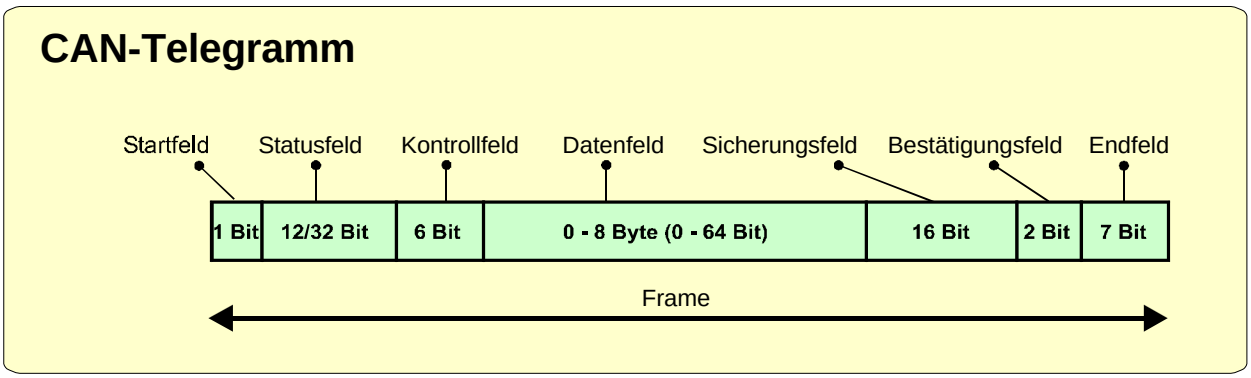


Abbildung 2: CAN-Telegramm (Bit-Stream)

In der ISO11898 sind zwei Telegrammformate definiert, die auch von CAN-Open unterstüzt werden. Dies sind das Standard-Format mit einem 11-Bit Identifier und das Extended-Format mit einem 29-Bit Identifier, der sich aus dem 11-Bit Identifier und dem 18-Bit Subindex zusammensetzt. Die Architektur des Telegramms ermöglicht die Verwendung beider Typen in einem Netz.

Telegramm-bereiche	**Erläuterung**
Startfeld	Startbit des Telegramms
Statusfeld	Enthält den Identifier und diesbezügliche Statusbits (siehe 1.2.1)
Kontrollfeld	Enthält u. a. die Anzahl der im Datenfeld enthaltenen Bytes (siehe 1.2.1)
Datenfeld	Hier können zusätzlich Informationen/Variablen (bis zu 8 Byte je Telegramm) übertragen werden. (die Anzahl der Datenbytes wird im Kontrollfeld übermittelt)
Sicherungsfeld	Sicherungsfeld (16 Bit). Es enthält ein Rahmensicherungswort zur Erkennung von etwa auftretenden Übertragungsstörungen
Bestätigungsfeld	Bestätigungsfeld (2 Bit). Im Bestätigungsfeld signalisieren die Empfänger dem Sender, dass die Datenübertragung korrekt oder fehlerhaft ist. Die Rückmeldung geschieht dadurch, dass diese Bits noch im Sendetelegramm von mindestens einem Empfänger überschrieben werden. Da der Sender seine eigene Nachrichtenübertragung mitverfolgt, erkennt er diese Rückmeldung und kann

	entsprechend reagieren. (z. B. Telegramm wiederholen)
Endfeld	Endfeld (7 Bit). Das Datenprotokoll endet durch die Mitteilung im Endfeld. Im Endfeld besteht die letzte Möglichkeit, Datenübertragungsfehler zu melden, die zu einer Wiederholung führen

Tabelle 1: Telegrammaufbau 1

1.2.1 CAN-Identifier und Kontrollfeld

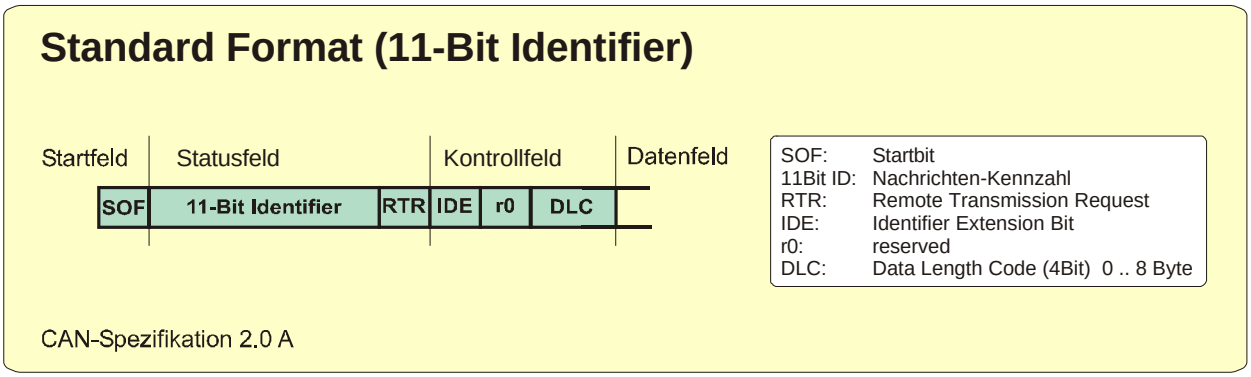


Abbildung 3: CAN-Telegramm im Standardformat 11-Bit ID

Damit ergibt sich eine minimale Telegrammlänge von 44 Bit bei 0 Datenbyte Informationsvolumen

