



Betriebshandbuch

Stand: 07.07.2000

Version 1.11

bearbeitet von:

Uwe Jaenisch

Markus Miertschink

Wilfried Sperber

Roland Tschakarow

Autorisiert von: Roland Tschakarow

Anzahl der Seiten: 231

AMTEC GmbH

Pankstrasse 8-10

D-13127 Berlin, Germany

Fon: (+49 30) 474 99 0 - 0

Fax: (+49 30) 474 99 0 - 99

World Wide Web

<http://www.amtec-robotics.com>

<http://www.PowerCube.de>

eMail

info@amtec-robotics.com

Support-eMail

support@amtec-robotics.com



Betriebshandbuch V 1.11

Inhaltsangabe

1. Einleitung

1.1 Copyright-Informationen	10
1.2 Versionshinweis	10
1.3 Software-Lizenzvertrag	11
1.3.1 AMTEC-Bestimmungen des Softwarelizenzvertrags	11
1.4 Bestimmungen zur Verwendung des PowerCube™-Logos	12
1.5 Abkürzungen und Begriffserläuterungen	13

2. Allgemeine Sicherheitshinweise und Verwendungszweck

2.1 Sorgfaltspflicht des Betreibers	14
2.2 Grundlegende Sicherheitsmaßnahmen bei Normalbetrieb	14
2.3 Bestimmungsgemäße Verwendung	15
2.4 Unzulässige Änderungen	15
2.5 Grundlegende Sicherheitsmaßnahmen bei Wartung und Instandhaltung	15
2.6 Arbeiten an der elektrischen Ausrüstung	16
2.7 Umweltschutzvorschriften	16
2.8 Haftung	17

3. Funktionsweise der PowerCube™-Produktlinie

3.1 Bestellbezeichnung der PowerCube™-Module	18
3.2 Funktionsprinzipien	18
3.2.1 Encoder	19
3.2.2 Referenzfahrt (Homing)	19
3.3 Produktvarianten	20
3.3.1 PR 70/90/110 - PowerCube™ Rotary	20
3.3.2 PLB 70/90/110 - PowerCube™ Linear Belt	21
3.3.3 PLS 70/90/110 - PowerCube™ Linear Screw	22
3.3.4 PW 70/90 - PowerCube™ Wrist	23
3.3.5 PG 70/90 - PowerCube™ Gripper	24

3.3.6 PDU 70/90/110 - PowerCube™ Drive Unit	25
3.3.7 PSM 70/90/110 - PowerCube™ System Motor	26
3.3.8 PLT 70 - PowerCube™ Lift'n Turn	27
3.4 Konfigurationshinweis	28
3.5 Zubehör	28
3.5.1 Komponenten für Stromversorgung und Kommunikation	28
3.5.2 Komponenten für mechanische Verbindungen	28
3.5.3 Steuerungskomponenten	30
3.5.4 PSD PowerCube™ Software und Dokumentation	30
3.5.5 Digitale Input-/Output CAN-Module	30
3.5.5.1 Allgemeine Beschreibung	30
3.5.5.2 Merkmale	30
3.6 Schnittstellen	31
3.6.1 Pinning der Input/Output-Verbinder	32
3.6.2 Pinning des I/O-Verbinders	32
3.6.3 Blockschaltbild eines möglichen Aufbaus	34
3.7 Allgemeine Beschreibung der StateMachine (Betriebssystem)	35
3.7.1 Motorregelung	36
3.7.2 Überwachung	36
3.7.3 Asynchrones Kommunikationsinterface	36
3.7.4 Rampengenerator	36
4. Inbetriebnahme	
4.1 Lieferumfang (je Modul)	37
4.2 Kundenseitig benötigte Infrastruktur	37
4.2.1 Anforderungen an die Stromversorgung	37
4.2.2 Anforderungen an das Steuergerät	37
4.3 Aufbau, Montage und Installation	38
4.3.1 ISA-Bus PC-CAN Interface iPC-I 320	39
4.3.2 PCI-Bus PC-CAN Interface iPC-I 165 PCI und iPC-I 320 PCI	41
4.3.3 PCMCIA-Karte TinCAN	42
4.4 RS232-Schnittstelle	44
4.5 Die CAN-Kartenkennung	45
4.5.1 IXXAT-Karten	45
4.5.2 Vector-Karten	45
4.5.3 C331 PCI Karte von Meilhaus	45

4.5.4 PC05 Karte von CO Comp.	45
4.6 Das Testprogramm PCubeDemo	46
4.6.1 Installation	46
4.6.2 Der Hauptdialog	46
4.6.3 Der Moduldialog	47
4.7 Normalbetrieb	50
4.8 Besondere Betriebssituationen	50
4.9 Betriebsstörungen und deren Beseitigung	50
4.10 Außerbetriebnahme	50

5. Betriebshinweise

5.1 Hinweise zur Pflege der PowerCube™-Produkte	51
5.2 Oberfläche der PowerCube™-Antriebsmodule	51
5.3 Wartungshinweise	51
5.3.1 Sicherungswechsel	51
5.3.2 Einstellung der Endlagen- und Referenzschalter	52
5.3.2.1 Linearmodule mit Zahnriemenantrieb	52
5.3.2.2 Linearmodule mit Kugelgewindetrieb	53
5.3.3 Einstellung der Zahnriemenspannung	57
5.3.4 Einstellung der Laufrollen	59
5.3.4.1 PLB 70	59
5.3.4.2 PLB 90	62
5.3.5 Grundlagen der Schmierung	65
5.3.5.1 Schmierung der Ölabstreifer	66
5.3.5.1.1 PLB 70	66
5.3.5.1.2 PLB 90	67
5.3.5.2 Schmierung der Kugelumlaufspindel	68
5.3.5.3 Schmierung der Linearführungen (im PG 70/90)	70

6. Konfiguration der PowerCube™-Module

6.1 Die Konfigurationssoftware „PowerConfig“	72
6.2 Installation	72
6.3 Das Hauptfenster	73
6.3.1 Die Menüleiste	73
6.3.1.1 Der Menüpunkt Datei	74
6.3.1.2 Der Menüpunkt Hardware	75

6.3.1.3 Der Menüpunkt Setup	76
6.4 Die Dialoge	76
6.4.1 Der Dialog „Initialization“	76
6.4.2 Die Benutzung des Konfigurationswizards	77
6.4.3 Die Property Pages - Modulparameter editieren	80
6.4.3.1 Der Konfigurationsdialog Identification	81
6.4.3.2 Der Konfigurationsdialog Drive Settings	83
6.4.3.3 Der Konfigurationsdialog Controller Settings	84
6.4.3.4 Der Konfigurationsdialog General Settings	86
6.4.3.5 Der Konfigurationsdialog I/O Settings	88
6.4.4 Der Dialog “Burning Module“	90
6.5 Fehlerbehandlung	92

7. Beschreibung des PowerCube™ - Kommunikationsprotokolls

7.1 Kommunikationsschnittstellen	93
7.2 PowerCube™ Kommunikationsprotokoll	93
7.2.1 PowerCube™-Identifizier für den CAN-Bus	94
7.2.2 PowerCube™-Identifizier für den Profibus	94
7.2.3 PowerCube™-Identifizier für die RS-Kommunikation	94
7.2.4 Übersicht über die PowerCube™-Command ID's	95
7.2.5 Datentypen	96
7.2.6 Aufbau der Datenströme	97
7.2.7 Übersicht über die PowerCube™-Parameter ID's	98
7.2.7.1 Konfigurationsbits im 32-Bit Wort Setup	101
7.2.7.2 Konfigurationsbits im 32-Bit Wort Config	102
7.2.7.3 Bedeutung der Bits im 32-Bit Wort DioSetup	103
7.2.8 Ausführung von Bewegungen	104
7.2.9 Kommandos an alle verbundenen Module in einer Kommandozeile senden	107
7.3 Betriebsverhalten des PowerCube™	108
7.3.1 Betriebsverhalten im Normalfall	108
7.3.2 Die Benutzung der Modul-Watchdogfunktion	109
7.3.3 Der Modulstatus: Das Statuswort CubeState	110
7.3.4 Betriebsweise bei Störungen	112

8. PowerCube™ DriveLib (M5API) - Programmierhandbuch

8.1 Leistungsumfang der PowerCube™-Systemsoftwaremodule	113
8.2 Systemvoraussetzungen	113
8.3 Allgemeine Installationshinweise	113
8.4 Typographie	114
8.5 Grundlagen der Programmierung mit der PowerCube™-Systembibliothek	114
8.5.1 Aufbau der Systembibliothek	114
8.5.1.1 Format der m5dll-Funktionen	115
8.6 Programmierung mit der m5dll-Bibliothek	115
8.6.1 Allgemeines	115
8.6.2 Dateien	115
8.6.3 Programmierkonventionen	115
8.6.4 Datentypen	116
8.6.5 Rückgabewert von Funktionen	116
8.6.6 Vorgehensweise bei Fehlern	117
8.7 Grundlagen der m5dll-Schnittstelle	117
8.7.1 Allgemeines	117
8.7.1.1 Bewegungsart der Module	117
8.7.2 Vorbereitung	117
8.7.2.1 Initialisierung der verwendeten Schnittstelle (Device)	117
8.7.3 Der Initialisierungsstring	118
8.7.3.1 Der Aufbau des Basisinitialisierungsstrings	118
8.7.3.2 Optionen im Initialisierungsstring	119
8.7.3.2.1 Debugmodus	119
8.7.3.2.2 Netzwerkspezifische Optionen	119
8.7.3.2.3 CAN-spezifische Optionen	119
8.7.3.2.4 RS232-spezifische Optionen	119
8.7.4 Einige allgemeine Sonderfunktionen	120
8.7.4.1 Abfragen der Anzahl der Module	120
8.7.4.2 Modultyp erfragen	120
8.7.4.3 Programmierung von Bewegungen	121
8.7.4.3.1 Einheiten	121
8.7.4.3.2 Verfahrbereich	121
8.7.4.3.3 Dynamik	122
8.7.4.3.4 Status des Antriebsmoduls abfragen	122
8.7.4.3.5 Dateien	123
8.8 Referenz	124
8.8.1 Übersicht	124

8.8.2 Initialisierungs- und Verwaltungsfunktionen	127
8.8.2.1 PCube_openDevice	127
8.8.2.2 PCube_closeDevice	128
8.8.2.3 PCube_getModulesCount	129
8.8.2.4 PCube_getModulesIdMap	130
8.8.2.5 PCube_getModuleType	131
8.8.3 Kommandofunktionen	132
8.8.3.1 PCube_syncModule	132
8.8.3.2 PCube_haltModule	133
8.8.3.3 PCube_resetModule	135
8.8.4 Statusfunktionen	136
8.8.4.1 PCube_getCubeState	136
8.8.4.2 PCube_querySyncEnd	137
8.8.4.3 PCube_queryEndPos	138
8.8.5 Bewegungsfunktionen	139
8.8.5.1 PCube_moveRamp	139
8.8.5.1a PCube_moveRampInc	141
8.8.5.1b PCube_moveRampExtended	142
8.8.5.2 PCube_moveStep	143
8.8.5.2a PCube_moveStepInc	144
8.8.5.2b PCube_moveStepExtended	145
8.8.5.3 PCube_moveVel	146
8.8.5.3a PCube_moveVelInc	147
8.8.5.4 PCube_moveCurrent	148
8.8.5.4a PCube_moveCurrentInc	149
8.8.5.4b PCube_moveCurrentExtended	150
8.8.6 Abfragen des aktuellen Modulzustands	151
8.8.6.1 PCube_getActPos	151
8.8.6.2 PCube_getActVel	152
8.8.6.3 PCube_getDeltaPos	153
8.8.6.4 PCube_getCurr	154
8.8.7 Abfragen und Setzen von Bewegungsgrenzwerten	155
8.8.7.1 PCube_getMaxPos	155
8.8.7.2 PCube_getMinPos	156
8.8.7.3 PCube_setMaxPos	157

8.8.7.4	PCube_setMinPos	158
8.8.7.5	PCube_getHomeOffset	159
8.8.7.6	PCube_setHomeOffset	160
8.8.7.7	PCube_getMaxVel	161
8.8.7.8	PCube_getMinVel	162
8.8.7.9	PCube_setMaxVel	163
8.8.7.10	PCube_setMinVel	164
8.8.7.11	PCube_getMaxAcc	165
8.8.7.12	PCube_setMaxAcc	166
8.8.7.13	PCube_getMinAcc	167
8.8.7.14	PCube_setMinAcc	168
8.8.7.15	PCube_getMaxCur	169
8.8.7.16	PCube_setMaxCur	170
8.8.7.17	PCube_getMinCur	171
8.8.7.18	PCube_setMinCur	172
8.8.7.19	PCube_getMaxDeltaPos	173
8.8.7.20	PCube_setMaxDeltaPos	174
8.8.8	Abfragen und Setzen der Koeffizienten des Lageregelkreises	175
8.8.8.1	PCube_getCO	175
8.8.8.2	PCube_getDamp	176
8.8.8.3	PCube_getAO	177
8.8.8.4	PCube_setCO	178
8.8.8.5	PCube_setDamp	179
8.8.8.6	PCube_setAO	180
8.8.8.7	PCube_recalcPIDParams	181
8.8.9	Abfragen und Setzen der internen Digital-Ein- und -Ausgänge	182
8.8.9.1	PCube_getDioData	182
8.8.9.2	PCube_setDioData	183
8.8.10	Abfrage von Vorgabewerten und sonstige Funktionen	184
8.8.11	Die I/O-Schnittstellen	186
8.8.11.1	EMS Thomas Wünsche I/O-Module	186
8.8.11.1.1	TW_initModules	186
8.8.11.1.2	TW_add Module	187
8.8.11.1.3	TW_getIO	188
8.8.11.1.4	TW_setIO	189
8.8.11.1.5	TW_getDigIOall	190
8.8.11.1.6	TW_setDigIOall	191

8.8.12 Besonderheiten bei speziellen Implementierungen	192
8.8.12.1 Die OR-PC5 CAN Schnittstelle	193
8.8.13 Beispiel-Implementierungen in unterschiedliche Arbeitsumgebungen	193
8.8.13.1 MS Visual C++	193
9. PCubeDemo - Ein Programmierbeispiel	
9.1 Typographie	194
9.2 Anforderungen an das Anwendungsprogramm	194
9.3 Konventionen	195
9.4 Anlegen eines Projekts	196
9.4.1 Hinzufügen der Menüleiste	198
9.5 Hinzufügen eines Listenelements	199
9.6 Anlegen eines Statusbars	200
9.7 Anlegen eines nichtmodalen Unter-Dialogs (CModulDlg)	202
9.7.1 Integration des nichtmodalen Dialoges	206
9.8 Grundprinzip des Programmaufbaus	209
9.9 Der Initialisierungs- und Scanthread	210
9.10 Funktionaler Ablauf des Programms	212
9.11 Fehlerabarbeitung	213
9.12 Die Funktionweise des Moduldialogs	215
9.12.1 Initialisierung	216
9.12.2 Scrollereignis bearbeiten	217
9.12.3 Die Funktion CModulDlg::DrawDialog()	220
9.12.4 Die Befehlsschaltflächen (Buttons)	223
9.12.5 Die Optionsfelder (Radio Buttons)	223
9.12.6 Tooltips (QuickInfos)	224
10. Anhang	
10.1 Weltweite Vertriebs- und Servicestellen	226
10.2 EG-Konformitätserklärung	227
10.3 FAQ's	228

1. Einleitung

Wir danken Ihnen für den Erwerb eines unserer Produkte. Mit Erwerb unserer Produkte steht Ihnen der weltweite Kundensupport für AMTEC-Produkte zur Verfügung. Unsere Mitarbeiter und Partner möchten Ihnen helfen, die gewünschten Resultate schnell und professionell zu erzielen.

Das Verzeichnis der Vertriebs- und Kundendienstpartner der AMTEC GmbH befindet sich im Anhang dieses Handbuchs.

1.1 Copyright-Informationen

© 1991-1999 AMTEC GmbH Germany

Alle Rechte vorbehalten. Die Reproduktion, Anpassung oder Übersetzung ist ohne vorherige schriftliche Einwilligung außer im Rahmen der Urheberrechtsgesetze verboten.

Garantieerklärung zur Dokumentation

Die in diesem Dokument enthaltenen Informationen können ohne vorherige Ankündigung verändert werden. AMTEC gibt bezüglich des Materials keine Garantie. Dies umfaßt auch - beschränkt sich jedoch nicht auf - die gesetzlichen Gewährleistungen, daß die Waren für den normalen Gebrauch bzw. für den vorbestimmten Zweck geeignet sind.

AMTEC haftet weder für Fehler in diesem Handbuch, noch für beiläufig entstandene Schäden oder Folgeschäden, die in Verbindung mit der Bereitstellung oder Verwendung dieses Handbuchs entstehen.

Warenzeichen

PowerCube™ ist ein eingetragenes Warenzeichen der AMTEC GmbH. Beachten Sie die Bedingungen zur Verwendung des PowerCube™-Logos in der Einleitung dieses Handbuchs.

Windows™ ist ein eingetragenes Warenzeichen der Microsoft Corporation.

1.2 Versionshinweis

Diese Dokumentation bezieht sich auf die PowerCube™-Produktfamilie ab [Version 3.5.00](#).

Die Versionsnummer kann folgenderweise aufgeschlüsselt werden:

- 3: bezieht sich auf die aktuelle Hardware-Version der PowerCube™-Module. Bei Änderungen an dieser Stelle wenden Sie sich bitte an Ihren Service-Partner.
- 5: bezieht sich auf die Software-Version der aktuellen API-Schnittstelle $mxdll$ ($x = 5$) sowie der optional angebotenen Software-Bibliotheken. Bei Änderungen an dieser Stelle wenden Sie sich bitte an Ihren Service-Partner.
- 00: bezeichnen Unterreleases. Neue Versionen können von Ihnen installiert werden.

1.3 Software-Lizenzvertrag

Die Verwendung der PowerCube™ Software unterliegt den nachstehend aufgeführten Bestimmungen des AMTEC Softwarelizenzvertrags. Indem Sie die Software installieren, erkennen Sie diese Vertragsbedingungen vollständig an.

1.3.1 AMTEC Bestimmungen des Softwarelizenzvertrags

Sofern kein gesonderter schriftlicher Vertrag mit AMTEC abgeschlossen wurde, unterliegt die Verwendung der mitgelieferten Software den folgenden Bestimmungen:

Lizenzerteilung. AMTEC gewährt Ihnen das Recht, eine (1) Kopie der Software zu verwenden. "Verwenden" bedeutet in diesem Fall, die Software zu speichern, zu laden, zu installieren, auszuführen oder anzuzeigen. Der Lizenznehmer darf die Software weder verändern noch irgendwelche Lizenzierungs- bzw. Steuerungsfunktionen der Software deaktivieren. Ist die Software für "gleichzeitige Nutzung" lizenziert, dürfen nicht mehr Benutzer die Software zur gleichen Zeit verwenden als maximal dazu ermächtigt.

Eigentumsrecht. Die Software ist Eigentum von AMTEC und durch sie urheberrechtlich geschützt. Die Lizenz beinhaltet kein Eigentumsrecht an der Software und stellt keineswegs einen Verkauf von irgendwelchen Rechten daran dar. Drittanbieter, mit denen AMTEC zusammenarbeitet, sind dazu befugt, im Falle einer Verletzung dieser Lizenzbedingungen juristische Schritte zum Schutz der Rechte einzuleiten.

Kopien und Anpassungen. Kopien und Anpassungen der Software dürfen ausschließlich zum Zwecke der Archivierung oder im Rahmen der autorisierten Verwendung der Software vorgenommen werden. Dabei müssen auf alle Kopien und Anpassungen alle in der Original-Software enthaltenen Copyright-Hinweise mit kopiert werden. Die Software darf nicht in ein öffentlich zugängliches System kopiert werden, sofern dies nicht ausdrücklich gestattet wurde.

De-Assemblieren oder Entschlüsseln untersagt. Die Software darf keinesfalls ohne vorherige schriftliche Genehmigung von AMTEC de-assembliert oder de-kompiliert werden. Die Entschlüsselung der Software ist untersagt, es sei denn, sie ist im Rahmen des Betriebes der Software erforderlich.

Übertragung. Ihre Lizenz erlischt automatisch, sobald die Software in irgendeiner Weise übertragen wird. Bei der Eigentumsübertragung müssen Sie dem Empfänger die gesamte Software einschließlich aller Kopien und der zugehörigen Dokumentation überlassen. Der Empfänger muß diese Lizenzbedingungen als Bedingung der Eigentumsübertragung anerkennen.

Beendigung des Vertrages. AMTEC hat das Recht, dem Lizenznehmer sofort die Lizenz zu entziehen, wenn dieser eine der Bedingungen dieses Vertrages verletzt. Bei vorzeitiger Beendigung des Vertrages ist der Lizenznehmer verpflichtet, die Software einschließlich der zugehörigen Dokumentation und aller Kopien oder Änderungen in jeglicher Form zu zerstören.

Exportbestimmungen. Es ist untersagt, die Software bzw. eine Kopie oder Anpassung zu exportieren oder zu reexportieren, wenn dadurch anwendbares Recht oder Rechtsvorschriften verletzt werden.

1.4 Bestimmungen zur Verwendung des PowerCube™-Logos

Als Anwender der PowerCube™ Produktfamilie steht es Ihnen frei, das PowerCube™-Logo für Ihre Dokumentationen und Präsentationen zu verwenden. Ein konsistentes "Look and Feel" ist dabei Voraussetzung für die Wiedererkennung und den Erfolg Ihrer Publikation. Wir bitten Sie jedoch aus rechtlichen Gründen, vor Herausgabe der Publikation, ein Belegexemplar an AMTEC zu schicken.

Beachten Sie dabei folgende Richtlinien:

Das PowerCube™-Logo muß schwarz dargestellt werden. Diese Darstellung erfolgt stets auf hellem Hintergrund.



Das PowerCube™-Logo sollte auf Drucksachen nie kleiner als 20 mm dargestellt werden.

Ränder:



In einem definierten Bereich um das Logo herum dürfen sich keine Schriftzüge oder Logos anderer Unternehmen befinden. Der einzuhaltende Freiraum rechts und links des Logos sollte jeweils 5% der Logobreite nicht unterschreiten. Der einzuhaltende Freiraum oberhalb und unterhalb des Logos sollte jeweils 25% der Logohöhe nicht unterschreiten.

Logo-Integrität

Änderungen am PowerCube™-Logo sind nicht zulässig. Um die Integrität des Logos aufrecht zu halten, verwenden Sie bitte stets das Original-Logo in elektronischer Form.

Das PowerCube™-Logo im Web

Verwenden Sie stets die Darstellung in schwarz auf weißem Hintergrund.

Warenzeichen

In allen Veröffentlichungen ist darauf hinzuweisen, daß PowerCube™ ein eingetragenes Warenzeichen der AMTEC GmbH ist.

1.5 Abkürzungen und Begriffserläuterungen

CPU

Central Processing Unit

RISC

Reduced Instruction Set Code

Tool-Center-Point

Der Tool-Center-Point ist der Punkt am Manipulator, auf den sich alle Bewegungen beziehen. Meistens ist dieser Punkt die Spitze eines Werkzeuges, das an den Manipulator montiert ist.

Weltkoordinaten

Die Weltkoordinaten (x) beschreiben ein rechtwinkliges Koordinatensystem mit den Achsen X, Y, Z und den Winkeln a , b , c . Die Winkel a , b , c beschreiben die Drehung um die Achsen X, Y, Z.

Die Weltkoordinaten können in Roboterkoordinaten umgerechnet werden, wobei es mehrere Lösungen geben kann. Sie werden häufig auch durch den Begriff „Orientierung“ bezeichnet.

Roboterkoordinaten

Roboterkoordinaten (q) sind die Koordinaten der Gelenke. Die Roboterkoordinaten können in eindeutige Weltkoordinaten umgerechnet werden.

Freiheitsgrad

Ein Freiheitsgrad beschreibt die Bewegung entlang einer Achse oder die Bewegung um eine Achse. Im Weltkoordinatensystem gibt es 6 Freiheitsgrade.

