

CANopen User Manual

Software Manual

Auflage September 2015

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.

© Copyright 2015 SYS TEC electronic GmbH. Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form ohne schriftliche Genehmigung der Firma SYS TEC electronic GmbH unter Einsatz entsprechender Systeme reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Kontakt	Direkt	Ihr lokaler Distributor
Adresse:	SYS TEC electronic GmbH Am Windrad 2 D-08468 Heinsdorfergrund GERMANY	
Angebots-Hotline:	+49 (0) 3765 / 38600-2110 info@systec-electronic.com	Sie finden eine Liste unserer Distributoren unter http://www.systec-electronic.com/distributors
Technische Hotline:	+49 (0) 3765 / 38600-2140 support@systec-electronic.com	
Fax:	+49 (0) 3765 / 38600-4100	
Webseite:	http://www.systec-electronic.com	

16. Auflage September 2015

Inhaltsverzeichnis

Einleitung	1
1 CANopen Grundlagen	3
1.1 Was ist CANopen?	3
1.2 Kommunikationsobjekte	6
1.2.1 PDO - Prozessdatenobjekte.....	6
1.2.2 SDO - Servicedatenobjekte.....	16
1.2.3 Synchronisationsobjekte	17
1.2.4 Time Stamp Object	18
1.2.5 Emergency	18
1.2.6 Layer-Setting-Service (LSS)	19
1.2.7 Netzwerkmanagement	22
1.3 CANopen Kommunikationsprofil.....	26
1.4 Übertragungsprotokolle	26
1.5 Objektverzeichnis	27
1.6 Fehlerbehandlung und -signalisierung.....	27
1.7 Telegramm-Tabelle (Pre-defined connection set).....	28
2 CANopen Anwenderschicht	29
2.1 Softwarestruktur	29
2.1.1 CANopen-Stack	32
2.1.2 CDRV - Hardwarespezifische Schicht.....	33
2.1.3 CCM – Applikationsspezifische Schicht	33
2.2 Verzeichnisstruktur	36
2.3 Datenstrukturen	38
2.4 Das Objektverzeichnis	42
2.4.1 Objektverzeichnis für Standard-I/O-Geräte	43
2.5 Instanziierung der CANopen-Schicht.....	46
2.5.1 Verwendung von Instanz-Handle	47
2.5.2 Verwendung von Instanz-Pointer	47
2.6 Hinweise für das Erstellen einer Applikation.....	48
2.6.1 Auswahl der benötigten Module und Konfiguration.....	49
2.6.2 Ablauf einer CANopen-Applikation.....	51
2.7 Funktionen der CCM Schicht.....	61
2.7.1 CcmMain-Modul.....	61
2.7.2 CcmSdoc	86
2.7.3 CcmDfPdo.....	98
2.7.4 CcmObj.....	101
2.7.5 CcmLgs.....	103
2.7.6 CcmStore	105
2.7.7 CcmNmtm und CcmNmtm	114
2.7.8 CcmSnPdo.....	121
2.7.9 CcmSync.....	121
2.7.10 CcmSyncEx	124
2.7.11 CcmEmcc.....	125
2.7.12 CcmEmcp	128
2.7.13 CcmHbc	132
2.7.14 CcmHbp	135
2.7.15 TgtCav	136

2.7.16	CcmBoot	145
2.7.17	CcmFloat.....	146
2.7.18	CcmStPdo.....	147
2.7.19	Ccm303.....	150
2.7.20	CcmLss	156
2.7.21	Kommunikations-Parameter und Prozessvariablen	167
2.8	Beschreibung der Funktionen des CANopen-Stacks.....	169
2.8.1	SDOS-Modul.....	169
2.8.2	SDOC-Modul.....	189
2.8.3	PDO-Modul	205
2.8.4	PDOSTC-Modul	219
2.8.5	OBD-Modul	222
2.8.6	COB-Modul	237
2.8.7	NMT-Modul	242
2.8.8	NMT Slave Modul.....	245
2.8.9	NMT Master Modul.....	248
2.8.10	Emergency Consumer Modul.....	253
2.8.11	Emergency Producer Modul.....	258
2.8.12	Heartbeat Consumer Modul	262
2.8.13	Heartbeat Producer Modul	265
2.9	Zusatzmodule für das CANopen.....	269
2.9.1	MPDO Modul - Multiplexed PDO.....	269
2.9.2	CcmMPdo Modul - Multiplexed PDO.....	271
2.10	Bedeutung der Return-Werte und Abortcodes.....	273
2.10.1	CANopen Return Codes	273
2.10.2	SDO-Abortcodes	282
2.10.3	Emergency-Error Codes	283
2.11	Konfiguration und Skalierung.....	284
2.11.1	Konfiguration des CANopen-Stacks.....	284
2.11.2	Konfiguration des Objektverzeichnisses	332
2.12	Besonderheiten für Hardware, Betriebssysteme und Entwicklungsumgebungen	334
2.12.1	Auswahl des Adressraum für die Ablage von Daten	334
2.12.2	Betriebssystem PxROS.....	334
2.12.3	Betriebssystem Linux	336
2.12.4	Betriebssystem Windows	342
3	Hinweise zur Portierung auf andere Zielsysteme.....	355
3.1	Die zentrale Definitionsdatei GLOBAL.H	356
3.2	Auswahl CAN-Treiber.....	359
3.3	Definition der CAN-Bitrate	360
3.4	Target-spezifische Einstellungen.....	362
3.4.1	Definition von Hardware-Eigenschaften	362
3.4.2	Definition Speicherverwaltung, Standardfunktionen.....	362
3.4.3	Definition Target-spezifischer Funktionen	363
3.5	Definition der Variablendarstellung der CPU (Big Endian, Little Endian)	365
3.6	Typische Konfiguration eines CANopen Gerätes als NMT-Slave	366
3.7	Typische Konfiguration eines CANopen Gerätes als NMT-Master	367
4	Hinweise zur CANopen-Zertifizierung	369
5	Index.....	371
6	Glossar.....	376
7	Literaturverzeichnis	378

Bildverzeichnis

Bild 1:	Überblick CANopen	3
Bild 2:	Kommunikationsmodell für PDOs	6
Bild 3:	Mapping von Einträgen aus dem Objektverzeichnis in ein PDO	8
Bild 4:	Datenübertragung von Objektdaten über SDO	16
Bild 5:	Aufbau einer Emergency-Nachricht	19
Bild 6:	Switch Mode Global Service	20
Bild 7:	Configure Bit Timing Service.....	20
Bild 8:	Antwort auf Configure Bit Timing Service.....	20
Bild 9:	Activate Bit Timing Service	20
Bild 10:	Configure Node-ID Service	21
Bild 11:	Antwort auf Configure Node-ID Service	21
Bild 12:	NMT-State Machine eines CANopen-Gerätes	22
Bild 13:	Antwort des NMT-Slaves auf den Node Guard Remote Frame	23
Bild 14:	Antwort des NMT-Slaves auf den Life Guard Remote Frame	24
Bild 15:	Heartbeat-Nachricht	24
Bild 16:	Allgemeine Softwarestruktur	31
Bild 17:	<i>Datenaustausch zwischen Applikation und Objektverzeichnis</i>	40
Bild 18:	NMT-State Machine nach CiA DS-301 V4.02	56
Bild 19:	Ablauf einer CANopen-Applikation.....	63
Bild 20:	Aufrufsequenz der Store Callback-Funktion für einen OD-Bereich.....	113
Bild 21:	Blinkrhythmen der LEDs nach CiA-303-3 (Werte in [ms])	150
Bild 22:	Aufrufreihenfolge der Ereignisse in der LSS-Callback-Funktion	166
Bild 23:	Aufrufreihenfolge der NMT-Ereignisse in der NMT-Callback-Funktion	167
Bild 24:	SDO-Server-Tabelle	170
Bild 25:	Schnittstellen für das Ändern von Parametern eines SDO-Servers	172
Bild 26:	Initiieren eines SDO-Downloads	176
Bild 27:	SDO-Client-Tabelle	189
Bild 28:	Schnittstellen für das Ändern von Parametern eines SDO-Clients	191
Bild 29:	Initiieren eines SDO-Downloads	192
Bild 30:	Abbildung der Variablenfelder	220
Bild 31:	Ereignisse der Objekt-Callback-Funktion bei SDO Zugriffe	236
Bild 32:	Ereignisse der Objekt-Callback-Funktion bei Zugriffen aus der Applikation	236
Bild 33:	Aufbau der CANopen-Software unter Linux.....	337
Bild 34:	Aufbau der CANopen-Software unter Windows	343

Tabellenverzeichnis

Tabelle 1:	Beispiel für Mapping-Parameter für das erste TPDO	7
Tabelle 2:	Mapping-Tabelle vor dem Ändern des Mappings	9
Tabelle 3:	Mapping-Tabelle nach dem Ändern des Mappings	10
Tabelle 4:	Kommunikations-Parameter für das erste TPDO	10
Tabelle 5:	Aufbau einer COB-ID für ein PDO	10
Tabelle 6:	Transmission Type für TPDOs	14
Tabelle 7:	Transmission Type für RPDOs	15
Tabelle 8:	SDO-Transferarten	17
Tabelle 9:	Baudratentabelle nach CiA DSP-305	21
Tabelle 10:	Knotenstatus eines CANopen-Gerätes	23
Tabelle 11:	Konfiguration des Heartbeat Consumers	25
Tabelle 12:	Aufbau eines Objekteintrages im Objektverzeichnis	27
Tabelle 13:	Predefined Master/Slave -Connection-Set [1]	28
Tabelle 14:	Software-Module im CANopen	33
Tabelle 15:	Dateien der CCM-Schicht	35
Tabelle 16:	Objektverzeichnis für Standard-I/O-Geräte	46
Tabelle 17:	Bedeutung der Instanz-Makros als Handle	47
Tabelle 18:	Bedeutung der Instanz-Makros als Handle	48
Tabelle 19:	Übersicht für die Auswahl der benötigten Softwaremodule	49
Tabelle 20:	Legende zur NMT-State Machine (Liste der Ereignisse und Kommandos) ..	57
Tabelle 21:	Unterstützte Kommunikationsobjekte in den NMT-States [4]	57
Tabelle 22:	Parameter der Struktur tCcmInitParam	68
Tabelle 23:	Parameter der Struktur tVarParam	72
Tabelle 24:	Bedeutung von pArg_p in der Error-Callback-Funktion	77
Tabelle 25:	Parameter der Struktur tNmtStateError	78
Tabelle 26:	Parameter der Struktur tPdoError	79
Tabelle 27:	Parameter der Struktur tSdocParam	88
Tabelle 28:	Parameter der Struktur tSdocTransferParam	91
Tabelle 29:	Mögliche Transferstatus-Werte in tSdocState	94
Tabelle 30:	Parameter der Struktur tSdocNetworkParam	97
Tabelle 31:	Parameter der Struktur tPdoParam	100
Tabelle 32:	Ereignisse für die Life Guard Callback-Funktion	105
Tabelle 33:	Zuordnung Subindex des Objektes 0x1010	109
Tabelle 34:	Parameter der Struktur tObdCbStoreParam	112
Tabelle 35:	Aufgaben der Callback-Funktion CcmCbStoreLoadObject	113
Tabelle 36:	Bedeutung der NMT-Kommandos	117
Tabelle 37:	Ereignisse der Master-Callback-Funktion	119
Tabelle 38:	Parameter der Struktur tEmcParam	128
Tabelle 39:	Ereignisse der Callback-Funktion CcmCbEmpcEvent	131
Tabelle 40:	Parameter der Struktur tHbcProdParam	134
Tabelle 41:	Bedeutung der Ereignisse für den Heartbeat Consumer	135
Tabelle 42:	Attribute für Store/Restore	143
Tabelle 43:	äquivalente Funktionen für statisches PDO Mapping	147
Tabelle 44:	Parameter der Struktur tPdoStaticParam	149
Tabelle 45:	Zustände der grünen LED nach CiA-303-3	151
Tabelle 46:	Zustände der roten LED nach CiA-303-3	151
Tabelle 47:	Werte für die Funktion Ccm303SetRunState	153
Tabelle 48:	Werte für die Funktion Ccm303SetErrorState	154
Tabelle 49:	Konfiguration des CANopen für LSS-Master oder Slave	156
Tabelle 50:	Konstanten für den LSS Modus	157
Tabelle 51:	vordefinierte Service Flags für den LSS-Master	160
Tabelle 52:	LSS Service Kommandos für den Service Inquire Identify	161

Tabelle 53:	Callback-Ereignisse des LSS-Masters.....	164
Tabelle 54:	Parameter der Struktur tLssCbParam.....	165
Tabelle 55:	Bedeutung der LSS-Ereignisse.....	166
Tabelle 56:	Auswirkungen der Objekteigenschaften auf den SDO-Transfer.....	173
Tabelle 57:	Abweisung des Initiierens eines SDO-Downloads beim SDO-Server.....	174
Tabelle 58:	Abweisung eines SDO-Segment-Downloads beim SDO-Server.....	175
Tabelle 59:	Abweisung des Initiierens eines SDO-Uploads beim SDO-Server.....	177
Tabelle 60:	Abweisung eines SDO-Segment-Downloads beim SDO-Server.....	177
Tabelle 61:	Auswahl des CRC-Berechnungsalgorithmus.....	178
Tabelle 62:	Parameter der Struktur tSdosInitParam.....	182
Tabelle 63:	Parameter der Struktur tSdosParam.....	186
Tabelle 64:	Abweisung der Download-Response durch den SDO-Client.....	192
Tabelle 65:	Abweisung eines Upload-Segmentes durch den SDO-Client.....	193
Tabelle 66:	Parameter der Struktur tSdocInitParam.....	195
Tabelle 67:	Von SdocNmtEvent verarbeitete NMT-Ereignisse.....	197
Tabelle 68:	Parameter der Struktur tSdocCbFinishParam.....	201
Tabelle 69:	Senden von PDO bezüglich ihres Transmission Types.....	207
Tabelle 70:	Transmission Type für Empfangs-PDOs.....	209
Tabelle 71:	Transmission Type für Empfangs-PDOs.....	209
Tabelle 72:	Konfiguration des OBD Moduls.....	222
Tabelle 73:	Einteilung des Objektverzeichnisses.....	227
Tabelle 74:	Ausführbare Anweisungen auf das Objektverzeichnis.....	227
Tabelle 75:	Knotenstatus eines CANopen Knotens.....	229
Tabelle 76:	Bedeutung der Parameterstruktur tObdCbParam.....	231
Tabelle 77:	Ereignisse der Callback-Funktion für den Objektzugriff.....	233
Tabelle 78:	Bedeutung der Parameterstruktur tObdVStringDomain.....	233
Tabelle 79:	Ermittlung der Anzahl der Kommunikationsobjekte.....	237
Tabelle 80:	Parameter der Struktur tCobParam.....	239
Tabelle 81:	Bedeutung der Kommunikationsobjekttypen.....	239
Tabelle 82:	Bedeutung der NMT Kommandos.....	244
Tabelle 83:	Bedeutung der Parameter der Struktur tNmtmSlaveParam.....	250
Tabelle 84:	Bedeutung der Parameter der Struktur tNmtmSlaveInfo.....	251
Tabelle 85:	Parameter der Struktur tMPdoParam.....	271
Tabelle 86:	SDO Abortcodes.....	282
Tabelle 87:	Emergency Error Codes nach [4].....	283
Tabelle 88:	Funktionspräfix für die CAN-Treiber.....	288
Tabelle 89:	Einstellung der Zeitüberwachung beim PDO-Modul.....	316
Tabelle 90:	Zusätzliche Parameter in der Struktur tCcmlInitParam.....	335
Tabelle 91:	Modulkonfiguration der CANOPMA.DLL und CANOPSL.DLL.....	344
Tabelle 92:	Modulkonfiguration der CCMMA.DLL und CCMSL.DLL.....	344
Tabelle 93:	Thread-Ereignisse für CANopen unter Windows.....	353
Tabelle 94:	Definition der Memory Types für verschiedene Targets.....	357
Tabelle 95:	Liste mit einer Auswahl von CAN-Treibern.....	360
Tabelle 96:	Liste einer Auswahl an Bitraten-Tabellen.....	361
Tabelle 97:	Liste der applikationsspezifischen Makros.....	362
Tabelle 98:	Liste von Target-spezifischen Funktionen.....	364

Einleitung

Dieses Handbuch beschreibt die Anwenderschicht sowie die unterstützten Kommunikationsobjekte des CANopen-Stacks für programmierbare CANopen-Geräte. Geräteprofile werden profilspezifisch in einem weiteren Handbuch beschrieben.

Das *Kapitel 1* vermittelt einige grundlegende Begriffe und Konzepte von CANopen. *Kapitel 2* erläutert die konkrete Umsetzung des CANopen-Protokolls und beschreibt die Anwenderfunktionen, -schnittstellen und Datenstrukturen.

Im *Kapitel 2.12* werden die Hardware-, Betriebssystem- und Entwicklungsumgebungsspezifische Besonderheiten bei der Anwendung des CANopen-Stacks beschrieben.

1 CANopen Grundlagen

CANopen ist eine vom CiA¹ autorisierte und standardisierte Profildfamilie für industrielle Kommunikation mit interoperabel arbeitenden Automatisierungsgeräten auf der Basis von CAN.

1.1 Was ist CANopen?

CANopen definiert eine Anwendungsschicht, ein Kommunikationsprofil sowie verschiedene Applikationsprofile.

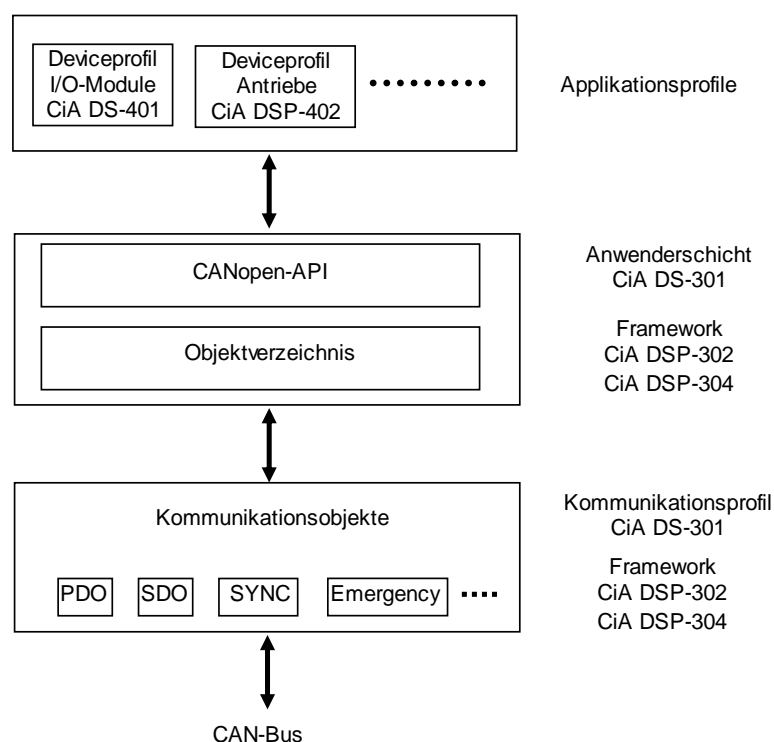


Bild 1: Überblick CANopen

Die Anwendungsschicht² stellt bestätigte und unbestätigte Dienste der Anwendung zur Verfügung und definiert die Kommunikationsobjekte. Dienste werden benutzt, um z.B. von einem Server Daten anzufordern.

¹ CAN in Automation e.V.

² Die Schnittstelle zur Applikation (API) wird durch die Anwendungsschicht nicht definiert und ist abhängig von der herstellereigenen Umsetzung.

Für den Austausch der Daten werden Kommunikationsobjekte verwendet. Kommunikationsobjekte existieren für die Übertragung von Prozess- und Servicedaten, zur Synchronisation von Prozessen oder einer Systemzeit, zur Übertragung von Fehlerzuständen sowie zur Steuerung und Überwachung des Knotenzustandes. Sie werden durch ihren Aufbau, durch Übertragungsarten und ihrem CAN-Identifizierer beschrieben. Die konkreten Parameter eines Kommunikationsobjektes wie z.B. der benutzte CAN-Identifizierer für die Datenübertragung, der Transmission Type¹ einer Nachricht, die Inhibit-Time² oder Event Time³ werden durch das Kommunikationsprofil festgelegt.

Der Ablauf und die Regeln einer Datenübertragung zwischen Kommunikationsobjekten wird durch Protokolle beschrieben (z.B. Download, Upload, ...).

Durch die Anwendungsschicht und die Kommunikationsobjekte ist jedoch noch nicht die Interpretation der übertragenen Daten definiert. Wie die Daten zu interpretieren sind, wird durch die Applikationsprofile bzw. die sogenannten Geräteprofile (Device Profile) beschrieben. Geräteprofile gibt es für verschiedenen Geräteklassen, so z.B. für Ein/Ausgabe-Module (CiA DS-401), Antriebssteuerungen (CiA DSP-402) und Mensch-Maschine-Schnittstellen (CiA DSP-403). Durch die Standardisierung der gerätespezifischen Interpretation von Daten wird es möglich, teilweise austauschbare Geräte herzustellen.

Jedes CANopen-Gerät verfügt als zentrale Datenstruktur über ein Objektverzeichnis (Object Dictionary – OD). Das Objektverzeichnis ist das Koppellement zwischen der Applikation und der Kommunikation über den CAN-Bus. Ein Zugriff auf Einträge kann sowohl durch die Applikation als über den CAN-Bus erfolgen. Einträge können aus Sicht des Programmierers als Variablen oder Felder verstanden werden.

Jedem Eintrag innerhalb des Objektverzeichnisses ist ein Index und Subindex zugeordnet. Darüber kann ein Eintrag eindeutig adressiert werden. Durch den CANopen-Stack werden API-Funktionen⁴ bereitgestellt, um Einträge im Objektverzeichnis zu definieren, zu lesen oder zu schreiben. Mit Hilfe von Kommunikationsobjekten kann über den CAN-Bus ebenfalls auf das Objektverzeichnis zugegriffen werden.

Für jeden Eintrag im Objektverzeichnis sind Eigenschaften zu definieren. Zu den Eigenschaften gehören der Datentyp (UNSIGNED8, REAL32, ...) und verschiedene Attribute wie z.B. das Zugriffsrecht (schreibend, lesend, konstant, ...), die Übertragung der Daten in einem PDO⁵ oder die Überwachung des Wertebereichs mit Bereichsgrenzen⁶.

Die Anwenderschicht und das Kommunikationsprofil ist in CiA DS-301 grundlegend beschrieben. Über CANopen Frameworks werden Erweiterungen von diesem Standard

¹ Der Transmission Type definiert die Verfahrensweise für das Auslösen einer Sendung. Es wird unterschieden zwischen zyklisch und azyklisch sowie synchron und asynchron.

² Die Inhibit Time ist die Zeit zwischen zwei Sendungen, die mindestens vergangen sein muss, bevor erneut gesendet werden darf.

³ Bei einem asynchronen TPDO wird nach Ablauf der Event Time das PDO gesendet.

⁴ Die Definition der API-Funktionen ist herstellerspezifisch.

⁵ Einträge können für die Übertragung als Prozessdatenobjekte in ein PDO „gemapped“ werden.