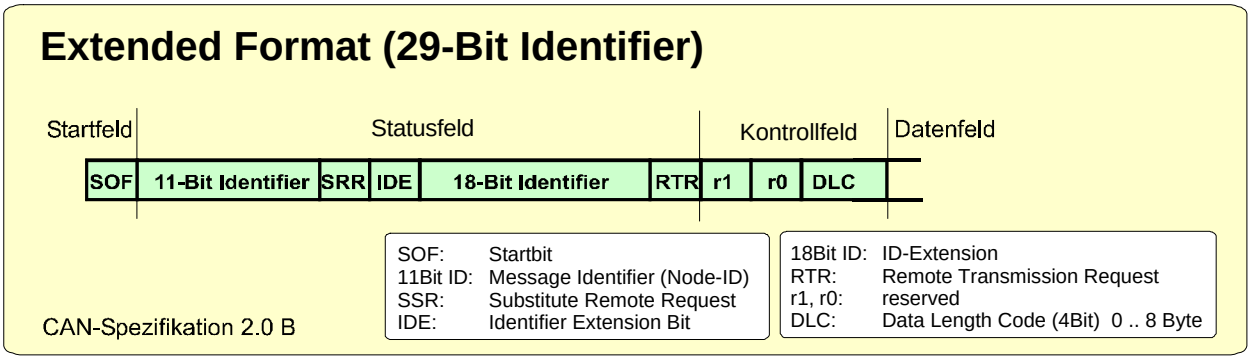


Abbildung 4: CAN-Telegramm im Standardformat 29-Bit ID

Damit ergibt sich eine minimale Telegrammlänge von 62 Bit bei 0 Datenbyte Informationsvolumen

Bits	**Erläuterung**
SOF	Startbit des Telegramms
11-Bit-ID	Nachrichten-Kenncode als 11-Bit-Wert. Der kleinste Wert besitzt die höchste Priorität
18-Bit-ID	Nachrichten-Kenncode-Erweiterung als 18-Bit-Wert. Bei gleicher 11-Bit-ID besitzt

	der kleinste Wert die höchste Priorität.
IDE	Kennzeichnet, ob das Telegramm einen 11- bzw. 29-Bit Identifier besitzt (dominant bei 11-Bit-ID).
RTR	Kennzeichnet, ob es sich bei diesem Telegramm um ein Datentelegramm oder um einen Anforderungsrahmen handelt.
SRR	Bestätigungsfeld (2 Bit). Im Bestätigungsfeld signalisieren die Empfänger dem Sender, dass sie das Datenprotokoll korrekt empfangen haben. Wird ein Fehler erkannt, teilen sie es dem Sender mit. Der Sender wiederholt die Übertragung.
r0 , r1	reserviert
DLC	Data Length Code (4 Bit). Im DLC steht die Anzahl der im Telegramm enthaltenen Datenbytes.

Tabelle 2: Telegrammaufbau 2

Der CAN-Bus ist damit ein universelles Übertragungsmedium für beliebige Daten. Wobei das Protokoll und die Datensicherungsmaßnahmen des Controllers Übertragungsfehler stark minimieren.

1.3 Die Arbitrierung im CAN-Netzwerk

Der CAN-Bus ist ein Netzwerk mit Multimaster Funktionalität. Alle Teilnehmer sind prinzipiell gleichberechtigt. Es ist von Seiten der Bus-Konzeption kein Frage-/Antwortverhalten vorgesehen. Vielmehr soll jeder Teilnehmer seine Datenübertragungen selbständig einleiten. Die Arbitrierung soll innerhalb einer Nachricht und ohne diese zu zerstören erfolgen.

Auf dem Bus werden die Pegel „aktiv“ entsprechend einer „0“ im CAN-Telegramm als dominant und „passiv“ entsprechend einer „1“ im CAN-Telegramm als rezessiv bezeichnet. Dabei überschreibt ein dominanter Pegel einen rezessiven Buszustand grundsätzlich. Dies bedeutet, dass ein CAN-Knoten, der einen rezessiven Pegel auf den Bus senden will, von einem dominant sendenden Knoten überschrieben wird.

Jeder CAN-Knoten hört auf dem Bus grundsätzlich mit. Ein Sendevorgang darf nur gestartet werden, wenn der Bus derzeit nicht von einem CAN-Telegramm belegt ist. Beim Senden wird dabei immer der aktuelle Buszustand mit dem eigenen Sendeframe verglichen.

Beginnen mehrere Controller eine Übertragung gleichzeitig, dann entscheidet das erste dominante Bit auf der Leitung über die Priorisierung (Dominanz hat Vorrang) der Nachricht. Erkennt ein CAN-Knoten, der einen rezessiven Pegel auf den Bus schreiben wollte, dass der Bus einen dominanten Pegel aufweist, wird der eigene Übertragungsvorgang abgebrochen und zu einem späteren Zeitpunkt erneut versucht. Die wichtigere Nachricht (Nachricht mit kleinstem Identifier) bleibt dadurch auf dem Bus fehlerfrei erhalten.

Als Samplepoint wird etwa ¾ einer Bitzeit verwendet.

Daraus ergeben sich folgende Forderungen:

1) Da eine „0“ im CAN-Telegramm den dominanten Pegel auf dem Bus repräsentiert folgt, dass die CAN-Identifier mit niedrigen Zahlen höher priorisiert sind. Je wichtiger also eine Busnachricht, desto niedriger muss der dafür verwendete Identifier gewählt werden.

2) Die Priorisierung erfolgt immer innerhalb eines Bits. Daraus folgt, dass alle Teilnehmer im Netzwerk innerhalb einer Leitungsverzögerung von 1 Bitzeit (genauer ¾) liegen müssen. Dies bezieht sich auf Hin- und Rückweg des Signals.

3) Da die Arbitrierung innerhalb des Identifiers erfolgen muss, darf es zu jedem Identifier nur einen Sender geben. Dieser darf jedoch von mehreren Knoten empfangen werden.

1.4 Buslängen

Aus den Anforderungen der Arbitrierung ergeben sich in etwa die folgenden Buslängen. Die Werte dienen als Anhaltspunkte, und können sich in Abhängigkeit der verwendeten Bustransceiver bzw. galvanischen Entkopplung verändern, insbesondere auch verschlechtern.