Manipulator

Ein Manipulator (Roboter) beschreibt die Summe aller Module. Jedes Modul ist verantwortlich für die Bewegung in oder um einen Freiheitsgrad. Die Bewegungen eines Manipulators dienen der Interaktion mit seiner Umwelt.

Punkt-zu-Punkt Bewegung

Eine Punkt-zu-Punkt Bewegung ist die Bewegung des Tool-Center-Points von Punkt A nach Punkt B, wobei die Bewegung der einzelnen Module nicht synchronisiert wird. Somit ist die Bahn des TCP nicht festgelegt.

Bahnsteuerung

Als Bahnsteuerung bezeichnet man eine synchronisierte Bewegung von Punkt A nach Punkt B, entlang eines vorgegebenen Weges.

2. Allgemeine Sicherheitshinweise und Verwendungszweck

Bei der Verwendung der PowerCube™-Module und deren Kombination müssen alle grundlegenden Sicherheitsvorschriften immer befolgt werden. Die Module, die Zubehörteile und Zusatzeinrichtungen dürfen erst nach Kenntnisnahme der Betriebsanleitung und nur durch hierfür unterwiesene Personen montiert und in Betrieb genommen werden.

Es sind nur die vom Hersteller empfohlenen oder in der Betriebsanleitung enthaltenen Zusatzeinrichtungen zu verwenden.

2.1 Sorgfaltspflicht des Betreibers

Die PowerCube™-Module werden unter Berücksichtigung einer Gefährdungsanalyse und nach sorgfältiger Auswahl der einzuhaltenden harmonisierten Normen, sowie weiterer technischer Spezifikationen konstruiert und gebaut. Sie entsprechen damit dem Stand der Technik und ermöglichen ein Höchstmaß an Sicherheit während des Betriebes.

Konstruktive Veränderungen an den PowerCube™-Modulen dürfen nur nach schriftlicher Genehmigung durch den Hersteller vorgenommen werden! Zuwiderhandlungen schließen jede Gewährleistung durch den Hersteller aus!

Die Sicherheit der PowerCube™-Module kann in der betrieblichen Praxis nur dann umgesetzt werden, wenn alle dafür erforderlichen Maßnahmen getroffen werden. Es unterliegt der Sorgfaltspflicht des Betreibers der PowerCube™-Module, diese Maßnahmen zu planen und ihre Ausführung zu kontrollieren.

Der Betreiber muß insbesondere sicherstellen, daß

- die PowerCube™-Module nur bestimmungsgemäß genutzt werden (siehe Kapitel [Produktvarianten](#)).
- die PowerCube™-Module nur in einwandfreiem, funktionstüchtigem Zustand betrieben werden und besonders die Sicherheitseinrichtungen regelmäßig auf ihre Funktionstüchtigkeit überprüft werden.
- erforderliche persönliche Schutzausrüstungen für Bedienungs-, Wartungs- und Reparaturpersonal zur Verfügung stehen und getragen werden.
- die Betriebsanleitung stets in einem leserlichen Zustand und vollständig am Einsatzort der PowerCube™-Module zur Verfügung steht.
- nur dafür qualifiziertes und autorisiertes Personal die PowerCube™-Module bedient, wartet und repariert. Bei der Wartung und Reparatur ist darauf zu achten, daß nur Originalteile verwendet werden.
- das Personal regelmäßig in allen zutreffenden Fragen von Arbeitssicherheit und Umweltschutz unterwiesen wird, sowie die Betriebsanleitung und insbesondere die darin enthaltenen Sicherheitshinweise kennt.
- alle an den PowerCube™-Modulen selbst angebrachten Sicherheits- und Warnhinweise nicht entfernt werden und leserlich sind.

2.2 Grundlegende Sicherheitsmaßnahmen bei Normalbetrieb

Die PowerCube™-Module dürfen nur von befugten Personen bedient werden, die die Betriebsanleitung kennen und danach arbeiten können.

Vor dem Einschalten der PowerCube™-Module ist zu überprüfen und sicherzustellen, daß

- sich keine Personen im Arbeitsbereich der PowerCube™-Module aufhalten.
- niemand durch das Anlaufen der PowerCube™-Module verletzt werden kann.
- die Stromversorgung der in der Betriebsanleitung spezifizierten Stromversorgung entspricht.

Vor jedem Produktionsbeginn sind die PowerCube™-Module, Zuleitungen und Zubehör auf sichtbare Schäden zu überprüfen und es ist sicherzustellen, daß sie nur in einwandfreiem Zustand betrieben werden! Festgestellte Mängel sofort sind sofort zu melden.

Vor jedem Produktionsbeginn sind Material und Gegenstände aus dem Arbeitsbereich der PowerCube™-Module zu entfernen, die nicht für die Produktion erforderlich sind.

Vor jedem Produktionsbeginn ist zu prüfen und sicherzustellen, daß alle Sicherheitseinrichtungen einwandfrei funktionieren.

Vor der Montage, Wartung und Einrichtung von Zusatzeinrichtungen und Zubehörteilen sind die PowerCube™-Module und deren Kombination netzfrei zu schalten.

2.3 Bestimmungsgemäße Verwendung

Es ist sicherzustellen, daß die Module im Betrieb nicht von den Spezifikationen (Nutzlast, Strombedarf u.ä.) abweichen, die in den betreffenden Datenblättern enthalten sind.

2.4 Unzulässige Änderungen

Die PowerCube™-Module dürfen kundenseitig nur von Personal geöffnet werden, das durch eine spezielle Schulung von AMTEC dafür autorisiert wurde (Ausnahmen s.u. [Wartungshinweise](#)). Es ist darauf zu achten, daß die Module keine Lasten bewegen, für die sie konstruktionsbedingt nicht ausgelegt sind. Die Module sind werksseitig bei Auslieferung nur für den Betrieb an einem bestimmten Bussystem geeignet (RS232, RS485, CAN-Bus oder Profi-Bus). Dies bedeutet, daß die PowerCube™-Module nicht ohne Konfigurationsänderungen an einer anderen Schnittstelle betrieben werden können! Diese Umstellung setzt das Öffnen der Module voraus, so daß eine Neukonfigurierung der Schnittstelle nur von AMTEC vorgenommen werden darf.

2.5 Grundlegende Sicherheitsmaßnahmen bei Wartung und Instandhaltung

Die in der Betriebsanleitung vorgeschriebenen Inspektions- und Wartungsintervalle sind Mindestforderungen und sollten nicht größer sein als die sonst im Anwendungsbereich üblichen.

Die Wartungs- und Reparaturanleitungen zu den Einzelkomponenten in dieser Betriebsanleitung sind zu beachten.

Vor Wartungs- und Reparaturarbeiten ist der Hauptschalter für die Stromversorgung auszuschalten und mit einem Vorhängeschloß zu sichern. Der Schlüssel zu diesem Schloß muß in Händen der Person sein, die die Wartungs- oder Reparaturarbeit ausführt.

Beim Austausch schwerer Maschinenteile sind nur geeignete und einwandfreie Lastaufnahmeeinrichtungen und Anschlagmittel zu verwenden.

Vor Wartungs- und Reparaturarbeiten ist sicherzustellen, daß alle eventuell zu berührenden Teile der PowerCube™-Module sich auf Raumtemperatur abgekühlt haben.

Vor der Ausführung von Wartungs- oder Reparaturarbeiten ist der Zugang zum Arbeitsbereich der PowerCube™-Module für unbefugte Personen zu sperren! Ein Hinweisschild ist anzubringen oder aufzustellen, das auf die Wartungs- oder Reparaturarbeit aufmerksam macht.

Umweltgefährdende Schmier-, Kühl- oder Reinigungsmittel sind ordnungsgemäß zu entsorgen.

Vor Inbetriebnahme nach Wartungs- oder Reparaturarbeiten

- sind gelöste Schraubverbindungen auf festen Sitz zu prüfen.
- ist sicherzustellen, daß vorher entfernte Behälterdeckel, Siebe oder Filter wieder eingebaut sind.

Nach Abschluß von Wartungs- oder Reparaturarbeiten und vor der Wiederaufnahme der Produktion ist sicherzustellen, daß

- alle für die Ausführung der Wartungs- oder Reparaturarbeiten benötigten Materialien, Werkzeuge und sonstige Ausrüsten aus dem Arbeitsbereich der Anlage entfernt wurden.
- eventuell ausgetretene Flüssigkeiten entfernt wurden.
- alle Sicherheitseinrichtungen der Anlage einwandfrei funktionieren.

2.6 Arbeiten an der elektrischen Ausrüstung

Reparaturarbeiten an elektrischen Ausrüstungen der PowerCube™-Module dürfen nur von einer ausgebildeten Elektrofachkraft ausgeführt werden.

- Elektrische Ausrüstungen sind regelmäßig zu überprüfen.
- Lose Verbindungen müssen wieder befestigt werden.
- Beschädigte Leitungen und Kabel sind sofort auszutauschen.
- Der Schaltschrank ist stets geschlossen zu halten! Zugang ist nur befugten Personen erlaubt.
- Schaltschränke und andere Gehäuse von elektrischen Ausrüstungen sind zur Reinigung niemals mit einem Wasserschlauch abzuspritzen.

2.7 Umweltschutzvorschriften beachten

Bei allen Arbeiten an und mit den PowerCube™-Modulen sind die gesetzlichen Pflichten zur Abfallvermeidung und ordnungsgemäßen Verwertung/Beseitigung von Abfallprodukten einzuhalten.

Insbesondere bei Installations-, Reparatur- und Wartungsarbeiten dürfen wassergefährdende Stoffe wie

- Schmierfette und -öle
- Hydrauliköle
- Kühlmittel
- lösungsmittelhaltige Reinigungsflüssigkeiten

nicht den Boden belasten oder in die Kanalisation gelangen.

Diese Stoffe müssen in geeigneten Behältern aufgefangen, aufbewahrt, transportiert und entsorgt werden.

2.8 Haftung

Die Haftung des Herstellers für Mängel und daraus entstehende Folgen wird ausgeschlossen, wenn die Mängel verursacht sind durch:

- nicht bestimmungsgemäße Verwendung der PowerCube™-Module,
- fehlerhafte Montage durch den Betreiber,
- fehlerhafte oder nachlässige Wartung,
- Verwendung ungeeigneter Schmierstoffe,
- natürlichen Verschleiß,
- weitere Verwendung nach Auftreten von Funktionsstörungen,
- weitere Verwendung nach erkannten oder vermutbaren Beschädigungen an Material und Zubehör,
- nicht schriftlich vom Hersteller genehmigte Eingriffe an PowerCube™-Modulen und deren Zubehör durch den Betreiber.

3. Aufbau und Funktionsweise der PowerCube™-Produktlinie

In den folgenden Kapiteln erhalten Sie detaillierte Informationen über den internen Aufbau und die Funktionsweise der PowerCube™-Produktlinie. Sie erfahren, welche Produktvarianten existieren und welche Schnittstellen das System bietet.

3.1 Bestellbezeichnung der PowerCube™-Module

Bitte beachten Sie, daß die Bestellbezeichnung der PowerCube™-Module auf dem Lieferschein in bestimmten Fällen aus internen abwicklungstechnischen Gründen nicht mit der Bezeichnung auf der Verpackung bzw. auf den PowerCube™-Modulen übereinstimmen.

Die nachfolgende Zusammenstellung gibt Ihnen eine Übersicht über alle Module (inklusive Optionen) und die dazugehörigen Bestellnummern.

3.2 Funktionsprinzipien

Das PowerCube™-Produktsystem integriert alle Elemente zur Motorsteuerung in den Antrieb selbst. Ein PowerCube™ verbindet folgende Komponenten auf kleinstem Raum:

- einen hochdynamischen elektronisch kommutierten Gleichstrommotor, in den meisten Modulen mit einstufigem Getriebe.
- einen extrem leistungsfähigen Torque Controller, der den Motor kommutiert.
- einen optischen Encoder (500 Strich auf 2000 Flanken hochgerechnet).
- einen hochpräzisen Motion Controller für die Bewegungssteuerung auf der Basis von Encoder-feedback.
- Überwachungselektronik für:
 - Motorstrom
 - Temperatur der Motorwicklung
 - Temperatur der Endstufentransistoren.
- eine serielle High-Speed Industriebusschnittstelle zur Vernetzung von bis zu 31 PowerCube™-Antrieben.
- ein I/O-Interface für zusätzliche Ein-/Ausgänge sowie 24 V-Stromversorgung im Notaus-Regime:
 - Typ I: 4 IN / 4 OUT
 - Typ II: Encoder-Signal-Ausgang
 - Typ III: Ausgang für externes Encoder-Signal

Die verwendeten Motoren sind bürstenlose Gleichstrommotoren. Der Vorteil dieser Motoren liegt in der beliebigen Positioniermöglichkeit und in dem geringen Verschleiß.

3.2.1 Encoder

In den Modulen befindet sich, an der Achswelle, ein inkrementaler Encoder (Lochscheibe, Lochmaske). Auf der Lochscheibe befinden sich 500 Schlitze und ein Nullpunktschlitz. Beide Scheiben werden von drei Lichtquellen durchleuchtet. Die Lichtimpulse werden von drei Detektoren in elektrische Impulse umgesetzt. Mit Hilfe der elektrischen Impulse ist nun eine genaue Positionsbestimmung möglich. Um einen Bezug zum umgebenden Koordinatensystem zu erhalten, muß das Modul eine Referenzfahrt (Homing) durchführen.

3.2.2 Referenzfahrt (Homing)

Vor dem ersten Fahrkommando muß das Antriebsmodul referenziert werden, das heißt es muß einen Referenzpunkt anfahren, um einen absoluten Bezug zum umgebenden Koordinatensystem zu erhalten. Die Referenzfahrt wird mit dem Kommando `>Home<` ausgelöst. Wird dieses Kommando aufgerufen, setzt sich das Modul in Richtung des Referenzschalters in Bewegung. Zunächst wird die im Modul eingestellte Flanke des Referenzschalters gesucht. Anschließend wird der Nullpunkt des Encoders angefahren. Hier stoppt das Modul die Fahrt und meldet die für den Referenzpunkt voreingestellte Position.

3.3 Produktvarianten

Die PowerCube™-Produktfamilie beinhaltet verschiedene Varianten von Antriebsmodulen, deren interner Aufbau gleichartig ist.

3.3.1 PR 70/90/110 - PowerCube™ Rotary

Drehantriebe mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit Harmonic Drive™-Getrieben 160:1
- Inkrementalgeber zur Lageerfassung und Bestimmung der Winkelgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Endlagen- und Referenzschalter
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabilisiertes Aluminiumgehäuse, blau eloxiert
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- Zusätzliche Stromversorgungs-Schnittstelle (48 V) für 500 W-Module
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (4 IN/ 4 OUT)
- Zusatzeingang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse
- Harmonic Drive™ Getriebe 100:1
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter
- Parallelausgang des internen Encodersignals
- Eingang für ein externes Encodersignal

Leistungsdaten

Baugröße	mm	70	90	110	70	90	110
Getriebeuntersetzung		100 : 1	100 : 1	100 : 1	160 : 1	160 : 1	160 : 1
Nominales Abtriebsmoment	Nm	13,1	34,6	65,6	18,7	49,2	93,2
Wiederholbares erhöhtes Abtriebsmoment	Nm	26	82	155	38	118	228
Geschwindigkeit (max.)	°/s	216	216	216	135	135	135
Lageerfassung	inc./°	555	555	555	888	888	888
Wiederholgenauigkeit der Positionierung	°	±0,02	±0,02	±0,02	±0,02	±0,02	±0,02
Aktionsbereich	°	±160	±160	±160	±160	±160	±160

Anschlußwerte

Baugröße	mm	70	90	110
Motor-Leistungsklasse	W	125	250	500
Masse	kg	1,8	3,8	6,6
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	5/ 12,5 0,25	10/ 25 0,5	1/ 1,5 0,5
Strombedarf aus 48 V (nom./ max.)	A	---	---	10/ 25
Spannungstoleranz auf 24 V/ 48 V	%	±5/ ±10	±5/ ±10	±5/ ±10

3.3.2 PLB 70/90/110 - PowerCube™ Linear Belt

Zahnriemengetriebene Linearmodule mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit Harmonic Drive™ Getriebe 100:1
- Zahnriemenscheibe direkt auf der Getriebe-Abtriebswelle
- Inkrementalgeber zur Lageerfassung und Bestimmung der Verfahrgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Endlagen- und Referenzschalter
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabilisiertes Aluminiumgehäuse, blau eloxiert
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- Zusätzliche Stromversorgungs-Schnittstelle (48 V) für 500 W-Module
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (4 IN/ 4 OUT)
- Zusatzzugang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse
- Harmonic Drive™ Getriebe 50:1
- Individuelle Hublängen
- O-Schlitten
- Hervorstehendes Wellenende zur mechanischen Kopplung zweier Achsen
- Erhöhte Wiederholgenauigkeit
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter
- Parallelausgang des internen Encodersignals
- Eingang für ein externes Encodersignal

Leistungsdaten

Baugröße	mm	70	90	110	70	90	110
Getriebeuntersetzung		50 : 1	50 : 1	50 : 1	100 : 1	100 : 1	100 : 1
Umfang Zahnriemenscheibe (eff.)	mm	150	225	300	150	225	300
Schubkraft (nom.)	N	280	520	740	500	800	1350
Geschwindigkeit (max.)	mm/ s	250	300	400	125	150	200
Lageerfassung	inc/ mm	666	444	333	1333	888	666
Wiederholgenauigkeit der Positionierung	mm	±0,05	±0,05	±0,05	±0,05	±0,05	±0,05
Aktionsbereich = Hub (Standard/ max.)	mm	200/ 5900	300/ 5800	400/ 5700	200/ 5900	300/ 5800	400/ 5700

Anschlußwerte

Baugröße	mm	70	90	110
Motor-Leistungsklasse	W	125	250	500
Masse (bei Standard-Hub)	kg	2,6	4,2	6,8
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	5/ 12,5 0,25	10/ 25 0,5	1/ 1,5 0,5
Strombedarf aus 48 V (nom./ max.)	A	---	---	10/ 25
Spannungstoleranz auf 24 V/ 48 V	%	±5/ ±10	±5/ ±10	±5/ ±10

3.3.3 PLS 70/90/110 - PowerCube™ Linear Screw

Spindelgetriebene Linearmodule mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit direktem Wellenausgang
- Kugelgestützte Linearführungen und Kugelgewindetriebe
- Inkrementalgeber zur Lageerfassung und Bestimmung der Verfahrgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Endlagen- und Referenzschalter
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabilisiertes Aluminiumgehäuse, blau eloxiert
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- Zusätzliche Stromversorgungs-Schnittstelle (48 V) für 500 W-Module
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (1 IN/ 4 OUT)
- Zusatzeingang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse
- Harmonic Drive™ Getriebe 50:1 und 100:1
- Individuelle Hublängen
- Erhöhte Wiederholgenauigkeit
- Erhöhte max. Geschwindigkeit
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter
- Parallelausgang des internen Encodersignals
- Eingang für ein externes Encodersignal

Leistungsdaten

Baugröße	mm	70	90	110	70	90	110	70	90	110
Getriebeuntersetzung		1 : 1	1 : 1	1 : 1	50 : 1	50 : 1	50 : 1	100 : 1	100 : 1	100 : 1
Durchmesser und Steigung der Spindel	mm	10x6	15x10	20x20	10x6	15x10	20x20	10x6	15x10	20x20
Schubkraft (nom.)	N	200	350	360	600	1200	2000	600	1200	2000
Geschwindigkeit (max.)	mm/ s	530	660	1500	10	13,2	24	5	6,6	12
Lageerfassung	inc./mm	333	200	100	16666	10000	500	33333	20000	10000
Wiederholgenauigkeit der Positionierung	mm	±0,02	±0,02	±0,02	±0,01	±0,01	±0,01	±0,01	±0,01	±0,01
Aktionsbereich = Hub (Standard/ max.)	mm	200/ ca. 500	400/ ca. 800	800/ ca. 1200	200/ ca. 500	400/ ca. 800	800/ ca. 1200	200/ ca. 500	400/ ca. 800	800/ ca. 1200

Anschlußwerte

Baugröße	mm	70	90	110
Motor-Leistungsklasse	W	125	250	500
Masse (bei Standard-Hub)	kg	4,4	14,9	30,1
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	5/ 12,5 0,25	10/ 25 0,5	1/ 1,5 0,5
Strombedarf aus 48 V (nom./ max.)	A	---	---	10/ 25
Spannungstoleranz auf 24 V/ 48 V	%	±5/ ±10	±5/ ±10	±5/ ±10

3.3.4 PW 70/90 - PowerCube™ Wrist

Handgelenkmodule mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren für zwei unabhängige Achsen (Neigen und Drehen)
- Zwei Harmonic Drive™ Getriebe
- Zwei Inkrementalgeber zur Lageerfassung und Bestimmung der Winkelgeschwindigkeit
- Zwei voll-digitale Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Endlagen- und Referenzschalter je Achse
- Magnetbremse für Neigeachse
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabiles Aluminiumgehäuse, blau eloxiert
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)

Optionen:

- Bus-Kommunikation RS 485, CAN oder Profibus DP

Leistungsdaten

Baugröße	mm	70	70	90	90
Achse		Neigen	Drehen	Neigen	Drehen
Getriebeuntersetzung		100:1	100:1	160:1	100:1
Nominales Abtriebsmoment	Nm	8,6	2,6	18,6	8,6
Wiederholbares erhöhtes Abtriebsmoment	Nm	10	3,8	38	10
Geschwindigkeit (max.)	°/s	216	360	135	216
Lageerfassung	inc./°	555	555	888	555
Wiederholgenauigkeit der Positionierung	°	±0,02	±0,02	±0,02	±0,02
Aktionsbereich	°	±100	±1080	±110	±1080

Anschlußwerte

Baugröße	mm	70	90
Motor-Leistungsklasse	W	40/ 20	100/ 40
Masse	kg	1,8	3,7
Strombedarf aus 24 V (nom.) zusätzlich für Magnetbremse	A	4,5/ 2,5 0,25	5/ 4,5 0,25
Strombedarf aus 24 V (max.)	A	11,5/ 7,5	12,5/ 11,5
Spannungstoleranz auf 24 V	%	±5	±5

3.3.5 PG 70/90 - PowerCube™ Gripper

2-Finger Parallelgreifermodule mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit Zahnrad-Getriebe, Spindeltrieb und Magnetbremse
- Inkrementalgeber zur Lageerfassung und Bestimmung der Verfahrgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Endlagen- und Referenzschalter
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Einstellbare Greifposition, Kraft und Geschwindigkeit für Außen- und Innengreifen
- Sicheres Greifen durch präzise Einstellung von Öffnungs- und Schließhub über spielfreie Führungen
- Geeignet zum Messen und Selektieren von Teilen
- Formstabile Aluminiumgehäuse, blau eloxiert
- Greifermodul ohne Greiferfinger
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)

Optionen:

- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Greiferfinger nach Kundenvorgabe

Leistungsdaten

Baugröße	mm	70	90
Greifkraft (max.)	N	200	500
Geschwindigkeit (max.)	mm/ s	2x20	2x20
Lageerfassung	inc./ mm	1.024	1.024
Wiederholgenauigkeit der Positionierung	mm	2x ±0,05	2x ±0,05
Aktionsbereich = Greifhub (max.)	mm	2x ±15	2x ±15

Anschlußwerte

Baugröße	mm	70	90
Motor-Leistungsklasse	W	20	40
Masse	kg	1,1	1,9
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	2,5/ 7,5 0,25	4,5/ 11,5 0,25
Spannungstoleranz auf 24 V	%	±5	±5