⁶ Es werden nur die Werte in einen Eintrag geschrieben, die innerhalb der Bereichsgrenzen liegen. Alle anderen Werte werden nicht akzeptiert.

für spezifische Anwendungen definiert. Die Frameworks legen weitere Regeln sowie spezifische Kommunikationsobjekte fest. Beispielsweise werden durch CiA DS-301 Objekt für das Netzwerkmanagement (Node Guarding, Life Guarding, ...) definiert. Die Anwendung dieser Objekte zur Überwachung von CANopen-Geräten wird durch den Framework beschrieben.

Es gibt folgende CANopen Frameworks:

- Framework für programmierbare CANopen-Geräte (CiA DSP-302)
- Framework für sicherheitsrelevante Datenübertragung (CiA DSP-304)

Zusammenfassend bietet CANopen folgende Vorteile:

- herstellernerutraler Standard, offene Struktur
- echtzeitfähige Kommunikation für Prozessdaten ohne Protokolloverhead
- modulare, skalierbare Struktur entsprechend den Anforderungen von einfachsten bis hin zu komplexen Automatisierungsgeräten
- umfangreiche Funktionalität für Kommunikation und Überwachungsaufgaben
- Unterstützung von Systemintegratoren durch Konfiguration und Überwachung
- an Interbus-S, Profibus und MMS orientierte Profile

CANopen bietet u.a. folgende Möglichkeiten zur Auto-Konfiguration von CAN-Netzwerken

- einheitlichen Zugriff auf Geräteparameter
- zyklische und ereignisgesteuerte Kommunikation
- Synchronisation von Geräten vor allem für Mehrfachsysteme

Zur Realisierung von CANopen kompatiblen Geräten bietet *SYS TEC electronic GmbH* folgende Produkte und Dienstleistungen

- eigene CANopen-Implementierung
- unabhängige Beratung
- Hard-und Softwareentwicklung
- Systemintegration und Zertifizierungsunterstützung
- CAN / CANopen Seminare

Die Ingenieure von *SYS TEC electronic GmbH* verfügen über umfangreiche Erfahrungen mit CAN und arbeiten in der SIG "Programmable Devices" und "CANopen Safety" mit.

1.2 Kommunikationsobjekte

Kommunikationsobjekte¹ (COB) werden zur Übertragung von Daten verwendet. Das Kommunikationsprofil definiert die Parameter der einzelnen Kommunikationsobjekte.

Je nach Kommunikationsobjekt gibt es verschiedene Übertragungsarten und Protokolle. Die Verbindung von Kommunikationsobjekten über den CAN-Bus erfolgt mit Hilfe der CAN-Identifizier. Der Empfänger eines Kommunikationsobjektes muss den gleichen COB-Identifizier (COB-ID, CAN-Identifizier) besitzen wie der Sender. Die Kommunikationsobjekte für unbestätigte Protokolle (PDO, Emergency) besitzen einen COB-Identifizier (COB-ID, CAN-Identifizier), Kommunikationsobjekte für bestätigte Protokolle (SDO) besitzen zwei COB-Identifizier (je Richtung ein Identifizier).

1.2.1 PDO - Prozessdatenobjekte

Für die schnelle Übertragung von Prozessdaten eignen sich Prozessdatenobjekte (PDO). Das Kommunikationsmodell für PDOs sieht einen PDO Producer und einen oder mehrere PDO Consumer vor.

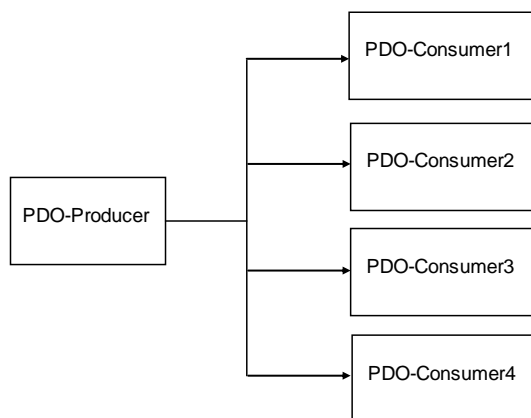


Bild 2: Kommunikationsmodell für PDOs

Der Empfang eines PDOs wird durch einen PDO-Consumer nicht bestätigt. Der PDO-Producer sendet ein PDO, diese PDOs werden als Sende-PDOs (TPDOs) bezeichnet. Der PDO-Consumer empfängt ein PDO, dieses wird als Empfangs-PDOs (RPDOs) bezeichnet. Der erfolgreiche Empfang wird nicht bestätigt. Für einen PDO-Producer können mehrere PDO-Consumer existieren. Ein PDO-Producer wird mit Hilfe seiner

¹ CANopen definiert den verschiedenen Aufgaben und Anforderungen angepasste Kommunikationsobjekte. So werden Prozessdaten mit Prozessdatenobjekte ohne Protokoll-Overhead in einer CAN-Nachricht übertragen. Servicedatenobjekte benutzen für den Datenaustausch zusätzliche Sicherungsmechanismen zur Überwachung des Transfers, die Länge des Objektes kann ein mehrfaches einer CAN-Nachricht sein.

COB-ID einem bzw. mehreren PDO-Consumern zugeordnet. Man bezeichnet das als PDO-Linking¹.

Die Übertragung eines PDOs wird durch ein Ereignis ausgelöst. Ereignisse können die Änderung einer Variablen, der Ablauf einer Zeit oder der Empfang einer Nachricht sein. Für die Übertragung eines PDOs wird direkt eine CAN-Nachricht verwendet, es entsteht kein Protokoll Overhead. Die Länge eines PDOs kann 0 - 8 Datenbytes betragen.

PDOs werden durch ihre sogenannten Mapping-Parameter und Kommunikationsparameter beschrieben. Max. können 512 TPDOs und 512 RPDOs definiert werden. Ein einfaches CANopen-Gerät unterstützt in der Regel vier PDOs. Die konkrete Anzahl der PDOs wird durch die Applikation oder das Geräteprofil festgelegt.

Mapping-Parameter - Wie ist ein PDO aufgebaut?

Ein PDO besteht aus aneinandergefügten Einträgen aus dem Objektverzeichnis. Die sogenannten Mapping-Parameter stellen den Bezug auf diese Einträge her. Ein Mapping-Parameter definiert über Index, Subindex und Anzahl der Bits die Quelle der Daten. Das Ziel, also die Platzierung innerhalb der CAN-Nachricht, wird durch die Reihenfolge der Mapping-Parameter in der Mapping-Tabelle und der Anzahl der Bits der jeweiligen Daten festgelegt.

Beispiel:

Index	Subindex	Objektdaten	Bedeutung
0x1A00	0	4	Anzahl der gemappten Einträge
	1	0x20000310	Der Eintrag auf Index 0x2000, Subindex 3 wird mit einer Länge von 16 Bit auf die Bytes 0 und 1 in der CAN-Nachricht gemappt.
	2	0x20000108	Der Eintrag auf Index 0x2000, Subindex 1 wird mit einer Länge von 8 Bit auf Byte 2 in der CAN-Nachricht gemappt.
	

Tabelle 1: Beispiel für Mapping-Parameter für das erste TPDO

Eine CAN-Nachricht kann max. 8 Datenbytes enthalten. Das heißt, mit Hilfe eines PDOs können Einträge aus dem Objektverzeichnis mit einer Länge von bis zu 8 Bytes übertragen werden.

¹ Bei komplexen Applikation kann das PDO-Linking mit Hilfe von Konfigurationstools grafisch unterstützt werden.

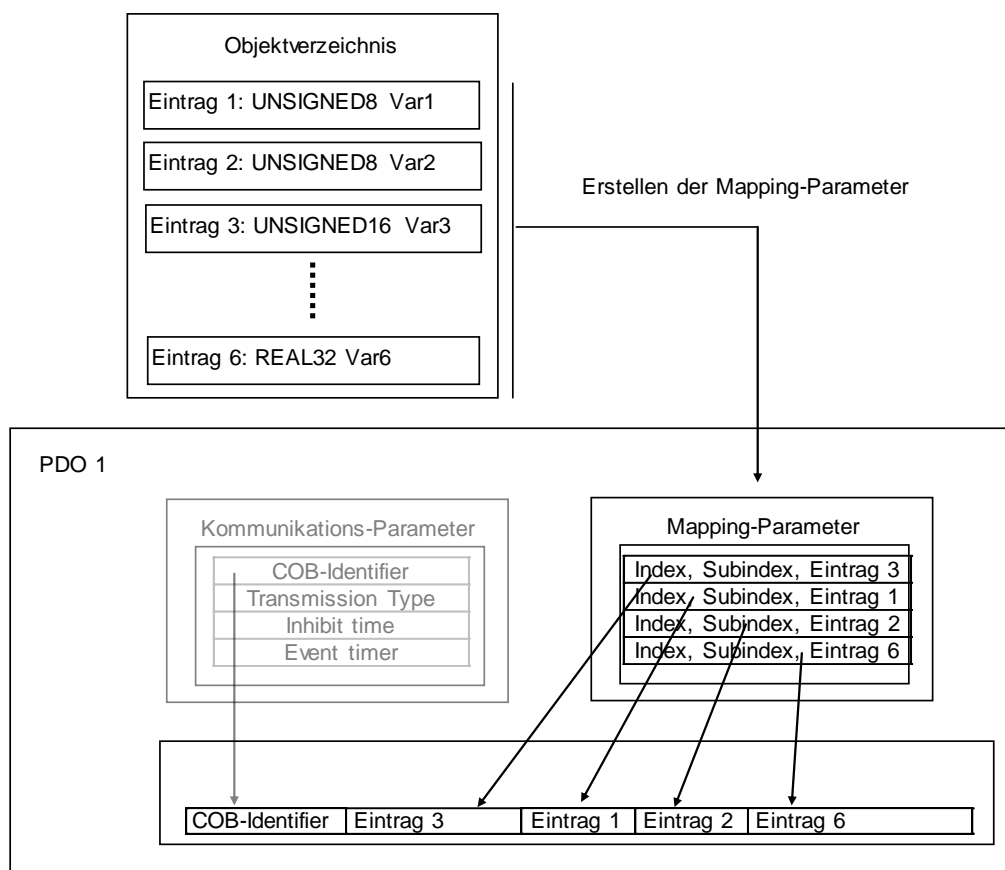


Bild 3: Mapping von Einträgen aus dem Objektverzeichnis in ein PDO

Die Mapping-Parameter sind Einträge im Objektverzeichnis (RPDOs: Index 0x1600 – 0x17FF, TPDOs: 0x1A00-0x1BFF) und können daher über den CAN-Bus mit Hilfe von Servicedatenobjekten (SDO) gelesen und, wenn erlaubt (Schreibrecht wurde erteilt), verändert werden. Das PDO-Mapping kann statisch erfolgen, eine Änderung ist dann nicht möglich. Je nach Geräteprofil oder Anwenderspezifikation kann das PDO-Mapping eines CANopen-Gerätes zur Laufzeit geändert werden. Man spricht dann vom dynamischen Mapping¹. Die Vorgehensweise wird an einem Beispiel erläutert:

¹ Dynamisches Mapping erfordert auch, dass die geänderten Mapping-Parameter in einem nicht-flüchtigen Speicher auf dem Target abgelegt werden. Ist dies nicht möglich, so muss durch einen Systemkonfigurator beim Starten des Netzwerkes das Mapping wiederhergestellt werden.

Ändern der Mapping-Parameter am Beispiel eines TPDOs:

In das erste TPDO werden Einträge des Objektverzeichnisses in folgender Reihenfolge und Länge gemappt:

- Eintrag Index 0x2000, Subindex 3, Länge 16 Bit
- Eintrag Index 0x2000, Subindex 1, Länge 8 Bit
- Eintrag Index 0x2000, Subindex 2, Länge 8 Bit
- Eintrag Index 0x6000, Subindex 6, Länge 32 Bit

Index	Subindex	Objektdaten	Bedeutung
0x1A00	0	4	Anzahl der gemappten Einträge
	1	0x20000310	UNSIGEND16 auf Index 0x2000, Subindex3
	2	0x20000108	UNSIGEND8 auf Index 0x2000, Subindex1
	3	0x20000208	UNSIGEND8 auf Index 0x2000, Subindex2
	4	0x60000620	REAL32 auf Index 0x6000, Subindex6

Tabelle 2: Mapping-Tabelle vor dem Ändern des Mappings

Die resultierende Länge für die CAN Nachricht zur Übertragung des PDOs beträgt 8 Bytes.

Das erste TPDO soll nun anstelle des Eintrags 0x6000, Subindex 6 den Eintrag 0x2000, Subindex 4 mit einer Länge von 16 Bits übertragen. Vor dem Ändern des Mapping-Parameters ist die aktuelle Konfiguration durch Schreiben einer 0 auf Subindex 0 der Mapping-Tabelle zu deaktivieren.¹

Hinweis:

Vor dem Mapping muss sichergestellt werden, dass der Subindex 0 des Mapping-Eintrages den Wert 0 besitzt. Ist dies nicht der Fall, wird beim Mapping-Versuch der SDO-Abort-Code 0x06010000 (nicht unterstützter Objektzugriff) zurückgegeben.

Mit Hilfe eines SDO-Downloads ist dann die neue Konfiguration in der Mapping-Tabelle zu hinterlegen. Durch Schreiben des Wertes 4 auf Subindex 0 der Mapping-Tabelle wird die Konfiguration gültig.

¹ Durch das Deaktivieren der aktuellen Konfiguration werden alle Mapping-Parameter ungültig und gelöscht.

Index	Subindex	Objektdaten	Bedeutung
0x1A00	0	4	Anzahl der gemappten Einträge
	1	0x20000310	UNSIGNED16 auf Index 0x2000, Subindex3
	2	0x20000108	UNSIGNED8 auf Index 0x2000, Subindex1
	3	0x20000208	UNSIGNED8 auf Index 0x2000, Subindex2
	4	0x20000610	UNSIGNED16 auf Index 0x2000, Subindex6

Tabelle 3: Mapping-Tabelle nach dem Ändern des Mappings

Die resultierende Länge für die CAN-Nachricht zur Übertragung des PDOs beträgt nun 6 Bytes.

Kommunikations-Parameter - Welche Übertragungsarten gibt es für ein PDO?

Die Kommunikations-Parameter definieren die Übertragungseigenschaften und die COB-ID (CAN-Identifizier) für die Übertragung eines PDOs. Mit Hilfe der Kommunikations-Parameter kann direkt Einfluss genommen werden auf die Häufigkeit der Übertragung eines PDOs und damit auf die Buslast.

Index	Subindex	Objektdaten	Bedeutung
1800h	0	Anzahl der folgenden Einträge	
	1	COB-ID	CAN-Identifizier für das PDO
	2	Transmission Type	Übertragungsart des PDO
	3	Inhibit Time	min. Sperrzeit für ein TPDO
	4	reserved	reserviert
	5	Event Time	max. Zeit zwischen zwei TPDOs

Tabelle 4: Kommunikations-Parameter für das erste TPDO

Die Kommunikations-Parameter eines PDOs sind Einträge im Objektverzeichnis (RPDOs: Index 0x1400 – 0x15FF, TPDOs: 0x1800-0x19FF) und können daher über den CAN-Bus mit Hilfe von Servicedatenobjekten (SDO) gelesen und, wenn erlaubt, verändert werden.

COB-ID (CAN-Identifizier, Subindex 1)

Die COB-ID dient zur Identifizierung und zur Definition der Priorität eines PDOs beim Buszugriff. Für jede CAN-Nachricht darf es nur einen Sender (Producer) geben. Es können jedoch mehrere Empfänger (Consumer) existieren.

Bit	31	30	29	28 – 11	10 - 0
11-bit-ID	0/1	0/1	0	00000000000000000000	11-bit Identifizier
29-bit-ID	0/1	0/1	1	29-bit Identifizier	

Tabelle 5: Aufbau einer COB-ID für ein PDO

Bit 30 definiert die Zugriffsrechte, Bit 30=0 bedeutet, dass ein Remote-Transmission-Request für dieses PDO erlaubt ist. Mit Hilfe von Bit 31 kann das PDO für eine Verarbeitung deaktiviert werden.

Ab DS301 V4.02 muss erst Bit 31 der COB-ID für PDOs auf 1 gesetzt werden, bevor die Bits 0 bis 29 geändert werden dürfen. Das gleiche gilt auch für das Ändern des Transmission Type (Subindex 2).

Für die ersten 4 PDOs definiert der CANopen-Standard COB-IDs (Default-Identifizier) in Abhängigkeit von der Knotennummer (Pre-defined Connection Set – *siehe Kapitel 1.7*). Bei Verwendung dieser Identifizier ist die Kommunikation zwischen Slaves nur über einen Master möglich. Das erhöht jedoch die Buslast auf dem CAN-Bus, da für die Übertragung von Informationen zwischen zwei Slaves die Nachricht erst vom Slave zum Master und von dort zu einem weiteren Slave weitergeleitet werden muss. CANopen bietet die Möglichkeit, den CAN-Identifizier für ein Kommunikationsobjekt anzupassen. So kann der CAN-Identifizier für ein TPDO ebenfalls für das RPDO verwendet werden. Dann ist es möglich, dass zwei Slaves direkt miteinander kommunizieren ohne einen Master zu benutzen. Man bezeichnet diese Zuordnung der CAN-Identifizier für PDOs als PDO-Linking.

Der Vorgang soll an einem konkreten Beispiel veranschaulicht werden:

Die Inputs 2 und 3 des Gerätes A sollen an die Outputs 1 und 3 des Geräts B übertragen werden. Beide Geräte unterstützen vollständiges Mapping.

Device A:

0x1000	Device Type
.....	
0x6000,1	Input 1, 8 Bit
0x6000,2	Input 2, 8 Bit
0x6000,3	Input 3, 8 Bit
....	

TPDO Mapping-Parameter:

0x1A00,0	Anzahl Einträge	2
0x1A00,1	1.Map Object	0x60000208
0x1A00,2	2.Map Object	0x60000308

TPDO Communication Parameter:

0x1800,0	Anzahl Einträge	2
0x1800,1	COB-ID	0x01C0
0x1800,2	Transmission Type	255
....		

Resultierendes TPDO:

COB-ID	DATA	
0x01C0	Input 2	Input 3

Device B:

0x1000	Device Type
.....	
0x6200,1	Output 1, 8 Bit
0x6200,2	Output 2, 8 Bit
0x6200,3	Output 3, 8 Bit
....	

RPDO Mapping-Parameter:

0x1600,0	Anzahl Einträge	2
0x1600,1	1.Map Object	0x6200,1
0x1600,2	2.Map Object	0x6200,3

RPDO Communication Parameter:

0x1400,0	Anzahl Einträge	2
0x1400,1	COB-ID	0x01C0
0x1400,2	Transmission Type	255
....		

Resultierendes RPDO:

COB-ID	DATA	
0x01C0	Output 1	Output 3

Sende und Empfangs-PDO haben den gleichen CAN-Identifizier 0x01C0. Somit empfängt Device B automatisch das PDO, welches von Device A gesendet wird. Der Empfänger interpretiert die Daten gemäß seinem Mapping. Das heißt, er leitet das erste Byte an Output 1 weiter und das zweite Byte an Output 3. Der Sender wiederum hat in genau diese Bytes seine Inputs 2 und 3 gelegt, womit die Zuordnung korrekt gelöst ist.

Übertragungsart (Transmission Type, Subindex 2)

Der Transmission Type definiert für ein TPDO, unter welchen Bedingungen Daten (z.B. Eingänge) erfasst und ein PDO gesendet, oder für ein RPDO Daten an Ausgänge übernommen werden. Die Übertragung kann dabei ereignisgesteuert, synchronisiert oder "gepollt" erfolgen.

a) TPDOs

Ein TPDO kann zyklisch oder azyklisch gesendet werden. Ein zyklisches Senden erfolgt nach Empfang einer zyklischen SYNC-Nachricht¹. Hierbei spielt es keine Rolle, ob sich die Daten der Eingänge geändert haben. Bei einem TPDO mit einem azyklischen Transmission Type wird das PDO nur beim Eintreten eines Ereignisses übertragen. Ein Ereignis kann der Empfang einer SYNC-Nachricht, eine Änderung der Eingangsdaten, der Ablauf einer Event Timer-Periode² oder ein Remote Frame sein.

Trans. Type	Datenerfassung	PDO senden
0	Die Daten (Eingänge) werden beim Empfang einer SYNC-Nachricht gelesen.	Haben sich die Daten zum vorhergehenden PDO geändert, so wird das PDO gesendet.
1 – 240	Die Daten werden beim Empfang der n-ten SYNC-Nachricht erfasst und gesendet. Der Transmission Type entspricht n.	
241-251	reserviert	
252	Die Daten werden beim Empfang einer SYNC-Nachricht erfasst.	Das PDO wird auf Anforderung durch einen Remote Frame gesendet.
253	Die Applikation ermittelt ständig die Daten.	
254	Die Applikation definiert das Ereignis für das Erfassen der Daten und Senden des PDOs. Ein Ereignis für das Senden des PDOs kann eine abgelaufene Periode des Event Timers sein. Die Periodendauer für den Event Timer wird über Subindex 5 parametrieret. Das Senden eines PDOs (unabhängig vom Ereignis und falls Event Timer parametrieret) startet stets eine neue Periode des Event Timers.	
255	Das Geräteprofil definiert das Ereignis für das Erfassen der Daten und Senden des PDOs. Ein Ereignis für das Senden des PDOs kann eine abgelaufene Periode des Event Timers sein. Die Periodendauer für den Event Timer wird über Subindex 5 parametrieret. Das Senden eines PDOs (unabhängig vom Ereignis und falls Event Timer parametrieret) startet stets eine neue Periode des Event Timers.	

Tabelle 6: Transmission Type für TPDOs

¹ Ein SYNC ist eine Nachricht ohne Dateninhalt, um Kommunikationsobjekte verschiedener Knoten zu synchronisieren. Durch einen SYNC-Producer kann zyklisch eine SYNC-Nachricht übertragen werden.

² Ein Event Timer kann dazu benutzt werden, eine PDO-Übertragung nach Ablauf der Event Time auszulösen, auch wenn sich die Daten des PDOs nicht geändert haben. Die Event Time wird mit Hilfe des Subindex 5 parametrieret.

b) RPDOs

Ein RPDO wird stets empfangen. Die Daten des RPDO werden jedoch nur bei Eintreten eines Ereignisses an die Ausgänge übertragen. Ein Ereignis kann der Empfang einer SYNC-Nachricht oder eine Änderung der empfangen Daten zum vorhergehenden RPDO sein. Optional kann für RPDOs unabhängig vom Transmission Type der Event Timer (Subindex 5) als Überwachungszeit parametrierbar werden. Wenn ein RPDO außerhalb der parametrisierten Event Time eintrifft, so wird die Applikation benachrichtigt (siehe [CcmCbError Kapitel 2.7.1.8](#)).

Trans. Type	PDO empfangen	Datenerfassung
0	Das PDO wird stets empfangen. Die Auswertung und gegebenenfalls Übernahme der Daten erfolgt beim nächsten gültigen SYNC.	Die Daten werden beim Empfang einer SYNC-Nachricht ausgewertet und bei einer Änderung an die Ausgänge übertragen. Die Übertragung des SYNC erfolgt azyklisch.
1 – 240	Das PDO wird stets empfangen. Die Auswertung und gegebenenfalls Übernahme der Daten erfolgt beim nächsten gültigen SYNC.	Die Daten werden beim Empfang der n-ten SYNC-Nachricht ausgewertet und bei einer Änderung an die Ausgänge übertragen. Der Transmission Type entspricht n. Die Übertragung des SYNC erfolgt zyklisch.
241-251	reserviert	
252	reserviert	
253		
254	Das PDO wird stets empfangen.	Die Applikation definiert das Ereignis für die Übernahme der Daten.
255	Das PDO wird stets empfangen.	Das Geräteprofil definiert das Ereignis für die Übernahme der Daten.