Bit Rate	Bus Länge	Nominale Bitzeit
1 Mbit/sec	30 m	1 usec
800 kBit/sec	50 m	1,25 usec
500 kBit/sec	100 m	2 usec
250 kBit/sec	250 m	4 usec
125 kBit/sec	500 m	8 usec
50 kBit/sec	1000 m	20 usec
20 kBit/sec	2500 m	50 usec
10 kBit/sec	5000 m	100 usec

Die Buslängen sind grobe Anhaltspunkte bei Verwendung von ISO11898 konformen Treibern, Standard Busleitungen ohne die Berücksichtigung von Optokopplern. Diese machen sich insbesondere bei hohen Übertragungsraten bemerkbar.

2 CANopen

Wie praktisch alle Feldbusse setzt auch der CAN auf dem OSI 7-Schichtmodel auf.

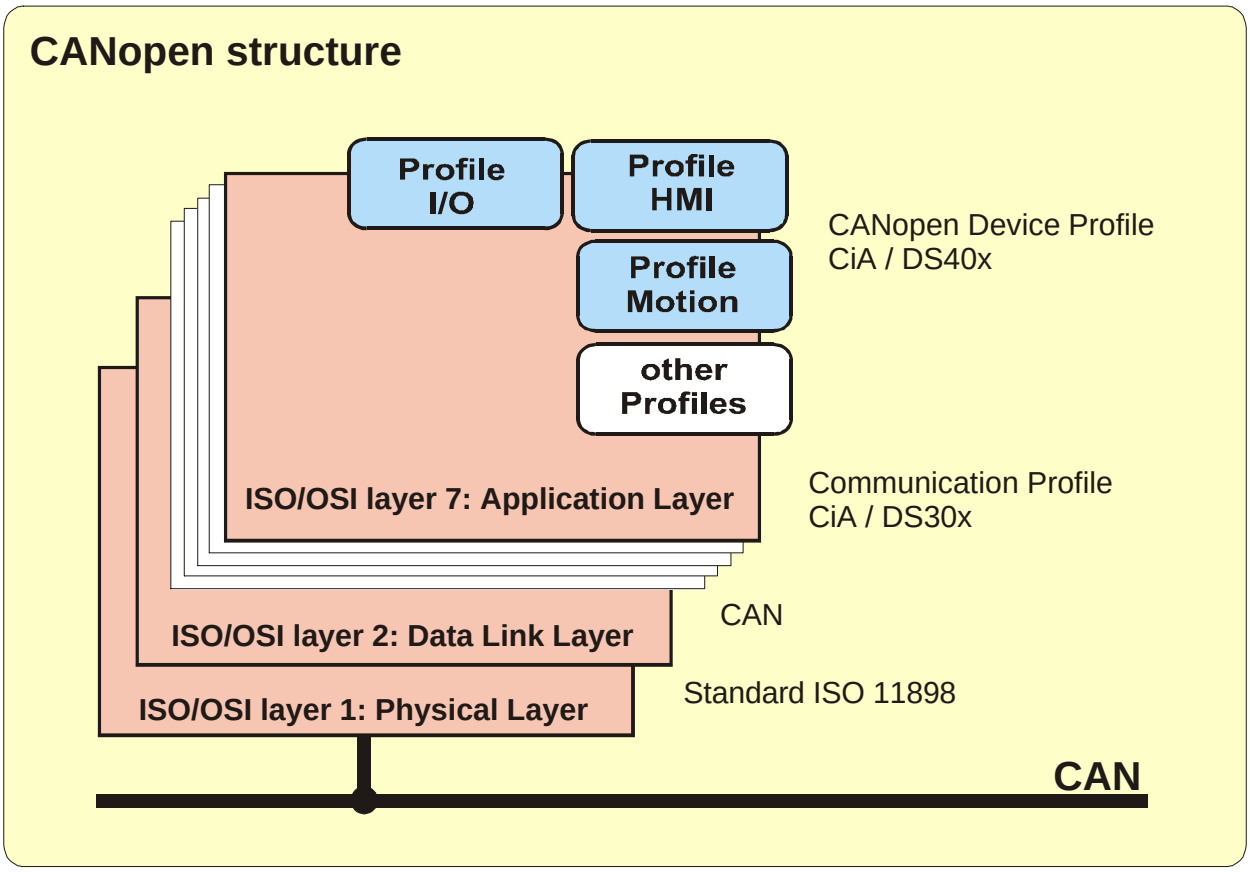


Abbildung 5: Strukturmodel von CANopen

CANopen ist der offene Protokollstandard für CAN in der Automatisierungstechnik und wurde im Verband „CAN in Automation“ (CiA) standardisiert. Das Protokoll nutzt den CAN-Bus als Übertragungsmedium und legt die Grundstrukturen für das Netzwerkmanagement, die Verwendung der CAN-Identifier (Nachrichten-adresse), das zeitliche Verhalten auf dem Bus, die Art der Datenübertragung, und anwendungsbezogene Profile fest. Dies soll gewährleisten, dass CANopen - Baugruppen unterschiedlicher Hersteller kombiniert werden können (*dass die Geräte die gleiche Sprache sprechen*).

CANopen definiert den Applicationslayer (die OSI-Schicht 7) als Kommunikationsprofil, das von der CiA im Standard DS30x spezifiziert wurde für alle Anwendungen gleich. Es legt fest, wie die Kommunikation zu erfolgen hat. Wie bei einigen anderen Feldbussen werden hierbei Echtzeitdaten und Parameterdaten unterschieden.

2.1 Geräteprofile

CANopen beschreibt die Eigenschaften von Geräten in sogenannten Geräteprofilen. Diese sind im Prinzip definierte Variablensätze. Abhängig vom Gerätetyp werden bestimmte Daten bzw. Parameter (in CANopen als Objekte bezeichnet) fest definiert.

Die Geräteprofile wurden von der CiA in den Standards DS40x beschrieben.