3.3.6 PDU 70/ 90/110 - PowerCube™ Drive Unit

Antriebsmodule mit Getriebe und freiem Wellenausgang mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit Harmonic Drive™ Getrieben 100:1
- Inkrementalgeber zur Lageerfassung und Bestimmung der Winkelgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabiles Aluminiumgehäuse, blau eloxiert
- Kundenspezifischer Wellenausgang
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- Zusätzliche Stromversorgungs-Schnittstelle (48 V) für 500 W-Module
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (4 IN/ 4 OUT)
- Zusatzeingang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse
- Harmonic Drive™ Getriebe 50:1
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter
- Parallelausgang des internen Encodersignals
- Eingang für ein externes Encodersignal
- Normflansch und Normwelle

Leistungsdaten

Baugröße	mm	70	90	110	70	90	110
Getriebeuntersetzung	°	50 : 1	50 : 1	50 : 1	100 : 1	100 : 1	100 : 1
Nominales Abtriebsmoment	Nm	7,1	18,7	35,4	13,1	34,6	65,6
Wiederholbares erhöhtes Abtriebsmoment	Nm	18	44	85	26	82	155
Geschwindigkeit (max.)	%/s	432	432	432	216	216	216
Lageerfassung	inc./°	278	278	278	555	555	555
Wiederholgenauigkeit der Positionierung	°	±0,02	±0,02	±0,02	±0,02	±0,02	±0,02
Aktionsbereich	U	±200	±200	±200	±100	±100	±100

Anschlußwerte

Baugröße	mm	70	90	110
Motor-Leistungsklasse	W	125	250	500
Masse	kg	1,6	3,2	6,6
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	5/ 12,5 0,25	10/ 25 0,5	1/ 1,5 0,5
Strombedarf aus 48 V (nom./ max.)	A	---	---	10/ 25
Spannungstoleranz auf 24 V/ 48 V	%	±5/ ±10	±5/ ±10	±5/ ±10

3.3.7 PSM 70/90/110 - PowerCube™ System Motor

Antriebsmodule ohne Getriebe mit Normflansch und Normwelle mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren mit direktem Wellenausgang
- Inkrementalgeber zur Lageerfassung und Bestimmung der Winkelgeschwindigkeit
- Volldigitaler Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabiles Aluminiumgehäuse, blau eloxiert
- Normflansch und Normwelle mit freiem Wellenende
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- Zusätzliche Stromversorgungs-Schnittstelle (48 V) für 500 W-Module
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (4 IN/ 4 OUT)
- Zusatzeingang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter
- Parallelausgang des internen Encodersignals
- Eingang für ein externes Encodersignal
- Kundenspezifischer Wellenausgang und/ oder Motorflansch

Leistungsdaten

Baugröße	mm	70	90	110
Getriebeuntersetzung		1 : 1	1 : 1	1 : 1
Nominales Antriebsmoment	Nm	0,194	0,553	1,0
Wiederholbares erhöhtes Antriebsmoment	Nm	0,582	1,8	3,8
Drehzahl (max.)	U/ min	5300	4000	4500
Lageerfassung	inc./ °	5,55	5,55	5,55
Wiederholgenauigkeit der Positionierung	inc.	±1	±1	±1
Aktionsbereich	U	±10.000	±10.000	±10.000

Anschlußwerte

Baugröße	mm	70	90	110
Motor-Leistungsklasse	W	125	250	500
Masse	kg	1,1	2,4	3,9
Strombedarf aus 24 V (nom./ max.) zusätzlich für Magnetbremse	A	5/ 12,5 0,25	10/ 25 0,5	1/ 1,5 0,5
Strombedarf aus 48 V (nom./ max.)	A	---	---	10/ 25
Spannungstoleranz auf 24 V/ 48 V	%	±5/ ±10	±5/ ±10	±5/ ±10

3.3.8 PLT 70 - PowerCube™ Lift'n Turn

Hub- und Drehmodul mit folgender Standardausstattung:

- Elektronisch kommutierte (EC-) Gleichstrommotoren für zwei unabhängige Achsen (Heben und Drehen)
- Zwei Inkrementalgeber zur Lageerfassung und Bestimmung der Winkel- und Verfahrgeschwindigkeit
- Zwei voll-digitale Positions-, Geschwindigkeits- und Stromregler (Motion Controller) mit Servo-Endstufe
- Referenzschalter je Achse, Endlagenschalter für Heben
- Dezentrale Steuerung, basierend auf einer 16-bit CPU
- Überwachung von Hardware-Endlagen, Motorstrom und Temperaturen
- Formstabilisiertes Aluminiumgehäuse, blau eloxiert
- RS 232 Kommunikations-Schnittstellen kombiniert mit Stromversorgung (24 V)
- I/O-Interface (optisch entkoppelt) mit digitalen Ein-/ Ausgängen (4 IN/ 4 OUT)
- Zusatzeingang für +24 V extern (Elektronikversorgung im NOT-AUS-Regime)

Optionen:

- Magnetbremse für Hubachse
- Bus-Kommunikation RS 485, CAN oder Profibus DP
- Eingänge für externe Referenz- und Endlagenschalter (Heben)

Leistungsdaten

Baugröße	mm	70	70
Achse		Heben	Drehen
Wiederholbares erhöhtes Abtriebsmoment	Nm	--	0,9
Schubkraft (max.)	N	235	--
Geschwindigkeit (max.)	mm/ s, U/ s	350	25
Nutzlast (max.)	kg	3,0	3,0
Lageerfassung	inc./ mm, inc./ °	375	16,6
Wiederholgenauigkeit der Positionierung	mm, °	±0,01	±0,1
Aktionsbereich	mm, °	200	±1080

Anschlußwerte

Baugröße	mm	70
Motor-Leistungsklasse	W	125/ 60
Masse	kg	2,85
Strombedarf aus 24 V (nom.) zusätzlich für Magnetbremse	A	4,5/ 2,5 0,25
Strombedarf aus 24 V (max.)	A	12,4/ 7,5
Spannungstoleranz auf 24 V	%	±5

3.4 Konfigurationshinweis

Alle PowerCube™-Module sind ab Werk mit Standardparametern vorkonfiguriert. Hierbei handelt es sich u.a. um folgende Einstellungen:

- maximale Geschwindigkeit, maximale Beschleunigung, maximaler Strom, Verfahrbereich, Getriebeuntersetzung, Moduladresse

Mit dem Programm **easyConfig** können Sie bequem PowerCube™-Module nachkonfigurieren.

Die von Ihnen geänderten Parameter werden im ROM (**R**ead **O**nly **M**emory) des Moduls gespeichert. Diese Änderungen bleiben so lange erhalten (auch nach Abschalten der Module), bis die Parameter erneut geändert werden. In der normalen Anwendung steht Ihnen diese Möglichkeit nicht zur Verfügung. Viele Parameter können auch online verändert werden. Dabei werden alle Angaben durch das Modul auf die ROMseitigen Maximalwerte begrenzt.

Die von Ihnen neu festgelegten Parameter werden in das PowerCube™-Modul "gebrannt", d.h. die Parameter werden im ROM überschrieben. Dieses "Brennen" der Parameter ist nur mit *easyConfig* möglich.

3.5 Zubehör

Ein Satz standardisierter mechanischer und elektrischer Zubehöerteile für die PowerCube™-Module ist verfügbar. Applikationsspezifische Sonderausführungen können angefertigt werden.

3.5.1 Komponenten für Stromversorgung und Kommunikation

- Master-Verbindungskabel, gerade, Länge 3m (für RS232, CAN, Profibus und RS485)
- PowerCube™ Abschlußwiderstand (für RS232/CAN, Profibus und RS485)
- PowerCube™ Verbindungskabel, gewandelt.
- PowerCube™ Verbindungskabel, gerade.
- PowerCube™ 48 V-Systemanschlußkabel, gerade, Länge 3m.
- PowerCube™ Bus- und/oder Spannungsverteiler (3-fach, 4-fach, 5-fach, 6-fach)

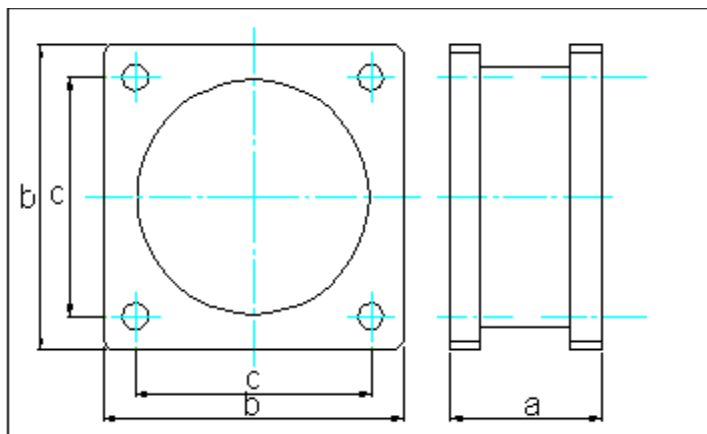
3.5.2 Komponenten für mechanische Verbindungen

- PowerCube™ Verbindungselement, gerade.
- PowerCube™ Verbindungselement, konisch.
- PowerCube™ Verbindungselement, Winkel 90°.

Die nächsten beiden Zeichnungen sollen Ihnen den Aufbau der Verbindungselemente näher erläutern.

Verbindungselement, gerade: $b \times b / a / b \times b$

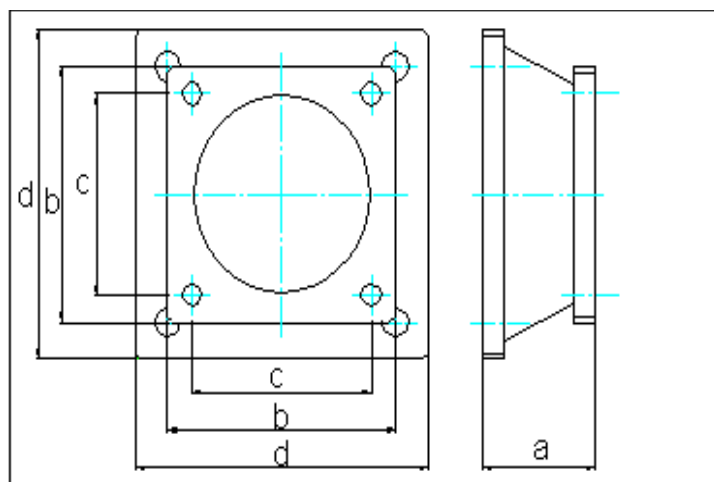
a = Systemmaß oder halbes Systemmaß



Verbindungselement, konisch: $d \times d / a / b \times b$

a = größeres oder halbes größeres Systemmaß

b, d = naheliegende Systemmaße (70/90, 90/110)



3.5.3 Steuerungskomponenten

Die Schnittstellenkarte oder Schnittstellenhardware in Ihrem Computer wird im allgemeinen als Master bezeichnet.

- PCM PowerCube™ CAN-Master: Kommunikationsinterface für den PC, einschließlich Treiber-Software für Win95 / NT.
- PPM PowerCube™ Profibus DP-Master: Kommunikationsinterface für den PC, einschließlich Treiber-Software für Win95 / NT.

3.5.4 PSD PowerCube™ Software und Dokumentation

- PowerCube™ Betriebshandbuch
- PowerCube™ DriveLib m5api.dll (WIN 95/NT) / Treiber-Software
- PowerCube™ easyConfig / Konfigurations-Software
- PowerCube™ quickStep / Test- und Bedien-Software
- PowerCube™ Software-Module für die Bahnsteuerung

3.5.5 Digitale Input-/Output CAN-Module (Modulbaureihe CST)

3.5.5.1 Allgemeine Beschreibung

Die CAN-Knoten der CST-Baureihe sind für dezentrale Ein-/Ausgabe in Meß-, Steuerungs- und Automatisierungsanwendungen vorgesehen. Ihr kompakter und robuster Aufbau mit komplett vergossener Elektronik erlaubt den Einsatz nicht nur im Schaltschrank, sondern auch in Prozeßnähe. Durch den integrierten Micro-Controller können sie Vorverarbeitungsaufgaben selbständig ausführen. Sie sind geeignet, in vielen Anwendungen die Kosten für Steuerungshardware und -installation zu senken bei gleichzeitiger Steigerung der Flexibilität. CST-Knoten sind modular aufgebaut. Grundeinheit ist das Basis-Modul mit Prozessor 80C32, CAN-Controller 82C200 und 8k ROM. Die Ein-/Ausgabe-Schnittstellen werden durch ein Applikations-Modul realisiert, das auf einer Seite des Basis-Moduls in Sandwich-Bauweise angeordnet ist. Verschiedene Applikations-Module sind verfügbar. Steuerungsfunktionen lassen sich effizient durch Einsatz einer programmierbaren Variante des Basismoduls realisieren. Weitere Informationen finden Sie in dem Dokument „Modulbaureihe CST - Anwenderhandbuch“ (csthb_d.pdf).

3.5.5.2 Merkmale

- Vielseitige Lösung für Ein-/Ausgabefunktionen am CAN-Bus
- CiA und ISO 11898 kompatible Busschnittstelle
- Automatische Baudraten-Erkennung aus einer Tabelle vordefinierter Standardwerte
- Eigenintelligenz durch Microcontroller 80C32 oder DS 80C320
- Robust durch vollständig vergossene Elektronik
- Kompakter Aufbau zum Einsatz bei beengten Platzverhältnissen
- Module für unterschiedliche Anwendungen verfügbar
- Modularer Aufbau für leichte Anpassung an Applikationsforderungen

3.6 Schnittstellen

Die PowerCube™-Module verfügen über zwei Steckverbinder, die zum Datenaustausch und zur 24 V-Spannungsversorgung dienen. Hier handelt es sich um einen 7-poligen Stecker mit Mischkontaktierung (Input) und eine 7-polige Buchse mit Mischkontaktierung (Output). Die Module der Baugröße 110 (500 W) verfügen zusätzlich über zwei 3-polige Steckverbinder (Input/Output) zur 48 V-Spannungsversorgung. Mit Ausnahme der Handgelenk(PW)- und Greifer(PG)-Module verfügen alle Module über einen weiteren 15-poligen Steckverbinder für zusätzliche Ein/Ausgänge. Zur Pin-Belegung der Steckverbinder siehe weiter unten.

	CAN	Profibus	RS485 (Halb-Duplex)	RS232
Norm	ISO 118 98	DIN 19245 EN 50 170 EN 50254	EIA 485	EIA 232
Übertragungsrate (* = default)	75 KBit/s...1MBit/s (default=250KBit/s)	9,6KBit/s...12MBit/s	9,6...115,2 KBit/s (default = 9,6 KBit/s)	1,2...115,2 KBit/s (default=9,6 KBit/s)
Anzahl Teilnehmer	31	31	31	8
Telegramm Datenlänge	max. 8 Byte	max. 8 Byte	max. 24 Byte	max. 24 Byte
Buslänge	1km (50 KBit/s)	1km (93 KBit/s)	1200m (100 KBit/s)	15m (19,2 KBit/s)
Kommunikation			Halbduplex	Vollduplex
Signale	Differential		Differential	single-ended
Ausgangsstrom			250 mA	500 mA
Ausgangsspannung			+/- 6V (max) +/- 1,5V (min)	+/- 25 V (max) +/- 5 V (min)
Input Level min.			0,2 V	+/- 3 V
Abschlußwiderstand (Ohm)	120	120	100	0K
Datensicherung	CRC (16 Bit)	FCS		keine

Sie können die Module mit einem von vier Kommunikationsprotokollen betreiben : CAN, Profibus-DP, RS485 (Halb-Duplex), RS232, das jedoch herstellerseitig vor einzustellen ist.

Die Belegung der Steckverbinder ist von der intern im PowerCube™ eingestellten Konfiguration abhängig. Das voreingestellte Kommunikationsprotokoll kann auf Wunsch geändert werden, ohne die Module oder die Verbindungskabel austauschen zu müssen. Wenden Sie sich in diesem Fall bitte an Ihren Service-Partner.

3.6.1 Pinning der Input/Output-Verbinder (7-polig)

	PowerCube™ mit CAN-Interface		PowerCube™ mit Profibus-Interface		PowerCube™ mit RS485-Interface		PowerCube™ mit RS232-Interface	
	Input	Output	Input	Output	Input	Output	Input	Output
A1	+24V	+24V	+24V	+24V	+24V	+24V	+24V	+24V
A2	0V	0V	0V	0V	0V	0V	0V	0V
	CAN		Profibus		RS485		RS485	
1	CanHI	CanHI	B	B	B	B	B	B
3	CanLO	CanLO	A	A	A	A	A	A
4	Schirm	Schirm	Schirm	Schirm	Schirm	Schirm	Schirm	Schirm
	RS232		RS232		RS232		RS232	
2	back	back	back	back	back	back	back	back
5	In	Out	In	Out	In	Out	In	Out

3.6.2 Pinning des zusätzlichen 15-poligen I/O-Interfaces

Pin	Farbcodierung I/O-Kabel	Typ I: 4In & 4Out	Typ II: Encoder Output	Typ III: External Encoder Input	
11	blau	GND extern #	GND extern #	n.c. #.	
4	lila	n.c. #	GND extern #	GND extern #	
12	grau	Input 3 *	Z1 *	Z1 *	
5	weiß/gelb	-	Z1 *	Z1 *	
13	grün	Output 0 *	Z2 *	Z2 *	
6	grau/rosa	Output 1 *	Z2 *	Z2 *	
14	gelb	Output 2 *	N *	N *	
7	gelb/braun	Output 3 *	N *	N *	
1	rot	+24V Logikversorgung mit Diode für externe Einspeisung von 24V (s. Seite 33)			
1	rosa				
9	weiß/grün				Input 0 (Referenz) ^
2	braun/grün				Input 1 (Endlage 1) ^
10	rot/blau				Input 2 (Endlage 2) ^
3	schwarz	GND intern			
15	braun	+24V Zusatzversorgung °	n.c.	n.c.	
8	weiß	GND extern °			

: interner Gebrauch, darf keinesfalls modifiziert werden.

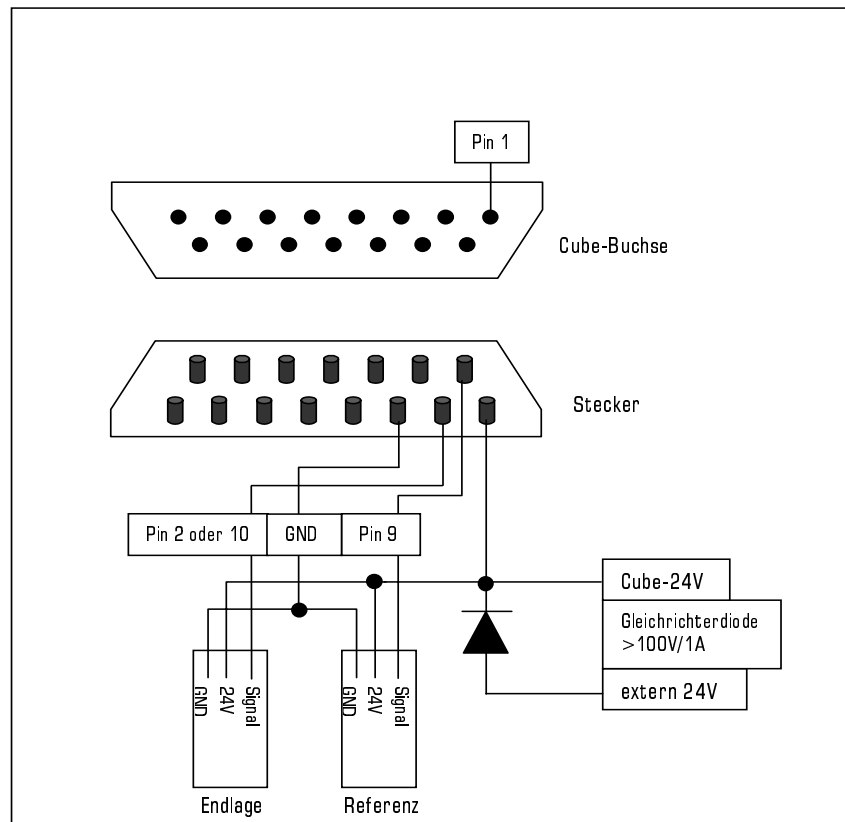
* : Bezugspegel GND extern.

^: Bezugspegel GND intern.

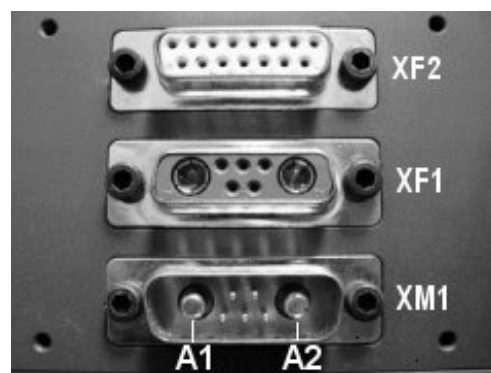
° : +24 V Zusatzversorgung, für Ausgangsschalter (extern mit 0,5 A absichern!).

Es gibt zwei Modi der Stromversorgung für die PowerCube™-Module:

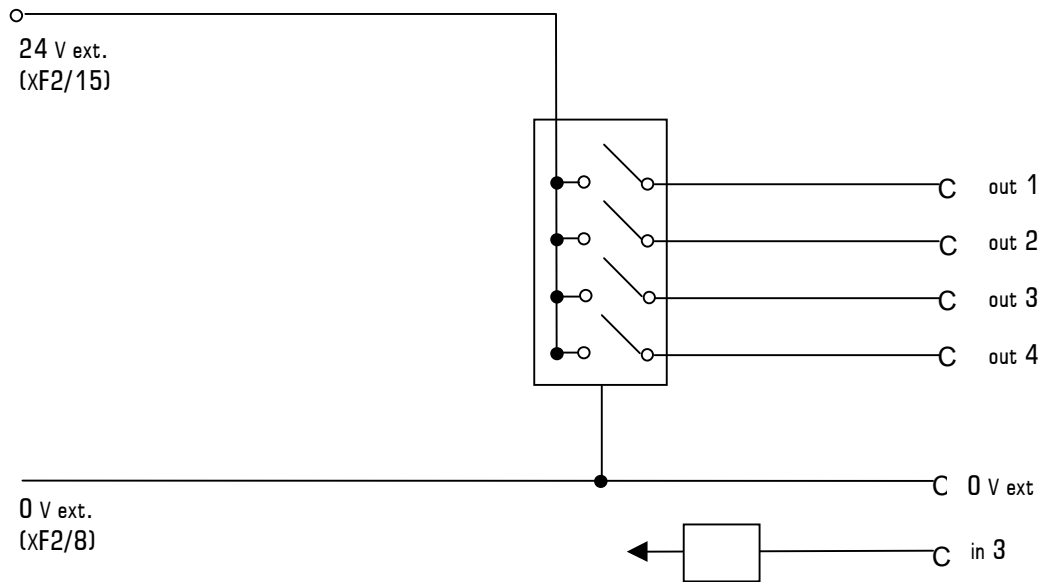
- Variante I: Single Power Supply am XM1.
XF2/Pin 1 und 3 wird intern von XM1 versorgt. +24 V gelangen über eine interne 1 A Sicherung an XF2/Pin 1. XM1/A2 ist verbunden mit XF2/Pin 3.
- Variante II: Power Supply am XM1 und Logic Supply am XF2/Pin 1 und 3. Bei der Anschaltung von Spannung und Signalen an XF2 (siehe Abbildung unten) achten Sie bitte auf die Implementierung der Schutzdiode (wie in der folgenden Zeichnung verdeutlicht). Diese Schutzdiode ist in den amtec-Steckern bereits integriert.