Tabelle 7: Transmission Type für RPDOs

Min. Sperrzeit (Inhibit Time, Subindex 3)

Die Sperrzeit (Inhibit Time) ist die minimale Zeit, die zwischen zwei Sendungen des TPDOs verstreichen muss. Dadurch ist es möglich, die Buslast auf ein Minimum zu reduzieren und einen hohen Datendurchsatz zu erzielen.

Die Sperrzeit wird in 100µs -Schritten als UNSIGNED16-Wert hinterlegt.

Event Time (Subindex 5)

a) TPDOs

Nach Ablauf der Event Time wird ein TPDO gesendet, auch wenn sich die Daten des PDOs nicht geändert haben. Der Event Timer wird nach jeder Sendung neu gestartet, unabhängig davon, ob die Sendung nach Ablauf der Event Time oder auf Grund einer Datenänderung ausgelöst wurde. Damit kann eine periodische Übertragung eines PDOs erreicht werden. Eine eingestellte Sperrzeit (Subindex 3) wird nicht berücksichtigt.

Wird die Event Time wieder auf Null zurückgestellt (Null ist der Default-Wert), wird der Event Timer deaktiviert. Das PDO wird nur dann gesendet, wenn sich die Daten geändert haben, wobei die Sperrzeit berücksichtigt wird.

b) RPDO

Für RPDOs mit einem Transmission Type 254 oder 255 kann der Event Timer (Subindex 5) als Überwachungszeit parametriert werden. Wenn innerhalb der parametrierten Event Time kein PDO eintrifft, so wird die Applikation benachrichtigt.

1.2.2 SDO - Servicedatenobjekte

Das Koppellement zwischen der Applikationsschicht und der Kommunikationsschicht ist das Objektverzeichnis. Im Objektverzeichnis können alle Daten des CANopen-Gerätes verwaltet werden. Die Daten werden in sogenannten Einträgen abgelegt. Ein Eintrag wird über Index und Subindex adressiert. CANopen definiert für den Zugriff auf Einträge sogenannte Servicedatenobjekte (SDO).

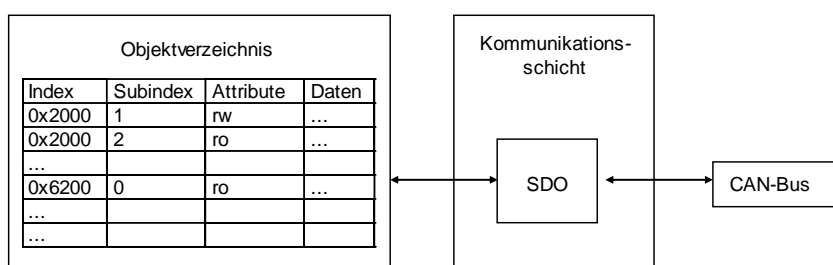


Bild 4: Datenübertragung von Objektdaten über SDO

Das verwendete Kommunikationsmodell beruht auf einer Client-Server-Struktur. Ein lesender oder schreibender Zugriff wird stets durch einen Client initiiert und von einem Server "bedient". Jedes CANopen-Gerät muss daher für einen Zugriff auf das Objektverzeichnis über einen SDO-Server verfügen.

Für die Übertragung werden zwei verschiedene COB-IDs (CAN-Identifizier) benötigt. Die erste COB-ID wird zur Übertragung des Request vom Client zum Server verwendet. Mit der zweiten COB-ID sendet der Server die Response zum Client. Um Kollisionen auf dem CAN-Bus auszuschließen, müssen die beiden Richtungen unterschiedliche CAN-Identifizier benutzen. Im Kommunikationsprofil [4] werden die COB-IDs für den Default-SDO-Server definiert. Jedes CANopen-Gerät kann bis zu 127 SDO-Server besitzen.

Für die Übertragung eines SDOs definiert der CANopen-Standard CiA DS-301 verschiedene Protokolle.

Protokoll	Datenlängen	Bemerkung
expedited transfer	1 – 4 Bytes	Beim Initiieren des Datentransfers werden bereits die Daten übertragen. Dieses Protokoll muss jedes CANopen-Gerät unterstützen.
segmented transfer	1 - >64kByte	Beim Initiieren des Transfers wird nur die Datenlänge übertragen. Die Datenübertragung erfolgt in Segmenten zu je 7 Datenbytes und ein Protokollbyte. Jedes Segment wird mit einer Response bestätigt.
block transfer	1 - > 64kByte	Beim Initiieren des Transfers wird nur die Datenlänge übertragen. Die Datenübertragung erfolgt in Segmenten zu je 7 Datenbytes und ein Protokollbyte. Bis zu 127 Segmente werden in einem Block übertragen. Es werden nur die Blöcke bestätigt. Durch die fehlende Bestätigung für jedes Segment erhöht sich der Datendurchsatz auf dem Bus bei großen Datenmengen.

Tabelle 8: SDO-Transferarten

Das Lesen von Einträgen wird als Upload und das Schreiben eines Eintrages als Download bezeichnet. Eine laufende Übertragung kann durch einen Server oder durch einen Client mit Hilfe eines Abort Transfer Service abgebrochen werden.

1.2.3 Synchronisationsobjekte

Der im CANopen verwendete Synchronisationsmechanismus basiert auf dem Producer-Consumer-Prinzip. Es existiert ein Producer im Netzwerk, der zyklisch die SYNC-Nachricht verschickt. Die SYNC-Nachricht enthält keine Daten.

Der Identifier wird über den Eintrag 0x1005 im Objektverzeichnis spezifiziert. Über diesen Eintrag wird auch definiert, ob das Gerät SYNC-Producer oder SYNC-Consumer ist.

Zwei weitere Objektverzeichnis-Einträge spezifizieren das Zeitverhalten bei der Übertragung. Das Zeitintervall zwischen zwei SYNC-Nachrichten wird im Eintrag Communication Cycle Time (0x1006) definiert. Die Zeitspanne, nach der die TPDOs nach Empfang der SYNC-Nachricht spätestens gesendet sein müssen, wird über den Eintrag SYNC Window (0x1007) definiert.

Für ein Gerät, das synchrone PDOs unterstützt, hat die SYNC-Nachricht folgende Bedeutung:

TPDOs: Aktualisierung der zu sendenden Daten und anschließendes Senden innerhalb des Synchronisationsfensters.

RPDOs: Ausgeben der im vorangegangenen Synchronisationsintervall empfangenen PDOs an die den PDOs zugeordneten Ausgangsdaten.

1.2.4 Time Stamp Object

CANopen verfügt über einen Mechanismus, alle Netzwerkteilnehmer zeitlich zu synchronisieren. Dieser Dienst basiert auf dem Producer-Consumer-Prinzip. Es existiert ein TIME-Producer im Netzwerk, der eine für alle Knoten (Consumer) gemeinsame Zeitreferenz bereitstellt.

Der Identifier der TIME-Nachricht wird über den Objektverzeichnis-Eintrag Time Stamp Object (0x1012) definiert.

1.2.5 Emergency

CANopen unterstützt die Applikation, Fehler über den CAN-Bus zu signalisieren. Fehler können in zwei Kategorien unterteilt werden:

- **Kommunikationsfehler**

Die Netzwerkschicht kann folgende Fehler registrieren:

- zeitlich gehäuftes Auftreten von Fehlern bei der Übertragung von Nachrichten
- Bus-off des CAN-Controllers¹
- Überlauf des Sendepuffers
- Überlauf des Empfangspuffers
- Ausfall des Heartbeat oder des Life Guarding
- CRC-Fehler beim SDO-Blocktransfer

- **Applikationsfehler**

Applikationsfehler sind Fehler wie Kurzschluss, Unterspannung, Temperaturüberschreitung, Code bzw. RAM-Fehler, nicht erlaubte Zustände wie z.B. Alarmer und Störungen.

Applikation und Netzwerkschicht registrieren die Fehler, es ist jedoch Aufgabe der Applikation diese Fehler zu verarbeiten bzw. zu signalisieren. Zur Signalisierung von Fehlern über den CAN-Bus stellt CANopen das Kommunikationsobjekt „Emergency“ zur Verfügung.

¹ Jeder CAN-Controller zählt intern seine Fehler. Bei einer fehlerfreien Kommunikation wird dieser Zähler dekrementiert. Übersteigt der Fehlerzähler ein Limit, so schaltet sich der CAN-Controller ab und nimmt nicht an der weiteren Kommunikation teil, es sei denn die Applikation setzt diesen Zustand zurück.

Identifizier	Daten								
	0	1	2	3	4	5	6	7	
0x080+ Knoten- nummer	Emergency Error Code		Error Register	herstellerspezifische Informationen					
	Index 0x1003		0x1001						

Bild 5: Aufbau einer Emergency-Nachricht

CANopen definiert in DS-301 sowie in den Device Profiles für die Übertragung von Fehlern „Error Codes“. Innerhalb einer Emergency können weitere herstellerspezifische Daten übertragen werden, die den Fehler genauer beschreiben. Der übertragene Fehlercode signalisiert immer den gerade aufgetretenen Fehler. Das Error Register ordnet Fehler Kategorien zu und signalisiert, ob Fehler innerhalb einer Kategorie immer noch anstehen. Das Verschwinden eines Fehlers überträgt ein CANopen-Gerät durch einen zurückgesetzten Fehlercode (High-Teil gleich Null). Gleichzeitig gibt das mit übertragene Error Register Aufschluss darüber, ob weitere Fehler anstehen.

Fehler, die durch einen nicht erlaubten Zugriff auf Objekteinträge oder gestörter Übertragung von SDO-Diensten entstehen, werden durch CANopen mit einem „Abort SDO Transfer Service“ gemeldet.

1.2.6 Layer-Setting-Service (LSS)

Für CANopen-Geräte, die auf Grund der Umgebungsbedingungen (IP65, Ex-Schutz) oder mechanischen Abmessungen keine Vorrichtungen zur mechanischen Konfiguration (z.B. DIP-Schalter) der Basisparameter (Baudrate, Knotennummer) besitzen, definiert CANopen den Layer-Setting-Service (LSS) in CiA DSP-305. Mit Hilfe des Layer Setting Service (LSS) kann ein LSS-Master die Baudrate und Knotennummer eines LSS-Slaves über den CAN-Bus ändern. Dabei versetzt der LSS-Master alle LSS-Slaves in einen Konfigurationsmodus. Dann übermittelt der LSS-Master die neue Baudrate mit dem Dienst „Configure Bit Timing“. Der LSS-Slave antwortet darauf mit einer CAN-Nachricht, in der er dem LSS-Master bekannt gibt, ob er diese neue Baudrate unterstützt oder nicht. Akzeptiert der LSS-Slave diese Baudrate, dann sendet der LSS-Master den Dienst „Activate Bit Timing“ an den LSS-Slave, dass er nach der Zeit „switch_delay“ diese neue Baudrate aktivieren soll. Nach der Aktivierung schaltet der LSS-Master den LSS-Slave wieder in den Operationsmodus.

Der LSS Service ist auch in der Lage die Knotenadresse eines LSS-Slaves zu verändern. Dazu versetzt der LSS-Master den LSS-Slave wieder in den Konfigurationsmodus. Dann gibt er dem LSS-Slave seine neue Knotenadresse bekannt. Der LSS-Slave antwortet darauf, um den LSS-Master mitzuteilen, ob diese Knotenadresse in den unterstützten Bereich liegt. Nach dem Zurückschalten in den Operationsmodus des LSS-Slaves führt dieser einen Software-Reset aus, so dass er die Kommunikationsobjekte mit seiner neuen Knotennummer konfigurieren kann (siehe Kapitel 1.7).

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E5	8	0x04	mod	reserviert					

Bild 6: Switch Mode Global Service

mod: neuer LSS-Modus
 0 = Operationsmodus einschalten
 1 = Konfigurationsmodus einschalten

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E5	8	0x13	tab	ind	reserviert				

Bild 7: Configure Bit Timing Service

tab: gibt an, welche Baudratentabelle verwendet werden soll
 0 = Baudratentabelle, die nach CiA DSP-305 definiert ist
 1 ... 127 = reserviert
 128 ... 255 = kann der Anwender selbst definieren

ind: Index innerhalb der Baudratentabelle, in der die neue Baudrate für das CANopen-Gerät abgelegt ist

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E4	8	0x13	err	spec	reserviert				

Bild 8: Antwort auf Configure Bit Timing Service

err: Fehlercode
 0 = erfolgreich ausgeführt
 1 = Baudrate wird nicht unterstützt
 2 ... 254 = reserviert
 255 = spezieller Fehlercode in **spec**

spec: herstellerspezifischer Fehlercode (wenn **err** = 255)

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E5	8	0x15	delay		reserviert				

Bild 9: Activate Bit Timing Service

delay: relative Zeit bis zum Einschalten der neuen Baudrate in ms

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E5	8	0x11	nid	reserviert					

Bild 10: Configure Node-ID Service

nid: neue Knotenadresse für den LSS-Slave
(Werte von 1 bis 127 erlaubt)

Identifizier	DLC	Daten							
		0	1	2	3	4	5	6	7
0x7E4	8	0x11	err	spec	reserviert				

Bild 11: Antwort auf Configure Node-ID Service

err: Fehlercode
 0 = erfolgreich ausgeführt
 1 = Knotenadresse ungültig (nur Werte 1 bis 127 erlaubt)
 2 ... 254 = reserviert
 255 = spezieller Fehlercode in **spec**

spec: herstellerspezifischer Fehlercode (wenn **err** = 255)

Tabellenindex	Baudrate	SYS TEC Definitionen in CDRV.H
0	1000 kBit/s	kBdi1Mbaud
1	800 kBit/s	kBdi800kBaud
2	500 kBit/s	kBdi500kBaud
3	250 kBit/s	kBdi250kBaud
4	125 kBit/s	kbdi125kBaud
5	100 kBit/s	kBdi100kBaud
6	50 kBit/s	kBdi50kBaud
7	20 kBit/s	kBdi20kBaud
8	10 kBit/s	kBdi10kBaud

Tabelle 9: Baudratentabelle nach CiA DSP-305

Hinweis:

Die CAN-Controller werden (abhängig von der verwendeten Hardware) unterschiedlich getaktet. Daher unterscheiden sich die Registerwerte für die jeweiligen Baudraten.

Es werden im Standard CiA DSP-305 weitere LSS-Dienste beschrieben. Die Beschreibung dieser Dienste soll aber nicht Bestandteil dieses Handbuches sein.

1.2.7 Netzwerkmanagment

Neben den Diensten für die Konfiguration und dem Datenaustausch existiert eine Reihe von Netzwerkdiensten zur Überwachung der Netzwerkteilnehmer. NMT-Dienste (Network Management) erfordern ein CANopen-Gerät im Netz, welcher die Aufgaben des NMT-Masters übernimmt. Dazu gehören unter anderem die Initialisierung der NMT-Slaves, die Verteilung der Identifier, die Knotenüberwachung und das Booten des Netzes.

1.2.7.1 Ablaufsteuerung (NMT State Machine)

CANopen definiert eine State Machine zur Steuerung der Gerätefunktionalität. Die Übergänge zwischen den Zuständen werden durch interne Ereignisse oder durch Dienste des NMT-Masters ausgelöst. Die einzelnen Zustände können mit der Applikation gekoppelt sein.

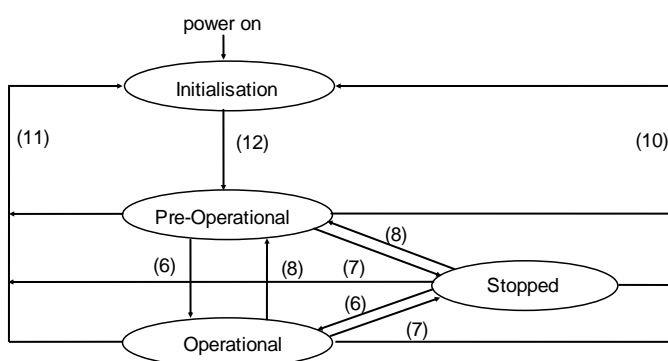


Bild 12: NMT-State Machine eines CANopen-Gerätes

Im Zustand „Initialisation“ werden die CANopen-Datenstrukturen eines Knotens durch die Applikation initialisiert. Der Standard CiA DS-301 definiert dafür eine Reihe von zwingend erforderlichen Einträgen im OD sowie die dafür benötigten Kommunikationsobjekte. Die Identifier für die Kommunikationsobjekte müssen in einer minimalen Gerätekonfiguration dem sogenannten Pre-Defined-Connection-Set entsprechen (siehe Kapitel 1.7). Die Device Profiles definieren weitere Einstellungen für die jeweilige Geräteklasse. Die vordefinierten Einstellungen für die Identifier für Emergency, PDOs und SDOs errechnen sich aus der Knotenadresse, die zwischen 1 und 127 liegen kann, addiert zu einem Basis-Identifier, der die Funktion festlegt.

Nach der Initialisierung wechselt der Knoten selbständig in den Zustand PRE-OPERATIONAL (12). Durch Senden der BOOTUP-Nachricht wird dieser Zustandswechsel einem NMT-Master mitgeteilt. Dieser Zustand ist dadurch gekennzeichnet, dass keine PDOs übertragen werden können. In diesem Zustand kann der Knoten per SDO über den CAN-Bus konfiguriert werden. NMT-Dienste und das Life Guarding stehen zur Verfügung.

In welchem Umfang die Konfiguration über den CAN-Bus mit Hilfe von SDO ausgeführt werden muss, hängt von der Applikation und den vorhandenen Ressourcen des CANopen-Gerätes ab. Können z.B. die Mapping- und Kommunikations-Parameter für PDOs nicht in einem nicht-flüchtigen Speicher gesichert werden, so muss nach erfolgtem

Initialisieren des CANopen-Gerätes diese Parameter über das Netzwerk übertragen werden, insofern sie vom Default-Wert abweichen. Für das CANopen-Gerät kann jedoch ein nicht-flüchtiger Speicher entfallen.

Nach Abschluss der Konfiguration durch die Applikation oder einem NMT-Master wird mit dem NMT-Dienst *Start_Remote_Node* (6) vom Zustand PRE-OPERATIONAL in den Zustand OPERATIONAL gewechselt. Dieser Zustandswechsel löst einmalig die Übertragung aller TPDOs aus unabhängig davon, ob ein Ereignis dafür vorliegt. Jede weitere Übertragung von PDOs erfolgt dann stets in Abhängigkeit von einem Ereignis.

Alle CANopen-Geräte unterstützen außerdem die Dienste *Stop_Remote_Node* (7), *Enter_PRE-OPERATIONAL_State* (8), *Reset_Node* (10), *Reset_Communication* (11). Mit *Reset_Node* werden die applikationsspezifischen Daten und die Kommunikations-Parameter des Knoten zurückgesetzt. Das Zurücksetzen erfolgt mit den PowerOn-Werten bzw. mit den Werten aus dem nicht-flüchtigen Speicher (falls sie dort gesichert wurden). Die CANopen-Datenstrukturen werden mit den Anfangswerten geladen. Ein Zustandswechsel infolge des NMT-Dienstes *Reset_Communication* bewirkt das Zurücksetzen ausschließlich für die Kommunikations-Parameter im CANopen-Stack.

Im Zustand STOPPED ist keine Kommunikation über PDO und SDO möglich. Ausnahmen bilden die NMT-Dienste und das Node bzw. Life Guarding sowie Heartbeat.

1.2.7.2 Node Guarding

Das Node Guarding ist eine Knotenüberwachung, die von einem NMT-Master durchgeführt wird, um den NMT Knotenstatus eines NMT-Slaves abzufragen und um festzustellen, ob dieser noch korrekt arbeitet. Dabei sendet der NMT-Master eine einzelne Node Guard-Nachricht (als Remote Frame mit dem CAN Identifier 0x700 + Knotenadresse des NMT-Slaves) an den Slave. Dieser antwortet darauf mit einer CAN-Nachricht, worin sein derzeitiger NMT Knotenstatus und ein zwischen zwei Nachrichten wechselndes Bit enthalten sind.

Identifizier	DLC	Daten
		0
0x700 + Knotenadresse	1	Status-Byte

Bild 13: Antwort des NMT-Slaves auf den Node Guard Remote Frame

Status-Byte	Knotenstatus
0x00	(BOOTUP)
0x04	STOPPED
0x05	OPERATIONAL
0x7F	PRE-OPERATIONAL

Tabelle 10: Knotenstatus eines CANopen-Gerätes

Das Bit 7 in dem Status Byte beginnt stets mit 0 und wechselt nach jedem Senden seinen Wert. Die Applikation muss aktiv dieses Bit toggeln. Damit wird sichergestellt, dass die Antwort-Nachricht des Slaves nicht nur in einem Full-CAN-Kanal abgelegt wurde. Der NMT-Master erhält dadurch auch eine Rückmeldung, dass die Applikation noch „lebt“.

1.2.7.3 Life Guarding

Alternativ zum Node Guarding kann eine Knotenüberwachung auch mit Hilfe des sogenannte Life Guarding erreicht werden. Im Unterschied zum Node Guarding sendet der NMT-Master zyklisch eine Life Guard-Nachricht (als Remote Frame mit dem CAN Identifier 0x700 + Knotenadresse des NMT-Slaves) an den Slave. Dieser antwortet darauf mit einer CAN-Nachricht, worin sein derzeitiger NMT Knotenstatus und ein zwischen zwei Nachrichten wechselndes Bit enthalten sind. Bei Ausbleiben der Antwort oder unerwartetem Status des Slave wird die NMT-Master Applikation informiert. Weiterhin kann der Slave den Ausfall des Masters detektieren. Das Life Guarding wird mit dem Aussenden der ersten Life Guard-Nachricht des Masters gestartet.

Identifizier	DLC	Daten
		0
0x700 + Knotenadresse	1	Status-Byte

Bild 14: Antwort des NMT-Slaves auf den Life Guard Remote Frame

Bedeutung des Status-Byte entspricht dem des Node Guardings (siehe Tabelle 10).

Die Life Guarding Überwachung beim NMT-Slave Knoten ist deaktiviert, wenn die Life Guard Time (Objekteintrag 0x100C im Objektverzeichnis) oder der Life Time Faktor (Objekteintrag 0x100D im Objektverzeichnis) gleich Null ist.

1.2.7.4 Heartbeat

Heartbeat ist ein Überwachungsdienst, für den kein NMT-Master nötig ist. Heartbeat basiert nicht auf Remote Frames, sondern arbeitet nach dem Producer / Consumer Model.

1.2.7.4.1 Heartbeat Producer

Der Heartbeat Producer sendet zyklisch eine Heartbeat-Nachricht. Als Zykluszeit gilt die im Objektverzeichnis auf Index 0x1017 eingetragene „Producer Heartbeat Time“. Als COB-ID wird 0x700 + Knotennummer verwendet. Im ersten Byte der Heartbeat-Nachricht ist der Knotenstatus des Heartbeat Producers enthalten.

Identifizier	DLC	Daten
		0
0x700 + Knotenadresse	1	Status-Byte

Bild 15: Heartbeat-Nachricht

Bedeutung des Status-Byte entspricht dem des Node Guardings (siehe Tabelle 10).