Geräteprofile (CiA)	für die Gerätetypen
DS 401	Digitale und analoge E/A
DS 402	Antriebe
DS 403	Bedienen und Beobachten
DS 404	Sensoren / Regler
DS 405	Programmierbare Geräte
DS 406	Encoder
DS …	(weitere Geräteprofile)

Tabelle 3: Beispiel DS40x - Geräteprofile

2.2 Objektverzeichnis

Das Objektverzeichnis ist die Zusammenstellung aller Variablen und Parameter (Objekte) eines CANopen Geräts. Dabei enthalten die Daten das Prozessabbild und mit den Parametern kann das Funktionsverhalten eines CANopen Gerätes beeinflusst werden.

Ein Objektverzeichnis ist so aufgebaut, dass einige Parameter für alle Geräte dieser Kategorie zwingend vorgeschrieben sind und andere frei definiert und verwendet werden dürfen.

Die Objekte erhalten in CANopen vornehmlich eine Nummer (den sogenannten Index), mit der sie eindeutig identifiziert und auch adressiert werden können. Die Objekte können als einfache Datentypen wie z. B. Bytes, Integers, Longs oder auch Strings realisiert sein. Bei komplexeren Strukturen wie z. B. Arrays und Strukturen, wird zur Adressierung der einzelnen Elemente ein Subindex eingeführt.

Die Struktur des Objektverzeichnisses, die Vergabe der Index-Nummern sowie einige Pflichteinträge sind in den Geräteprofilen spezifiziert.

Für den Anwender ist das Objektverzeichnis als EDS-file (Electronic Data Sheet) gespeichert. Im EDS-file sind alle Objekte mit Index, Subindex, Name, Datentyp, Defaultwert, Minima, Maxima und Zugriffs-möglichkeiten (Lesen-/Schreiben, Übertragung nur per SDO oder auch per PDO usw.) gespeichert.

Mit der EDS-Datei wird somit die komplette Funktionalität eines CANopen Geräts beschrieben.

Prinzipielle Vergabe der Objekt-Index-Nummern

Object Index (hex)	**Object**
0000	Not used
0001 - 001F	Static Data Types
0020 - 003F	Complex Data Types
0040 - 005F	Manufacturer Specific Complex Data Types
0060 - 007F	Device Profile Specific Static Data Types
0080 - 009F	Device Profile Specific Complex Data Types
00A0 - 0FFF	Reserved for further use
1000 - 1FFF	Communication Profile Area
2000 - 5FFF	Manufacturer Specific Profile Area
6000 - 9FFF	Standardized Device Profile Area
A000 - FFFF	Reserved for further use

Tabelle 4: Allgemeine Struktur eines CANopen Objektverzeichnisses

Beispiel für ein Objektverzeichnis: Ausschnitt aus dem Verzeichnis des CANopen Chip CO4011

Index	*Sub-Index*	*Name*	*Acc.*	*PDO-Mapping*
0005	*-*	*Dummy 8*	*ro*	*Yes*
0006	*-*	*Dummy 16*	*ro*	*Yes*
100B	*-*	*Node ID*	*ro*	*-*
100C	*-*	*Guard Time*	*rw*	*-*
100D	*-*	*Life Time Factor*	*rw*	*-*
100E	*-*	*COB-ID Guard*	*rw*	*-*
1010	*-*	*Store Parameters *1)*	*wo*	*-*
1011	*-*	*Reload Default Parameter *1)*	*wo*	*-*
1014	*-*	*COB ID Emergency*	*rw*	*-*
1015	*-*	*Inhibit Time Emergency*	*rw*	*-*
1017	*-*	*Producer Heartbeat Time*	*rw*	*-*
1018		*Identity Object*		
	0	*(no. of subentries)*	*ro*	*-*
	1	*Vendor ID*	*ro*	*-*
	2	*Product Code*	*ro*	*-*
	3	*Revision Number*	*ro*	*-*
2000	*-*	*Device Manufacturer *3)*	*ro*	*-*
2100	*-*	*New Node ID *4)*	*rw*	*-*
2101	*-*	*System Configuration*	*ro*	*-*
6000	*0 to n*	*Read digital input 8-bit*	*ro*	*Yes*
6002	*0 to n*	*Polarity input 8-bit*	*rw*	*-*
6003	*0 to n*	*Filter constant enable 8-bit*	*rw*	*-*
6005		*Global interrupt enable*	*rw*	*-*
6006	*0 to n*	*Interrupt mask: any change*	*rw*	*-*
6007	*0 to n*	*Interrupt mask rising edge*	*rw*	*-*

CANopen Kommunikation

Der Datenaustausch in CANopen erfolgt in Form von Telegrammen, mit denen die Nutzdaten übertragen werden. Unterschieden werden dabei die Servicedatenobjekte, die zur Übermittlung der Servicedaten von und zum Objektverzeichnis verwendet werden und Prozessdatenobjekte die zum Austausch der aktuellen Prozesszustände dienen. Zusätzlich werden Telegramme für das Netzwerk Management und für die Fehlermeldungen definiert.

Im Prinzip kann mit Hilfe von SDOs auf alle Einträge des Objektverzeichnisses zugegriffen werden. In der Praxis werden SDOs meist nur zur Initialisierung während des Boot-Vorgangs verwendet. Innerhalb eines SDOs kann immer nur auf ein Objekt zugegriffen werden. SDOs werden grundsätzlich beantwortet.

PDOs sind im Prinzip eine Zusammenfassung von Objekten (Variablen bzw. Parameter) aus dem Objektverzeichnis. In einem PDO können maximal 8 Byte, die aus verschiedenen Objekten zusammengesetzt sein können, bestehen. Man sagt, die Objekte sind in ein PDO gemappt.