Implementierung der Schutzdiode am 15-poligen I/O-Interface XF2

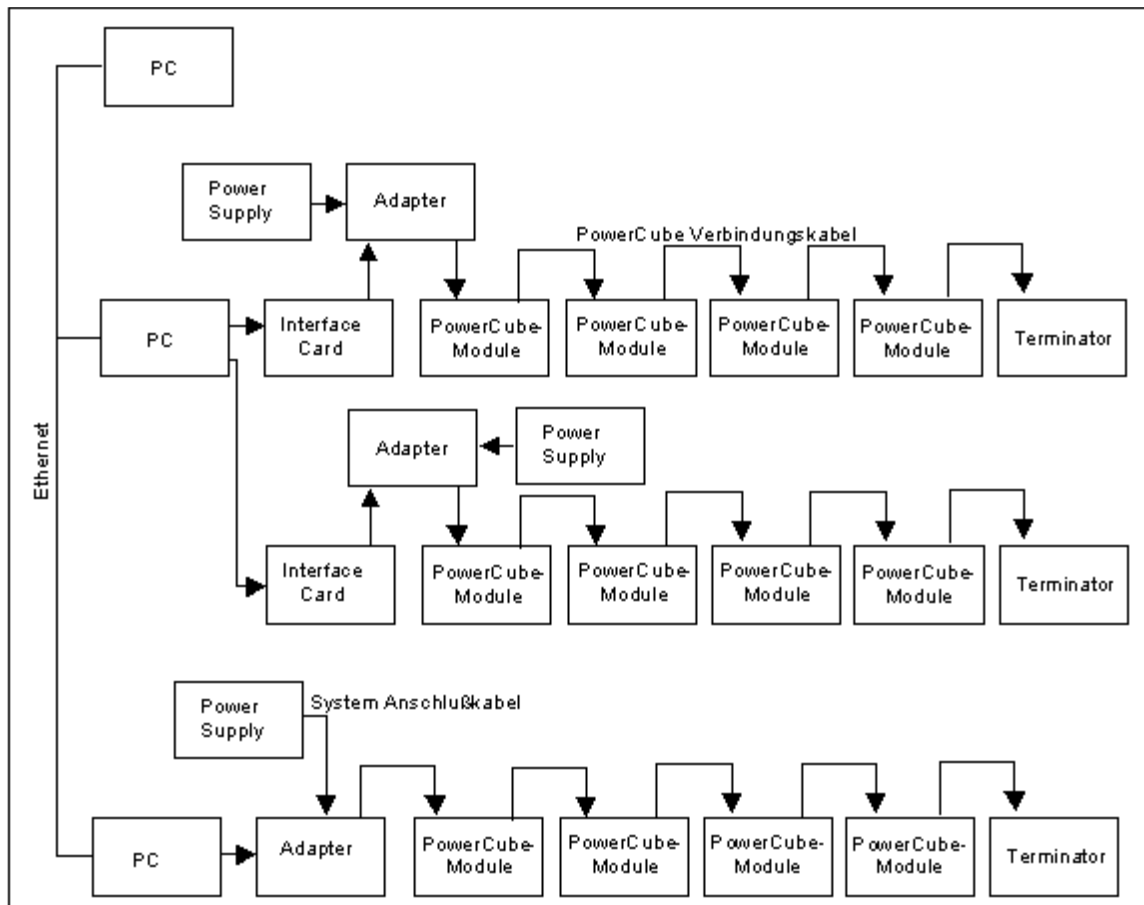


Bezeichnung der Buchseneingänge der PowerCube™-Module ab Versionsnummer 3.5.00



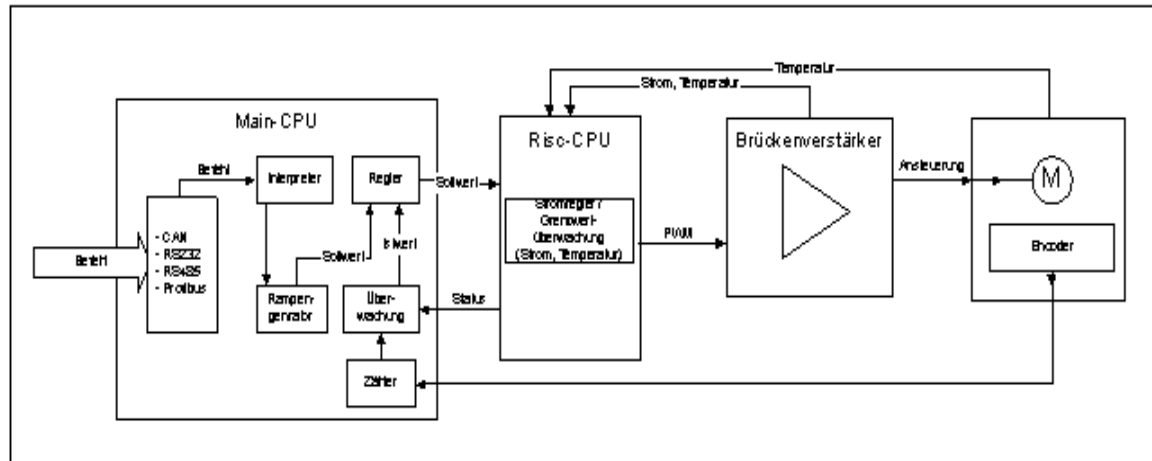
Beschaltung der Ausgangsgruppe und des 3. Eingangs

3.6.3 Blockschaltbild eines möglichen Aufbaus

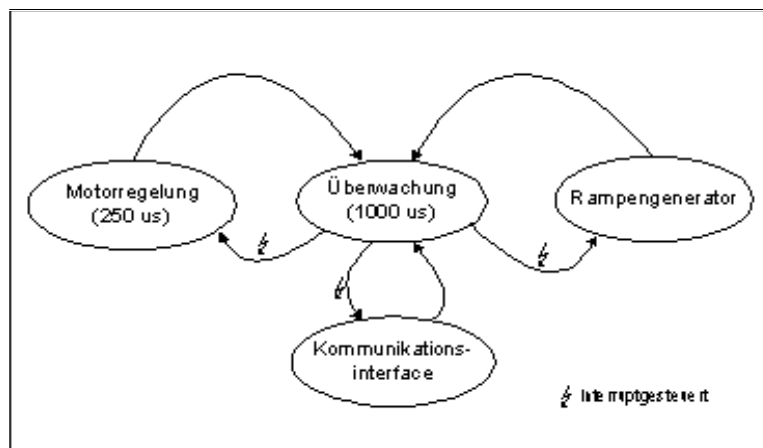


3.7 Allgemeine Beschreibung der State Machine (Betriebs system)

Das folgende Schaubild zeigt in vereinfachter Form die Funktionsweise der PowerCube™-Module .



Im Hauptprozessor der PowerCube™-Module laufen vier Regel- und Steuerkreise (Tasks).



3.7.1 Motorregelung

Die Hauptaufgabe des Mikroprozessors liegt in der Regelung des Motors. Dieser Regelkreis arbeitet mit einer Periode von 250 μs . Das stark vereinfachte Schaubild soll Ihnen einen Einblick in die Funktionsweise des Regelkreises verschaffen.

3.7.2 Überwachung

Die Überwachungsschleife wiederholt sich alle 1000 μs (1ms) und überprüft dabei die folgenden Werte:

- Endlage
- Überstrom
- Unterspannung
- Hallgeber
- Temperatur Motor
- Temperatur Endstufe
- Schleppfehler.

Der Betriebszustand der PowerCube™-Module kann über das Statuswort [CubeState](#) abgefragt werden. Dort werden auch die Reaktionen des Betriebssystems auf eventuelle Fehlerzustände beschrieben.

3.7.3 Asynchrones Kommunikationsinterface

Das Interface arbeitet asynchron, d.h. auf Anfrage, und muß dabei die folgenden Punkte erfüllen:

Interpretation der Anfrage

Ausführung der Anfrage

Antworten

3.7.4 Rampengenerator

Bei Bewegung der Module besteht die Aufgabe darin, einen bestimmten Weg nach vorgegebenen Parametern zurückzulegen. Diese Aufgabe läßt sich in drei Bereiche einteilen. Zunächst erfolgt eine konstante Beschleunigung, die in eine konstante Geschwindigkeit übergeht und in einer konstanten negativen Beschleunigung (bremsen) endet. Die Berechnung dieser Rampe erfolgt im Interpolator. Dieser modulinterne Interpolator berechnet unter Berücksichtigung des bisherigen Rampenverlaufs alle 250 μs eine neue Sollposition für den Lagereger.

4. Inbetriebnahme

Bevor Sie mit der Inbetriebnahme beginnen, müssen Sie sich mit den [Sicherheitshinweisen](#) zur PowerCube™-Produktfamilie vertraut gemacht haben.

4.1 Lieferumfang

Zu jedem PowerCube™-Modul wird ein Datenträger (CD-ROM) mit der aktuellen Treiber-Bibliothek [m5apiw32.dll](#) zur Ansteuerung des Moduls sowie dieses Betriebshandbuch im pdf-Format mitgeliefert. Weiterhin gibt es für die von AMTEC vertriebenen CAN-Bus-Karten sowie für die RS232-Schnittstelle die Testprogramme *m5apitst* (Console-Programm) und *PCubeDemo* (fensterbasiertes Programm), um die Funktionsfähigkeit der Module zu testen (s.u. [Das Testprogramm PCubeDemo](#)). Schließlich ist im Lieferumfang auch das Konfigurationstool [easyConfig](#) enthalten, das in der Lage ist, bestimmte Parameter im ROM eines Moduls zu verändern, indem diese im Konfigurationsfenster neu gesetzt und dann in das Flash-ROM des PowerCube™-Moduls "gebrannt" werden. Die oben beschriebene Software samt Dokumentation, aktuelle Updates davon sowie Beispielanimationen von PowerCube™-Modulen können aus dem Internet von der Adresse www.amtec-robotics.com/download.html heruntergeladen werden. Readme-Dateien sollten, soweit vorhanden, vor der Installation sorgfältig gelesen werden.

Weitere spezialisierte Software-Bibliotheken sowie das Programm *quickStep* (Ablaufsteuerung im Point-To-Point-Modus) sind optional gegen Aufpreis verfügbar (siehe [Zubehör](#)).

Stromversorgung und -anschlußkabel (Spezifikationen siehe [Produktvarianten](#)) sind nicht im Standardlieferumfang eines Moduls enthalten und bei Bedarf separat zu bestellen.

4.2 Kundenseitig benötigte Infrastruktur

4.2.1 Anforderungen an die Stromversorgung

Gleichspannung am Ausgang	24 V ± 5 %
Gleichstrom am Ausgang	Modul-Anzahl x Modul-Nennstrom x 1,2
Welligkeit der Spannung am Ausgang	Restwelligkeit < 150 mVSS Schaltspitzen < 240 mVSS
Strombegrenzung	ca. 1,2 x Nennstrom
Eingang	- Überspannungsfest - mit Einschaltstrombegrenzung
Beispielproduktreihe	SIEMENS SITOP Baureihe

4.2.2 Anforderungen an das Steuergerät (PC oder SPS)

Computer + Betriebssystem	Pentium-PC + Windows 9x/NT
---------------------------	----------------------------

oder

SPS mit serieller Kommunikationsschnittstelle RS232, RS485, CAN-Bus bzw. Profi-Bus-DP	Beispiel-Produkte (PROFIBUS-DP): BOSCH CL 200 DP + BT5 Siemens SIMATIC S7-300 +OP7 Siemens SIMATIC S7-400 +OP17 (in Vorbereitung)
---	--

PowerCube™-Module können auch als Gesamtpaket mit Stromversorgung und Steuergerät erworben werden.

4.3 Aufbau, Montage und Installation

Im allgemeinen erfordern einzeln gelieferte PowerCube™-Module keine besondere Montage. Bei Konstruktionen, die aus Modulkombinationen (Manipulator) bestehen, werden von AMTEC, falls benötigt, gesondert Montageanleitungen mitgeliefert.

Vor Inbetriebnahme der PowerCube™-Module muß sichergestellt werden, daß die korrekte Stromversorgung ordnungsgemäß installiert wurde, der richtige Abschlußwiderstand (sowohl an den Modulen als auch an der Schnittstellenkarte) angebracht sowie das Modul sachgemäß an einer stabilen Stützkonstruktion befestigt wurde. Zur Stromversorgung sind die Datenblätter der entsprechenden PowerCube™-Module zu konsultieren (siehe [dort](#)). Die Abschlußwiderstände werden gemäß der Schnittstellenspezifikation von AMTEC mitgeliefert. Bei der Befestigung müssen sowohl das Gewicht der PowerCube™-Module und der daran eventuell befestigten Nutzlast als auch die darauf wirkenden Kräfte ausreichend berücksichtigt werden. Um die Module über einen PC ansteuern zu können, muß dieser über eine entsprechende Schnittstellenkarte verfügen (siehe [Schnittstellen](#)).

Abb. 1 zeigt exemplarisch eine Modulkombination mit der zum Betrieb notwendigen Infrastruktur (Stromversorgung, Interfacekarte, PC sowie der benötigten Verkabelung)



Abb. 1

Auf der Abbildung sind folgende Komponenten dargestellt:

- A: Modulkombination
- B: Buchse mit eingestecktem Master-Verbindungskabel (Strom- und Datenleitung)
- C: Verbindungskabel zwischen Einzelmodulen, gewendelt
- D: Stromversorgung

- E: Schnittstellenkarte (mit Installationsdiskette und -handbuch sowie Abschlußwiderstand)
- F: Master-Verbindungskabel mit eingestecktem Stromkabel (schwarz) und Datenkabel (grau; Verbindung zur Schnittstellenkarte)
- G: PC (in dem die Schnittstellenkarte eingebaut sein muß)
- H: eingesteckter Abschlußwiderstand

AMTEC bietet verschiedene Schnittstellen-Karten für unterschiedliche Bussysteme an (siehe Zubehör: [Steuerungskomponenten](#)). Es ist jedoch zu beachten, daß die Module nur für ein bestimmtes Bussystem geeignet sind und beim Wechsel auf ein anderes Kommunikationsprotokoll von AMTEC neu konfiguriert werden müssen.

Nachstehend wird die Installation unterschiedlicher, von AMTEC vertriebener CAN-Karten beschrieben:

4.3.1 ISA-Bus PC-CAN Interface iPC-I 320

a) Installation unter Windows NT 4.0

Hardwareinstallation:

- **Schritt 1:** Im Bios überprüfen, ob Interrupts (IRQs) für ISA-Erweiterungskarten reserviert wurden.
- **Schritt 2:** Die vom Betriebssystem verwalteten und bereits vergebenen Interrupts ermitteln (Aufruf des WinNT-Hilfsprogramms "Windows NT Diagnostics" über den Startbutton/Programms/Administrative Tools (Common)/Windows NT Diagnostics. Anwahl der Registerkarte *Resources*, Button *IRQ*).
- **Schritt 3:** Die vom Betriebssystem reservierten Speicherbereiche feststellen (Button *Memory*).
- **Schritt 4:** Die CAN-Karte auf freien Interrupt bzw. Basisadresse konfigurieren. Vom Hersteller der CAN-Karte voreingestellten Werte:
 - Interrupt: IRQ 5 (über Jumper einstellbar: IRQ 3, 4, 5, 7, 9, 10, 11, 12, 14, 15),
 - Basisadresse (reservierter Arbeitsspeicher): D000 - D1FF (über DIP-Schalter wählbar: ein Bereich von C000h - FE00h, in 8 kB-Schritten einstellbar).
- **Schritt 5:** Den Interrupt der CAN-Karte im BIOS für ISA-Steckkarte reservieren.
- **Schritt 6:** PC stromlos machen, Stecker ziehen und danach noch einmal Netzschalter betätigen (wichtig für ATX-Boards).
- **Schritt 7:** PC öffnen, CAN-Karte in einem geeigneten ISA-Steckplatz so befestigen, daß eine ausreichende Kontaktierung gewährleistet ist.
- **Schritt 8:** PC schließen, an das Stromnetz anschließen und einschalten.

Sollten nach Einschalten und Booten des Systems Fehler auftreten, sind die Schritte 4 bis 7 gegebenenfalls zu wiederholen. Anderenfalls kann mit der Installation des Kartentreibers begonnen werden.

Treiberinstallation:

Falls WinNT-Treiber nicht mitgeliefert wurde (nicht auf der Diskette "iPC-I 320 - Test- und Installationsprogramm" enthalten!), Programmpaket für Windows NT unter der Internet-Adresse <ftp://vci:xa5i196ps@www.stzp.de/vci> herunterladen. Es enthält neben dem erforderlichen Treiber auch nützliche Utilities wie den CAN-Minimon.

- **Schritt 9:** Auf der Internet-Seite das Verzeichnis *Winnt* auswählen und von dort die Datei *vcint112.exe* (vollständiges Setup-Programm) oder die beiden gepackten Archive *disk1.zip* und *disk2.zip* (um Installationsdisketten zu erstellen) in ein temporäres Verzeichnis herunterladen.
- **Schritt 10:** Sicherstellen, daß der Benutzer zum Zeitpunkt der Installation Administratorrechte hat.
- **Schritt 11:** Treiber installieren durch Ausführen der einzelnen *EXE*-Datei oder der *setup*-Routine auf Disk 1 der entpackten Installationsdisketten.
- **Schritt 12:** Den voreingestellte Installationspfad "c:\Programme\IXXAT\Vcint114" den eigenen Wünschen anpassen.
- **Schritt 13:** Den Computer neu starten.
- **Schritt 14:** Zum Test der Funktionsfähigkeit des Treibers das neu installierte Programm *Minmon32* starten.
- **Schritt 15:** Im Dialogfenster *Board Type* die eingebaute CAN-Karte sowie ihre Einstellungen (Segment: Basisadresse, IRQ Nr.: Interrupt) auswählen und den OK-Button drücken. Im Protokollfenster (*VCI-Status*) sollten alle Tests mit OK quittiert werden, danach schließt sich dieses Fenster selbstständig. Alle Anzeigen in der Status-Box im Fenster *CAN-MINIMON1* sollten auf grün wechseln.
- **Schritt 16:** Beim Auftreten von Fehlern werden im Hauptmenü der Menüpunkt *Config/Boardtype* ausgewählt und alle Einstellungen im *Board Type*-Fenster überprüft.
- **Schritt 17:** Liegen dort keine Fehleingaben vor, so sind gegebenenfalls andere Interrupts bzw. Basisadressen auf ihre Verwendbarkeit zu testen. Zum Auffinden einer freien Basisadresse kann das Hilfsprogramm "TCFRTEND" verwendet werden

b) Installation unter Win95/98

Hardwareinstallation:

- **Schritt 1:** Im Bios überprüfen, ob Interrupts (IRQs) für ISA-Erweiterungskarten reserviert wurden.
- **Schritt 2:**
Win 95: Systemressourcen (Interrupts, belegte Speicherbereiche) im *Device Manager* (Kontextmenü von *Computer*), Registerkarte *Resources*, anzeigen lassen.
Win 98: Systemressourcen über *Settings/Control Panel/System*, Registerkarte *Device Manager* einsehen.
Dann freien Interrupt wählen.
- **Schritt 3:** Die ausgewählte Basisadresse in der *system.ini* (im Ordner *Windows*) unter dem Unterpunkt [386ENH] eintragen (z.B. mit Notepad), beispielsweise:
EMMExclude = D000-D1FF
- **Schritte 4 bis 8:** Siehe „[Installation unter Windows NT 4.0](#)“

Treiberinstallation:

- **Schritt 9:** Falls nicht beiliegend, kann die Treibersoftware unter der Internet-Adresse <ftp://vci:xa5i196ps@www.stzp.de/vci> aus dem Ordner *Win95* auf ein beliebiges Verzeichnis des eigenen Rechners heruntergeladen werden.
- **Schritt 10:** Treiber installieren durch Ausführen der einzelnen *EXE*-Datei oder der *setup*-Routine auf Disk 1 der entpackten Installationsdisketten.
- **Schritt 11:** Den voreingestellte Installationspfad "c:\Programme\IXXAT\Vcint11x" den eigenen Wünschen anpassen.
- **Schritt 12:** Den Computer neu starten.
- **Schritt 13:** Zum Test der Funktionsfähigkeit des Treibers das neu installierte Programm *Minmon32* starten.
- **Schritt 14:** Im Dialogfenster *Board Type* die eingebaute CAN-Karte sowie ihre Einstellungen (Segment: Basisadresse, IRQ Nr.: Interrupt) auswählen und den OK-Button drücken. Im Protokollfenster (*VCI-Status*) sollten alle Tests mit OK quittiert werden, danach schließt sich dieses Fenster selbstständig. Alle Anzeigen in der Status-Box im Fenster *CAN-MINIMON1* sollten auf grün wechseln.
- **Schritt 15:** Beim Auftreten von Fehlern werden im Hauptmenü der Menüpunkt *Config/Boardtype* ausgewählt und alle Einstellungen im *Board Type*-Fenster überprüft.
- **Schritt 16:** Liegen dort keine Fehleingaben vor, so sind gegebenenfalls andere Interrupts bzw. Basisadressen auf ihre Verwendbarkeit zu testen. Zum Auffinden einer freien Basisadresse kann das Hilfsprogramm "TCFRTEND" verwendet werden

4.3.2 PCI-Bus PC-CAN Interface iPC-I 165 PCI und iPC-I 320 PCI

a) Installation

Die Installation der PCI-Interfaces läuft unter Windows 9x/NT 4.0 prinzipiell gleich ab.

- **Schritt 1:** PC stromlos machen, Stecker ziehen und danach noch einmal Netzschalter betätigen (wichtig für ATX-Boards).
- **Schritt 2:** PC öffnen, CAN-Karte in einem geeigneten PCI-Steckplatz so befestigen, daß eine ausreichende Kontaktierung gewährleistet ist.
- **Schritt 3:** PC schließen, an das Stromnetz anschließen und einschalten.

Sollten nach Einschalten und Booten des Systems Fehler auftreten, sind die Schritte 1 bis 3 gegebenenfalls zu wiederholen. Anderenfalls kann mit der Installation des Kartentreibers begonnen werden.

Treiberdeinstallation:

Sollte sich auf Ihrem Rechner bereits ein installierter Treiber (z.B. VCI-Version 112) befinden, so sollte dieser zuerst entfernt werden. Bevor Sie dies tun müssen Sie aber unbedingt die bereits installierten „Devices“ auf disable schalten, da sonst die Treiber nicht sauber deinstalliert werden! Aufruf über den Startbutton/Settings/Control Panel/Devices (Doppelklick). Dort müssen Sie für jedes Xat...-Device den Button HW Profiles klicken und in dem erscheinenden Fenster den Button Diasable klicken.

Anschließend können Sie über Startbutton/Settings/Control Panel/Add-Remove Software (Doppelklick) die bereits installierte Software entfernen.

Treiberinstallation:

- **Schritt 4:** Falls Win9x/NT-Treiber nicht mitgeliefert wurde, Programmpaket für Windows 9x bzw. Windows NT unter der Internet-Adresse <ftp://vci:xa5i196ps@www.stzp.de/vci> herunterladen. Es enthält neben dem erforderlichen Treiber (VCI 114) auch nützliche Utilities wie den CAN-Minimon.
- **Schritt 5:** Treiber installieren durch Ausführen der einzelnen *EXE*-Datei oder der *setup*-Routine auf Disk 1 der entpackten Installationsdisketten.
- **Schritt 6:** Den voreingestellte Installationspfad "c:\Programme\IXXAT\Vcixx114" den eigenen Wünschen anpassen.
- **Schritt 7:** Den Computer neu starten.
- **Schritt 8:** Zum Test der Funktionsfähigkeit des Treibers das neu installierte Programm *Minmon32* starten.
- **Schritt 9:** Im Dialogfenster *Board Type* die eingebaute CAN-Karte sowie ihre Einstellungen (Board Nr. : Zähler des eingebauten VCI-Boards; bei nur einem eingebauten Board immer 0) auswählen und den OK-Button drücken. Im Protokollfenster (*VCI-Status*) sollten alle Tests mit OK quittiert werden, danach schließt sich dieses Fenster selbstständig. Alle Anzeigen in der Status-Box im Fenster *CAN-MINIMON1* sollten auf grün wechseln.
- **Schritt 10:** Beim Auftreten von Fehlern wird im Hauptmenü der Menüpunkt *Config/Boardtype* ausgewählt und alle Einstellungen im *Board Type*-Fenster werden überprüft. Beim Austausch einer älteren ISA-Karte muß weiterhin sichergestellt sein, daß sich keine Überreste des alten VCI 112-Treibers im Systemverzeichnis befinden (siehe Treiberdeinstallation). Manchmal hilft es wenn Sie in diesem Fall in das Installationsverzeichnis wechseln. Dort gibt es einen Pfad namens „Sysfiles“. In diesem Pfad befinden sich einige *.reg Dateien. Klicken Sie diese doppelt, so werden die Treiber in der Registry eingetragen (falls dies bei der Installation fehlgeschlagen ist). Anschließend müssen Sie Ihren Rechner neu booten.