Im Gegensatz zum Node bzw. Life Guarding wird das Bit 7 nicht nach jedem Senden verändert. Es enthält immer den Wert 0. Dies ist hier auch nicht nötig, weil ein Full CAN-Controller diese Nachricht nicht automatisch senden kann, da dieses Protokoll nicht auf Remote Frames basiert. Es ist Aufgabe der Applikation die Übertragung der Heartbeat-Nachricht auszuführen.

Der Heartbeat Producer ist deaktiviert, wenn die Producer Heartbeat Time (Objekteintrag 0x1017 im Objektverzeichnis) gleich Null ist.

1.2.7.4.2 Heartbeat Consumer

Der Heartbeat Consumer wertet die vom Producer gesendeten Heartbeat-Nachrichten aus. Für jeden zu überwachenden Producer wird jeweils ein Subindexeintrag auf Index 0x1016 im Objektverzeichnis verwendet. Dort ist die Knotennummer des Producers und die „Consumer Heartbeat Time“ eingetragen:

Bit	31...24	23...16	15...0
Wert	0x00	Knotennummer	Consumer Heartbeat Time

Tabelle 11: Konfiguration des Heartbeat Consumers

Die Aktivierung des Consumers erfolgt bei gültigem Eintrag im Objektverzeichnis (Wert ist ungleich 0) und einer empfangenen Heartbeat-Nachricht. Läuft die für einen Producer konfigurierte Heartbeat Time ab, ohne dass eine Heartbeat-Nachricht empfangen wurde, dann wird vom Consumer ein Ereignis an die Applikation gemeldet. Die Überwachung eines Producers wird deaktiviert, indem im entsprechenden Subindexeintrag eine Null konfiguriert wird.

1.3 CANopen Kommunikationsprofil

Das CANopen-Kommunikationsprofil CiA DS-301 [4] definiert die Kommunikations-Parameter für Kommunikationsobjekte, die jedes CANopen-Gerät unterstützen muss. Darüber hinaus wird das Kommunikationsprofil in CANopen-Frameworks und Device Profiles gerätespezifisch ergänzt.

Folgende CANopen Frameworks wurden vom CiA verabschiedet:

- Framework für programmierbare CANopen-Geräte (CiA DSP-302)
- Framework für sicherheits-relevante Datenübertragung (CiA DSP-304)

Eine Auswahl an CANopen Device Profiles:

- Geräteprofil für Ein-/Ausgabe-Module (CiA DSP-401) [7]
- Geräteprofile für Antriebssteuerungen (CiA DSP-402)
- Geräteprofil für Anzeige- und Bediengeräte (CiA DSP-403)
- Geräteprofil für Sensoren und messwertverarbeitende Module (CiA DSP-404)
- Geräteprofil für SPSen nach IEC 61131-3 (CiA DSP-405)
- Geräteprofil für Encode (CiA DSP-406)
- Geräteprofil für Proportionalventile (CiA DSP-408)

Unter Kommunikations-Parameter werden die CAN-Identifizierer eines COB, Sperrzeiten und Übertragungsart für PDOs, ... verstanden. Die Kommunikations-Parameter sind Bestandteil des Objektverzeichnisses und können somit auch von außen gelesen und, wenn freigegeben, geändert werden. Einige Parameter wurden im *Kapitel 1.2* erläutert, für weitere Informationen sei auf die oben erwähnten CANopen-Frameworks und Device Profiles verwiesen.

1.4 Übertragungsprotokolle

Die Übertragung von Kommunikationsobjekten wird durch Übertragungsprotokolle geregelt. Diese werden ebenfalls im CANopen-Kommunikationsprofil CiA DS-301 beschrieben und sind nicht Gegenstand dieses Manuals.

Es sei jedoch erwähnt, dass der Umfang der realisierbaren Protokolle eingeschränkt werden kann. Dadurch ergibt sich eine Einsparung bei den Ressourcen für Code und Daten. Im *Kapitel 2.11* wird beschrieben, auf welche Weise das geschehen kann.

1.5 Objektverzeichnis

Das Objektverzeichnis (Object Dictionary, OD) ist das Koppellement zwischen Applikation und CAN-Bus, um Daten mit einer Applikation und über den CAN-Bus auszutauschen. CANopen definiert Dienste und Kommunikationsobjekte für den Zugriff auf Einträge des Objektverzeichnisses. Jeder Eintrag wird über Index und Subindex adressiert. Die Eigenschaften eines Eintrages im OD wird über seinen Typ (UINT8, UIN16, REAL32, Visible String, Domain, ...) und seine Attribute (read-only, write-only, const, read-write, mappable) festgelegt.

Das OD kann bis zu 65536 Indexeinträge und pro Index 0 – 255 Subindizes enthalten. Sie werden jedoch durch die Kommunikationsprofile bzw. Geräteprofile vordefiniert. Typ und Attribute für Subindizes unterhalb eines Index können verschieden sein.

Index	Subindex	Typ	Attribute
0x2000	0	UINT8	const
	1	UINT32	read-write
	2
	3		

Tabelle 12: Aufbau eines Objekteintrages im Objektverzeichnis

Einträge sind mit Default-Werten vorbelegbar. Mit Hilfe von SDOs kann ein Wert eines Eintrags geändert werden, insofern es das Attribute zulässt (read-write und write-only; nicht für read-only und const). Ein Wert kann auch von der Applikation selbst geändert werden (Attribute read-write, write-only und read-only; nicht für const).

Das OD ist in Abschnitte unterteilt. Der Abschnitt 0x1000 – 0x1FFF wird für die Definition der Parameter für die Kommunikationsobjekte und die Ablage von allgemeinen Informationen (Hersteller, Gerätetyp, Seriennummer, ...) verwendet. Die Einträge ab Index 0x2000 – 0x5FFF sind für die Ablage von herstellerspezifischen Einträgen reserviert. Ab 0x6000 folgen dann Einträge, die gerätespezifisch durch die Device Profiles oder Frameworks beschrieben sind.

CANopen definiert in CiA-301 eine Reihe von „Pflicht“-Einträgen, die ein Gerät stets besitzen muss. Diese Einträge sind mit Mandatory gekennzeichnet. Durch die Device Profiles werden diese „Pflicht“-Einträge ergänzt.

Die Erstellung eines Objektverzeichnisses ist Inhalt eines weiteren Handbuchs (L-1024).

1.6 Fehlerbehandlung und –signalisierung

Zur Fehlersignalisierung existieren im CANopen verschiedene Mechanismen:

- **Emergency-Objekt:** Dies ist eine hoch priorisierte 8-Byte-Nachricht, die die Fehlerinformation enthält. Für eine detaillierte Beschreibung *siehe 1.2.5*.
- **Error Register:** Dies ist ein 1-Byte Eintrag im Objektverzeichnis (Index 0x1001). Darüber wird das Vorhandensein eines Fehlers sowie die Fehlerart signalisiert.
- **Pre-Defined-Error-Field:** Dies ist eine Fehlerliste, die im Objektverzeichnis (Index 0x1003) abgelegt wird. Diese Liste enthält den Emergency Error Code sowie eine gerätespezifische Zusatzinformation. Die Liste ist so aufgebaut, dass der letzte Fehler auf dem Subindex 1 liegt.

1.7 Telegramm-Tabelle (Pre-defined connection set)

Für einfache Netzwerkkonfigurationen mit einem Master und bis zu 127 Slaves definiert CANopen Default-COB-IDs (CAN-Identifizier) in Abhängigkeit vom Dienst und der Knotennummer des Slaves. Für jeden Dienst wurde ein Funktionscode festgelegt. Aus Funktionscode und Knotennummer¹ wird dann die COB-ID gebildet.

CAN-Identifizier										
Bit 10	9	8	7	6	5	4	3	2	1	0
Funktionscode				Knotennummer						

Objekt	Funktionscode	Knotennummer	COB-ID	Index Objektverzeichnis	im
Broadcast-Nachrichten					
NMT	0000	-	0	-	
SYNC	0001	-	0x80	0x1005, 0x1006, 0x1007	
TIME STAMP	0010	-	0x100	0x1012, 0x1013	
Punkt-zu-Punkt-Nachrichten					
Emergency	0001	1-127	0x81-0xFF	0x1014, 0x1015	
TPDO1	0011	1-127	0x181-0x1FF	0x1800	
RPDO1	0100	1-127	0x201-0x27F	0x1400	
TPDO2	0101	1-127	0x281-0x2FF	0x1801	
RPDO2	0110	1-127	0x301-0x37F	0x1401	
TPDO3	0111	1-127	0x381-0x3FF	0x1802	
RPDO3	1000	1-127	0x401-0x47F	0x1402	
TPDO4	1001	1-127	0x481-0x4FF	0x1803	
RPDO4	1010	1-127	0x501-0x57F	0x1403	
Default-SDO (tx)	1011	1-127	0x581-0x5FF	0x1200	
Default-SDO (rx)	1100	1-127	0x601-0x67F	0x1200	
NMT Error Control	1110	1-127	0x701-0x77F	0x1016, 0x1017	

Tabelle 13: Predefined Master/Slave -Connection-Set [1]

¹ Die Knotennummer kann mit lokal oder mit Hilfe der LSS-Dienste über CAN eingestellt werden.

2 CANopen Anwenderschicht

Das vorliegende Kapitel beschreibt die Datenstrukturen und API-Funktionen der *SYS TEC electronic GmbH* –spezifischen Umsetzung des CANopen-Standards CiA DS-301. Bei der Implementierung wurde die Unterstützung weiterer CANopen-Standards vorgesehen, aber auch hardware- und Compiler spezifische Besonderheiten berücksichtigt. Die API bietet Schnittstellen, die vom Anwender zur Erweiterung um gerätespezifische Eigenschaften genutzt werden können. Langjährige Erfahrungen der Ingenieure von *SYS TEC electronic GmbH* bei der Integration oder Portierung des CANopen-Stacks auf verschiedenen Kundenapplikationen haben aber auch zur Erweiterung des Standards geführt. Abweichungen vom CANopen-Standard werden daher besonders gekennzeichnet.

Aufbau, Erstellung und Konfiguration eines Objektverzeichnisses werden in einem separaten Manual beschrieben.

2.1 Softwarestruktur

Bevor die einzelnen API-Funktionen erläutert werden, wird hier die Softwarestruktur und File-Struktur beschrieben. Das schafft die Grundlage, um sich später bei der Implementierung "zurechtzufinden".

Grundsätzlich kann in CANopen-Stack, applikations- und hardware-spezifische Module unterteilt werden.

Der CANopen-Stack ist in einzelne Module gegliedert. Bei der Definition von Modulen wurde der Grundsatz verfolgt, den Umfang des CANopen-Stacks (Funktionsumfang, Datenumfang) skalierbar zu gestalten. Ein Teil der Module sind als Basismodule zu verstehen und sind fester Bestandteil eines CANopen-Stacks. Andere Module müssen zur Realisierung einer Aufgabenstellung nicht vorhanden sein. Das betrifft zum großen Teil CANopen-Dienste, die laut CANopen-Standard, optional oder alternativ für andere Dienste eingesetzt werden können. Um ohne weiteres einzelne Module weglassen zu können, darf es keinen Funktionsaufruf innerhalb der modularisierten Softwareschicht quer zu einem anderen Modul geben, sondern nur zu einer darunterliegenden oder darüberliegenden (als Callback-Funktion)¹.

Die applikationsspezifische Schicht „CANopen Controlling“ (CCM Modul) kontrolliert das Zusammenspiel der einzelnen Module. Die CCM-Schicht ist nicht zwingend für den Einsatz in der Applikation notwendig, stellt jedoch ein einfaches Interface bei der Verwendung von mehreren CANopen-Instanzen zur Verfügung und kapselt sequentielle Funktionsaufrufe von mehreren API-Funktion (z.B. Initialisierung, Definition von PDOs) in Funktionen.

Die hardware-spezifische Schicht kapselt die besonderen Eigenschaften eines CAN-Controllers sowie Mikrocontrollers. Eine Portierung auf eine neue Hardware wird dadurch vereinfacht und kann sich auf das Austauschen des Treibers für den CAN-Controller und die mikrocontrollerspezifische Initialisierung reduzieren.

¹ Dadurch wird es möglich, einzelne Module/Dienste bei der Erstellung wegzulassen, ohne beim Linken der Applikation Fehlermeldungen über unreferenzierte Funktionen zu erhalten.

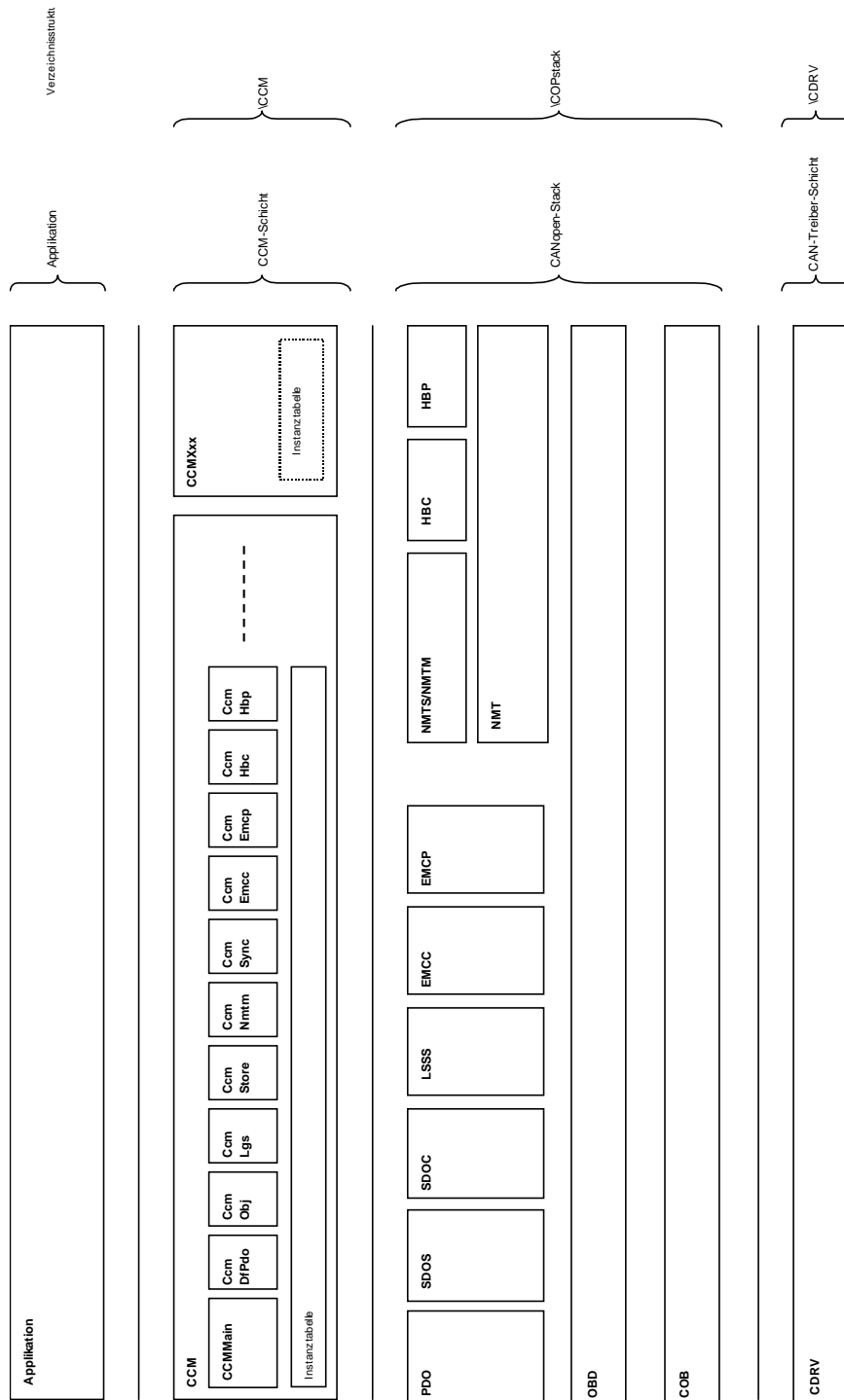


Bild 16: Allgemeine Softwarestruktur

2.1.1 CANopen-Stack

Der CANopen-Stack ist portable, d.h. unabhängig von der Hardware bzw. Applikation, implementiert.

COB	Die COB-Schicht liefert Dienste für die Übertragung von Kommunikationsobjekten und stellt somit eine Basisschicht dar, die in jeder Konfigurationsvariante notwendig ist.
OBD	Das OBD-Modul liefert die globale Datenstruktur für alle CANopen-Instanzen. Hier werden alle vom Anwender konfigurierbaren Datenstrukturen erzeugt. Das betrifft z.B. das Objektverzeichnis sowie Tabellen zur Verwaltung von PDOs und SDO-Server bzw. Clients.
NMT	Dieses Module realisiert die NMT – State Machine und ruft die Callback-Funktion für den NMT – State Wechsel in das CCM – Modul.
NMTS	Dieses Modul liefert Dienste für Node Guarding, Life Guarding und Bootup als NMT-Slave. NMTS kann nicht gleichzeitig mit NMTM in einer CANopen-Instanz verwendet werden.
NMTM	Dieses Modul liefert Dienste für Node Guarding, Life Guarding und Bootup als NMT-Master. NMTM kann nicht gleichzeitig mit NMTS in einer CANopen-Instanz verwendet werden.
HBP	Dieses Modul liefert Dienste für einen Heartbeat Producer. Es ist erlaubt, dass Heartbeat Producer und Consumer in einer CANopen-Instanz gemeinsam existieren. Es ist nicht möglich Heartbeat und Life Guarding für den eigenen Knoten gleichzeitig zu aktivieren.
HBC	Dieses Modul liefert Dienste für einen Heartbeat Consumer. Es ist erlaubt, dass Heartbeat Producer und Consumer in einer CANopen-Instanz gemeinsam existieren. Es ist nicht möglich, Heartbeat und Life Guarding für denselben Knoten gleichzeitig zu aktivieren.
PDO	Dieses Modul liefert Dienste zum Definieren und Übertragen von PDOs. In diesem Modul werden ebenso Dienste für einen SYNC Producer und Consumer realisiert.
PDOSTC	Mit diesem Modul steht der gleiche Dienst wie beim PDO Modul zur Verfügung, jedoch handelt es sich hier um die Realisierung des statischen PDO Mapping.
SDOS	Dieses Modul liefert Dienste zur Verwaltung von SDO Servern und Servicedatenobjekten (SDO) sowie den Protokollen für die Übertragung von Servicedatenobjekten als Server. Die unterstützten Protokolle (expedited, segmented, block) sind konfigurierbar.
SDOC	Dieses Modul liefert Dienste zur Verwaltung von SDO Clients und Servicedatenobjekten (SDO) sowie den Protokollen für die Übertragung von Servicedatenobjekten als Client. Die unterstützten Protokolle (expedited, segmented, block) sind konfigurierbar.
LSSSLV	Dieses Modul liefert Dienste für das Setzen von Bit-Timing und Knotennummer aus Sicht eines LSS-Slaves.
LSSMST	Dieses Modul liefert Dienste für das Setzen von Bit-Timing und Knotennummer aus Sicht eines LSS-Masters
EMCC	Dieses Modul liefert Dienste für einen Emergency Consumer. Es ist erlaubt, dass Emergency Producer und Consumer in einer CANopen-Instanz gemeinsam existieren.

EMCP	Dieses Modul liefert Dienste für einen Emergency Producer. Es ist erlaubt, dass Emergency Producer und Consumer in einer CANopen-Instanz gemeinsam existieren.
------	--

Tabelle 14: Software-Module im CANopen

2.1.2 CDRV - Hardwarespezifische Schicht

Die CDRV-Module stellen dem CANopen-Stack eine einheitliche Schnittstelle für verschiedene CAN-Controller zur Verfügung. Die besonderen Eigenschaften und "Eigenheiten" der CAN-Controller werden so im CDRV-Treiber berücksichtigt. Eine Portierung auf eine neue Hardwareplattform kann durch Erstellen bzw. Anpassen des CDRV-Treibers erfolgen.

Die CDRV-Treiber sind instanzierbar. Interessant wird diese Lösung für Targets mit mehreren CAN-Controllern. Dort können mehrere CANopen-Schnittstellen geschaffen werden, um somit mehrere CANopen-Netze aus einer Applikation heraus zu bedienen. Der Einsatz von mehrkanaligen CAN-Karten auf dem PC (wie pcNetCAN, PCI-CAN oder USB-CANmodule) wird damit möglich.

Bei der Erstellung/Konfiguration des CANopen-Stacks sind folgende Fälle zu berücksichtigen:

- Das Target unterstützt verschiedene CAN-Controller (z.B. Mikrocontroller C167CR mit integriertem CAN-Controller und einem externem CAN-Controller SJA1000). Für jeden CAN-Controller ist ein Hardwaretreiber einzubinden. Von jedem Hardwaretreiber existiert eine Instanz.
- Das Target unterstützt n CAN-Controller (z.B. C167CS mit zwei integrierten CAN-Controllern). Für den CAN-Controller ist ein Hardwaretreiber jedoch mit N Instanzen einzubinden.

Kapitel 2.11 beschreibt die Einstellungen für die Auswahl und Konfiguration der Hardwaretreiber.

Für nähere Informationen zum CDRV-Modul siehe L-1023 „CAN-Treiber – Softwaremanual“.

2.1.3 CCM – Applikationsspezifische Schicht

Die applikationsspezifische Schicht „CANopen Controlling Module“ (CCM) kontrolliert das Zusammenspiel der einzelnen Module. Die CCM-Schicht ist nicht zwingend für den Einsatz in der Applikation notwendig, stellt jedoch ein einfaches Interface bei der Verwendung von mehreren CANopen-Instanzen zur Verfügung und kapselt sequentielle Funktionsaufrufe von mehreren API-Funktion (z.B. Initialisierung, Definition von PDOs) in Funktionen.

Die CCM-Schicht enthält eine ganze Reihe von kleinen Funktionsmodulen. Bei der Erstellung der Applikation kann der Anwender geeignete Module einbinden oder als Vorlage für eigene Erweiterungen der CCM-Schicht verwenden. Diese Erweiterungen betreffen zum Beispiel die Reaktion auf bestimmte Ereignisse, die während eines

CANopen Prozesses auftreten können. In jedem Fall muss nicht der gesamte Umfang an Modulen zu einer Applikation hinzugebunden werden.¹

Modul	Beschreibung	Funktionen
CCMMAIN.C	Dieses Modul enthält die globale Initialisierung und Prozess-Funktionen für das CANopen sowie die Reaktion auf die wichtigsten Ereignisse (Zustandswechsel der NMT-State Machine, Fehler bei der Übertragung)	- CcmInitCANopen - CcmShutDownCANopen - CcmDefineVarTab - CcmConnectToNet - CcmProcess - CcmCbNmtEvent - CcmCbError
CCMOBJ.C	Dieses Modul enthält Funktionen zum Zugriff auf das Objektverzeichnis.	- CcmWriteObject - CcmReadObject
CCMDFPDO.C	Dieses Modul enthält eine Funktion zum Definieren der PDOs über eine Tabelle.	- CcmDefinePdoTab
CCMSTORE.C	Dieses Modul definiert die Funktionen für die Ablage von Objektdaten aus dem Objektverzeichnis in den nicht-flüchtigen Speicher.	- CcmInitStore - CcmStoreCheckArchivState - CcmCbStore - CcmCbRestore - CcmCbStoreLoadObject - CcmStoreRestoreDefault
CCMSYNC.C	Dieses Modul definiert Funktionen für den SYNC Consumer. Es unterstützt die Konfiguration des SYNC.	- CcmInitSyncConsumer - CcmConfigSyncConsumer - CcmConfigSyncProducer - CcmCbSyncReceived - CcmGetSyncCounter
CCMEMCC.C	Dieses Modul definiert Funktionen für den Emergency Consumer. Es unterstützt das Anlegen zu überwachender CANopen-Geräte.	- CcmInitEmcc - CcmEmccDefineProducerTab - CcmCbEmccEvent
CCMEMCP.C	Dieses Modul unterstützt die Konfiguration des Emergency Producers. Es bietet eine Funktion zum Löschen des Pre-defined Error Field.	- CcmConfigEmcp - CcmSenEmergency - CcmClearPreDefinedErrorField - CcmCbEmcpEvent
CCMHBC.C	Dieses Modul definiert die Funktionen für den Heartbeat Consumer. Es unterstützt das Anlegen zu überwachender CANopen-Geräte.	- CcmInitHbc - CcmHbcDefineProducerTab - CcmCbHbcEvent
CCMHBP.C	Dieses Modul unterstützt die Konfiguration des Heartbeat Producers.	- CcmConfigHbp

¹ Das „Weglassen“ von nicht benötigten Modulen wird teilweise von Linkern unterstützt. Ein Modul kann daher durchaus innerhalb eines IDE-Projektes enthalten sein, wird aber, da kein Funktionsaufruf in dieses Modul erfolgt, beim Linken nicht berücksichtigt.