PDO (Process Data Object)	**SDO (Service Data Object)**
- Echtzeitdaten - Telegramm wird nicht beantwortet (schnellere Übertragung) - Hochpriore Identifier - Max. 8 Bytes / Telegramm - Vorvereinbartes Datenformat	- System-Parameter - Telegramm wird beantwortet (langsamere Übertragung) - Niederpriore Identifier - Daten auf mehrere Telegramme verteilt - Indexadressierbare Daten

Tabelle 5: Kommunikationsarten bei CANopen

CANopen verzichtet zu Gunsten des Netzwerkmanagements auf die Multimasterfähigkeit des CAN-Busses und führt einen CAN-Master ein, der die Aufgaben des Netzwerkmanagements übernimmt. Alle weiteren CAN-Knoten werden als sogenannte Slave-Baugruppen implementiert.

Für das Netzwerkmanagement und Fehlermeldungen gibt es vordefinierte logische Kommunikationskanäle wie

- Kommunikationsobjekte für den Boot-Up (das Aufstarten des Netzes)
  Starten, Anhalten, Zurücksetzen eines Knotens usw.
- Kommunikationsobjekte für die dynamische Identifier-Verteilung nach DBT
- Kommunikationsobjekte für das Nodeguarding und Lifeguarding
  Damit kann eine Überwachung des Netzwerks durchgeführt werden.
- Ein Kommunikationsobjekt für die Synchronisation
- Kommunikationsobjekte für Notfallmeldungen (Emergency)

Diese sind in CANopen fest definiert und besitzen einen globalen Nachrichtencharakter (Broadcast).

2.3 Default Identifier Verteilung

Eine Default-Identifier-Verteilung spart Konfigurationsaufwand. Dabei wird die Knotennummer in den Identifier eingebettet. Die Default-Identifier-Verteilung ist in CANopen wie folgt definiert:

Identifier 11-Bit (binär)	Identifier (dezimal)	Identifier (hexadezimal)	Funktion
00000000000	0	0	Netzwerkmanagement
00010000000	128	80h	Synchronisation
0001xxxxxxxx	129 - 255	81h - FFh	Emergency
0011xxxxxxxx	385 - 511	181h - 1FFh	PDO1 (tx)
0100xxxxxxxx	513 - 639	201h - 27Fh	PDO1 (rx)
0101xxxxxxxx	641 - 767	281h - 2FFh	PDO2 (tx)
0110xxxxxxxx	769 - 895	301h - 37Fh	PDO2 (rx)
0111xxxxxxxx	897 - 1023	381h - 3FFh	PDO3 (tx)
1000xxxxxxxx	1025 - 1151	401h - 47Fh	PDO3 (rx)
1001xxxxxxxx	1153 - 1279	481h - 4FFh	PDO4 (tx)
1010xxxxxxxx	1281 - 1407	501h - 57Fh	PDO4 (rx)
1011xxxxxxxx	1409 - 1535	581h - 5FFh	SDO senden
1100xxxxxxxx	1537 - 1663	601h - 67Fh	SDO empfangen
1110xxxxxxxx	1793 - 1919	701h - 77Fh	NMT Error Controll
xxxxxxxx = Knotennummer 1 - 127			

Tabelle 6: Default-Identifier

CANopen ermöglicht eine komplett freie Identifierkonfiguration.

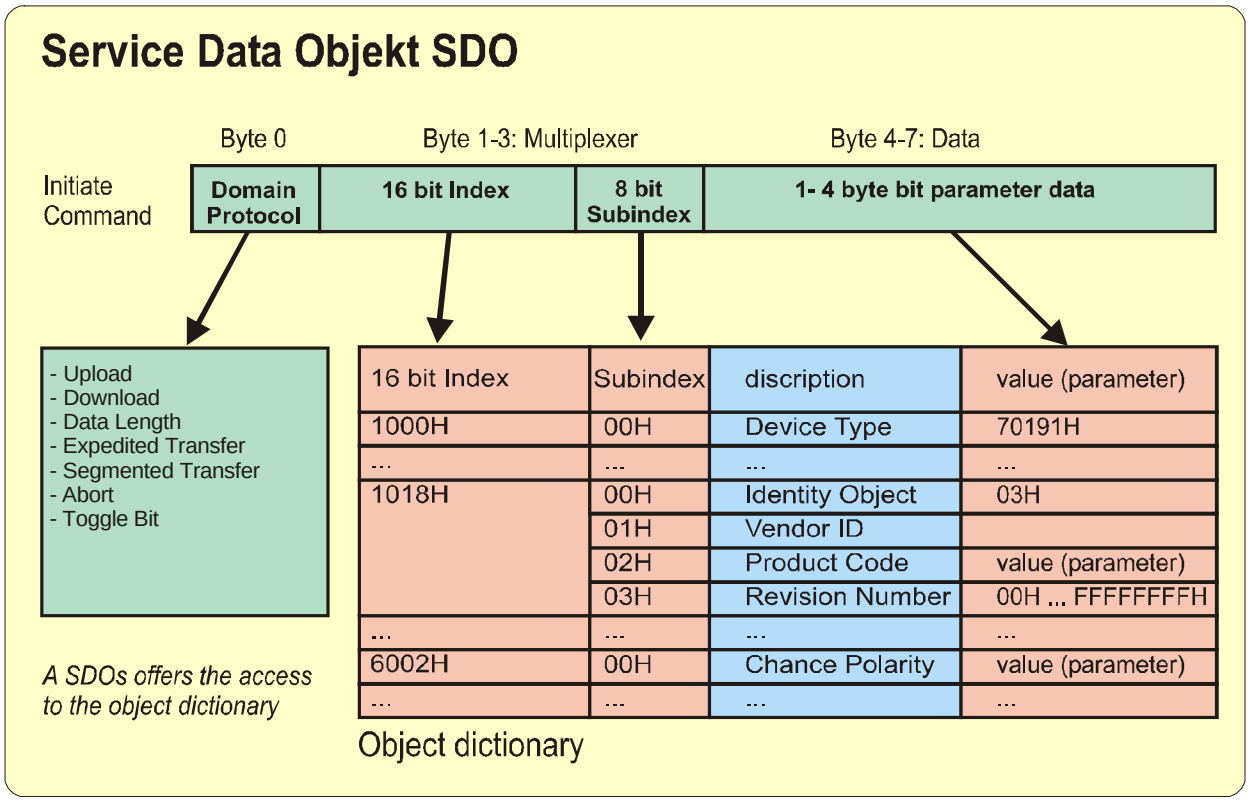
Das Servicedatenobjekt SDO

Alle CANopen Baugruppen besitzen ein sogenanntes Objektverzeichnis, über das auf alle Parameter die von der Baugruppe bereitgestellt werden, zugegriffen werden kann.