ACHTUNG: Der [Initstring](#) für den Zugriff auf PowerCube™-Module über PCI-Interfaces ändert sich: VCI:x,0,1 (VCI: ID der PCI-Karte [iPC-I 165 PCI = 6, iPC-I 320 PCI = 7], Zähler der eingebauten Boards, 1=> Interrupt wird automatisch ausgewählt)..

4.3.3 PCMCIA-Karte TinCAN

Sollen PowerCube™-Module auch an einem Laptop über eine CAN-Schnittstellenkarte angesteuert werden, so bietet sich die Verwendung der PCMCIA-Karte TinCAN von IXXAT an. Die mitgelieferte Installationsdiskette enthält Treiber für DOS und Windows 95.

a) Installation unter Windows NT 4.0

Hardwareinstallation:

- **Schritt 1:** TinCAN-Karte-Karte in einen freien PCMCIA-Slot stecken.
- **Schritt 2:** Überprüfen, ob die Karte vom PCMCIA-Controller (zu finden unter *Settings|Control Panel|PC Card(PCMCIA)*) korrekt erkannt wurde. Hier wird auch die Nummer des Slots, indem die Karte steckt, angezeigt.
- **Schritt 3:** Überprüfen freier Interrupts und Basisadressen wie bei [Installation der iPC-I 320-ISA-Karte](#) beschrieben.

Treiberinstallation:

- **Schritt 4:** Falls nicht beigelegt, Treiber aus dem Internet von der Adresse <ftp://vci:xa5i196ps@www.stzp.de/vci>, Ordner *WinNT*, herunterladen und installieren.

- **Schritt 5:** Im CAN-Minimon unter *Board Type* das Board *tinCAN* und den korrekten PCMCIA-Slot einstellen.
- **Schritt 6:** Test durchführen, wie bei der [Installation der iPC-I 320-ISA-Karte](#) beschrieben.

b) Installation unter Windows 95/98

Hardwareinstallation:

- **Schritt 1:** TinCAN-Karte in einen freien PCMCIA-Slot stecken.
- **Schritt 2:** Im Bios überprüfen, ob Interrupts (IRQs) für ISA-Erweiterungskarten reserviert wurden.
- **Schritt 3:**
Win 95: Systemressourcen (Interrupts, belegte Speicherbereiche) im *Device Manager* (Kontextmenü von *Computer*), Registerkarte *Resources*, anzeigen lassen.
Win 98: Systemressourcen über *Settings/Control Panel/System*, Registerkarte *Device Manager* einsehen.
Dann freien Interrupt wählen.
- **Schritt 4:** Die ausgewählte Basisadresse in der *system.ini* (im Ordner *Windows*) unter dem Unterpunkt [386ENH] eintragen (z.B. mit Notepad), beispielsweise:
EMMExclude = D000-D1FF

Treiberinstallation:

- **Schritt 5:** Den Inhalt der Installationsdisk in ein beliebiges Verzeichnis der Festplatte kopieren (Speicherbedarf ca. 2MB!):
- **Schritt 6:** Die Datei *unpack.bat* in einer DOS-Box starten (generiert mehrere Unterverzeichnisse mit Dateien).
- **Schritt 7:** Im Verzeichnis *Driver\Win95* über das Kontextmenü (rechter Mausklick, Untermenüpunkt *installieren*) der Datei *TC2_NEW.INF* die Installation des Kartentreibers durchführen.
- **Schritt 8:** Im Verzeichnis *Test* in einer DOS-Box die Datei *Test.bat* ausführen.
- **Schritt 9:** Beim Fehlschlagen dieser Testroutine das Programm *PCM_VIEW.EXE* ausführen, das ausführlichere Informationen über den Fehlerstatus der Karte bzw. des Softwaretreibers liefert.
- **Schritt 10:** Zur Behebung eventuell auftretender Fehler unterschiedliche Interrupts und insbesondere mehrere verschiedene Basisadressen testen:
- **Schritt 11:** Wie bei der Installation unter Windows NT beschrieben, kann auch das oben beschriebene VCI-Programmpaket mit CAN-Monitor dazu verwendet werden, einen gültigen Treiber zu installieren und die Karte auf ihre Funktionsfähigkeit zu überprüfen.

Wichtig: Die PowerCube™-DriveLib *m5dll* unterstützt nur die VCI-Software bis zur Version 112, d.h. daß die über die *Vci..114.exe*-Dateien installierten Treiber mit der aktuellen *m5dll*-Bibliothek Fehlerzustände hervorrufen. Ab VCI-Treiberversion 114 wird eine spezielle Bibliothek *VCI_W32.dll* benötigt, um die CAN-Befehle korrekt umzusetzen. Diese Datei ist nicht mit der vom VCI-Treiber im Systemverzeichnis installierten gleichnamigen DLL identisch. Sie wird mit allen AMTEC-Softwareprodukten mitgeliefert und muß im gleichen Verzeichnis wie die AMTEC-Anwendung liegen.

4.4 RS232-Schnittstelle

Um die PowerCube™-Module über eine RS232-Schnittstelle betreiben zu können, ist lediglich eine freie serielle Schnittstelle am Steuer-PC erforderlich.

Wichtig: Es ist darauf zu achten, daß es zwei Typen von Abschlußwiderständen gibt! Typ I unterstützt CAN/Profibus/RS232-Interfaces, Typ II unterstützt RS 485/RS232-Schnittstellen.

Zur Ansteuerung der PowerCube™-Module wird eine Funktionsbibliothek in Form einer *dll* mitgeliefert. Diese wird benötigt, um das Testprogramm *m5apitst* auszuführen und gegebenenfalls eigene Steuerungsprogramme für die Module entwickeln zu können. Die Funktionsbibliothek (m5dll, Stand: 21.05.99) wird installiert, indem die Datei *setupex.exe* im Verzeichnis *m5apiw32* ausgeführt wird und den Anweisungen auf dem Bildschirm gefolgt wird. Standardmäßig wird die *dll* in das Verzeichnis *c:\Programme\Amtec GmbH\M5DLL* (englische Installation: *c:\Program Files\Amtec GmbH\M5DLL*) kopiert. Werden kundenseitig eigene Programme zur Steuerung der PowerCube™-Module auf Basis der Funktionsbibliothek realisiert, so ist die Datei **m5apiw32.dll** in das Installationsverzeichnis dieser Software zu kopieren, um deren Funktionsfähigkeit zu garantieren. Bei der Installation der Funktionsbibliothek wird gleichzeitig auch das Testprogramm *m5apitst* mitinstalliert.

Das Konfigurationstool **PowerConfig** wird installiert, indem die Datei *setup.exe* im Verzeichnis *PowerConfig* ausgeführt wird und den Anweisungen auf dem Bildschirm gefolgt wird.

4.5 Die CAN-Kartenkennung

4.5.1 IXXAT-Karten

Die Zusammenarbeit der Schnittstelle mit der *m5apiw32.dll* erfordert die Angabe eines Initstrings, der unter anderem bei VCI-CAN-Interfaces der Firma IXXAT die Kartenkennung beinhaltet (siehe "[Der Aufbau des Basis-Initialisierungsstrings](#)"). Typische Initstrings sind z.B.:

ISA-Interface-Karte: VCI:1,d000,5.

PCI-Interface-Karte: VCI:6,0,1

Darin bedeuten:

- VCI: Schnittstelle wird als VCI-CAN-Interface der Firma IXXAT identifiziert
- 1: Kartenkennung
- bei ISA-Interface-Karten (Kartenkennung 1 und 2): d000 --> Basisadresse der CAN-Karte; bei PCI-Interface-Karten (Kartenkennung 6 und 7): Die Boardnummer. Wird nur eine Interface-Karte benutzt: immer 0.
- 5: Interrupt, den die Karte belegt (nur bei ISA-Boards; bei PCI-Boards grundsätzlich auf 1 oder gar nicht gesetzt)

Im Folgenden werden die Kartenkennungen der VCI-CAN-Interfaces der Firma IXXAT beschrieben:

0	iPC-I 165
1	iPC-I 320
2	CANdy 320
3	tinCAN V2
4	PCI 01, PCI 02, PCI 03
5	iPC-I 386
6	iPC-I 165 PCI
7	iPC-I 320 PCI

Für die passive Karte i-PC03 wird unter Linux folgender Initialisierungsstring benötigt: „CAN4LINUX:X,0“ (X = Busnummer).

4.5.2 Vector-Karten

Die zur Zeit unterstützte PCI-CAN-Karte der Firma Vector (CAN-AC2-PCI) benötigt keinen Identifier und kann mit folgenden Initialisierungsstrings angesprochen werden:

Windows (Win NT/9x): „VECTOR:0,0“

Linux: „VECTOR:X,0“ (X = Busnummer)

4.5.3 C331 PCI Karte von Meilhaus

Die PCI-CAN-Karte C331 von Meilhaus benötigt folgende Initialisierungsstrings:

Windows (Win NT/9x): „ESD4WIN:a,0“

QNX: „ESD4QNX:X,0“ (X = Busnummer)

4.5.4 PC05 Karte von OR Comp.

Unter Linux wird folgender Initialisierungsstring benötigt: „ORCAN:0,0“.

4.6 Das Testprogramm PCubeDemo

4.6.1 Installation

Die Installation des Programms erfolgt durch Starten des zugehörigen Installationsprogramms Setupex.exe. Nach Angabe des Installationsverzeichnisses erfolgt die Wahl des Setup-Typs:

- *Typical.*
Die Applikationsdateien werden wie folgt abgelegt:
Die Dateien „PCubeDemo.exe“ und „m5apiw32.dll“ werden ins Installationsverzeichnis kopiert. In das Systemverzeichnis des Betriebssystems werden die Dateien „MFC42.DLL“ und „MSVCRT.DLL“ abgelegt. Schließlich wird im Verzeichnis des Betriebssystems die Datei „PCubeDemo.ini“ erzeugt.
Unterstützung von Programmentwicklern:
Das Verzeichnis „Source“ wird im Installationspfad angelegt und die zum Erstellen des Visual C++ V.6.0-Projekts notwendigen Dateien in dieses Verzeichnis kopiert.
- *Compact.*
Nur die Applikationsdateien werden wie oben beschrieben angelegt.
- *Custom.*
Zusätzlich zu den Applikations- und Programmentwicklerdateien wird die Option „VCI 114-Support“ angeboten. Bei Auswahl des VCI 114-Support wird die Programm-Bibliothek „VCI_W32.dll“ in das Installationsverzeichnis kopiert. Diese wird benötigt, um CAN-Karten der Firma IXXAT, die mit der neuen Treiberversion 114 angesteuert werden müssen (PCI-Versionen), zusammen mit der m5apiw32.dll betreiben zu können.

ACHTUNG: Ältere Karten, die mit der Treiberversion 112 betrieben werden, arbeiten nicht mit dieser Programmbibliothek zusammen!

4.6.2 Der Hauptdialog

Das Programm öffnet nach dem Start den Hauptdialog. Im Menüpunkt *Device* muß der Unterpunkt *Initstring* gewählt werden. Im Dialog „Board Init-String“ wird der passende Initstring eingegeben (vergleiche "[Der Aufbau des Basis-Initialisierungsstrings](#)") und die Eingabe mit einem <Return> abgeschlossen. Der so ausgewählte Initstring wird in der Datei „PCubeDemo.ini“ abgespeichert und beim nächsten Programmstart automatisch geladen. Soll der eingegebene Initstring nicht verwendet werden, so kann der Unterdialog mit einem Mausklick auf das Kreuz rechts oben in der Titelleiste geschlossen werden, ohne daß der Initstring übernommen wird. Dann wird der Menüpunkt *Device/Rescan* gewählt, worauf das Programm versucht, Module zu finden, die an der im Initstring angegebene Schnittstelle angeschlossen sind. Wird die Schnittstelle bereits von einem anderen Programm verwendet, so versucht die m5apiw32.dll durch automatisches Verwenden des Initstrings „NET: LOCALHOST“, über das Netzwerk auf die angeschlossenen Module zuzugreifen. In der Listbox werden alle angeschlossenen PowerCube™-Module mit ID und Status auflistet.

Der Hauptdialog verfügt über eine Menüleiste mit den Einträgen *Device* (s.o.) und *Exit* (Programm verlassen) sowie einem Halt-Button, der die Bewegung aller angeschlossenen Module unterbricht (Not-Halt). Fehler- bzw. Statusmeldungen des Programms werden im Hauptdialog über eine Statusleiste zur Anzeige gebracht.

Ein rechter Mausklick auf ein Modul (im Listenelement) öffnet ein kontextsensitives Popup-Menü, in dem folgende Optionen für **die zuvor selektierten Modulen** wählbar sind:

- Initialisieren (Homing-Procedure wird ausgelöst)
- Reset (Fehlerbits und Halt-Flag werden gelöscht)

- Halt (Bewegung des Moduls wird angehalten, Halt-Flag wird gesetzt)
- Stromwert auf 0 setzen (Modul kann manuell bewegt werden, wenn Halt-Flag nicht gesetzt ist)

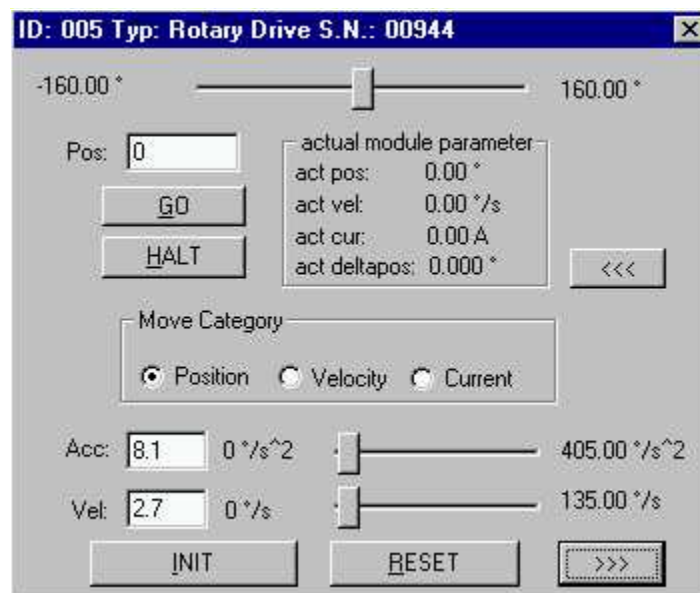
4.6.3 Der Moduldialog

Ein Doppelklick auf ein Modul öffnet den Moduldialog, der die folgenden Punkte realisiert:

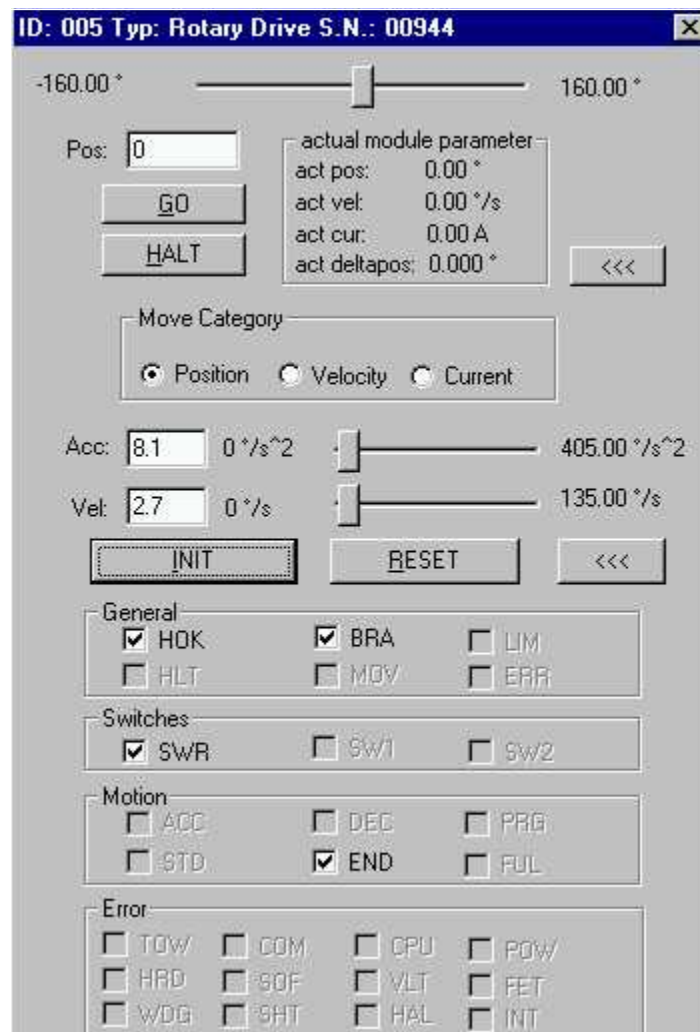
- Auswahlbox zur Auswahl der Bewegungsart (Ramp, Velocity, Current). Eine Änderung in dieser Auswahlbox aktualisiert die Anzeige.
- Einen RESET-/INIT-Button. (Zurücksetzen oder Referenzfahrt eines Moduls)
- Slider zum Einstellen der Bewegung. Die anzuzeigenden Bereichsgrenzen sind die Modulparameter maxpos/minpos, maxvel/minvel, maxcur/mincur.
- Parallel zum Slider gibt es ein Eingabefeld, um die Daten über Tastatur eingeben zu können.
- Slider und Eingabefeld sind synchronisiert.
- Einen GO-/HALT-Button.
- Zwei Eingabefelder für Beschleunigung und Geschwindigkeit. Parallel dazu jeweils einen Schieberegler. Schieberegler und Eingabefeld sind synchronisiert.



Der Moduldialog präsentiert sich zunächst in einer einfachen Form. Die Titelleiste enthält die ID, den Typ und die Seriennummer des ausgewählten Moduls. Die Initialisierung des Moduldialogs ermöglicht anfänglich nur eine Bewegung des Moduls im Rampenmodus, sobald die Referenzfahrt erfolgreich abgeschlossen wurde. Diese Bewegung kann entweder durch Ziehen des Sliders oder durch Eingabe einer Zielposition in das Editfeld und anschließendem Drücken des **Go**-Buttons ausgelöst werden. Geschwindigkeit und Beschleunigung sind hier bei Linearmodulen auf die halben vom Betriebssystem des Moduls vorgegebenen Maximalwerte beschränkt, bei Rotary-Modulen jeweils auf 1/50 des Maximalwerts. Wird der obere Slider im Positionsmodus bis zur gewünschten Endposition gezogen und losgelassen, springt er an die aktuelle Position des Moduls zurück und ermöglicht eine visuelle Kontrolle des Bewegungsfortschritts, indem er ständig der aktuellen Modulposition folgt. Der Wert im Editfeld *Pos* zeigt dabei die Position an, an der der Slider losgelassen wurde und die die Endposition der Bewegung repräsentiert. Im Frame *actual module parameter* werden die aktuelle Position, Geschwindigkeit und Stromaufnahme sowie der aktuelle Schleppfehler (Abweichung Ist-Position von Soll-Position) des angewählten Moduls dargestellt. Der **Halt**-Button beendet die Modulbewegung sofort. Anschließend sind Slider und Go-Button deaktiviert, bis ein **Reset** ausgelöst wurde. Dazu öffnet man mit dem ">>>>"-Button einen weiteren Teil des Dialogs:



Durch Betätigen des **Reset**-Buttons werden evtl. aufgetretene Fehlerbits sowie das Halt-Bit, das nach Betätigen des **Halt**-Buttons gesetzt wird, wieder zurückgesetzt. Gleichzeitig werden die unteren beiden Slider für die Geschwindigkeits- bzw. Beschleunigungskontrolle im Positionsmodus wieder auf ihre Ausgangswerte (s.o.) zurückgesetzt. Das Drücken des **Init**-Buttons bewirkt, daß das Modul eine Referenzfahrt durchführt. In diesem Teil des Dialogs können nun auch Geschwindigkeit und Beschleunigung der Rampenbewegung individuell durch Ziehen des entsprechenden Sliders oder Eingabe eines Zielwerts in das Editfeld variiert werden. Die Bewegungsmodi des Moduls können durch Betätigen der entsprechenden Radiobuttons im Frame *Move Category* ausgewählt werden. Im Geschwindigkeits (*Velocity*)- sowie im Strom (*Current*)-Modus ändert sich die Beschriftung und Funktionalität des oberen Sliders und des zugehörigen Editfeldes. Beim Ziehen des Sliders fährt das Modul nun mit konstanter Geschwindigkeit bzw. Stromaufnahme, bis die Endlagen erreicht sind. Durch Loslassen des Sliders kann die Fahrt sofort unterbrochen werden; die Geschwindigkeit bzw. die Stromaufnahme werden in diesem Fall sofort auf Null zurückgesetzt. Werden in das Editfeld, das nun die Bezeichnung *Vel* bzw. *Cur* trägt, Eingaben für Geschwindigkeit bzw. Stromaufnahme gemacht und dann der **Go**-Button gedrückt, so fährt das Modul solange, bis die Endlagen erreicht sind, der **Halt**-Button gedrückt wurde oder es zu einem Schleppfehler kommt. Ein Schleppfehler tritt z.B. dann auf, wenn sich ein Hindernis im Weg des Moduls befindet oder so hohe Geschwindigkeiten bzw. Stromaufnahmen vorgegeben wurden, daß die voreingestellte Beschleunigung nicht ausreicht und die Differenz zwischen Ist- und Soll- Position zu groß wird. Beschleunigung und Geschwindigkeit können über die unteren beiden Slider bzw. deren zugehörigen Editfeldern nur im Positionsmodus (*Rampmode*) beeinflußt werden. Daher sind sie im Geschwindigkeits- bzw. Strommodus nicht zugänglich. Ein weiterer ">>>"-Button öffnet den Dialog vollständig:



Hier werden nun die im Statuswort **CubeState** enthaltenen Statusbits sichtbar gemacht. Eine gecheckte (mit Häkchen versehene) Checkbox bedeutet, daß das Statusbit gesetzt ist, eine ausgegraute (disabled) Checkbox zeigt ein nicht gesetztes Statusbit an. Die Bedeutung der Statusbits wird mittels Tooltips (kleines Informationsfenster, das sich öffnen, wenn man länger als eine halbe Sekunde mit dem Mauszeiger über einer Statusbox verweilt) erklärt. Die in PCubeDemo verwendeten Statusbit-Bezeichner korrespondieren mit folgenden Statusflags im Statuswort CubeState (siehe ab S. 103):

HOK	<=>	STATE_HOME_OK	BRA	<=>	STATE_BRAKEACTIVE
LIM	<=>	STATE_CURLIMIT	HLT	<=>	STATE_HALTED
MOV	<=>	STATE_MOTION	ERR	<=>	STATE_ERROR
SWR	<=>	STATE_SWR	SW1	<=>	STATE_SW1
SW2	<=>	STATE_SW2	ACC	<=>	STATE_RAMP_ACC
DEC	<=>	STATE_RAMP_DEC	PRG	<=>	STATE_INPROGRESS
STD	<=>	STATE_RAMP_STEADY	END	<=>	STATE_RAMP_END
FUL	<=>	STATE_FULLBUFFER	TOW	<=>	STATE_TOW_ERROR
COM	<=>	STATE_COMM_ERROR	CPU	<=>	STATE_CPU_OVERLOAD

POW	<=>	STATE_POWERFAULT	HRD	<=>	STATE_BEYOND_HARD
SOF	<=>	STATE_BEYOND_SOFT	VLT	<=>	STATE_POW_VOLT_ERR
FET	<=>	STATE_POW_FET_TEMP	WDG	<=>	STATE_POW_WDG_TEMP
SHT	<=>	STATE_POW_SHORTCUR	HAL	<=>	STATE_POW_HALLERR
INT	<=>	STATE_POW_INTEGRALERR			

4.7 Normalbetrieb

Vor Inbetriebnahme der Module, insbesondere von Modulkombinationen (Manipulator), ist ihr Aktionsradius zu ermitteln und zu kennzeichnen. Es ist dafür zu sorgen, daß sich während des Betriebs der PowerCube™-Module niemand im gekennzeichneten Gefahrenbereich aufhält.