Modul	Beschreibung	Funktionen
CCM303.C	Dieses Modul wird für die Anzeige der internen Zustände im CANopen benötigt. Dabei werden zwei LEDs nach dem CiA Standard CiA-303-3 angesteuert.	<ul style="list-style-type: none"> - Ccm303InitIndicators - Ccm303ProcessIndicators - Ccm303SetRunState - Ccm303SetErrorState
CCMLSS.C	Dieses Modul implementiert API-Funktionen und eine Callback-Funktion für den LSS-Master sowie die Callback-Funktion des LSS-Slave	<ul style="list-style-type: none"> - CcmLssmSwitchMode - CcmLssmConfigureSlave - CcmLssmInquireIdentity - CcmLssmIdentifySlave - CcmCbLssmEvent - CcmCbLsssEvent

Tabelle 15: Dateien der CCM-Schicht

Diese Liste nennt einige wichtige Dateien der CCM-Schicht. Der Umfang der CCM-Schicht wird ständig erweitert und ist daher nicht vollständig. Die Beschreibung der Funktionen, Parameter und Anwendung ist im jeweiligen CCM-Modul zu finden.

2.2 Verzeichnisstruktur

Wo sind welche Dateien zu finden?

Verzeichnis	Inhalt
\ccm	Dateien der CCM-Schicht
\copstack	Dateien des CANopen-Stacks
\cdrv	Dateien der hardware-spezifischen Schicht
\examples	Dateien mit Beispielapplikationen für Mikrocontroller-Projekte und Projekte mit Betriebssystemen. Hier sind die C-Files mit der main-Funktion abgelegt. Zusätzlich gibt es Headerfiles, die Definitionen für die vereinfachte Handhabung des CANopen Stacks (z.B. <i>BDITABDF.H</i> und <i>APPMCO.H</i>) im Unterverzeichnis <code>\examples\include</code> .
\include	Dieses Verzeichnis enthält sämtliche Interface-Dateien für CANopen. In der Applikation sind die Dateien <i>GLOBAL.H</i> , <i>COP.H</i> einzubinden.
\objdicts	Dieses Verzeichnis enthält vordefinierte Objektverzeichnisse für verschiedene Device Profiles. Jedes Objektverzeichnis besteht aus 3 Dateien, die zusammengehören; <i>OBJDICT.C</i> , <i>OBJDICT.H</i> und <i>OBDCFG.H</i> . Die Auswahl des Objektverzeichnisses erfolgt über die Festlegung der entsprechenden Include-Pfade in der Projekteinstellung.
\ds401_3p	Objektverzeichnis für DSP-401 mit 3 RPDOs und 3 TPDOs, NMT-Slave
\ds401_4pstc	Objektverzeichnis für DSP-401, statisches PDO-Mapping, NMT-Slave
\ds401_7p	Objektverzeichnis für DSP-401 mit 7 RPDOs und 7 TPDOs, NMT-Slave
\o401p3m	Objektverzeichnis für DSP-401 mit 3 RPDOs und 3 TPDOs, NMT-Master
\o401p7m	Objektverzeichnis für DSP-401 mit 7 RPDOs und 7 TPDOs, NMT-Master
\o401p3s_hpt	Objektverzeichnis für DSP-401 mit 3 RPDOs und 3 TPDOs, NMT-Slave und High Precision Time Stamp
\domain_string	Objektverzeichnis mit Domain- und String-Objekten im herstellerspezifischen Bereich.
\target	...
\sd-ezdsp-tms320f2812	Dieses Verzeichnis enthält die Projektverzeichnisse für verschiedene Beispielapplikationen. Für jedes Projekt gibt es ein Konfigurationsfile <i>COPCFG.H</i> , das die verwendete Hardware, die unterstützten Eigenschaften und Protokolle definiert. Zusätzlich sind für verschiedene Targets (Unterverzeichnisse) hardware-spezifische Dateien (Startup-Dateien) für die Initialisierung enthalten.
\explorer16	Projekte für Spectrum Digital Eval-Board (TI TMS320F2812, CodeComposer Studio)
\explorer16	Projekte für Microchip Eval-Board Explorer16 (Microchip dsPIC33F, MPLAB IDE)

Verzeichnis	Inhalt
\keil-mcb2300	Projekte für das Keil Eval-Board MCB2300 (NXP LPC2368, Keil IDE)
\phycore-at98c51cc01	Projekte für das phyCORE-CC01 (Atmel AT89C51CC01, Keil IDE)
\pmc14	Projekte für das SYS TEC COMBI modul C14 (Infineon XC161, Keil IDE)
x86	Projekte Intel x86 und kompatible Prozessoren (Betriebssysteme eCos, Linux, Windows)
\KC167	Nur Target-Dateien für KitCON-167 (Infineon C167)
\PC565	Nur Target-Dateien für phyCORE-565 (Freescale MPC565)
...	
\projects ...	Dieses <u>veraltete</u> Verzeichnis enthält die Projektverzeichnisse für verschiedene Beispielapplikationen. Das neue Verzeichnis für die Projekte ist \Target.

Die Include-Files wurden in den C-Files ohne Pfadangabe eingebunden. Für einen fehlerfreien Übersetzungslauf ist der Pfad auf das Include-Verzeichnis und Objdict-Verzeichnis für den Compiler bzw. des IDE-Projektes zu definieren.

2.3 Datenstrukturen

Im folgenden Kapitel werden die verwendeten Datenstrukturen erläutert. Es gibt Datenstrukturen, die für den Datenaustausch zwischen Applikation und CANopen benutzt werden. Andere Datenstrukturen dienen zur Verwaltung und Steuerung des Ablaufs von Verarbeitungsvorgängen, Diensten oder Protokollen innerhalb eines Moduls und werden daher nur der Vollständigkeit erwähnt.

Als Schnittstelle zur Applikation werden folgende Datenstrukturen verwendet:

- Jede CANopen-Instanz¹ verfügt über ein eigenes Objektverzeichnis (OD). Das Objektverzeichnis ist das Koppellement zwischen der Applikation und der Kommunikationsschicht und enthält alle Daten des CANopen-Gerätes. Einträge im Objektverzeichnis werden über Index und Subindex adressiert. Einträge können mit Hilfe von Servicedatenobjekten (SDO, *siehe Kapitel 1.2.2*) über den CAN-Bus oder mit Hilfe von API-Funktion (*siehe Kapitel 2.7.4, Kapitel 2.8.5*) durch die Applikation gelesen und geschrieben werden. Mit Hilfe von API-Funktionen des OBD-Moduls kann die Adresse und die Größe eines Eintrages ermittelt werden. Hier ist dann ein Zugriff über Pointer auf die Daten des Objekteintrages möglich (*siehe Kapitel 2.8.5*).

Einträge des Objektverzeichnisses können jedoch auch mit Variablen oder Feldern der Applikation verknüpft werden. Das hat den Vorteil, dass ein Zugriff auf die Daten ohne die Verwendung einer API-Funktion des CANopen-Stacks oder eines Pointers möglich ist. Ausschließlich auf diese Weise definierte Variablen können durch ein PDO übertragen werden. Die Übertragung per SDO oder der Zugriff mit Hilfe von API-Funktionen wird dadurch nicht eingeschränkt. Auf Grund der Vielseitigkeit in der Anwendung und Änderbarkeit dieser Einträge werden diese als Var-Entry (variabler Objekteintrag) bezeichnet.

Wie oben erwähnt, können diese Var-Entries in PDOs eingebettet werden, vorausgesetzt das Mappen des Eintrages wurde mit dem Attribut `kObdAccPdo` erlaubt. Um eine schnelle und unkomplizierte Änderung einer Variablen an die Applikation zu melden, kann bei der Definition von Var-Entries eine Variablen-Callback-Funktion einschließlich eines Argumentenpointers hinterlegt werden. Eine Änderung des Eintrags über den CAN-Bus durch ein PDO führt dann zum Aufruf dieser Callback-Funktion, wobei der Argumentenpointer als Parameter übergeben wird.

Das Objektverzeichnis ist als Tabelle organisiert. Jeder Tabelleneintrag entspricht einem Index. Diese Index-Tabelle befindet sich im ROM. Innerhalb eines Index gibt es eine weitere Tabelle mit je einem Eintrag pro Subindex. Die Subindex-Tabelle kann sowohl im ROM als auch im RAM abgelegt werden. Der Aufbau der Tabellen ist hinsichtlich Zugriffsgeschwindigkeit und Speicherplatzbedarf optimiert. Das Erstellen eines Objektverzeichnisses wird mit Hilfe von Makros unterstützt.

Ein Eintrag für einen Subindex enthält den Typ des Objekteintrages, die Zugriffsrechte, den Anfangswert, die Bereichsgrenzen und den Pointer auf die Daten.

¹ Der CANopen-Stack und die Hardware-Treiber sind instanzierbar. D.h., der Funktionsumfang von CANopen kann auf mehrere Dateninstanzen angewendet werden. Das schafft die Voraussetzung, mehrere unabhängige CANopen-Schnittstellen auf einem Target zu unterstützen.

Die Verwaltungsstruktur für Var-Entries benötigt auf Grund der ergänzten Eigenschaften zusätzliche Einträge für die Callback-Funktion und den Argumentenpointer.

Bei einem statischen Objektverzeichnis wird während des Compilerlaufs die Verwaltungsstruktur für das Objektverzeichnis angelegt. Ein Ändern der Tabellen ist zur Laufzeit nicht möglich. Daher muss die spätere Verwendung eines Eintrages bekannt sein. Bei einem dynamischen OD werden die Verwaltungsstrukturen des Objektverzeichnisses zur Laufzeit erstellt.

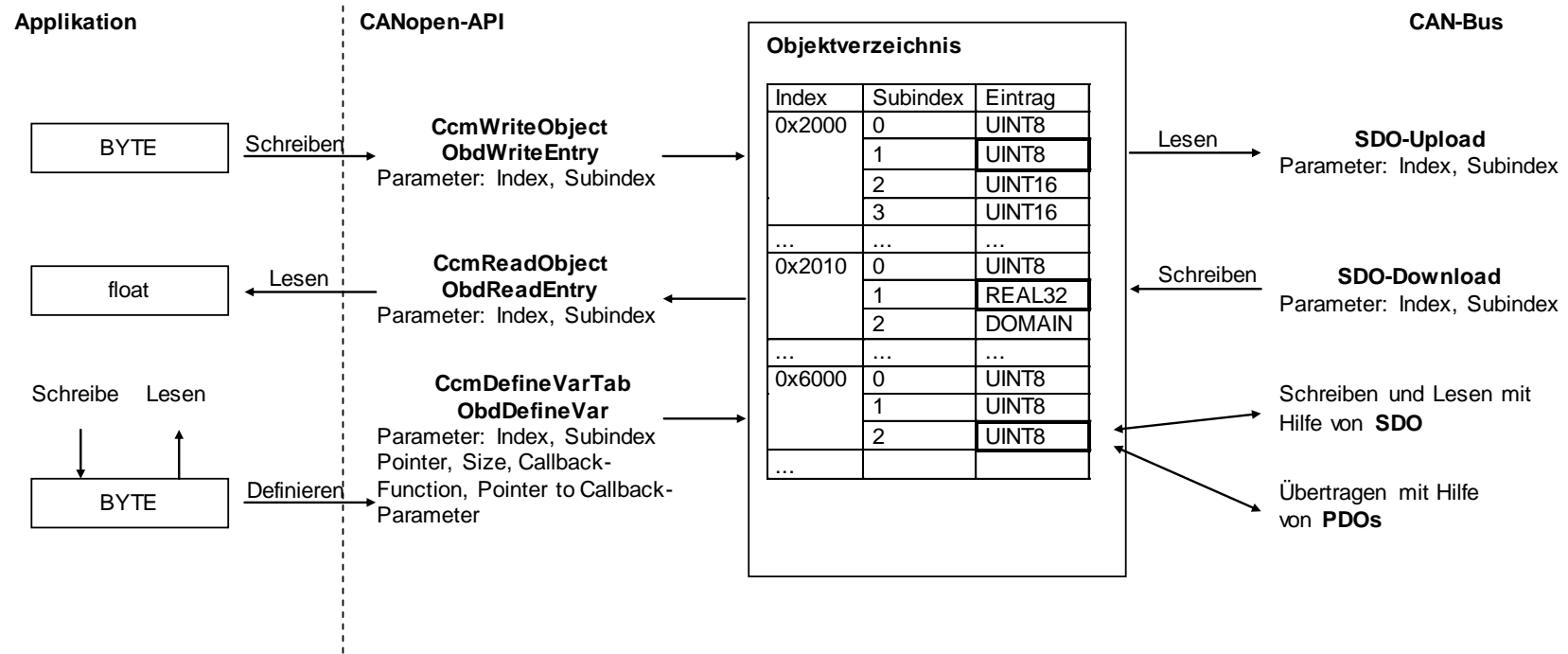


Bild 17: Datenaustausch zwischen Applikation und Objektverzeichnis

Variablen und Felder der Applikation, die mit Hilfe von PDOs zu übertragen sind oder als DOMAIN oder Strings deklariert wurden, müssen im Objektverzeichnis Var-Entries registriert werden¹. Die CCM-Schicht stellt dafür die Funktion [CcmDefineVarTab](#) zur Verfügung, die diesen Vorgang mit Hilfe von Tabellen automatisiert.

- Für die Übergabe von komplexen Parametern an Funktionen werden Strukturen verwendet. Die Strukturen werden als Parameter einer Funktion erläutert. Vor einem Aufruf der Funktion ist eine solche Struktur zu initialisieren.

Strukturen und Tabellen für die Verwaltung interner Abläufe und Einstellungen:

- Für die Verwaltung von PDOs, SDO-Server, SDO-Client, ... werden intern Tabellen verwendet. Die Größe der Tabellen (d.h. Anzahl der Einträge) richtet sich nach der definierten Anzahl von PDOs, SDO-Server usw. (*siehe Kapitel 2.11*). Die Tabellen werden durch den Compiler beim Übersetzen des Objektverzeichnisses erstellt. Um Speicherressourcen und Rechenzeit zu sparen, sind einzelne Einträge aus den Tabellen direkt verknüpft mit den Einträgen des Objektverzeichnisses. Die Tabellen werden durch die Initialisierungsfunktion des jeweiligen Moduls initialisiert.
- Jedes Modul verfügt über eine Instanztafel. Die Instanztafel enthält alle Variablen für ein Modul. Die Variablen werden benutzt, um Verarbeitungszustände und Parameter innerhalb eines Moduls abzulegen. Bis auf das CCM-Modul ist eine Instanztafel nur innerhalb eines Moduls gültig und daher als „static“ deklariert. Das Anlegen und Verarbeiten von Einträgen einer Instanztafel wird durch Makros unterstützt (*siehe Kapitel 2.5*).

¹ Beim Anlegen des Objektverzeichnisses werden die Datenstrukturen für die Verwaltung angelegt, nicht jedoch der Speicher (d.h. die Variable oder das Feld) für die Ablage von Daten.

2.4 Das Objektverzeichnis

Das Objektverzeichnis wird über drei Dateien *OBJDICT.C*, *OBJDICT.H* und *OBDCFG.H* definiert. Eine genaue Beschreibung zur Erstellung eines eigenen Objektverzeichnisses wird im Handbuch *L-1024* beschrieben. Wenn Sie das CANopen als Library erhalten haben, dann wird auch das Objektverzeichnis als Library ausgeliefert. In diesem Fall sind die oben genannten drei Dateien nicht in der Verzeichnisstruktur zu finden.

Bei der Auslieferung der CANopen-Software werden Standardvarianten angeboten diese werden in den nachfolgenden Kapiteln aufgelistet.

Bei der Auflistung der Objekte werden Abkürzungen für den Objekttyp, den Datentyp und für die Attribute verwendet. Diese Abkürzungen haben folgende Bedeutungen:

Objekttypen:

var = Objekt enthält einen Wert, auf den per SDO oder von der Applikation aus zugegriffen werden kann (Variable).

Datentypen:

u8 = Unsigned 8-Bit
u16 = Unsigned 16-Bit
u32 = Unsigned 32-Bit
i8 = Interger 8-Bit
i16 = Interger 16-Bit
i32 = Integer 32-Bit
vstr = Visible String

Attribute:

ro = read only; Objekt kann per SDO gelesen und von der Applikation gelesen und geschrieben werden.
rw = read write; Objekt kann per SDO und von der Applikation gelesen und geschrieben werden.
wo = write only; Objekt kann per SDO und von der Applikation nur geschrieben werden.
const = constant; Objekt kann per SDO und von der Applikation nur gelesen und nicht geschrieben werden.
mapp = Objekt kann in ein PDO gemappt werden
store = Objekt kann im nicht-flüchtigen Speicher gesichert werden (*siehe Kapitel 2.7.6*)

2.4.1 Objektverzeichnis für Standard-I/O-Geräte

Es stehen mehrere Objektverzeichnisse für Standard-I/O-Geräte zur Verfügung. Das OD **ds401_3p** beinhaltet 3 TPDOs und 3 RPDOs, das OD **ds401_7p** beinhaltet 7 TPDOs und 7 RPDOs. Sonst gleichen sich die beiden Objektverzeichnisse in allen Objekten. Das OD **o401p3m** besitzt ebenfalls 3 TPDOs und 3 RPDOs, enthält jedoch als CANopen Master die Objekte 0x100C und 0x100D nicht. Dafür enthält es zusätzlich das Objekt 0x1016 (für den Heartbeat Consumer) und das Objekt 0x1280 (für den ersten SDO Client). **O401p7m** gleicht **o401p3m**, hat jedoch 7 TPDOs und 7 RPDOs. Für die CANopen Kits stehen die beiden ODs **o401p2ks** (für den Slave) und **o401p2km** (für den Master) zur Verfügung. Beide beinhalten nur 2 TPDOs und 2 RPDOs wobei der Master keinen Heartbeat Consumer besitzt (Objekt 0x1016 fehlt).

Index	Sub-index	Name	Objekt-typ	Daten-typ	Attribute	Default-Wert
0x1000		device type	var	u32	ro	0x000F-0191
0x1001		error register	var	u8	ro	0
0x1003		predefined error field	array			
	0	number of errors	var	u8	ro, rw mit 0 zum löschen	0
	1..4	standard error field	var	u32	ro	0
0x1005		COB-ID SYNC	var	u32	rw, store	0x080
0x1006		communication cycle period	var	u32	rw, store	0
0x1007		synchronous window length	var	u32	rw, store	0
0x1008		manufacturer device name	var	vstr	const	"CANopen Slave"
0x1009		manufacturer hardware version	var	vstr	const	"V1.00"
0x100A		manufacturer software version	var	vstr	const	"V5.xx"
0x100C ¹		guard time	var	u16	rw, store	0
0x100D ¹		life time factor	var	u8	rw, store	0
0x1010		store parameters	array			
	0	largest subindex supported	var	u8	const	3
	1	save all parameters	var	u32	rw	0
	2	save communication parameters	var	u32	rw	0
	3	save application parameters	var	u32	rw	0
0x1011		restore default parameters	array			
	0	largest subindex supported	var	u8	const	3
	1	restore all default parameters	var	u32	rw	0

Index	Sub-index	Name	Objekt- typ	Daten- typ	Attribute	Default- Wert
	2	restore communication default parameters	var	u32	rw	0
	3	restore application default parameters	var	u32	rw	0
0x1012		COB-ID time stamp message	var	u32	rw, store	0x100
0x1014		COB-ID emergency message	var	u32	rw, store	0x8000- 0000
0x1015		inhibit time EMCY	var	u16	rw, store	0
0x1016 ²		consumer heartbeat time	array			
	0	number of entries	var	u8	const	5
	1..5	consumer heartbeat time	var	u32	rw	0
0x1017		producer heartbeat time	var	u16	rw, store	0
0x1018		identity object	record			
	0	number of entries	var	u8	const	4
	1	vendor ID	var	u32	ro	0x3F
	2	product code	var	u32	ro	0
	3	revision number	var	u32	ro	0x0A05
	4	serial number	var	u32	ro	1
0x1280 ³		client SDO parameter	record			
	0	number of entries	var	u8	const	3
	1	COB-ID client to server	var	u32	rw, store	0x8000- 0000
	2	COB-ID server to client	var	u32	rw, store	0x8000- 0000
	3	node ID server	var	u8	rw, store	0x00
0x1400		receive PDO parameter	record			
	0	largest subindex supported	var	u8	ro	5
	1	COB-ID used by PDO	var	u32	rw, store	0x8000- 0000
	2	transmission type	var	u8	rw, store	255
	3	inhibit time	var	u16	rw, store	0
	5	event timer	var	u16	rw, store	0
0x14xx
0x1600		receive PDO mapping	record			
	0	number of mapped application objects in PDO	var	u8	rw, store	0
	1..8	PDO mapping for the n- th application object to be mapped	var	u32	rw, store	0
0x16xx
0x1800		transmit PDO parameter	record			
	0	largest subindex supported	var	u8	ro	5

Index	Sub-index	Name	Objekt-typ	Daten-typ	Attribute	Default-Wert
	1	COB-ID used by PDO	var	u32	rw, store	0x8000-0000
	2	transmission type	var	u8	rw, store	255
	3	inhibit time	var	u16	rw, store	0
	5	event timer	var	u16	rw, store	0
0x18xx
0x1A00		transmit PDO mapping	record			
	0	number of mapped application objects in PDO	var	u8	rw, store	0
	1..8	PDO mapping for the n-th application object to be mapped	var	u32	rw, store	0
0x1Axx
0x6000		read input 8-bit	array			
	0	number of inputs 8-bit	var	u8	const	16
	1..16	read input	var	u8	ro, mapp	0
0x6100 ⁴		read input 16-bit	array			
	0	number of inputs 16-bit	var	u8	const	8
	1..8	read input	var	u16	ro, mapp	0
0x6200		write output 8-bit	array			
	0	number of outputs 8-bit	var	u8	const	16
	1..16	write output	var	u8	rw, mapp	0
0x6300 ⁴		write output 16-bit	array			
	0	number of outputs 16-bit	var	u8	const	8
	1..8	write output	var	u16	rw, mapp	0
0x6401 ⁴		read analogue input 16-bit	array			
	0	number of analogue input 16-bit	var	u8	const	4
	1..4	analogue input	var	i16	ro, mapp	0
0x6402 ⁴		read analogue input 32-bit	array			
	0	number of analogue input 32-bit	var	u8	const	4
	1..4	analogue input	var	i32	ro, mapp	0
0x6411 ⁴		write analogue output 16-bit	array			
	0	number of analogue output 16-bit	var	u8	const	4
	1..4	analogue output	var	i16	rw, mapp	0
0x6412 ⁴		write analogue output 32-bit	array			
	0	number of analogue output 32-bit	var	u8	const	4
	1..4	analogue output	var	i32	rw, mapp	0

Index	Sub-index	Name	Objekt-typ	Daten-typ	Attribute	Default-Wert
0x6424 ⁴		analogue input interrupt upper limit integer	array			
	0	number of analogue inputs	var	u8	const	4
	1..4	analogue input	var	i32	rw, store	0
0x6425 ⁴		analogue input interrupt lower limit integer	array			
	0	number of analogue inputs	var	u8	const	4
	1..4	analogue input	var	i32	rw, store	0

Tabelle 16: Objektverzeichnis für Standard-I/O-Geräte

Fußnoten:

- 1) Nicht vorhanden in den ODs für den Master o401p3m, o401p7m und o401p2km.
- 2) Nur vorhanden in den ODs für den Master o401p3m und o401p7m.
- 3) Nur vorhanden in den ODs für den Master o401p3m, o401p7m und o401p2km.
- 4) Nicht vorhanden in den ODs für das CANopen Einstiegs-Kit o401p2ks und o401p2km.