Wie aus dem Objektverzeichnis ersichtlich ist, besitzen die Objektdaten einen 16-Bit Index, über den ein Parameter direkt angewählt werden kann. Zudem existiert noch zu jedem Index ein 8-Bit Sub-Index, der eine weitere Untergliederung innerhalb eines Indexes ermöglicht.

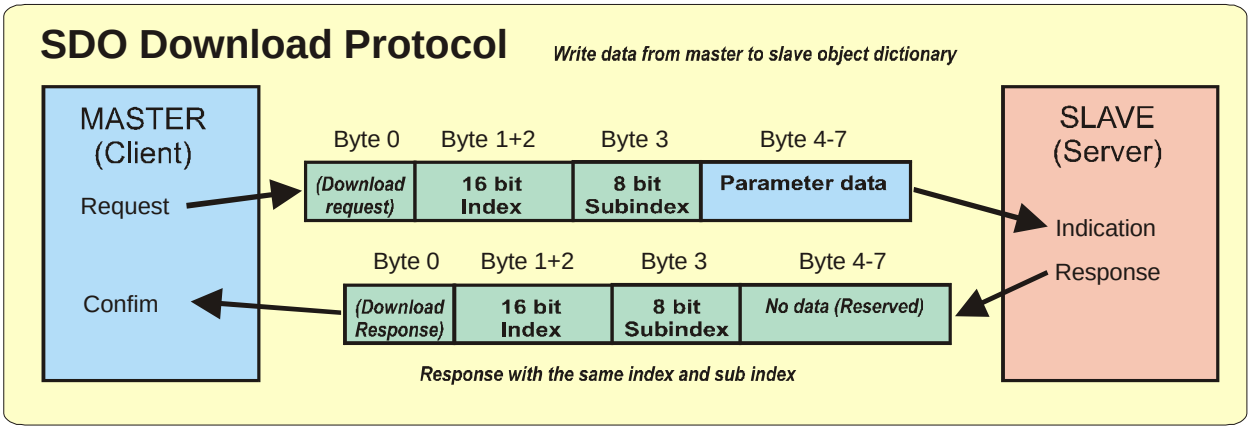
Daher muss ein Servicedaten-Telegramm eine Protokollstruktur besitzen, die genau bestimmt, welcher Parameter angesprochen wird und was mit dem Parameter geschehen soll.

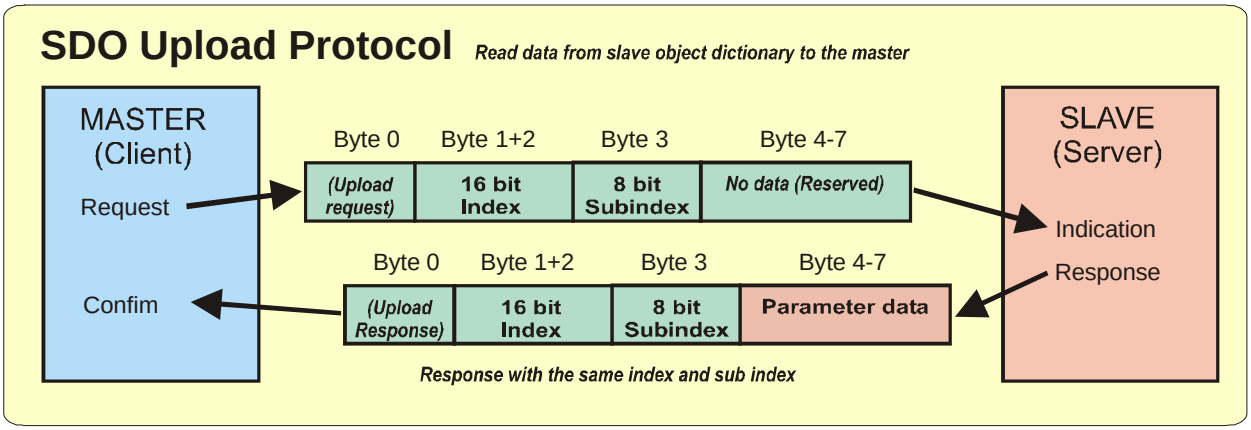
Ein Servicedatenobjekt setzt sich aus einem „Domainprotokoll (8-Bit)“, dem „Index (16-Bit)“, dem „Sub-Index (8-Bit)“ und bis zu 4 Datenbytes zusammen. Dabei beinhaltet das Domainprotokoll die Aktion, die auf den mit Index und Sub-Index verwiesenen Parameter zu erfolgen hat. Sollen Parameter neue Werte erhalten, können diese in den Datenbytes mit übertragen werden.



Die 8 Byte des SDOs (wie hier Dargestellt) sind im Datenbereich der CAN-Message untergebracht. Die Adressierung des Knotens erfolgt mit dem SDO im Message-Identifier.

Ein SDO-Transfer besteht immer aus mindestens zwei Telegrammen.