Die PowerCube™-Module müssen in regelmäßigen Abständen gewartet und bestimmte Modulteile auf betriebsbedingte Veränderungen untersucht werden (siehe dazu im Kapitel [Wartungshinweise](#)).

Weitere Hinweise finden sich im Kapitel Kommunikationsschnittstelle: [Betriebsweise im Normalfall](#).

4.8 Besondere Betriebssituationen

Bei Modulen ohne Bremse ist darauf zu achten, daß nach Abschalten des Antriebs die Lage der Module durch eventuell daran befestigte Lasten verändert werden kann.

4.9 Betriebsstörungen und deren Beseitigung

Das Betriebssystem der PowerCube™-Module verfügt über eine interne Verwaltung des Antriebsstatus. Dieser Status kann mit dem Statuswort [CubeState](#) abgefragt werden. Die [Bedeutung](#) der einzelnen Bits im Statuswort kann im Kapitel Kommunikationsschnittstelle, Rubrik "Betriebsweise im Normalfall" nachgeschlagen werden, wo auch mögliche Strategien zur Fehlerbehebung vorgeschlagen werden. Es ist ratsam, dieses Statuswort für die einzelnen Module regelmäßig zu abzufragen (polling), um auf eventuell auftretende Fehlerzustände reagieren zu können.

Beim Auftreten von Fehlerzuständen, die nicht über das Statuswort diagnostiziert werden können, empfiehlt es sich, das Modul abzuschalten (stromlos zu machen) und zunächst nach erkennbaren physischen Fehlerquellen (Hindernis im Fahrtweg, Stromführung beschädigt u.ä.) zu suchen. Sollte dies nicht zur Beseitigung des Fehlerzustands führen, dann ist es ratsam, sich zur Fehlerbehebung an AMTEC oder einen seiner Servicepartner zu wenden.

Weitere Hinweise siehe im Kapitel Kommunikationsschnittstelle unter der Rubrik "[Betriebsweise bei Störungen](#)".

4.10 Außerbetriebnahme

Bei der Außerbetriebnahme sind die Hinweise für [besondere Betriebssituationen](#) zu berücksichtigen.

5. Betriebshinweise

Die Beachtung der nachfolgenden Hinweise ist für die Lebensdauer der Produkte entscheidend.

5.1 Hinweise zur Pflege der PowerCube™-Produkte

Die PowerCube™-Produktfamilie ist für den industriellen Einsatz entwickelt worden und aus diesem Grund sehr robust.

Reinigen Sie den PowerCube™ in regelmäßigen Abständen mit einem trockenen Tuch. Entfernen Sie grobe Verschmutzungen, die sich möglicherweise in Hohlräumen und an Kanten bilden. Achten Sie darauf, daß offenliegende Zahnriemen nicht mit Schmutz zugesetzt werden, weil dies die Funktion und Positioniergenauigkeit des PowerCube™ beeinträchtigen kann.

5.2 Oberfläche der PowerCube™-Antriebsmodule

Die Gehäuse der PowerCube™-Antriebsmodule werden aus Aluminium gefertigt. Sie haben eine hohe Oberflächenqualität, die durch eine Oxidschicht veredelt wird. Die Oberflächengüte der PowerCube™-Antriebe ist für ihre Verwendung in Manipulatoren und Robotersystemen von hoher Bedeutung, weil die geometrische Ausrichtung von der Oberflächengüte abhängig ist.

Die Veredelung durch die PowerCube™-blaue Oxidschicht ist somit ein wesentliches Funktionsmerkmal der PowerCube™-Produkte. Aus Fertigungsgründen kann es in der Farbgebung zu geringfügigen Abstufungen kommen, die die Qualität des Produkts aber nicht beeinträchtigen.

Beachten Sie bei der Arbeit mit den PowerCube™-Antrieben, daß die Oberfläche kratzempfindlich ist. Oxidschichten mit Kratzstellen können technologisch bedingt nicht nachbehandelt werden. Im Rahmen der Fertigung werden alle PowerCube™-Module mit einer Spezialfolie weitestgehend vor Kratzwirkung geschützt. Geringfügige Kratzstellen können nicht vollständig vermieden werden und sind von der Amtec-Qualitätssicherung für die Auslieferung zugelassen.

5.3 Wartungshinweise

Alle Wartungsarbeiten dürfen nur bei abgeschalteten PowerCube™-Modulen und von fachkundigem Personal durchgeführt werden, um Schäden an den Modulen zu vermeiden.

5.3.1 Sicherungswechsel

Derzeit sollten die Module bei einem notwendigen Sicherungswechsel zu Amtec eingeschickt werden, da die Prozedur das Öffnen des PowerCube™-Moduls erfordert. Bei neuen Modul-Baureihen wird der Wechsel über eine abnehmbare Stromzuführung realisiert werden können. Dazu wird das Handbuch aktualisiert, welches den Sicherungswechsel ausführlich beschreiben wird.

5.3.2 Einstellung der Endlagen- und Referenzschalter

Endlagen- und Referenzschalter können von Kundenseite nur bei Linearmodulen mit Zahnriemenantrieb sowie mit Kugelgewindetrieb neu justiert bzw. eingestellt werden.

5.3.2.1 Linearmodule mit Zahnriemenantrieb

Der Endschalternocken dient der Markierung der elektrischen Endlagenbegrenzung und des Referenzpunktes der Lineareinheit auf dem Zahnriemen. Dieser Stahlstift muß einen Abstand von ca. 0,4 bis 0,8 mm vom End- bzw. Referenzschalter haben (Abb. 2). Die Module werden von Amtec mit der entsprechenden Einstellung ausgeliefert, während des Betriebs können allerdings Änderungen des Abstands auftreten, die eine Nachjustierung der Endschalter erforderlich machen.

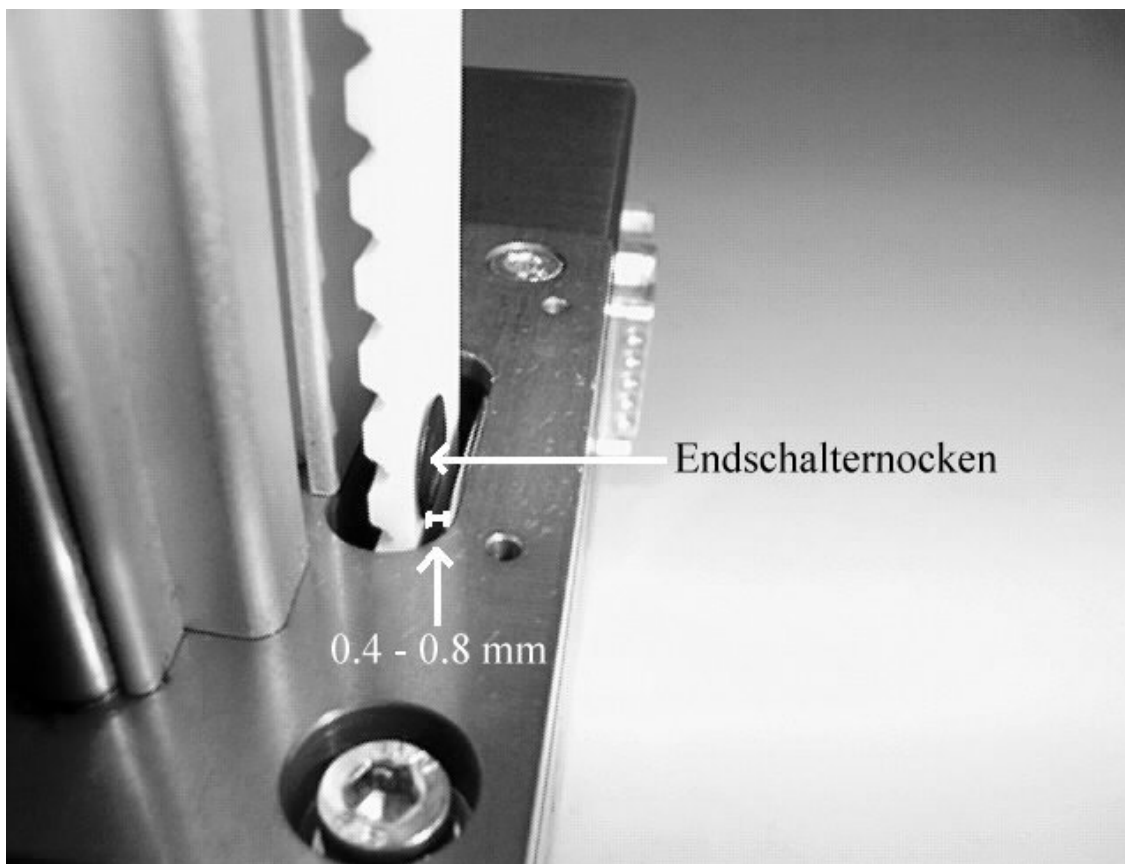


Abb. 2

Die Endlagen- und Referenzschalter befinden sich am Steuer-/Antriebs-Modul C des in Abb. 3 exemplarisch dargestellten Linear-Moduls:

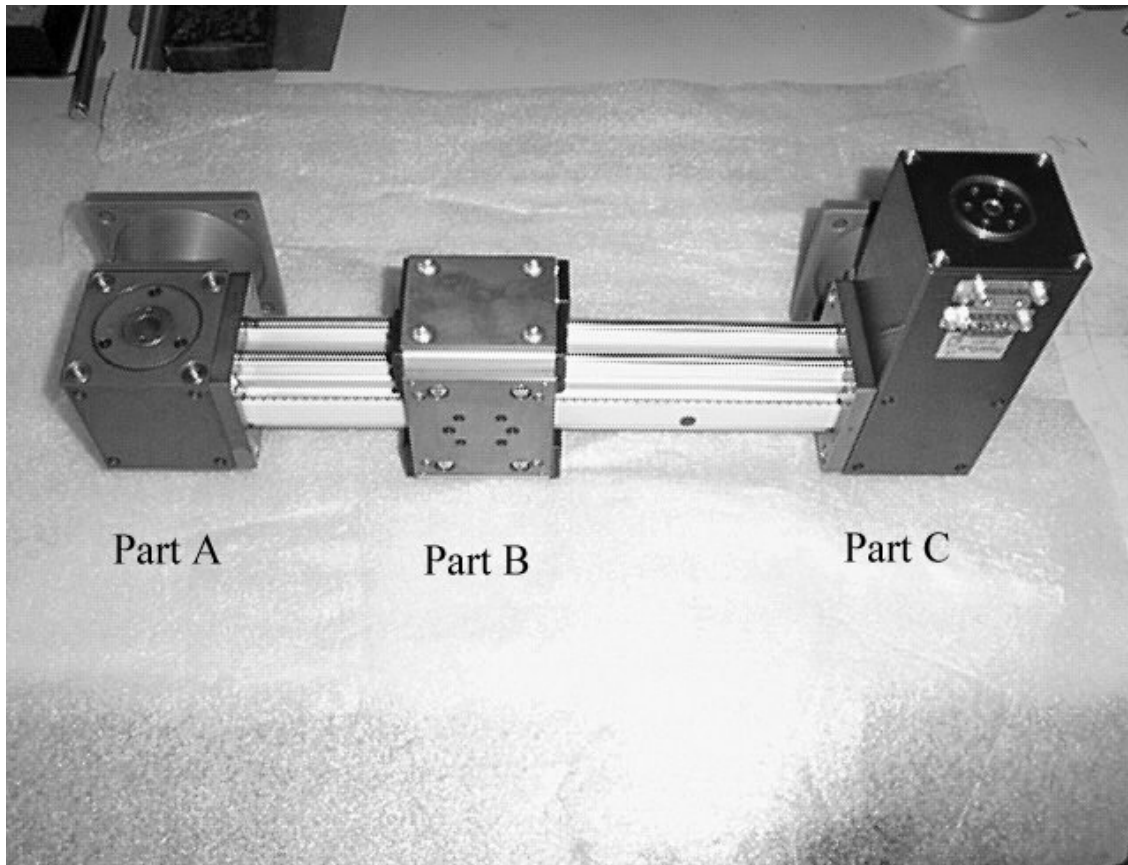


Abb. 3

Der Abstand des Endlagenschalters vom Endschalternocken kann durch Lösen der beiden Schrauben in den Positionen A und B in Abb. 4 und Drücken mit einem dünnen Stift oder Draht an Position C bzw. einem Gegendruck von der anderen Seite nachjustiert werden (Abb 4):

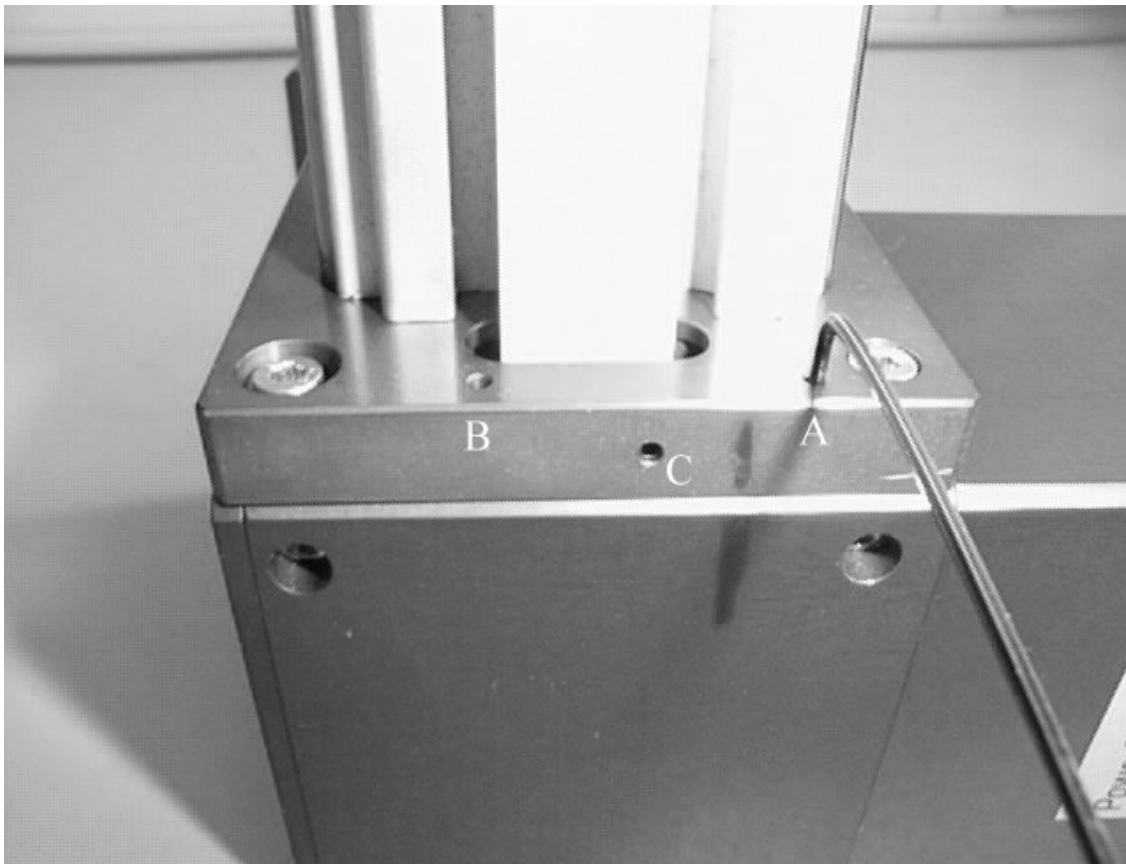


Abb. 4

Alle PowerCube™-Module werden von Amtec mit eingestellten und geprüften Endlagen- und Referenz-Schaltern ausgeliefert. Das Verändern der Lagen insbesondere der Endschalternocken sollte nur von Amtec vorgenommen werden und geschieht sonst auf eigene Gefahr.

5.3.2.2 Linearmodule mit Kugelgewindetrieb

Abb. 5 zeigt ein Linearmodul mit Kugelgewindetrieb in der Gesamtansicht:

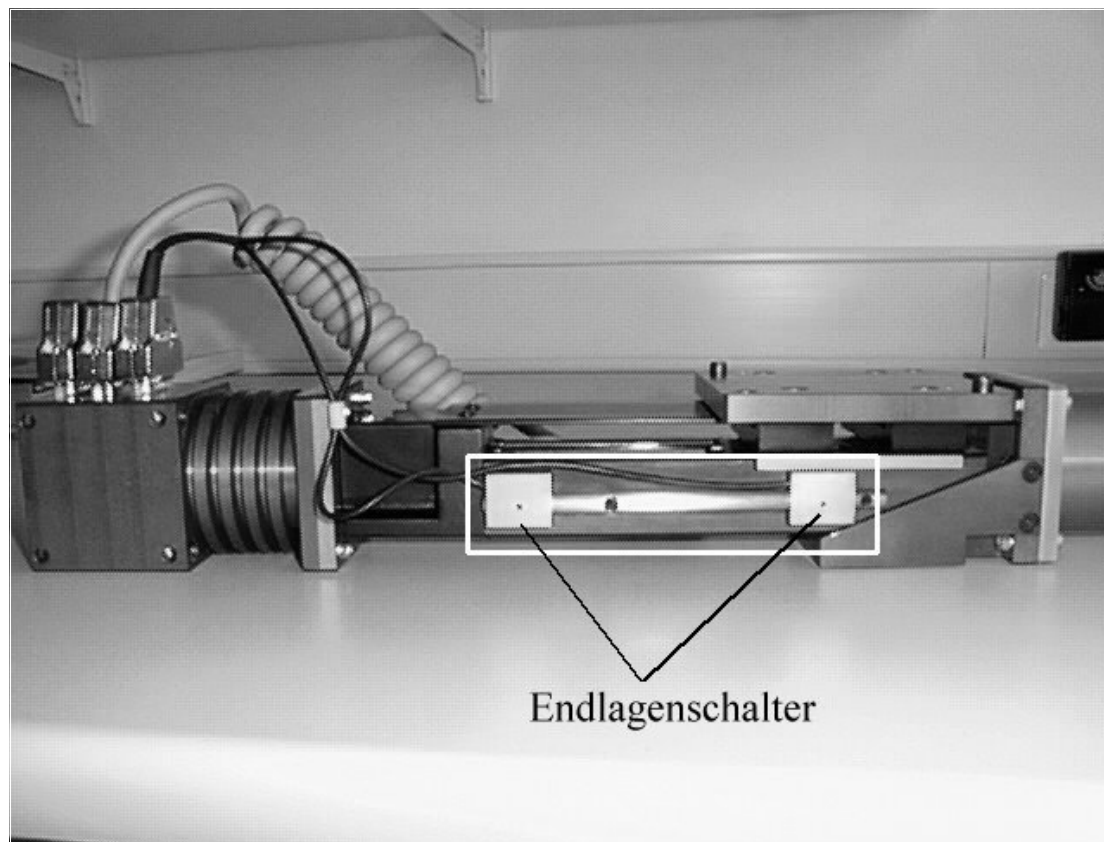


Abb. 5

Der weiß umrandete Ausschnitt zeigt die Lage der Endschalter am Modul sowie der Achse, auf der sie bewegt werden können.

Die Endlagenschalter können mit einem Imbus-Schlüssel gelöst werden (Abb. 6):

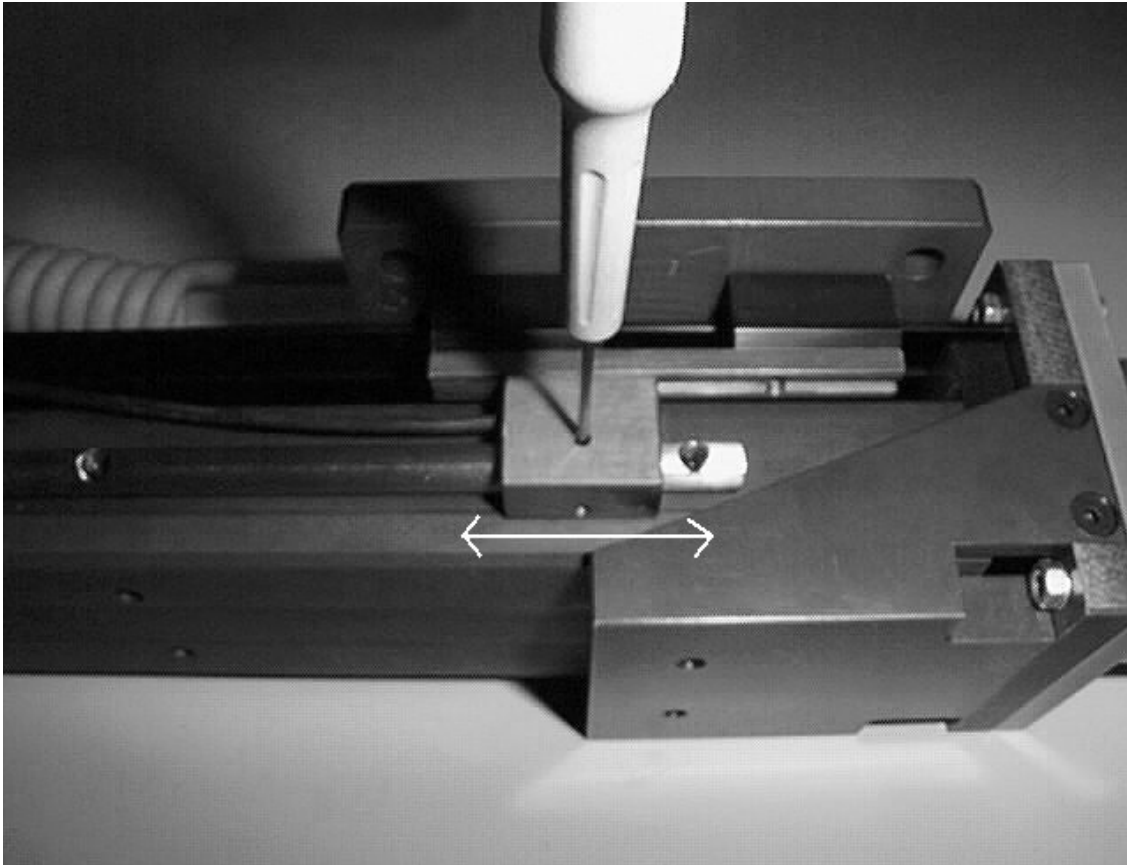


Abb. 6

Dann kann die Lage der Endscharter den Betriebserfordernissen angepasst werden. Es empfiehlt sich unbedingt, die ursprüngliche Position der Endscharter zu markieren, um gegebenenfalls die Ausgangslage wiederherstellen zu können.