2.5 Instanziierung der CANopen-Schicht

Der CANopen-Stack, das CCM-Modul als auch die Hardware-Treiber sind instanzierbar. D.h., der Funktionsumfang von CANopen kann auf mehrere Dateninstanzen angewendet werden. Das schafft die Voraussetzung, mehrere unabhängige CANopen-Schnittstellen auf einem Target zu unterstützen.

Für das Erzeugen von Instanzen werden alle globalen und statischen Variablen in sogenannten Instanztabellen abgelegt. Jeder Tabelleneintrag entspricht genau einer CANopen-Instanz. Ein Eintrag wird durch eine Struktur beschrieben. Die Funktionen erhalten beim Aufruf einen Verweis auf die zu verarbeitende Instanz in Form eines Instanz-Pointers oder eines Instanz-Handles.

Die Anzahl der Instanzen und somit die Anzahl der Einträge in einer Instanztabelle wird beim Übersetzungslauf als Konstante definiert. Diese Konstante heißt `COP_MAX_INSTANCES` für das CANopen und wird in der Datei `COPCFG.H` definiert. Für die Instanziierung der CAN-Treiber gibt es dafür ein eigenes Define namens `CDRV_MAX_INSTANCES`, die ebenfalls in der Datei `COPCFG.H` definiert ist (siehe Kapitel 2.11.1). Der Zugriff auf Strukturelemente einer Instanz erfolgt ausschließlich über Makros.

Bei der Definition von mehreren Instanzen wird bei einem Funktionsaufruf stets ein Verweis auf die zu verarbeitende Instanz in Form einer Adresse auf eine Instanztabelle (siehe Kapitel 2.5.2) oder Instanz-Handles (siehe Kapitel 2.5.1) als Parameter übergeben. Wurde nur eine Instanz definiert, so entfällt dieser Parameter. In der Beschreibung der API-Funktionen ist dieser Parameter stets aufgeführt. Die Definition des Instanz-Parameters erfolgt mit Hilfe von Makros. Diese Makros werden durch den Präprozessor des Compilers in Abhängigkeit der definierten Anzahl von Instanzen aufgelöst.

Beispiel:

Wird nur eine Instanz verwendet, so entfällt der Instanz-Parameter.

```
CcmConnectToNet ();
```

Bei mehreren Instanzen ist der Instanz-Parameter anzugeben.

```
CcmConnectToNet (HandleInstance0);
```

In der Datei *INSTDEF.H* sind Makros für die Deklaration und Übergabe von Instanzparametern an Funktionen sowie Makros für den Zugriff auf Einträge einer Instanztabelle definiert. Die Verwendung dieser Makros unterstützt das Schreiben von Funktionen, die unabhängig von der Anzahl der Instanzen sind. In der Regel ist die Anzahl der Instanzen (CANopen-Schnittstellen) durch die Applikation fest definiert.

2.5.1 Verwendung von Instanz-Handle

Ein Instanz-Handle wird als Verweis auf die aktuelle Instanz beim Aufruf einer Funktion der CCM-Schicht oder beim Aufruf einer Callback-Funktion der Applikation verwendet.

Werden mehrere Instanzen in einer CANopen Applikation verwendet, dann haben die Instanz-Makros den folgenden Inhalt:

Das Makro ...	entspricht im C-Source ...
Für die Deklaration von Parametern in einer Parameterliste einer Funktion:	
CCM_DECL_INSTANCE_HDL	tCopInstanceHdl InstanceHandle
CCM_DECL_INSTANCE_HDL_	tCopInstanceHdl InstanceHandle,
CCM_DECL_PTR_INSTANCE_HDL	tCopInstanceHdl MEM* pInstanceHandle
CCM_DECL_PTR_INSTANCE_HDL_	tCopInstanceHdl MEM* pInstanceHandle,
Für die Übergabe von Parametern an die aufzurufende Funktion:	
CCM_INSTANCE_HDL	InstanceHandle
CCM_INSTANCE_HDL_	InstanceHandle,

Tabelle 17: Bedeutung der Instanz-Makros als Handle

Wird nur eine Instanz verwendet, dann sind die Instanz-Makros ohne Inhalt.

2.5.2 Verwendung von Instanz-Pointer

Ein Instanz-Pointer wird als Verweis auf die aktuelle Instanz beim Aufruf einer Funktion einer tieferliegenden Schicht (z.B. Aufruf der Funktion [SdosProcess](#) durch eine Funktion aus dem CCM-Modul) verwendet.

Werden mehrere Instanzen in einer CANopen Applikation verwendet, dann haben die Instanz-Makros den folgenden Inhalt:

Das Makro ...	entspricht im C-Source ...
Für die Deklaration von Parametern in einer Parameterliste einer Funktion:	
MCO_DECL_INSTANCE_PTR	void MEM* pInstance
MCO_DECL_INSTANCE_PTR_	void MEM* pInstance,
MCO_DECL_PTR_INSTANCE_PTR	void MEM* MEM* pInstancePtr
MCO_DECL_PTR_INSTANCE_PTR_	void MEM* MEM* pInstancePtr,
Für die Übergabe von Parametern an Modul-eigene Funktion:	
MCO_INSTANCE_PTR	pInstance
MCO_INSTANCE_PTR_	pInstance,
MCO_PTR_INSTANCE_PTR	pInstancePtr
MCO_PTR_INSTANCE_PTR_	pInstancePtr,
Für die Übergabe von Parametern an Modul-fremde Funktion:	
MCO_INSTANCE_PARAM(par)	par
MCO_INSTANCE_PARAM_(par)	par,

Tabelle 18: Bedeutung der Instanz-Makros als Handle

Wird nur eine Instanz verwendet, dann sind die Instanz-Makros ohne Inhalt.

2.6 Hinweise für das Erstellen einer Applikation

Der Anwender der CANopen-Schicht muss wissen, welche Funktionen in welchen Betriebszuständen auszuführen sind, um die gewünschte Funktionalität zu erzielen. Durch Erläutern der internen Mechanismen und Abläufe wird das Verständnis für die gewählte Lösung oder Einschränkung aufgebaut. Weiterhin wird erläutert, welche Aufgaben durch den Anwender zu erledigen sind, um die gewünschte Funktion zu erreichen.

Für die korrekte Funktion des CANopen-Protokolls ist eine bestimmte Reihenfolge bei der Ausführung der Funktionen einzuhalten. Anderenfalls sind evtl. Datenstrukturen nicht vorhanden bzw. nicht initialisiert, ein Funktionsaufruf führt dann zu einem Fehler oder undefinierten Verhalten.¹

Die Reihenfolge bei der Ausführung der verschiedenen Funktionen wird mit den einzelnen Zuständen der NMT-State Machine gekoppelt. Diese Vorgehensweise hat den Vorteil, dass der Zustand sehr genau beschrieben werden kann. Die NMT-State Machine ist durch den Standard CiA DS-301 definiert, es gibt eine Reihe von Sekundärliteratur mit ergänzenden Hinweisen und Beispielen, um das Verständnis zu vertiefen.

Dieses Kapitel beschreibt ganz allgemein den Aufbau einer Applikation. Die Applikation wird dazu in Bereiche unterteilt, die Bereiche werden nummeriert. Die folgenden Kapitel mit der Beschreibung der einzelnen Module nehmen dann Bezug auf diese Bereiche, um die Stellen zu benennen, die für die Integration des gewünschten Moduls oder CANopen-Dienstes anzupassen sind.

¹ In der Debug-Version werden verschiedene Plausibilitätschecks durchgeführt und im Fehlerfall als PRINTF-Ausgabe gemeldet.

2.6.1 Auswahl der benötigten Module und Konfiguration

Für das Erstellen eines CANopen-Gerätes werden verschiedene Dienste von CANopen und Eigenschaften für Objekteinträge benötigt. Beim Erwerb einer CANopen-Library ist der Umfang der unterstützten Dienste definiert und nicht änderbar, dagegen kann bei Einsatz des CANopen-Source die Auswahl der Dienste konfiguriert und somit den applikativen Erfordernissen angepasst werden.

Dienste sind in Modulen des CANopen-Stacks gekapselt. Die folgende Übersicht zeigt, welches Modul für welchen Dienst benötigt wird. Bei Anwendung des Source Codes sind die benötigten Module bei der Codegenerierung einzubeziehen und die passenden Einstellung im File *COPCFG.H* (siehe Kapitel 2.11) vorzunehmen. Module, die als Basismodule aufgeführt sind, müssen stets bei der Codegenerierung einbezogen werden. Optionale Module können, falls der jeweilige Dienst nicht benötigt wird, entfallen.

Dienst/Merkmal	Modul	Kategorie
Initialisieren CANopen	<i>CCMMAIN.C</i>	Basismodul
Verwalten von PDOs	<i>PDO.C</i> oder <i>PDOSTC.C</i>	optional
SDO-Server	<i>SDOSCOMM.C</i>	Basismodul
SDO-Client	<i>SDOC.C</i>	optional
CRC-Berechnung beim SDO-Blocktransfer	<i>SDOCRC.C</i>	optional
Heartbeat-Producer	<i>HBP.C</i>	optional
Heartbeat-Consumer	<i>HBC.C</i>	optional
Emergency-Producer	<i>EMCP.C</i>	Basismodul
Emergency-Consumer	<i>EMCC.C</i>	optional
Life Guarding Master	<i>NMTM.C</i>	Die Module Nmtm.c und Nmts.c sind in SO-877 stets alternativ zu verwenden.
Life Guarding Slave	<i>NMTS.C</i>	
Node Guarding Master	<i>NMTM.C</i>	
Node Guarding Slave	<i>NMTS.C</i>	
LSS – Slave	<i>LSSSLV.C</i>	optional
LSS-Master	<i>LSSMST.C</i>	optional
Anlegen von Kommunikationsobjekten zur Übertragung von Nachrichten	<i>COB.C</i>	Basismodul
Funktionen für den Zugriff auf Objekteinträge	<i>OBD.C</i>	Basismodul
NMT-State Machine	<i>NMT.C</i>	Basismodul
Funktionen für den Zugriff auf maschinenabhängige Datenformate für den Mikrocontroller Xxx	<i>AMIXXX.C</i>	Basismodul
Treiber für den jeweiligen CAN-Controller (Xxx) bzw. Betriebssystem	<i>CDRVXXX.C</i>	Basismodul
Interfacefunktionen für die Anpassung von hardware-spezifischen CAN-Controller-Anschaltungen	<i>CCIXXX.C</i>	Basismodul
Baudratentabelle mit den unterstützten Baudraten	<i>BDITABXXX.C</i>	Basismodul

Tabelle 19: Übersicht für die Auswahl der benötigten Softwaremodule

Module der CCM-Schicht sind bis auf das Modul *CCMMAIN.C* optional. Die Module unterstützen den Anwender bei der Konfiguration der Applikation. Es liegt in der Hand des Anwenders zu entscheiden, welche Module eingebunden werden.

Ein Emergency Producer wird stets unterstützt, auch wenn laut Standard CiA DS-301 dieser Dienst optional ist. Es hat sich jedoch in der Praxis gezeigt, dass für die Diagnose eines fehlerhaften Zustands in einer Applikation dieser Dienst unverzichtbar ist.

Der Umfang der unterstützten Dienste und Protokolle kann innerhalb eines Moduls weiter reduziert werden (*siehe Kapitel 2.11*). Das ist besonders dann interessant, wenn sehr wenig Code und Datenspeicher auf dem Target zur Verfügung steht. Dazu sind Einstellungen im File *COPCFG.H* vorzunehmen.

Der CANopen-Stack ist unabhängig von einem spezifischen CAN-Controller implementiert. Für die Anbindung eines CAN-Controllers ist das spezifische Treiber-Modul *CdrvXxx.c* und evtl. ein Modul *CciXxx.c* einzubinden. Das Modul *CciXxx.c* wird dann benötigt, wenn die Anschaltung eines Stand-Alone-Controllers an einen Mikrocontroller auf unterschiedliche Weise erfolgen kann.¹ Weitere Informationen dazu sind im Handbuch „CAN-Treiber“ (*L-1023*) zu finden.

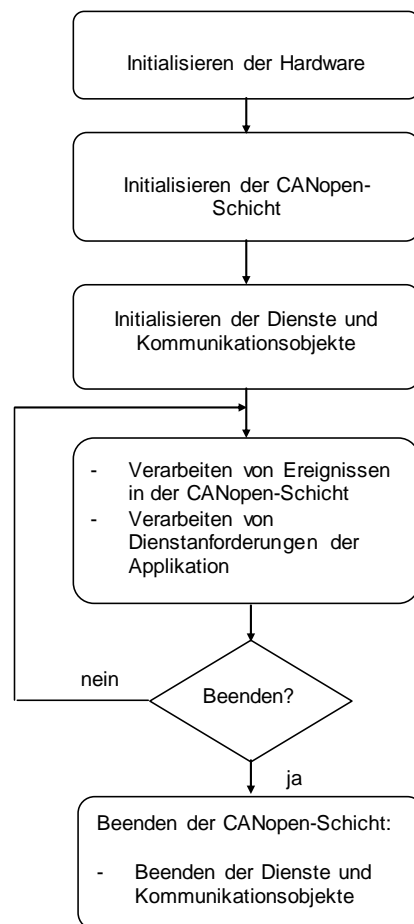
Die Baudratentabelle enthält Werte für die Baudraten-Register BTR0 und BTR1 für verschiedene Baudraten. Die Berechnung dieser Werte basiert auf der Taktfrequenz des CAN-Controller und nicht der Quarz- bzw. Oszillatorfrequenz. Die Taktfrequenz des CAN-Controllers wird in der Regel durch Teilen oder Vielfachen aus der Oszillatorfrequenz des CAN-Controllers bzw. Mikrocontrollers abgeleitet.

¹ Der INTEL 82C527 kann sowohl über ein paralleles als auch ein serielles Interface angebunden werden.

2.6.2 Ablauf einer CANopen-Applikation

Eine CANopen-Applikation hat folgenden prinzipiellen Ablauf:

1. Initialisieren der Hardware
2. Anlegen der Datenstrukturen (Objektverzeichnis, Tabellen, Strukturen, Variablen, Instanzen) und Verknüpfen der konfigurierten Module, Konfiguration der Knotennummer
3. Initialisieren der Dienste (Kommunikations-Parameter, Anlegen der Kommunikationsobjekte)
4. Verarbeiten von Ereignissen und Ausführen von Dienstanforderungen der Applikation
5. Beenden der CANopen-Schicht, wenn erforderlich



2.6.2.1 Initialisieren der Hardware

Noch vor dem Initialisieren der CANopen-Schicht ist durch die Applikation die Hardware zu initialisieren. CANopen benötigt für die korrekte Funktion eine Zeitbasis, aufgelöst in 100µs, sowie in der Debug-Version eine Schnittstelle zur Ausgabe von Fehlermeldungen. Wird ein Fehler auf Grund einer fehlerhaften Konfiguration oder Parametrierung festgestellt, so ruft die CANopen-Schicht in der Debug-Version in einigen Fällen die Standard C-Funktion **printf** auf. Die Ausgabe des seriellen Datenstroms auf ein Terminal ist gegebenenfalls für das Target anzupassen.

Der globale Interrupt des Mikrocontrollers ist freizugeben, die Interrupt Service Routine des CAN-Controllers ist einzubinden (Festlegen des Interrupt-Vektors und der Interrupt-Priorität).

Bei der Auslieferung der CANopen-Schicht liegen für verschiedene Targets die Files *TARGET.C* bei. In diesen Files sind Funktionen zum Initialisieren eines Timers, der seriellen Schnittstelle sowie zur Freigabe des Globale und CAN-Controller-spezifischen Interrupts enthalten.

Beispiel für das Initialisieren der Hardware:

```
void main (void)
{
...
// disable globale interrupt
TgtEnableGlobalInterrupt (FALSE);

// init target (timer, interrupts, ...)
TgtInit (); // init general
TgtInitSerial (); // init serial interface
TgtInitTimer (); // init system time
TgtInitCanIsr (); // init CAN-controller interrupt

// enable global interrupt
TgtEnableGlobalInterrupt (TRUE);

...
}
```

Beim Einsatz eines Betriebssystems wird die Hardware in der Regel durch das Betriebssystem initialisiert. Evtl. sind Funktionen für das Initialisieren des Betriebssystems auszuführen.

2.6.2.2 Initialisieren der CANopen-Schicht, Anlegen der Datenstrukturen

Jedes Modul des CANopen-Stacks bzw. der Cdrv-Schicht (CAN-Treiber CdrvXxx.c) verfügt über eine Funktion für das Initialisieren und Parametrieren des Moduls. Die Init-Funktion ist für jede Instanz einmal auszuführen. Dieser Schritt ist Voraussetzung dafür, dass weitere Funktionen innerhalb des Moduls korrekt arbeiten.

Die Funktion [CcmlInitCANopen](#) führt dieses grundlegende Initialisieren der CANopen-Schicht aus. Innerhalb dieser Funktion werden die Init-Funktionen der einzelnen Module aufgerufen. Es sind nun die Voraussetzungen geschaffen, Variablen der Applikation (z.B. für die Ablage von Prozessdaten) mit der CANopen-Schicht zu verknüpfen.

Beispiel für das Initialisieren der CANopen-Schicht:

Im folgenden Beispiel wird für ein CANopen-Gerät (Mikrocontroller Infineon C167CR) mit einer Instanz das Initialisieren der CANopen-Schicht vorbereitet und ausgeführt.

Zunächst muss die Headerdatei *COPINC.H* eingebunden werden. Diese definiert alle API-Funktionen, Strukturen und Konstanten für den CANopen Stack. Sie bindet automatisch weitere Headerdateien ein, die für den CANopen Stack benötigt werden.

Das CANopen erhält die Knotennummer 0x41, als Baudrate wird 1 Mbit/s gewählt. Der Takt für den CAN-Controller beträgt 10MHz bei einer CPU-Frequenz von 20MHz. Bei der Auswahl der Baudraten-Tabelle ist zu beachten, dass sich die aufgeführte Taktfrequenz auf die Taktfrequenz des CAN-Controllers bezieht und nicht der Oszillatorfrequenz des CAN-Controllers bzw. der CPU entspricht. Bei Mikrocontrollern mit einem integrierten CAN-Controller bzw. bei Stand-Alone-CAN-Controllern wird in der Regel der Takt aus der Oszillatorfrequenz durch Teilen bzw. Vervielfachen abgeleitet.

```
#include "copinc.h"
#include "include/bditabdf.h" // BDI table definitions
#include "include/apmco.h" // helping macros for application

#define NODE_ID 0x41 // Node ID is 0x41

// define index to baudrate table for 1 Mbit/sec
#define BAUDRATE kBd11Mbaud

// define the baudrate table for 10MHz CAN controller clock
// extern CONST WORD ROM awCdrvBdiTable10_g[9];
// #define CDRV_BDI_TABLE_PTR awCdrvBdiTable10
// #define CDRV_BDI_TABLE_SIZE sizeof(awCdrvBdiTable10_g)
```

Viele Baudratentabellen sind in der Headerdatei *BDITABDF.H* in Abhängigkeit des verwendeten Targets vordefiniert. In den meisten Fällen kann die Definition aus dieser Headerdatei verwendet werden. Fügen Sie anstelle der Definition von Konstanten *CDRV_BDI_TABLE_PTR* und *CDRV_BDI_TABLE_SIZE* die Headerdatei *BDITABDF.H* aus dem Verzeichnis *C:\system\cop\examples\include* ein.

Eine Akzeptanzfilterung ist in unserem Beispiel nicht vorgesehen. Jeder CAN-Identifizierer darf empfangen werden. In einer als „const“ deklarierten **tCcmInitParam**-Struktur werden die Parameter abgelegt. Durch die Applikation wird eine Funktion definiert (*TgtEnableCanInterrupt1*), die das Sperren und Freigeben des CAN-Controller-Interrupts ausführt. Eine Callback-Funktion (**AppCbNmtEvent**) für das Verarbeiten von Zustandsänderungen der NMT-State Machine wird definiert. Die Funktion **ObdInitRam** für das Initialisieren der internen Datenstrukturen muss stets eingetragen werden.

```
CONST tCcmInitParam ROM CcmInitDefaultParam_g =
{
    NODE_ID, // node id
    BAUDRATE, // index to baudrate
    CDRV_BDI_TABLE_PTR, // pointer to BDI table (CAN bit rate)
    CDRV_BDI_TABLE_SIZE, // size of BDI table
    0xFFFFFFFFL, // Acceptance Mask Register
    0x00000000L, // Acceptance Code Register
    {{0}}, // CAN-Controller address
    TgtEnableCanInterrupt1, // function pointer to enable CAN interrupt
    AppCbNmtEvent, // pointer to NMT-Callback function
    ObdInitRam // init function for OD
};
```

In diesem Beispiel sind alle Einträge der Struktur fest und müssen zur Laufzeit nicht geändert werden. Daher wird die Struktur im ROM abgelegt. Muss zur Laufzeit die Knotenadresse oder Baudrate geändert bzw. mit einem DIP-Switch konfiguriert werden, so ist die Struktur im RAM abzulegen, so dass die Einträge (**m_blnitNodeId**, **m_BaudIndex** usw.) durch die Applikation verändert werden können.

In unserem Beispiel wird die Struktur **CcmInitDefaultParam_g** in das RAM kopiert und mit Hilfe des Makros **APP_INIT_HW_PARAM** verändert. Dieses Makro ist in der Datei **APPMCO.H** in Abhängigkeit des verwendeten Targets definiert. Hier wird z.B. die Basisadresse des CAN-Controllers eingetragen.

Durch Aufruf der Funktion [CcmInitCANopen](#) wird die CANopen-Schicht initialisiert. Der erste Aufruf von [CcmInitCANopen](#) erfolgt stets mit dem Parameter **kCcmFirstInstance**. Dadurch wird die Funktion veranlasst, die interne Instanztafel zu löschen.