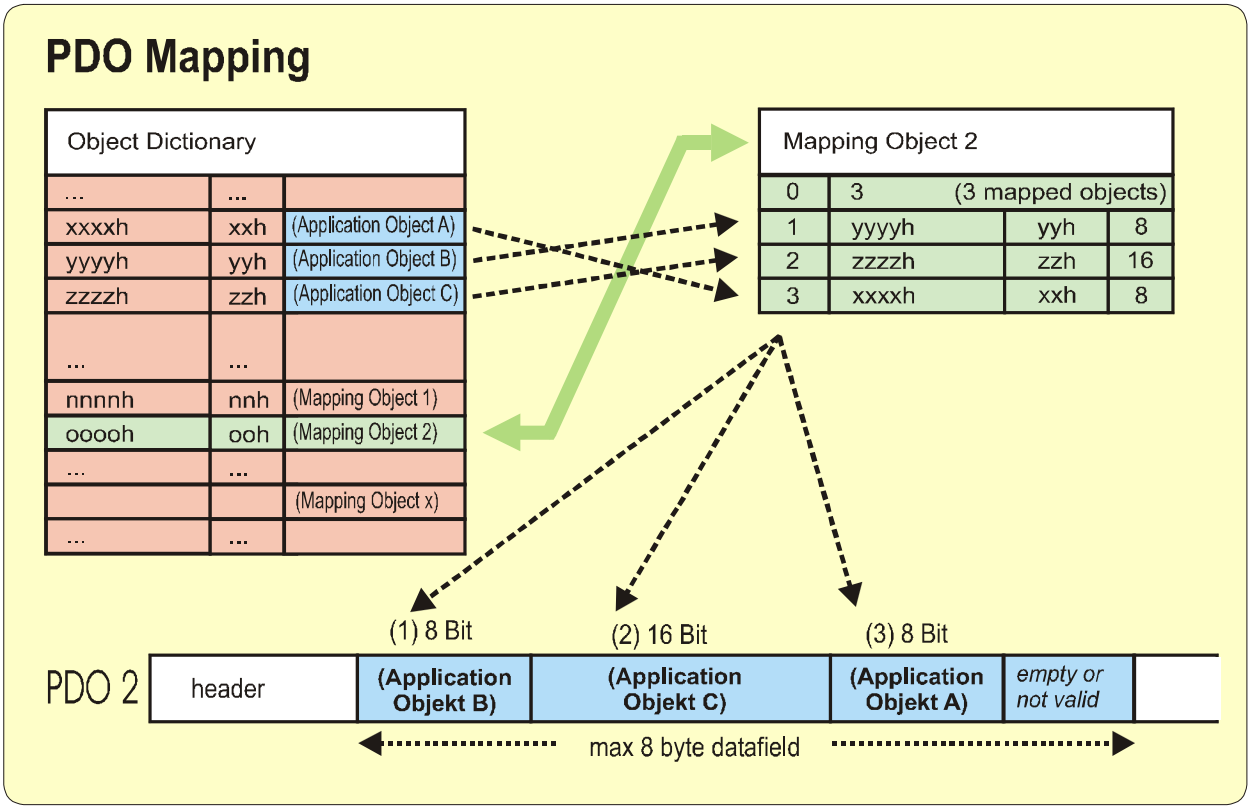
2.4 Das Prozessdatenobjekt PDO

Der Prozessdatenaustausch mit CANopen ist wiederum CAN-Bus pur, also ohne Protokoll-Overhead. Die Broadcast–Eigenschaft des CAN-Bus bleibt vollständig erhalten. Eine Nachricht kann somit von allen Knoten empfangen und ausgewertet werden (Producer-Consumer-Model). Das starre Master/Slave-Prinzip ist damit beim Datenaustausch mit PDOs aufgehoben.

Da im Telegramm die Protokollstruktur fehlt, ist es erforderlich, dass der/die Teilnehmer am Bus, für die diese Daten bestimmt sind wissen, wie die Information im Datenbereich des PDOs eingebettet sind (welches Bit/Byte ist welcher Wert). Diese Deklaration geschieht daher im vorab bei der Initialisierung des Netzes mit dem sogenannten PDO-Mapping, das es ermöglicht, die gewünschten Informationen an einer bestimmten Stelle im Datenbereich eines PDOs zu platzieren.

2.4.1 PDO-Mapping

Um eine variable Konfiguration der PDO Daten zu ermöglichen, wird das Mapping selbst wiederum in einem speziellen Mapping-Objekt vorgenommen. Dieses ist im Prinzip eine Tabelle, in welche die zu mappenden Objekte eingetragen werden.



Der Austausch von Prozessdaten kann generell auf verschiedene Arten geschehen, die in einem Netzwerk gleichzeitig (in gemischter Form) vorkommen können:

2.4.2 Ereignisgesteuerter Datentransfer

Dabei werden die Daten eines Knoten als Nachricht verschickt, sobald eine Änderung an dem bisherigen Zustand erfolgt ist.

1) *Wechselt beispielsweise der Pegel an einem digitalen Eingang einer CAN-I/O-Baugruppe, so wird dies das Versenden der zugeordneten Nachricht (PDO) veranlassen.*
2) *Besitzt beispielsweise ein Knoten Schwellwerte für einen Analogwert, und wird die Schwelle erreicht, wird ebenfalls die zugeordnete Nachricht (PDO) verschickt.*

2.4.3 Polling mit Remote Frames

Beim Remote Frame Polling fordert generell der CAN-Knoten, der als Master im Netzwerk fungiert, die gewünschte Information per Abfrage (mittels Remote Frame) an. Derjenige Knoten, der diese Information (bzw. die erforderlichen Daten) besitzt, antwortet dann mit dem Versand der angeforderten Daten.

Da bei CANopen der Message-Identifier auch die Baugruppenadressierung enthält, (ein Teil der ID wird als Knoten-Nummer spezifiziert) wird die Abfrage normalerweise an eine spezielle Baugruppe gerichtet sein. Alle anderen CAN-Controller im Netz ignorieren die Abfrage.