5.3.3 Einstellung der Zahnriemenspannung

Modulteil A in [Abb. 3](#) bezeichnet die Umlenkeinheit, in dem sich der Exzenter zum Spannen des Zahnriemens befindet. Zuerst müssen die 3 Schrauben gelöst werden, die den Exzenter fixieren:

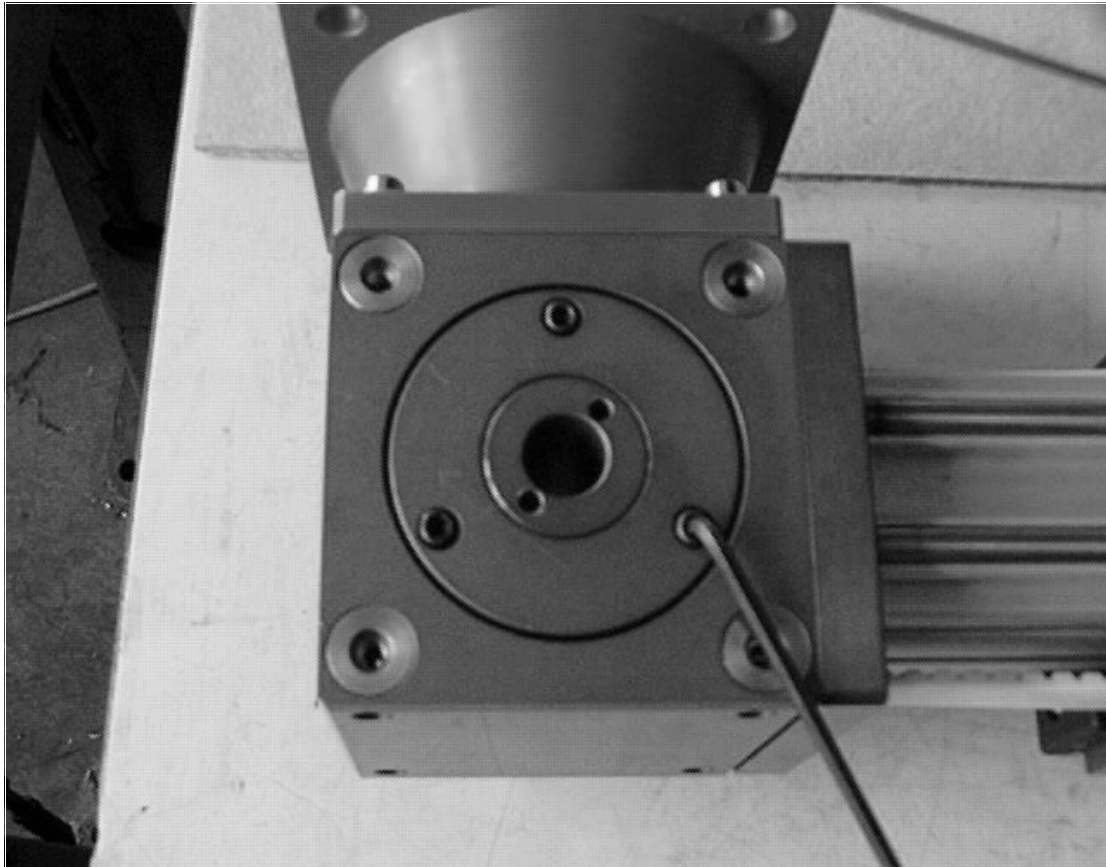


Abb. 7

Dann wird mittels einer Spezialzange oder eines anderen geeigneten Werkzeugs der Exzenter soweit gedreht (gegen oder auch im Uhrzeigersinn), bis die gewünschte Zahnriemenspannung eingestellt wurde:

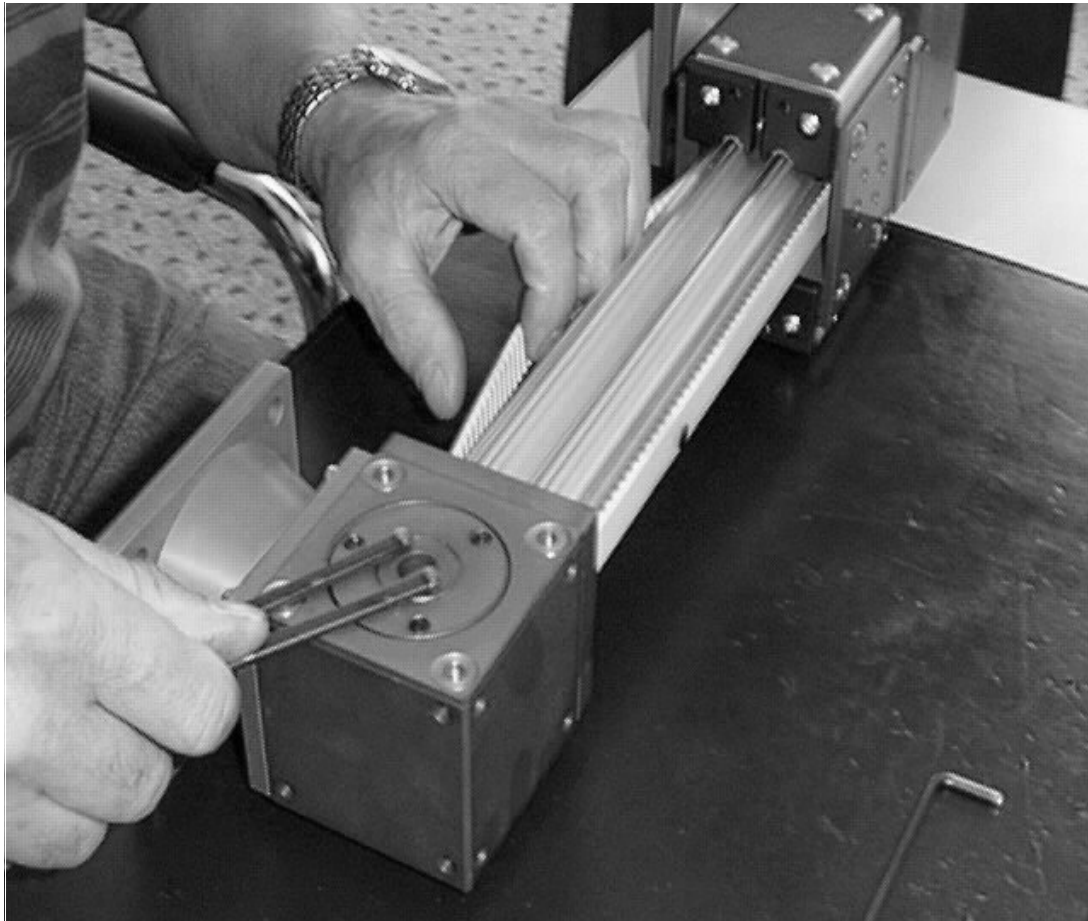


Abb. 8

5.3.4 Einstellung der Laufrollen

Es gibt die PowerCube™-Linearmodule in verschiedene Ausführungen mit unterschiedlichen Lagereinheiten: PLB 70, PLB 90 und PLB 110 (70 mm, 90 mm und 110 mm Kantenlänge des Modulgehäuses). Die Linearmodule PLB 70 und PLB 90 haben pro Lagereinheit je eine Reihe fester Laufrollen (nicht verstellbar) und eine Reihe Laufrollen mit exzentrischen Bolzen, die einen Verstellbereich von $\pm 1,5$ mm ermöglichen. Diese Verstellmöglichkeit erleichtert es, eine spielfreie Führungseinheit zu realisieren und wird für die verschiedenen Modultypen im Folgenden beschrieben:

5.3.4.1 PLB 70

Am Linear Belt 70-Modul sind zunächst die Verbindungsplatten der Rollenführungen des oberen und unteren Teilschlittens zu lösen, wie in Abb. 9 dargestellt:

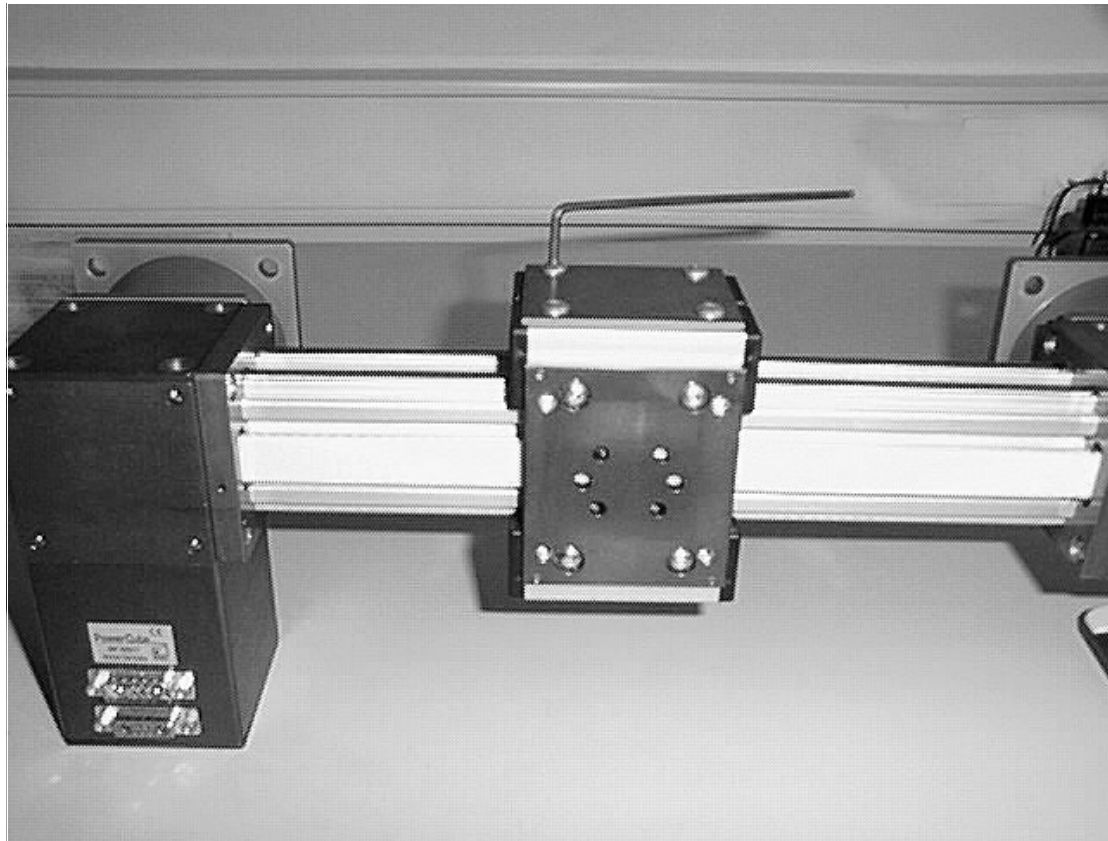


Abb. 9

Nun können die Rollenführungen auseinandergeschoben werden, wodurch die Rollenlager frei zugänglich werden (Abb. 10):

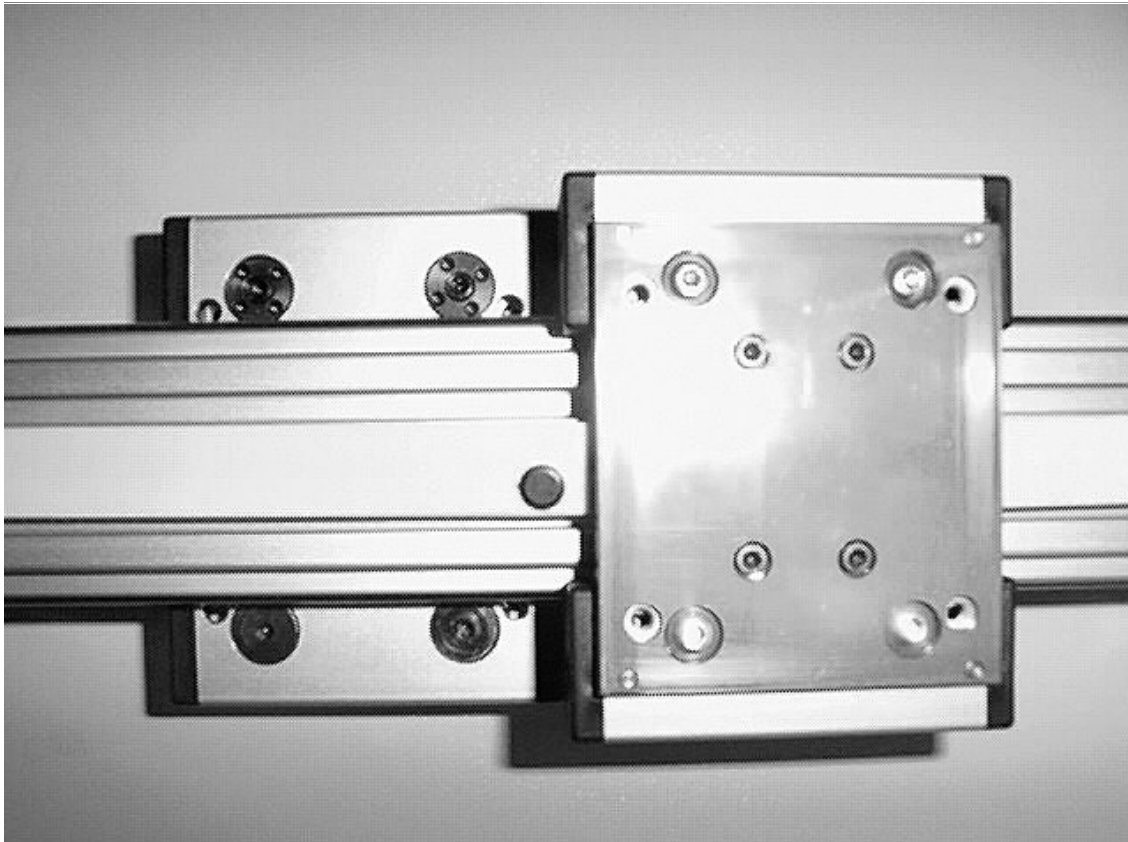


Abb. 10

Die oberen Lagereinheiten mit einem Loch in der Mitte und vier konzentrisch angeordneten Löchern außen können verstellt werden, indem mit einem Spezialwerkzeug (Zange mit Zapfen) durch Drehung außen die Arretierung gelöst wird, während mit einem Imbus-Schlüssel in der Mitte der exzentrisch angeordnete Bolzen gedreht wird und so die Lage der Rolle zur Führungsschiene verändert werden kann (Abb. 11).

Die Gegen-Lagereinheiten sind zentrisch und können daher nicht verstellt werden.



Abb. 11

5.3.4.2 PLB 90

Abb. 12 zeigt einen fertig montierten Schlitten eines PLB 90-PowerCube™-Linearmoduls.

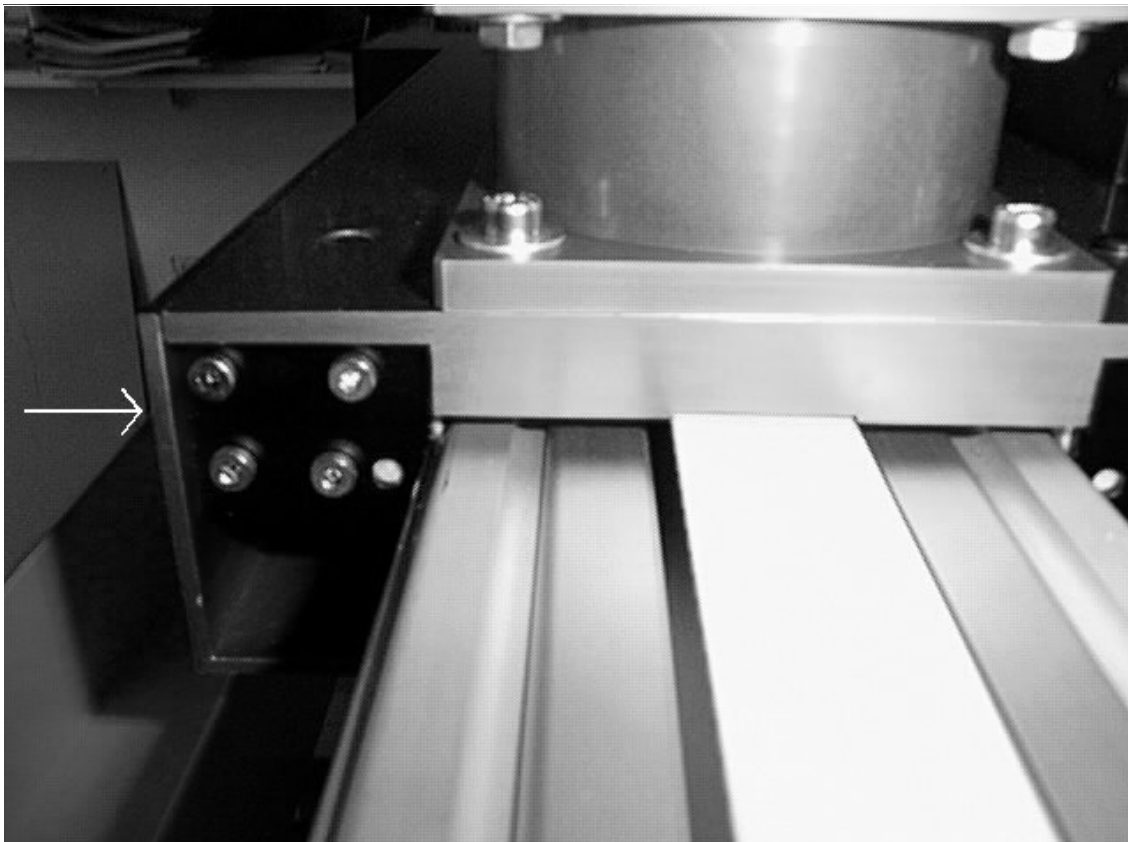


Abb. 12

Wenn die mit einem Pfeil markierte Seitenverkleidung entfernt wird, erhält man Zugang zu den dahinter befindlichen Rollen. Auch hier existieren wie beim PLB 70-Modul zwei verschiedene Lagereinheiten.

Zwei der Laufrollen sind zentrisch gelagert (z) und können nicht verstellt werden, die auf der gegenüberliegenden Seite befindlichen Rollen dagegen lagern auf exzentrischen Bolzen (e). Abb. 13 zeigt eine Aufsicht auf die Unterseite eines demontierten Schlittens mit den entsprechend gekennzeichneten Rollen (e = exzentrisch; z = zentrisch).



Abb. 13

Die exzentrische Rollenlagerung wird in Abb. 14 detaillierter dargestellt. Um die Lage der Rollen verstellen zu können, muß die Sechskantmutter gelöst werden; mit einem Vierkant- oder Maulschlüssel kann dann die Stellung des Exzenterbolzens variiert werden. Dazu kann der Schlitten auf der Führungsschiene verbleiben.

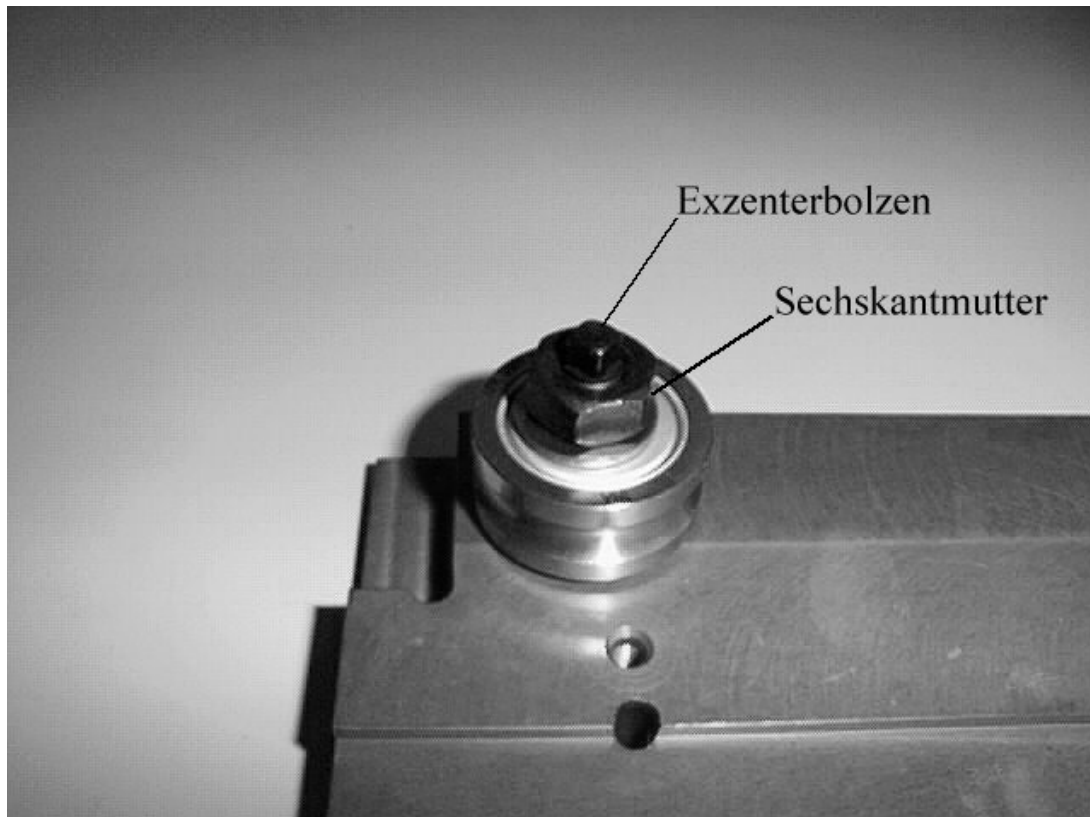


Abb. 14

5.3.5 Grundlagen der Schmierung

Damit die Funktion der Linearführungssysteme nicht beeinträchtigt wird und über einen längeren Zeitraum erhalten bleibt, ist eine Schmierung entsprechend der Umgebungsbedingungen und der spezifischen Anforderungen unbedingt durchzuführen. Im allgemeinen wird eine Nachschmierfrist spätestens nach 100 km Laufleistung bzw. alle 3 Monate zugrundegelegt. Spezielle Einsatzbedingungen können diese Frist verkürzen. Es wird angeraten, bei Modulen mit Linearführungssystemen, die besonderen Belastungen unterliegen, die Führungselemente in kürzeren Abständen auf ausreichende Schmierung zu prüfen. Die Abschmierintervalle sind dann gegebenenfalls individuell den besonderen Anforderungen anzupassen. Zu berücksichtigende Faktoren sind u.a.

- Extreme Betriebstemperaturen
- Kondens- oder Spritzwassereinwirkung
- hohe Schwingungsbeanspruchung
- Einsatz im Vakuum oder Reinräumen
- Beaufschlagung mit speziellen Medien (z.B. Dämpfe, Säuren oder Kohlenwasserstoffe)
- hochdynamischer Betrieb
- permanente kleine Hubbewegungen
(Hubweg < 2 Wagen-/Mutterlängen).

Für den Einsatz unter normalen Betriebsbedingungen des Linearführungssystems werden Schmierstoffe mit folgenden Mindestanforderungen empfohlen:

Schmierstoff	DIN-Kennzeichen	DIN-Nummer	Bemerkung
Schmierfett	KP 2 - K	51502/51825	Lithium-Seifenfett
Schmieröl	CLP32 - 100	51517 Teil 3	ISO VG 32 - 100

Die Schmierstoffmenge ist abhängig von der Hublänge. Bei besonders langen Hüben sind kürzere Schmierintervalle oder eine größere Schmierstoffmenge erforderlich, damit der Ölfilm über die ganze Länge der Laufbahnen nicht abreißt.

Sind die Linearführungssysteme Kühlmitteln ausgesetzt, sollte ein Schmierstoff mit einer dynamischen Viskosität von ca. 68 cst oder ein besonders emulsionsbeständiger Schmierstoff eingesetzt werden. Außerdem ist in kürzeren Intervallen und größeren Mengen nachzuschmieren. Eine Ölschmierung ist für Linearführungssysteme, die mit hohen Belastungen und Geschwindigkeiten verfahren werden, zu empfehlen.

Ein besonders emulsionsbeständiges Gleitöl (dynamische Viskosität: ca. 68 cst) ist z.B. Mobil Vectra Oil No. 2S.

5.3.5.1 Schmierung der Ölabbreifer in Linearmodulen mit Zahnriemenantrieb

5.3.5.1.1 PLB 70

Abb. 15 zeigt die Seitenansicht eines Schlittens. Es gibt insgesamt 8 Abstreifsysteme mit jeweils einer Schmierstelle, die in oben genannten Abständen mit Schmiermittel versorgt werden müssen.