Das Objektverzeichnis wird angelegt, die Einträge mit Default-Werten initialisiert (Default-Werte können bei der Definition des Objektverzeichnisses hinterlegt werden). Einträge des Objektverzeichnisses sind jedoch noch nicht mit der Applikation verknüpft.

```
tCcmInitParam MEM CcmInitParam_g;

void main (void)
{
...
    // enable global interrupt
    TgtEnableGlobalInterrupt (TRUE);

    // copy default values to RAM
    COP_MEMCPY (&CcmInitParam_g, CcmInitDefaultParam_g, sizeof (CcmInitParam_g));

    // init hardware specific parameters for the first CAN controller
    APP_INIT_HW_PARAM (CcmInitParam_g, 0);

    // initialize first instance of CANopen
    Ret = CcmInitCANopen (&CcmInitParam_g,
        kCcmFirstInstance);
    if (Ret != kCopSuccessful)
    {
        goto Exit;
    }

...
Exit:
...
}
```


2.6.2.3 Konfiguration der Knotennummer mit LSS

Bei der Benutzung des LSS-Dienstes zur Konfiguration der Knotennummer ist zu beachten, dass noch vor einem Wechsel vom NMT-State INITIALISATION nach PRE-OPERATIONAL die LSS-State Machine auszuführen ist, wenn die Knotennummer ungültig ist.

Sollte die Applikation nach Ausführen von [CcmInitCANopen](#) noch keine gültige Knotennummer haben (laut LSS Spezifikation CiA DS-305 V1.01 ist 0xFF als ungültige Knotennummer definiert), dann ist die Funktion [CcmProcessLssInitState](#) zyklisch in einer Schleife aufzurufen. Hier wartet CANopen, bis über den LSS-Service eine gültige Knotennummer initialisiert wurde. Ist das geschehen, dann gibt diese Funktion einen Rückgabewert ungleich kCopLssInvalidNodeId zurück. Dann darf diese zyklische Schleife beendet und [CcmConnectToNet](#) gerufen werden. Mit [CcmConnectToNet](#) wird dann die NMT-State Machine gestartet. Während dieses Aufrufes wird die NMT-Callback-Funktion in der Applikation mit mehreren Ereignissen aufgerufen. Was innerhalb dieser Ereignisse zu tun ist, wird im *Kapitel 2.6.2.4* näher beschrieben.

Beispiel:

```
...
Ret = CcmInitCANopen (&CcmInitParam_g, CcmFirstInstance);
if (Ret != kCopSuccessful)
{
    goto Exit;
}

...
// run LSS init state process until NodeId is valid
do
{
    Ret = CcmProcessLssInitState ();
} while (Ret == kCopLssInvalidNodeId);
...

Ret = CcmConnectToNet ();
if (Ret != kCopSuccessful)
{
    goto Exit;
}
```

Wird die Knotennummer während dem zyklischen Ausführen [CcmProcess](#) erneut geändert, dann wird automatisch (im CCM Modul) eine Re-Initialisierung der CANopen-Schicht durchgeführt. Dabei werden die Ereignisse **kNmtEvResetNode**, **kNmtEvResetCommunication** und **kNmtEvEnterPreOperational** erneut in der NMT-Callback-Funktion der Applikation gemeldet.

Achtung:

Seit CANopen Version 5.57 ist die Funktion [CcmProcessLssInitState](#) nur als Platzhalter und aus Kompatibilitätsgründen implementiert. Wird der CANopen Stack mit einer ungültigen Knotenadresse gestartet, dann werden dennoch die Funktionen [CcmConnectToNet](#) und [CcmProcess](#) ausgeführt, wobei die NMT State Machine im Zustand INITIALISING stehen bleibt. Wird dem CANopen Stack von außen (per LSS Master) eine gültige Knotenadresse vergeben, dann wird diese erst mit dem NMT Kommando „Reset Communication“ aktiv. Dieses Ereignis wird der Applikation über die NMT Callback-Funktion gemeldet.

2.6.2.4 Initialisieren der Dienste und Kommunikationsobjekte, Ausführen von Diensten

Im vorhergehenden Schritt wurden die grundlegenden Datenstrukturen erzeugt und initialisiert. Das CANopen-Gerät besitzt eine gültige Knotennummer. Der jetzt folgende Schritt verknüpft die Variablen der Applikation mit den Einträgen im Objektverzeichnis und initialisiert die Dienste und Kommunikationsobjekte für die Datenübertragung. Hierbei werden die auszuführenden Schritte (Aktionen) den States innerhalb der NMT-State Machine zugeordnet.

Nach Ausführung der Funktion [CcmInitCANopen](#) befindet sich das CANopen-Gerät im Zustand *INITIALISING* der NMT-State Machine.

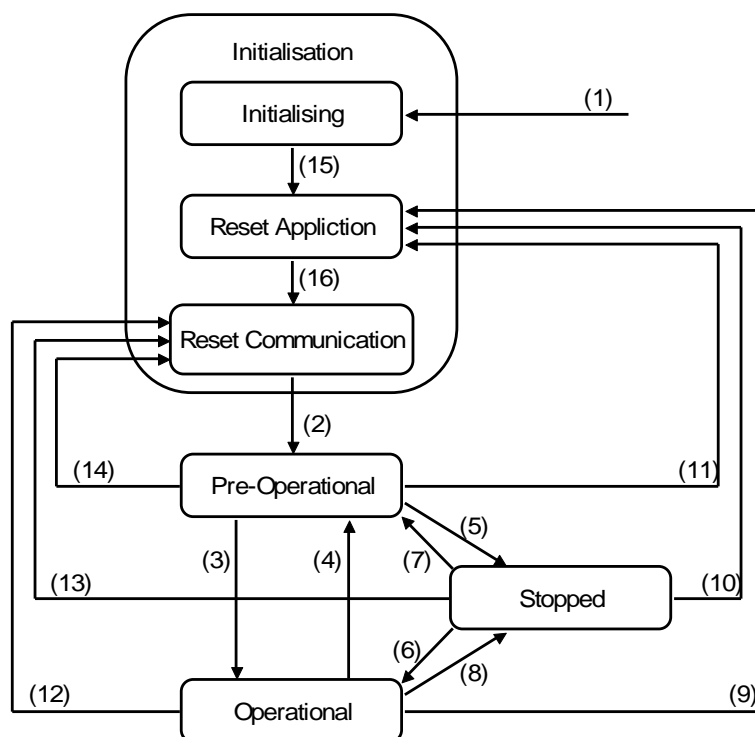


Bild 18: NMT-State Machine nach CiA DS-301 V4.02

	Ereignis	NMT Ereignis als Konstante
(1)	Power-on oder Hardware-Reset	kNmtEvEnterInitialising
(2)	Automatischer Wechsel nach PRE-OPERATIONAL nach Abschluss von INITIALISATION	kNmtEvEnterPreOperational
(3), (6)	NMT – Befehl: Start_Remote_Node	kNmtEvEnterOperational
(4), (7)	NMT – Befehl: Enter_Pre_Operational_Node	
(5), (8)	NMT – Befehl: Stop_Remote_Node	kNmtEvEnterStopped
(9), (10), (11)	NMT – Befehl: Reset_Node	kNmtEvResetNode
(12), (13), (14)	NMT – Befehl: Reset_Communication	kNmtEvPreResetCommunication kNmtEvResetCommunication kNmtEvPostResetCommunication
(15)	Automatischer Wechsel nach RESET-APPLICATION nach Abschluss von INITIALISING	kNmtEvResetNode
(16)	Automatischer Wechsel nach RESET-COMMUNICATION nach Abschluss von RESET-APPLICATION	kNmtEvPreResetCommunication kNmtEvResetCommunication kNmtEvPostResetCommunication

Tabelle 20: Legende zur NMT-State Machine (Liste der Ereignisse und Kommandos)

Laut Standard CiA DS-301 sind in den verschiedenen NMT-States folgende Dienste zu unterstützen:

Kommunikationsobjekte	INITIALISING	PRE-OPERATIONAL	OPERATIONAL	STOPPED
PDO			X	
SDO		X	X	
SYNC		X	X	
Time Stamp		X	X	
Emergency		X	X	
Boot-Up	X			
NMT		X	X	X

Tabelle 21: Unterstützte Kommunikationsobjekte in den NMT-States [4]

Die Funktion [CcmConnectToNet](#) startet die Ausführung der State Machine mit dem State *INITIALISING*. Nach Abschluss eines States wechselt die State Machine selbständig in den nächsten State bis der State *PRE-OPERATIONAL* erreicht wird. Die Funktion [CcmConnectToNet](#) kehrt dann wieder zurück. Während der Ausführung der einzelnen States bzw. Ereignisse werden Module des CANopen-Stacks mehrfach über ihre **XxxNmtEvent**-Funktion gerufen. Ebenso erfolgt ein Aufruf der NMT-Callback-Funktion **AppCbNmtEvent** in der Applikation, vorausgesetzt eine Funktion wurde parametrisiert (Eintrag **m_fpNmtEventCallback** der Struktur **tCcmlnitParam**). Beim Aufruf einer NMT-Callback-Funktion wird das NMT-Ereignis als Parameter übergeben (Ereignis wird als Parameter übergeben, *siehe Tabelle 20*).

Innerhalb der Ausführungssequenz der **XxxNmtEvent**-Funktionen eines NMT-States wird die Funktion **AppCbNmtEvent** als letzte Funktion gerufen, um zuvor gesetzte Standardwerte gegebenenfalls applikationsspezifisch zu verändern.

In den folgenden Beispielen wird die Funktion **AppCbNmtEvent** als NMT-Callback-Funktion der Applikation gerufen. Die Beispiele gehen davon aus, dass nur eine Instanz konfiguriert wurde. Bei Verwendung von mehreren Instanzen ist der Instanzparameter zu ergänzen.

State INITIALISING:

Dieser State wird nur einmal nach einem Power-on oder RESET ausgeführt. In diesem State sind daher Init-Funktionen der Module (z.B. [CcmlnitLgs](#), [CcmlnitStore](#), ...) auszuführen. In diesem State sind alle Variablen der Applikation mit den variablen Einträgen des Objektverzeichnisses zu verknüpfen. Nach Fertigstellung wechselt die State Machine automatisch in das Ereignis *RESET APPLICATION*.

Beispiel:

```
tCopKernel PUBLIC AppCbNmtEvent (tNmtEvent NmtEvent_p)
{
tCopKernel Ret = kCopSuccessful;

    // which event is called?
    switch (NmtEvent_p)
    {
        // after power-on link all variables with OD
        case kNmtEvEnterInitialising:

            // linking of variables for CANopen with OD
            Ret = CcmDefineVarTab (aVarTab_g,
                sizeof (aVarTab_g) / sizeof (tVarParam));

            break;

        ...
    }
}
```

State RESET APPLICATION:

In diesem Ereignis wurden bereits alle herstellerspezifischen Objekte (ab 0x2000 bis 0x5FFF) und alle gerätespezifischen Objekt (ab 0x6000 bis 0x9FFF) auf ihre Power-on Werte zurückgesetzt. Unter Power-on Wert ist hier der Default-Wert aus dem Objektverzeichnis oder zuletzt gespeicherte Wert im nicht-flüchtigen Speicher zu verstehen. Die Applikation kann nachträglich den Wert der Prozessvariablen verändern.

Beispiel:

```

tCopKernel PUBLIC AppCbNmtEvent (tNmtEvent NmtEvent_p)
{
tCopKernel Ret = kCopSuccessful;

    // which event is called?
    switch (NmtEvent_p)
    {
        case kNmtEvResetNode:

            // reset prozess vars
            wDigiOut = 0;
            ...
            break;
    }
}

```

State RESET COMMUNICATION:

Hier wurden alle Kommunikations-Parameter (ab 0x1000 bis 0x1FFF) auf ihre Power-on Werte zurückgestellt. Unter Power-on Wert ist hier der Default-Wert aus dem Objektverzeichnis oder zuletzt gespeicherte Wert im nicht-flüchtigen Speicher zu verstehen. Die Kommunikationsobjekte für alle Module des CANopen-Stacks wurden angelegt. Die Applikation kann nun alle PDOs konfigurieren. Dabei werden die Einstellungen durch die Default-Werte im OD bzw. die Werte aus dem nicht-flüchtigen Speicher überschrieben. Die State Machine wechselt automatisch in den State *PRE-OPERATIONAL*. Ein CANopen-Slave signalisiert diesen State-Wechsel durch Senden der BOOTUP-Nachricht.

Beispiel:

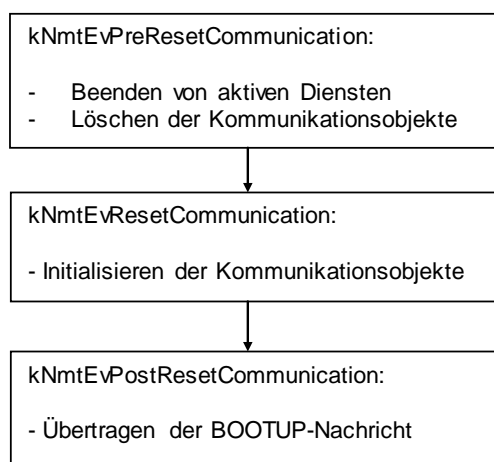
```

tCopKernel PUBLIC AppCbNmtEvent (tNmtEvent NmtEvent_p)
{
tCopKernel Ret = kCopSuccessful;

    // which event is called?
    switch (NmtEvent_p)
    {
        // reset all communication objects (0x1000-0x1FFF)
        case kNmtEvResetCommunication:
            Ret = CcmDefinePdoTab (
                (tPdoParam GENERIC*) &aPdoTab_g[0],
                sizeof (aPdoTab_g) / sizeof (tPdoParam));
            ...
            break;
    }
}

```

Abweichend von der NMT-State Machine im *Bild 18* wurden innerhalb dieses States zwei weitere States implementiert.



Im State *PRE-RESET-COMMUNICATION* werden evtl. aktive Dienste beendet und die Kommunikationsobjekte gelöscht. Im State *POST-RESET-COMMUNICATION* wird das Übertragen der BOOTUP-Nachricht ausgelöst. Damit signalisiert ein CANopen-Slave, dass das Initialisieren abgeschlossen ist. Die State Machine wechselt in den State *PRE-OPERATIONAL*.

State PRE-OPERATIONAL:

In diesem State ist die Kommunikation über SDO möglich. Life Guarding, Node Guarding oder Heartbeat werden ausgeführt, wenn diese Dienste durch die Applikation konfiguriert wurden. Mit Hilfe von SDOs können über den CAN-Bus Kommunikations-Parameter und Mapping-Parameter für PDOs geändert werden. Das CANopen-Gerät wechselt in den Zustand *OPERATIONAL* nach Empfang des Dienstes *Start_Remote_Node* eines NMT-Masters¹ oder nach Aufruf der Funktion [CcmBootNetwork](#) bzw. [CcmSendNmtCommand \(0x00, kNmtCommStartRemoteNode\)](#).

Nach Ausführen dieses Ereignisses wird die Funktion [CcmConnectToNet](#) beendet. State Wechsel werden dann durch Empfangen von NMT Kommandos ausgelöst. Innerhalb der Funktion [CcmProcess](#) erfolgt dann die Verarbeitung.

Die Funktion [CcmProcess](#) muss in einer Schleife zyklisch gerufen werden. Je öfter sie gerufen wird, desto stabiler reagiert die CANopen-Schicht auf zeitliche Ereignisse.

Innerhalb von [CcmProcess](#) werden zunächst empfangene Kommunikationsobjekte zur weiteren Verarbeitung an die internen Module von CANopen weitergeleitet. Sollte ein Ereignis auftreten, welches für die Applikation wichtig ist, dann wird eine Callback-Funktion gerufen. Die meisten dieser Callback-Funktionen befinden sich im CCM-Modul

¹ Für Netzwerkanwendungen ohne NMT-Master kann der Wechsel in den State *OPERATIONAL* auch durch Aufruf der Funktion `NMTExecCommand(kNmtCommEnterOperational)` erzwungen werden.

oder sind Bestandteil der Applikation und können daher durch den Anwender angepasst werden. Weiterhin prüft die Funktion [CcmProcess](#) einige zeitliche Abläufe, bei denen unter Umständen eine CAN-Nachricht gesendet werden muss. Zum Beispiel können PDOs auf Änderung oder nach Ablauf des Event Timers gesendet werden. Ebenso wird ein SDO-Abort gesendet, wenn der SDO-Server beim segmented Transfer eine Nachricht vom SDO-Client erwartet, diese aber nicht eintrifft.

State OPERATIONAL:

Der Übergang vom Zustand *PRE-OPERATIONAL* nach *OPERATIONAL* löst die Übertragung aller asynchronen TPDOs aus (Senden der Anfangswerte). Weiterhin werden in diesem State PDOs bei einem Ereignis übertragen (z.B. Event Timer abgelaufen, SYNC-Nachricht empfangen, Änderung der Prozessvariablen). Werden PDOs empfangen, dann werden die Daten in das OD aufgenommen und die Applikation durch Aufruf einer evtl. parametrisierten Callback-Funktion benachrichtigt.

State STOPPED:

In diesem Zustand wird die Ausführung aller Dienste gestoppt mit Ausnahme der NMT-Dienste (auch Node Guarding und Heartbeat).

2.6.2.5 Beenden einer CANopen-Applikation

Eine CANopen-Applikation wird durch Ausführen der Funktion [CcmShutDownCANopen](#) beendet. Die Funktion ruft für jedes konfigurierte Modul des CANopen-Stacks die Funktion [XxxDeleteInstance](#) auf. Die Module beenden ihre Dienste und Löschen die Kommunikationsobjekte. Nach Ausführung der Funktion [CcmShutDownCANopen](#) sind die Datenstrukturen der CANopen-Schicht ungültig.

2.7 Funktionen der CCM Schicht

In den folgenden Kapiteln werden die API-Funktionen der CCM-Schicht beschrieben. Die Beschreibung umfasst die Syntax der Funktion, die Parameter, die Rückgabewerte und Erläuterungen zur Anwendung. Die Bedeutung der Return Codes und die unterstützten Abortcodes sind im *Kapitel 2.10* beschrieben.

In den folgenden Funktionsbeschreibungen wird teilweise der Platzhalter Xxx in Funktionsnamen verwendet. Dabei steht das Xxx für die Bezeichnung des jeweiligen CANopen-Moduls.

Beispiel: [XxxInit](#)-Funktion bedeutet alle Initialisierungsfunktionen der verschiedenen CANopen Module [SdosInit](#), [Pdoinit](#),

2.7.1 CcmMain-Modul

Das CcmMain-Modul fasst die wichtigsten Funktionen für das CANopen in Funktionen zusammen. Es vereinigt die Grundfunktionalität des CANopen und regelt die Verarbeitung der wichtigsten Ereignisse. Es wird empfohlen, dieses Modul stets einzusetzen.

Bei dieses Dokument handelt es sich um eine gekürzte Fassung!

Wenn Sie eine vollständige Version dieses Dokuments
erhalten möchten, kontaktieren Sie uns bitte per E-Mail:
support@systemec-electronic.com

4 Hinweise zur CANopen-Zertifizierung

Für die CANopen-Zertifizierung beim CiA ist zu beachten:

- Eine Zertifizierung kann stets nur für ein Gerät erfolgen und nicht für eine Software.
- Der CANopen-Stack wurde mit dem CANopen-Chip der Fa. SYS TEC electronic GmbH zertifiziert.

Certificate No.: CiA200002-301V30/11-013

- Dadurch können wir Ihnen demonstrieren, dass eine Zertifizierung mit unserem CANopen-Stack möglich ist.

Eine Zertifizierung hängt aber auch von einer Reihe von Faktoren ab, die wir nicht direkt beeinflussen können.

Daher die folgenden Hinweise.

- Die Einträge im OD müssen mit denen im ***.EDS-File** übereinstimmen. Dies betrifft insbesondere den Device Name (Index 0x1008, die Hard- und Softwareversion (Index 0x1009 bzw. 0x100A) usw.
- Die Anzahl der PDOs muss mit den tatsächlich vorhandenen PDOs im OD übereinstimmen
- Alle Indizes, die in der Software vorhanden sind, müssen auch im EDS-File eingetragen sein. Es darf keine versteckten Einträge geben.
- Die Einträge auf den Index 0x100C und 0x100D (Life Guarding) müssen per Default auf null stehen.
- Der Index 0x1003, Subindex 0 darf nur mit einer 0 beschrieben werden und es muss dann das Error Field gelöscht werden. Das Schreiben einer Zahl > 0 wird mit einem Fehler beantwortet.
- Die Mapping-Parameter Subindexe 0 (z.B. 0x1600/0; 0x1601/0; 0x1A00/0 usw.) dürfen nur mit Werten bis max. 64 beschrieben werden, sonst erfolgt eine Fehlermeldung. Da unsere CANopen-Software per Default Byte-Mapping unterstützt, werden Werte >8 abgewiesen.
- Die Beantwortung von RTR-Anfragen an den Knoten muss (unabhängig vom TxType) immer möglich sein.

Unter Beachtung der oben genannten Punkte steht einer Zertifizierung nichts mehr im Wege.

Hinweis:

Wir verifizieren unsere Software selbst mit dem CiA Conformance Testtool der jeweils aktuellen Version. Dadurch können wir z.B. auch Vorabtest Ihrer Geräte bei uns in Haus gemeinsam mit Ihnen durchführen.