1) *Der Master besitzt beispielsweise eine visuelle Prozessdatenanzeige auf der soeben das Menü für die Drehzahlanzeige Servomotor 2 geöffnet wurde. Da die Drehzahl des Motors für den sonstigen Ablauf nicht benötigt wird, kann der Master diese solange zyklisch per Remote Frame abfragen, solange das Anzeigemenü geöffnet ist.*

2.4.4 Synchronisierter Modus

CANopen ermöglicht es Eingänge und Zustände verschiedener Teilnehmer gleichzeitig abzufragen und Ausgänge bzw. Zustände gleichzeitig zu ändern. Hierzu dient das Synchronisationstelegramm (SYNC). Das Sync-Telegramm ist ein Broadcast an alle Busteilnehmer mit hoher Priorität ohne Dateninhalt. Das Sync-Telegramm wird von einem Busteilnehmer (in der Regel vom Master) zyklisch in festen Intervallen (Kommunikationszyklus) versandt.

Baugruppen, die im synchronisierten Modus arbeiten, lesen ihre Eingänge (aktuellen Zustände) bei Empfang der Sync-Nachricht aus und senden die Daten direkt anschließend, sobald der Bus dies zulässt.

Ausgangsdaten (über den Bus instruierte Zustandsänderungen) werden erst nach dem nächsten Sync-Telegramm zu den Ausgängen geschrieben (bzw. ausgeführt).

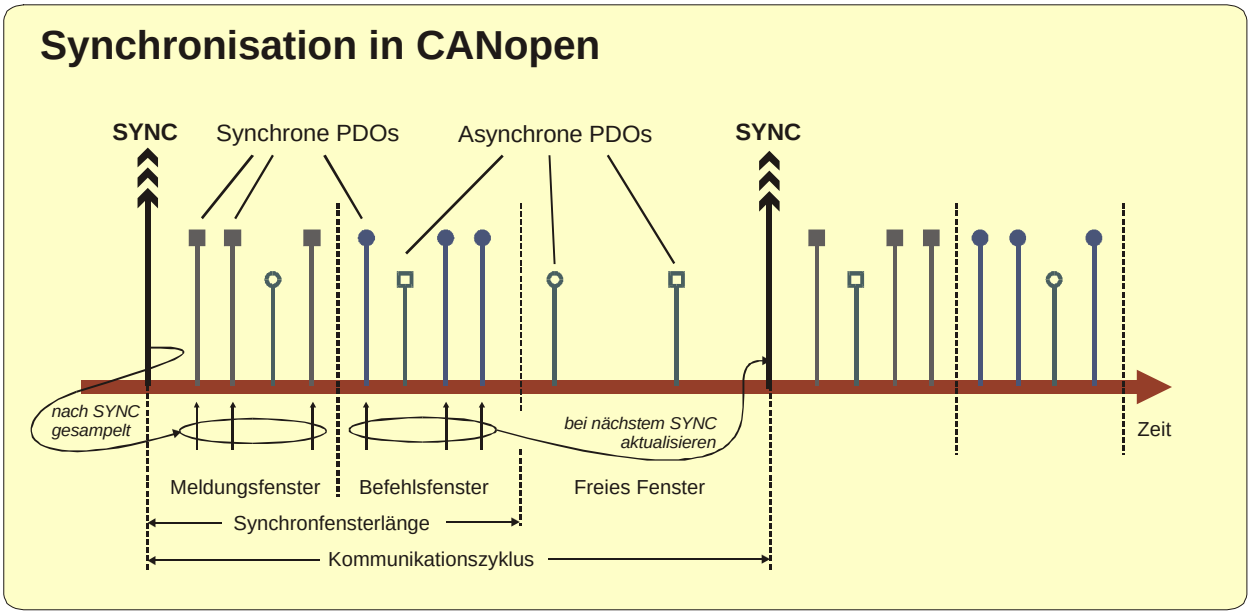


Abbildung 6: Synchronisation in CANopen

Das Intervall zwischen zwei Sync-Telegrammen ist dafür nominal unterteilt in ‚Meldungsfenster', ‚Befehlsfenster' und ‚Freies Fenster'. Jedoch wird die Länge der Nachrichtenfenster nicht überwacht. Dadurch ist es möglich, Eingangsdaten der einen Baugruppe auch im synchronisierten Modus direkt auf die Ausgänge einer anderen durchzuschalten.

Wie man an der bisherigen Beschreibung des Verfahrens leicht erkennt, führt dieser Modus zu einer enormen Busbelastung, wenn alle Teilnehmer ihren Datenaustausch nach jedem Sync-Telegramm beginnen. In der Realität wird jedoch nicht jeder Teilnehmer auf jede Sync-Nachricht reagieren müssen. Hierzu wurden folgenden zwei Verfahren definiert:

2.4.4.1 Ereignisgesteuerte Synchronisation

Wie beim Ereignisgesteuerten Datentransfer wird auch hier keine Nachricht verschickt, solange der Zustand der Parameter (Eingänge bei I/O-Modulen) unverändert ist. D. h., es wird beim Erhalt eines Sync-Telegramms verglichen, ob sich seit der letzten Aktualisierung eine Änderung der eigenen Parameter / Daten ergeben hat. Ist dies nicht der Fall, werden auch keine Daten versandt. Sind jedoch Änderungen zu verzeichnen, werden diese nach Erhalt der Sync-Nachricht verschickt.

2.4.4.2 Unterteilte (Timer-) Synchronisation

Bei der Timer-Synchronisation besitzt der Teilnehmer eine Konstante n, die angibt, beim wievielten Sync-Telegramm eine Datenaktualisierung durchzuführen ist. (Ist n = 3, erfolgt eine Aktualisierung bei jedem dritten SYNC). Diese kann vom Hersteller in der Baugruppe fest eingestellt, oder im Objektverzeichnis der Baugruppe einstellbar sein.

2.4.5 Zeitgesteuerte Übertragung

Dabei wird die Übertragung eines PDOs durch den Ablauf einer einstellbaren Zeit veranlasst.