5 Index

AMI-Interface	365
Big Endian	365
Bitratentabelle.....	360
Callback-Funktion	
AppCbNmtEventUnfiltered	84
AppCbUnknownCobld.....	82
CcmCbEmccEvent	127
CcmCbEmcpEvent	131
CcmCbError	76, 80
CcmCbHbcEvent.....	135
CcmCbLgsEvent	105
CcmCbLssmEvent.....	163
CcmCbLsssEvent.....	80, 164
CcmCbNmtEvent.....	75
CcmCbNmtrmEvent.....	118
CcmCbRestore.....	109
CcmCbStore.....	108
CcmCbStoreLoadObject	111
CcmCbSyncReceived	123
CcmGetSyncCounter	124
COB-Callback-Funktion	237, 242
EMCC-Callback-Funktion... 125, 126, 128, 253, 256	
Error-Callback-Funktion	77, 318
HBC-Callback-Funktion..... 132, 133, 135, 262, 265	
Lgs-Callback-Funktion	104, 105, 246
LSS-Callback-Funktion	165
Master-Callback-Funktion .. 114, 115, 117, 118, 120, 248, 253	
NMT Kommando	247
NMT-Callback-Funktion .32, 55, 57, 66, 72, 76, 167, 243	
NMT-Kommandos	84
Objekt-Callback-Funktion... 101, 108, 110, 169, 171, 172, 180, 190, 224, 225, 230	
PDO-Callback-Funktion	100, 208, 213
SDO-Callback-Funktion ...90, 93, 95, 193, 199, 200, 202, 204	
Store-Callback-Funktion.....	106, 111
SYNC-Callback-Funktion ... 121, 122, 124, 206, 219	
tCcmCbUnknownCobld.....	82
tCcmNmtEventCbUnfiltered	84
unbekannte CAN-Nachrichten	82
Variablen-Callback-Funktion	38, 72, 73, 208
CANopen DLL.....	343
CAN-Treiber.....	360
Ccixxx.c	359
CCM_CONVERT_LSSCMD_TO_LSSFLAG	160
Ccm303	150
CcmLss.....	156
CcmMPdo	271
CcmStPdo.....	147
CDRVTGT.H.....	360
Cdrvxxx.h.....	359
CiA-302.....	291
CiA-302-7	96, 205
CiA-303-3.....	150, 291
COP_CB_DIRECT_CALL	290
COP_FREE	362
COP_MALLOC.....	362
COPCFG.H	360, 362
DAM	320, 377
dynamische Speicherverwaltung	362
Emergency-Fehlercodes	283
Fehlercodes.....	273
free362	
Funktion	
Ccm303InitIndicators	152
Ccm303ProcessIndicators.....	152
Ccm303SetErrorState.....	154
Ccm303SetRunState	153
CcmBootNetwork	145
CcmClearPreDefinedErrorField	130
CcmConfigEmcp	129
CcmConfigHbp.....	136
CcmConfigLgm	119
CcmConfigLgs	104
CcmConfigSyncConsumer	122
CcmConfigSyncProducer	123
CcmConnectToNet	71
CCmConvertFloat	146
CcmDefinePdoTab	99
CcmDefineSlaveTab.....	115
CcmDefineStaticPdoTab	148
CcmDefineVarTab	71
CcmEmccDefineProducerTab	126
CcmEnterCriticalSectionPdoProcess	353
CcmExecNmtCommand	82
CcmGetNmtState.....	81
CcmGetNodeId	81
CcmHbcDefineProducerTab.....	133
CcmInitCANopen	64
CcmInitEmcc.....	125
CcmInitHbc	133
CcmInitLgs	104
CcmInitNmtrm	115
CcmInitNmtrmEx.....	120
CcmInitStore	106
CcmInitSyncConsumer.....	122
CcmInitSyncConsumerEx.....	125
CcmLeaveCriticalSectionPdoProcess	354
CcmLockCanopenThreads	340
CcmLockedCopyData	341
CcmLssmConfigureSlave	158
CcmLssmIdentifySlave	161
CcmLssmInquireIdentity	160
CcmLssmSwitchMode	157
CcmPdoSendMPDO.....	271
CcmProcess.....	75
CcmProcessLssInitState.....	74
CcmReadObject.....	103
CcmRegisterErrorCallback	77, 80
CcmSdocAbort.....	95
CcmSdocDefineClientTab	86
CcmSdocGetState	93
CcmSdocIndicateNetwork	96
CcmSdocStartTransfer	90
CcmSendEmergency.....	130

CcmSendNmtCommand	116	PdoSendMPDO	270
CcmSendThreadEvent	341, 352	PdoSetSyncCallback	218
CcmShutDownCANopen	70	PdoSignalStaticPdo	221
CcmSignalCheckVar	121	PdoSignalVar	214
CcmSignalStaticPdo	150	PdoStaticDefineVarField	221
CcmStoreCheckArchivState	107	SdocAbort	204
CcmStoreRestoreDefault	114	SdocAddInstance	195
CcmTriggerNodeGuard	117	SdocDefineClient	197
CcmUnlockCanopenThreads	340	SdocDeleteInstance	196
CcmWriteObject	102	SdocGetState	202
CobCheck	240	SdocIndicateNetwork	205
CobDefine	238	SdocInit	194
CobProcessRecvQueue	242	SdocInitTransfer	199
CobSend	241	SdocNmtEvent	196
CobUndefine	240	SdocProcess	203
EmccAddInstance	254	SdosAbort	188
EmccAddProducerNode	257	SdosAddInstance	182
EmccDeleteInstance	255	SdosDefineServer	185
EmccDeleteProducerNode	257	SdosDeleteInstance	183
EmccInit	254	SdosInit	181
EmccNmtEvent	255	SdosNmtEvent	184
EmccSetEventCallback	256	SdosProcess	187
EmcpAddInstance	259	SdosUndefineServer	187
EmcpDeleteInstance	259	TgtCalcCrc16	179
EmcpInit	258	TgtCavCheckValid	144
EmcpNmtEvent	260	TgtCavClose	140
EmcpProcess	261	TgtCavCreate	137
EmcpSendEmergency	260	TgtCavDelete	138
HbcAddInstance	263	TgtCavGetAttrib	143
HbcDeleteInstance	263	TgtCavInit	136
HbcInit	262	TgtCavOpen	139
HbcNmtEvent	264	TgtCavRestore	142
HbcSetEventCallback	265	TgtCavShutDown	137
HbpAddInstance	266	TgtCavStore	141
HbpDeleteInstance	266	global.h	356
HbpInit	265	Indicator Specification	150
HbpNmtEvent	267	Indicator Specification	291
HbpProcess	268	Intel-Format	365
NmtExecCommand	82, 243	Kernel Treiber	337, 343
NmtmAddSlaveNode	248	kLssmEvIdentifyAnySlave	164
NmtmConfigLgm	249	kLssmEvInquireData	164
NmtmDeleteSlaveNode	248	kLssmEvResult	164
NmtmGetSlaveInfo	251	Knotennummer	81, 229
NmtmProcess	253	Knotenstatus	81, 229
NmtmSendCommand	252	Kommunikationsobjekt	237
NmtmTriggerNodeGuard	250	bestätigte	6
NmtsProcess	246	unbestätigte	6
NmtsSendBootup	245	Konfiguration	284
NmtsSetLgCallback	246	CCM_MODULE_INTEGRATION ..	86, 98, 103, 114, 291
ObdAccessOdPart	226	CCM_PROCESS_RECV_COUNT	292
ObdDefineVar	228	CCM_STORE_FILE_SYSTEM	66, 292
ObdGetEntry	223	CCM_USE_PDO_CHECK_VAR	293
ObdGetNodeId	81, 229	CCM_USE_STORE_RESTORE	106, 291
ObdGetNodeState	81, 228	CDRV_CAN_SPEC	295
ObdReadEntry	225	CDRV_IDINFO_ALGO	296
ObdRegisterUserOd	230	CDRV_IDINFO_ENTRIES	298
ObdWriteEntry	224	CDRV_IMPLEMENT_RTR	301
PdoAddInstance	210	CDRV_MAX_INSTANCES	294
PdoDefineCallback	213	CDRV_MAX_RX_BUFF_ENTRIES_HIGH	332
PdoDeleteInstance	211	CDRV_MAX_RX_BUFF_ENTRIES_LOW	332
PdoForceAsynPdo	219	CDRV_MAX_RX_POLLING	300
PdoInit	210	CDRV_MAX_TX_BUFF_ENTRIES_HIGH	332
PdoNmtEvent	211	CDRV_MAX_TX_BUFF_ENTRIES_LOW	332
PdoProcessAsyn	216	CDRV_TIMESTAMP	298
PdoProcessCheckVar	215	CDRV_USE_BASIC_CAN	295
PdoProcessSync	217	CDRV_USE_DELETEINST_FUNC	300
PdoSend	213		

CDRV_USE_ERROR_ISR.....	299
CDRV_USE_HIGHBUFF.....	295
CDRV_USE_HPT.....	300
CDRV_USE_IDVALID.....	296
CDRV_USE_NO_ISR.....	299
CDRV_USE_NO_RXBUFF.....	299
CDRV_USE_NO_TXBUFF.....	298
CDRV_USE_SETBAUDRATE_FUNC.....	302
CDRV_USED_CAN_CONTROLLER.....	294
CDRVLIN_USE_NEW_HWPARAM_API.....	302
CDRVWIN_USE_CYCLIC_TX_INTERFACE.....	301, 305
COB_IMPLEMENT_SET_PARAM.....	304
COB_MAX_RX_COB_ENTRIES.....	237, 333
COB_MAX_TX_COB_ENTRIES.....	237, 333
COB_MORE_THAN_128_ENTRIES.....	303
COB_MORE_THAN_256_ENTRIES.....	303
COB_SEARCHALGO.....	303
COB_USE_ADDITIONAL_API.....	304
COB_USE_CB_UNKNOWN_COBID.....	305
COB_USE_CYCLIC_TX_INTERFACE.....	305
COB_USE_RTR_CONSUMER.....	304
COP_MAX_INSTANCES.....	183, 286
COP_USE_CDRV_FUNCTION_POINTER.....	65, 287
COP_USE_DELETEINST_FUNC.....	70, 290
COP_USE_OPERATION_SYSTEM.....	289
COP_USE_SMALL_TIME.....	289
COP_USE_TGTOS_API.....	289
DEF_DEBUG_LVL.....	284
EMCC_MAX_CONSUMER.....	253, 333
EMCP_CHECK_COBID_ORDER.....	315
EMCP_ENABLE_ERROR_WRITE.....	315
EMCP_PROCESS_TIME_CONTROL.....	313
EMCP_USE_EVENT_CALLBACK.....	314
EMCP_USE_INHIBIT_TIME.....	315
EMCP_USE_PREDEF_ERROR_FIELD.....	314
HBC_IGNORE_BOOTUP.....	331
HBC_MAX_EMCY_VALUES.....	293
HBC_PROCESS_TIME_CONTROL.....	313
HBC_USE_ADDITIONAL_API.....	332
HBP_PROCESS_TIME_CONTROL.....	313
LSSM_CONFIRM_TIMEOUT.....	331
LSSM_PROCESS_DELAY_TIME.....	331
LSSM_PROCESS_TIME_CONTROL.....	313
LSSS_PROCESS_TIME_CONTROL.....	313
NMTM_MAX_SLAVE_ENTRIES.....	333
NMTM_PROCESS_TIME_CONTROL.....	313
NMTS_PROCESS_TIME_CONTROL.....	313
NMTS_USE_CB_MONITOR_ALL_COMMAND.....	314
NMTS_USE_LIFEGUARDING.....	103, 313
NMTS_USE_NODEGUARDING.....	313
OBD_CALC_OD_SIGNATURE.....	307
OBD_CHECK_FLOAT_VALID.....	306
OBD_CHECK_OBJECT_RANGE.....	222, 306
OBD_IMPLEMENT_ARRAY_FCT.....	310
OBD_IMPLEMENT_DEFINE_VAR.....	310
OBD_IMPLEMENT_INIT_MOD_TAB.....	311
OBD_IMPLEMENT_PDO_FCT.....	308
OBD_IMPLEMENT_READ_WRITE.....	310
OBD_INCLUDE_A000_TO_DEVICE_PART.....	308
OBD_OBJ_SIZE_BIG.....	222
OBD_OBJ_SIZE_MIDDLE.....	222
OBD_OBJ_SIZE_SMALL.....	222
OBD_SUPPORTED_OBJ_SIZE.....	222, 305
OBD_USE_DYNAMIC_OD.....	306
OBD_USE_STRING_DOMAIN_IN_RAM.....	307
OBD_USE_USTRING.....	311
OBD_USE_VARIABLE_SUBINDEX_TAB.....	308
OBD_USER_OD.....	230
PDO_CHECK_COBID_ORDER.....	321
PDO_DISABLE_FORCE_PDO.....	319
PDO_GRANULARITY.....	316
PDO_IMPLEMENT_CHECK_VAR.....	322
PDO_MAX_EMCY_VALUES.....	293
PDO_MORE_THAN_255_ENTRIES.....	322
PDO_PROCESS_TIME_CONTROL.....	313, 316
PDO_USE_ADDITIONAL_API.....	321
PDO_USE_BIT_MAPPING.....	319
PDO_USE_DEF_LINKING_IN_OD.....	322
PDO_USE_DEF_MAPPING_IN_OD.....	323
PDO_USE_DUMMY_MAPPING.....	319
PDO_USE_ERROR_CALLBACK.....	318
PDO_USE_EVENT_TIMER.....	317
PDO_USE_MPDO_DAM_CONSUMER.....	320
PDO_USE_MPDO_DAM_PRODUCER.....	320
PDO_USE_MPDO_SAM_CONSUMER.....	320
PDO_USE_MPDO_SAM_PRODUCER.....	320
PDO_USE_REMOTE_PDOS.....	317
PDO_USE_STATIC_MAPPING.....	98, 318
PDO_USE_SYNC_CONS_COUNTER.....	323
PDO_USE_SYNC_PDOS.....	317
PDO_USE_SYNC_PROD_COUNTER.....	323
PDO_USE_SYNC_PRODUCER.....	318
PDO_VARCB_BEFOR_ENCODE.....	321
SDO_BLOCKSIZE_DOWNLOAD.....	325
SDO_BLOCKSIZE_UPLOAD.....	325
SDO_BLOCKTRANSFER.....	178, 325
SDO_CALCULATE_CRC.....	178, 193, 326
SDO_MAX_CLIENT_IN_OBD.....	196
SDO_MAX_SERVER_IN_OBD.....	183
SDO_SEGMENTTRANSFER.....	192, 193, 326
SDO_USE_SDO_ROUTER.....	329
SDOC_DEFAULT_TIMEOUT.....	328
SDOC_NETWORK_RESPONSE_TIMEOUT.....	97, 329
SDOC_PROCESS_TIME_CONTROL.....	313
SDOC_USE_ADDITIONAL_API.....	328
SDOC_USE_NETWORK_INIDICATION.....	96, 205, 329
SDOR_MAX_ROUTING_ENTRIES.....	330
SDOR_ROUTER_FORWARDING.....	330
SDOR_SDO_CLIENT_INDEX.....	330
SDOS_DEFAULT_TIMEOUT.....	328
SDOS_MULTI_SERVER_SUPPORT.....	327
SDOS_PROCESS_TIME_CONTROL.....	313
SDOS_USE_ADDITIONAL_API.....	327
TARGET_HARDWARE.....	286
Konformität.....	369
Konstante.....	
CCM_LSSFLAGS_ALL.....	160
CCM_LSSFLAGS_SLAVE_ADDRESS.....	160
EMCP_EVENT_ERROR_DELETEALL.....	131
EMCP_EVENT_ERROR_LOG.....	131
kCobTypForceRmtRecv.....	239
kCobTypForceSend.....	239
kCobTypRecv.....	239
kCobTypRmtRecv.....	239
kCobTypRmtSend.....	239
kCobTypSend.....	239
kLssmCmdInquireNodeld.....	161
kLssmCmdInquireProductCode.....	161

kLssmCmdInquireRevisionNr	161
kLssmCmdInquireSerialNr	161
kLssmCmdInquireVendorId	161
kLssmEvActivateBitTiming	163
kLssmEvActivateBusContact	163
kLssmEvDeactivateBusContact	163
kLssmEvModeSelective	163
kLssmEvTimeout	164
kLssModeConfiguration	157
kLssModeOperation	157
kLssModeSelective	157
kLsssEvActivateBitTiming	165
kLsssEvActivateBusContact	165
kLsssEvConfigureBitTiming	165
kLsssEvConfigureNodeId	165
kLsssEvDeactivateBusContact	165
kLsssEvEnterConfiguration	165
kLsssEvEnterOperation	166
kLsssEvPreResetNode	166
kLsssEvSaveConfiguration	165
kNmtCommEnterOperational	117, 244
kNmtCommEnterPreOperational	117, 244
kNmtCommEnterStopped	117, 244
kNmtCommInitialize	244
kNmtCommResetCommunication	117, 244
kNmtCommResetNode	117, 244
kNmtCommStartRemoteNode	117, 244
kNmtCommStopRemoteNode	117, 244
kNmtErrCtrlEvBootupReceived	119
kNmtErrCtrlEvHbcConnected	135
kNmtErrCtrlEvHbcConnectionLost	135
kNmtErrCtrlEvHbcNodeStateChanged	135
kNmtErrCtrlEvLgConnected	105
kNmtErrCtrlEvLgLostConnection	105
kNmtErrCtrlEvLgMsgLost	105
kNmtErrCtrlEvLgNoAnswer	119
kNmtErrCtrlEvLgNodeStateChanged	119
kNmtErrCtrlEvLgSuspended	119
kNmtErrCtrlEvLgToggleError	119
kNodeStateInitialisation	229
kNodeStateOperational	229
kNodeStatePreOperational	229
kNodeStateStopped	229
kObdAccVar	228
kObdCommClear	113
kObdCommCloseRead	112
kObdCommCloseWrite	112
kObdCommOpenRead	112
kObdCommOpenWrite	112
kObdCommReadObj	112
kObdCommWriteObj	112
kObdDirInit	227
kObdDirLoad	227
kObdDirRestore	227
kObdDirStore	227
kObdEvCheckExist	232
kObdEvInitWrite	232
kObdEvPostRead	232
kObdEvPostWrite	233
kObdEvPreRead	232
kObdEvPreWrite	233
kObdEvWrStringDomain	232
kObdPartAll	227
kObdPartDev	112, 227
kObdPartGen	112, 227
kObdPartMan	112, 227
kObdPartUser	112
kObdPartUsr	227
SDO_CRC_POLYNOM	178
SDO_CRC_TABLE	178
SDO_CRC_TARGET	178
SDO_NO_CRC	178
Linux	336
Little Endian	365
LSS Adresse	157, 160
LSS Modus	20, 157
LSS_INVALID_NODEID	81, 229
LSS-Master	19, 156
LSS-Slave	19, 156
malloc	362
Motorola-Format	365
MPDO	269, 271, 320, 377
NMT	
BOOTUP	22
Fehler	77
Heartbeat Consumer	25
Heartbeat Producer	24
Initialisation	22
Kommando	82, 243
Life Guarding	23, 24
Master	23
NMT-Dienst	23
Node Guarding	23
OPERATIONAL	23, 60, 229
PRE-OPERATIONAL	22, 55, 57, 59, 70, 71, 77, 229
Reset Communication	23, 100
Reset Node	100
Slave	23
State Machine	22, 77, 243
STOPPED	23, 229
PDO	6, 26, 32, 49, 57, 205, 291
Callback-Funktion	208, 213
Empfangssignalisierung	208
Event Time	15, 61, 208, 317
Fehler	78, 318
Inhibit Time	15, 208
Initialisierung	210
Konfiguration	59, 99
PDO-Linking	7, 10, 60, 99
PDO-Mapping	7, 9, 38, 60, 72, 99
Remote Anforderung	317
Sperrzeit	15
Synchronisation	17, 122, 124, 217, 317
Transmission Type	16
Übertragung	4, 7, 14, 38, 41, 72, 208, 214, 215, 216
PDOSTC	32, 205
Pre-Defined-Connection-Set	22, 28
PxROS	334
Returncodes	273
SAM	320, 377
SDO-Abortcodes	282
SDO-Client	189
Blocktransfer	193
Callback-Funktion	200
Client definieren	197
CRC-Berechnung	193
Exp. Download	192
Exp. Upload	193
Initialisierung	194
Instanz hinzufügen	195
Instanz löschen	196
Konfiguration	190

NMT-Event	196	tPdoParam	100
Prozess-Funktion	203	tPdoStaticParam	148
Segm. Download	192	tSdocCbFinishParam	201
Segm. Upload	193	tSdocInitParam	194
Status ermitteln	202	tSdocNetworkParam	96
Status ermitteln (CCM)	93, 95	tSdocParam	87, 198
Transfer abrechnen	204	tSdocTransferParam	90, 200
Transfer starten	199	tSdosInitParam	182
Transfer starten (CCM)	90	tSdosParam	185
SDO-Client-Tabelle	189	tVarParam	72
SDO-Gateway	96, 205	tWindowsParam	349
statisches PDO Mapping	32, 147, 148, 205, 219, 318	TARGET.C	363
Struktur		TARGET.H	362
tCcmInitParam	65, 335, 348	TARGET_HARDWARE	359, 362
tCobCdrvFct	287	TgtCanIsrxxx	364
tCobParam	238	TgtEnableCanInterrupt	363
tEmcParam	128	TgtEnableGlobalInterrupt	363
tHbcProdParam	134	TgtGetCanBase	363
tLinuxParam	338	TgtGetTickCount	363, 364
tLssAddress	158	TgtInit	363
tLssCbParam	165	TgtInitCanIsr	363
tLssBitTiming	159	TgtInitSerial	363
tLssIdentifyParam	162	TgtInitTimer	363
tLssResult	164	TgtMemCpy	362, 364
tMPdoParam	270	TgtMemSet	362, 364
tNmtmSlaveInfo	251	TgtTimerIsr	363
tNmtmSlaveParam	249	Typ	
tNmtStateError	77	tSdocState	94
tObdCbParam	231	tThrdPriority	349
tObdCbStoreParam	112	tVxDType	349
tObdVStringDomain	233	Windows	342
tPdoError	78	Zertifizierung	369

6 Glossar

Anwenderschicht:

CiA-301: Definition des Kommunikationsprofils und der Applikationsschicht

Framework:

CiA-302: Framework für programmierbare CANopen-Geräte

CiA-304: Framework für sicherheitsrelevante Kommunikation

Kommunikationsprofil Spezifikation von Übertragungsprotokollen, Kommunikationsobjekten, Datenobjekten

Geräteprofil Spezifikation von gerätespezifischen Diensten und Eigenschaften

CiA-401 CiA Draft Standard 401
Device Profile für Generic IO-Module

Objektverzeichnis (OD): Das Objektverzeichnis (OD) ist die zentrale 0
Datenstruktur eines CANopen-Gerätes zur Ablage alle 27
Gerätedaten. Es bildet das Koppellement zwischen
der Applikation und der Kommunikationsschicht. Ein
Eintrag des ODS wird über Index und Subindex
adressiert.

Kommunikationsobjekt: Objekt zur Übertragung von Daten zwischen CANopen- 0
Geräten 6

TPDO Kommunikationsobjekt zum Senden von Prozessdaten 1.2.1
(Transmit Process Data Object) 14

RPDO Kommunikationsobjekt zum Empfangen von 1.2.1
Prozessdaten (Receive Process Data Object) 15

Tx-Typ	Transimission Type eines PDOs. Dieser entspricht immer dem Subindex 2 der Kommunikations-Parameter eines PDOs (Objektindex 0x1400 bis 0x15FF und 0x1800 bis 0x19FF).
MPDO	Multiplexed PDO – Übertragung von Daten wie ein SDO mit der Möglichkeit diese an mehrere CANopen-Geräte zu senden.
DAM	Destination Address Mode – Übertragungsmodus bei MPDO, wobei der Producer das Objekt als Ziel im OD der Consumer angibt.
SAM	Source Address Mode – Übertragungsmodus bei MPDO, wobei der Producer das Objekt als Quelle im eigenen OD angibt. Der Producer besitzt eine Scanner-Liste, die alle Objekte beinhaltet, die gesendet werden. Die Consumer besitzen eine Dispatcher-Liste, die zu jedem Quellobjekt des Producers ein Zielobjekt des Consumers beinhaltet.

7 Literaturverzeichnis

- [1] „CANopen User Manual“, Software Manual, SYS TEC electronic GmbH, L-1020d, dieses Handbuch
- [2] „CAN-Treiber“, Software Manual, SYS TEC electronic GmbH, L-1023d
- [3] „CANopen Objektverzeichnis“, Software Manual, SYS TEC electronic GmbH, L-1024d
- [4] „CANopen - Application Layer and Communication Profile“, CiA¹ 301 Work Draft, Version V4.2.0.72, Juny 2012
- [5] „CANopen - Framework for CANopen Managers and Programmable CANopen Devices“, CiA¹ Draft Standard Proposal 302, V4.1, February 2009
- [6] „CANopen - Interface and Device Profile for IEC 61131-3 Programmable Devices“, CiA¹ Draft Standard 405, V2.0, 21. 05. 2002
- [7] „CANopen - Device Profile for Generic I/O Modules“, CiA¹ Draft Standard 401, V3.0, June 2008

¹ CiA CAN in Automation e.V.

Dokument: CANopen User Manual
Dokumentnummer: L-1020d_16 Auflage September 2015

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
Am Windrad 2
D-08468 Heinsdorfergrund
GERMANY
Fax : +49 (0) 3765 / 38600-2140

Veröffentlicht von

© SYS TEC electronic GmbH 2015

SYS TEC
ELECTRONIC
Best.-Nr. L-1020d_16
Printed in Germany