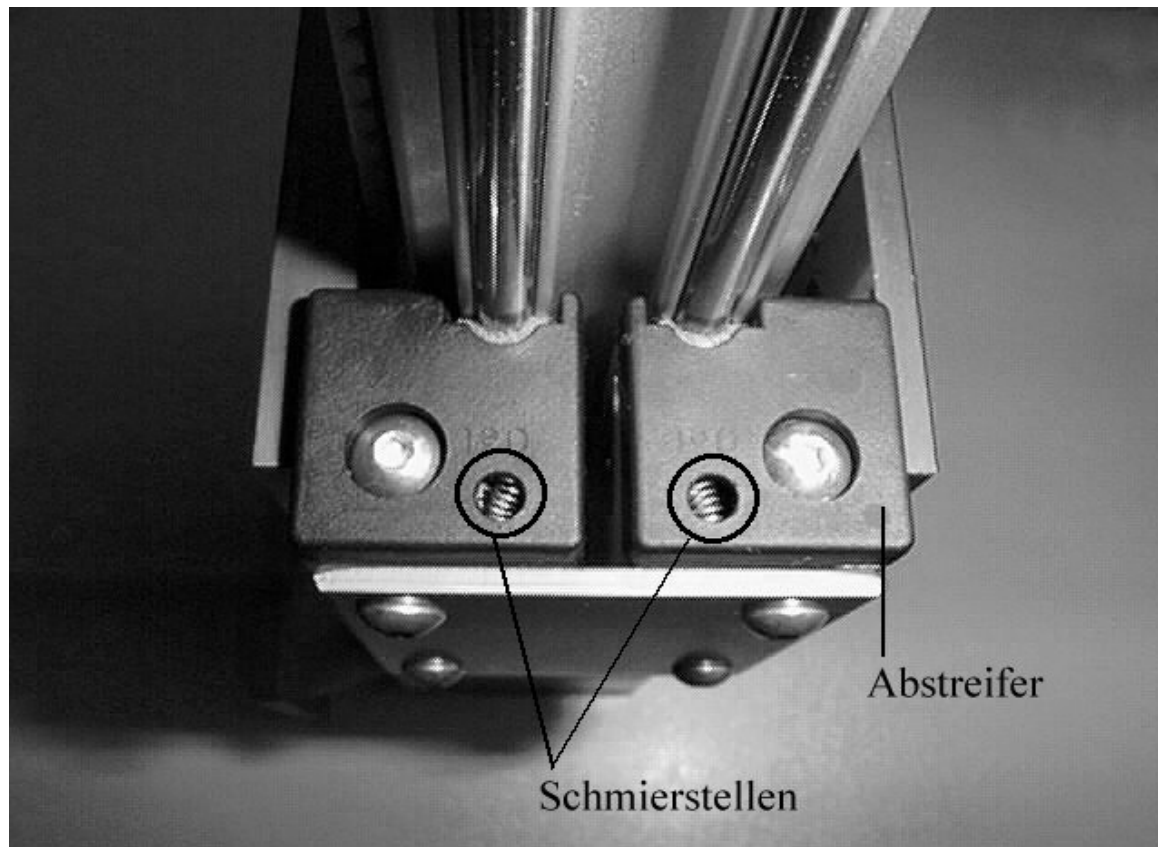


Abb. 15

5.3.5.1.2 PLB 90

Das Schmiermittel wird an der Stelle, die in Abb. 16 mit einem Kreis markiert ist, angewendet. Hier sind pro Schlitten vier Abstreifer mit jeweils einer Schmierstelle zu berücksichtigen.

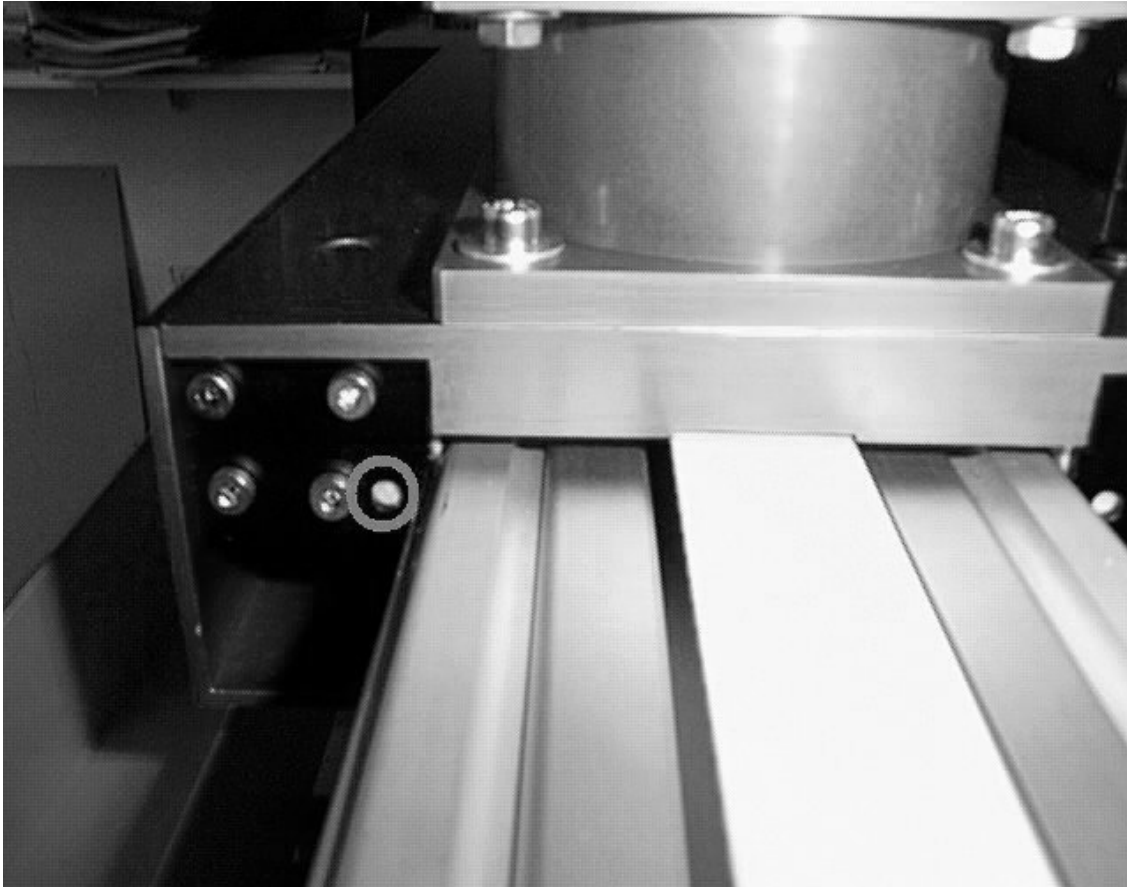


Abb. 16

5.3.5.2 Schmierung der Kugelumlaufspindel

Abb. 17 zeigt ein Linearmodul mit Kugelgewindetrieb in der Gesamtansicht:

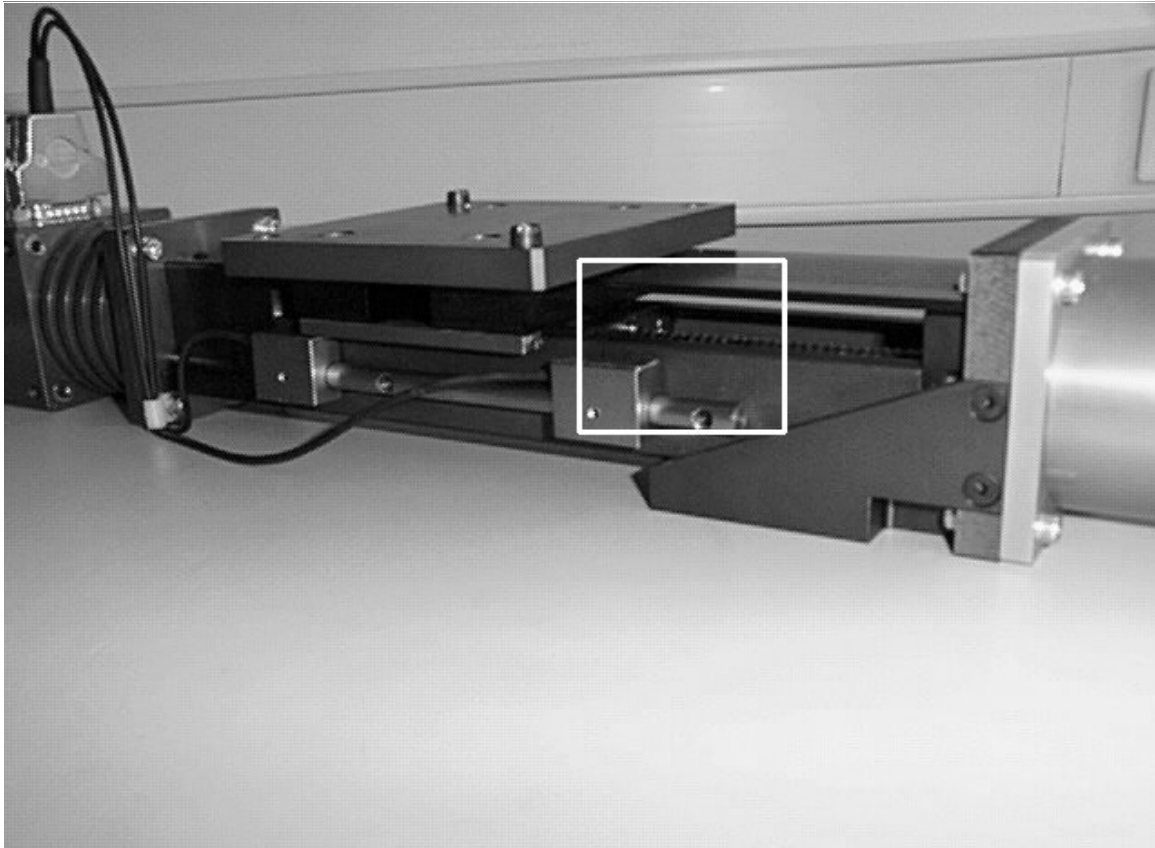


Abb. 17

Der weiß umrandete Ausschnitt in Abb. 17 enthält die Schmierstelle und wird in Abb. 18 vergrößert dargestellt:

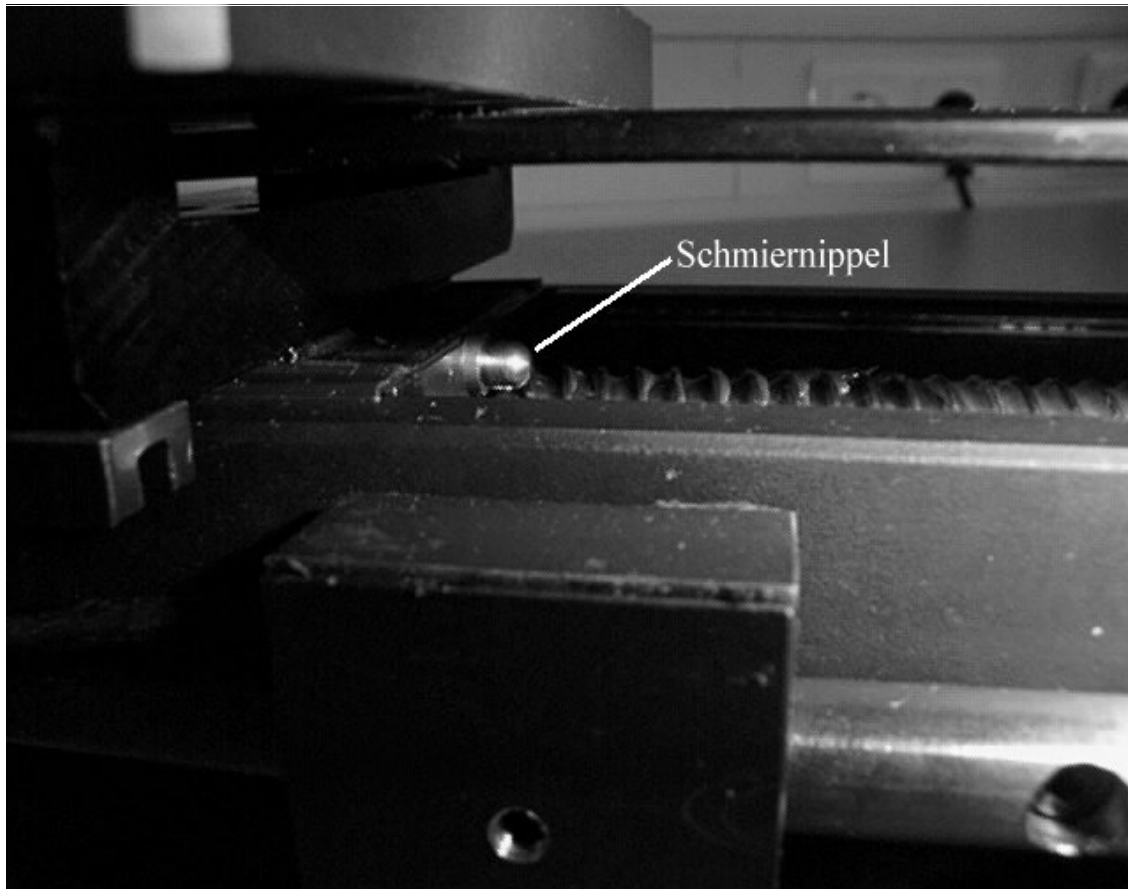


Abb. 18

Mit der Fettpresse Typ MG70 (Vertrieb: THK; Kontakt: THK GmbH, Headquarters Düsseldorf, Hubert-Wollenberg-Str. 15, D-40878 Ratingen, Tel.: ++49 +21 02 74 25-0, Fax: ++49 +21 02 74 25-2 99, E-mail: info@thk.de, Internet: www.thk.de) und den mitgelieferten Abschmieradaptoren können alle Baugrößen der Linearführungssysteme in PowerCube™-Linearmodulen mit Kugelumlaufspindeln (Linear Spindel 70 / 90 / 110) abgeschmiert werden.

5.3.5.3 Schmierung der Linearführungen (am Greifer)

Abb. 19 zeigt ein 2-Finger-Parallelgreifermodul (Gripper 70 / 90) in der Gesamtansicht:

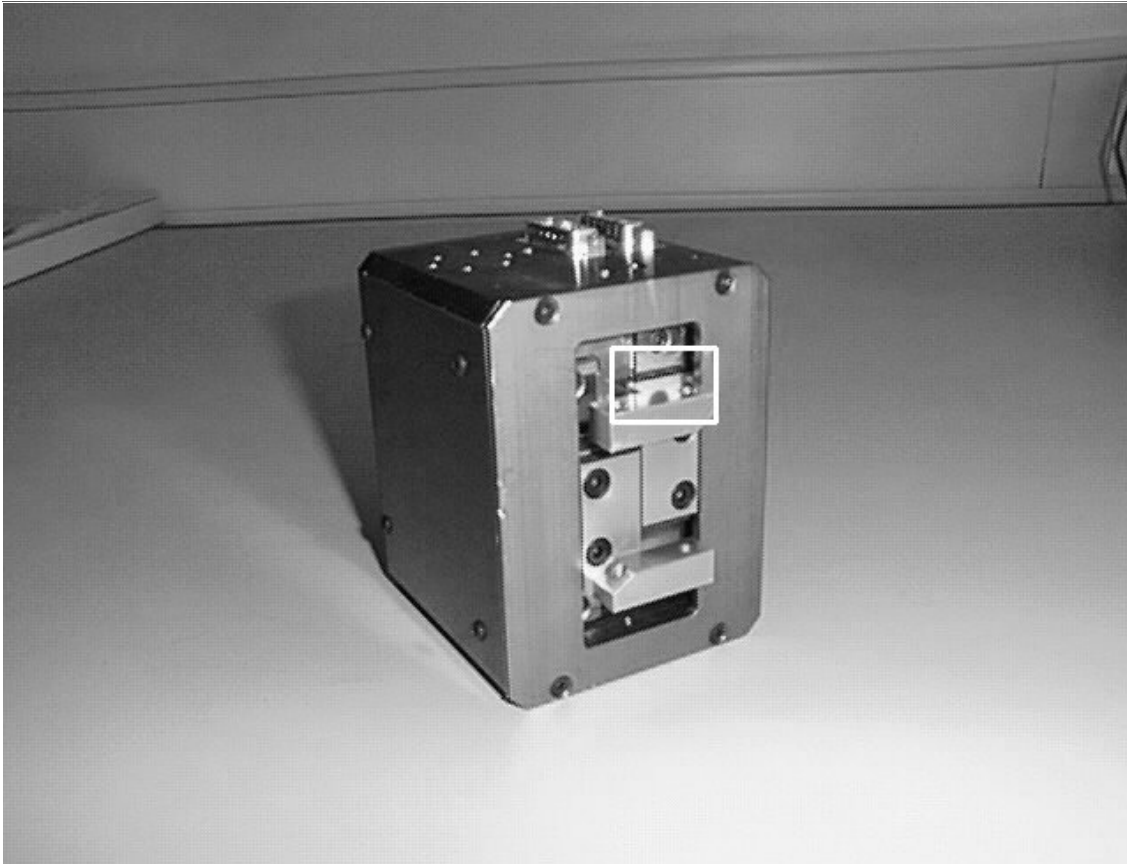


Abb. 19

Der weiß umrandete Ausschnitt in Abb. 19 enthält die Schmierstelle eines Greifers (der zweite Greifer wird an entsprechender Stelle unten abgeschmiert) und wird in Abb. 20 vergrößert dargestellt:

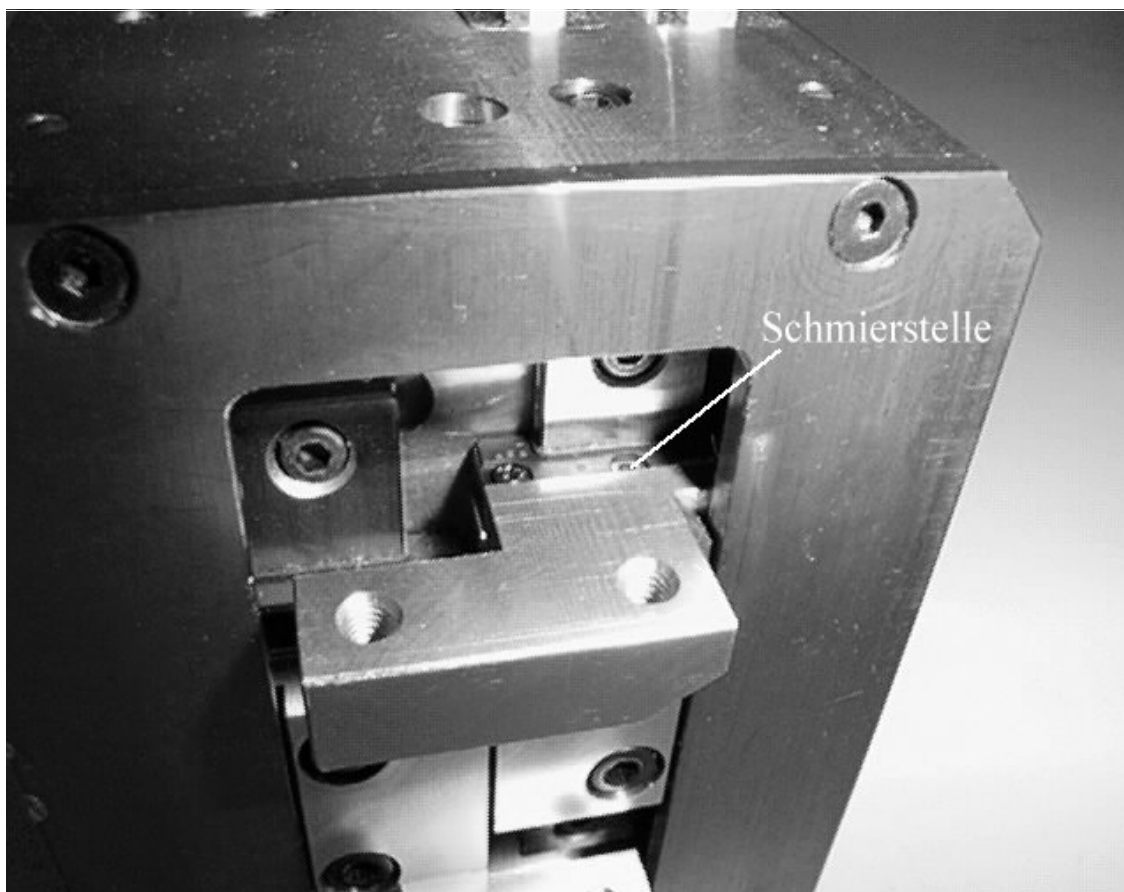


Abb. 20

An der in Abb. 20 bezeichneten "Schmierstelle" sitzt normalerweise ein Schmiernippel ähnlich der in Abb. 18 gezeigten Schmierstelle des Linearmoduls mit Kugelgewindetrieb. Zum Abschmieren kann wieder die oben erwähnte Fettpresse Typ MG70 mit entsprechendem Adapter verwendet werden. Sind die Schmiernippel wie auf diesem Bild abgeschraubt worden, so genügt es auch, etwas Öl auf die Laufschiene unterhalb der Greiferbacken zu geben.

6. Konfiguration der PowerCube™-Module

Das Programm PowerCube™-PowerConfig ersetzt das bisher bekannte Konfigurationsprogramm PowerCube™-easyConfig und dient der Konfiguration aller PowerCube™-Module inclusive derer mit den neuen Betriebssystemversionen ab 3.5.00.

6.1 Die Konfigurationssoftware "PowerConfig"

Das Produkt PowerConfig ist ein spezielles Software-Werkzeug für die PowerCube™-Produktfamilie und sollte nur von Anwendern benutzt werden, die über gute Kenntnisse der internen Parameter der PowerCube™-Module verfügen. PowerConfig unterstützt den Anwender beim Neuparametrieren von Modulen mit einer Reihe von Sicherheitshinweisen und Abfragen. Trotzdem wird dringend geraten, vor Anwendung von PowerConfig zum Brennen neuer Modulparameter Rücksprache mit AMTEC halten, um Schäden am Modul zu vermeiden, die durch das Brennen falscher Parameter in das Modul entstehen können. Bitte achten Sie darauf, daß die Konfiguration immer nur mit einem Modul am Bus durchgeführt wird! Dabei ist zu berücksichtigen, daß sich PowerCube™ Handgelenksmodule (PW) sowie Lift'n Turn-Module (PLT) wie zwei Module verhalten, da sie zwei konfigurierbare Bewegungsachsen besitzen. Weiterhin muß sowohl auf der Interface-Karte wie auf dem zu brennenden Modul ein Abschlußwiderstand angebracht sein!

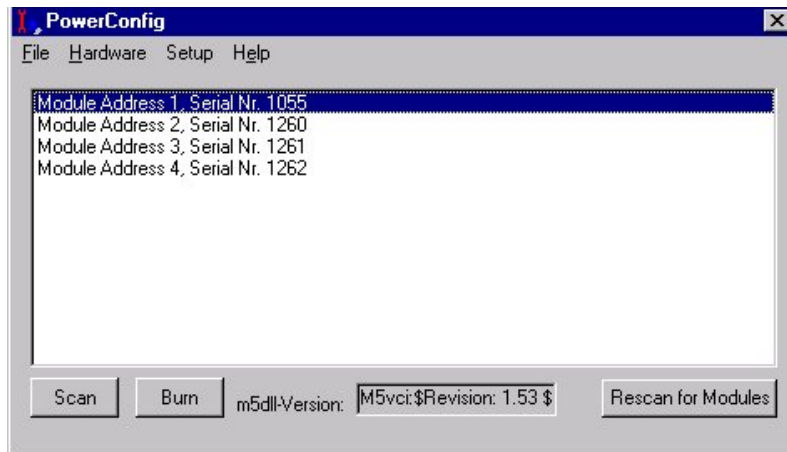
Bitte beachten Sie **unbedingt** den Abschnitt über [Fehlerbehandlung](#) am Ende des Kapitels **vor** der Installation.

6.2 Installation

- Stellen Sie sicher, daß vor der Installation von PowerConfig die Testsoftware für Ihre Module korrekt läuft.
- Die Distribution von PowerConfig wird mit dem neuen Windows-Installer von Microsoft ausgeliefert. Sollte das Windows-Installationsprogramm noch nicht auf Ihrem Rechner installiert sein, kann es mit dem Aufruf von setup.exe installiert werden und verbleibt als Teil des Betriebssystems. Unter Umständen muß der Rechner danach neu gestartet werden.
- Die eigentliche Installation von PowerConfig wird durch Doppelklicken auf die Installationsdatei PowerConfig.msi ausgelöst.
- Klicken Sie 'Next' im daraufhin erscheinenden Welcome-Fenster
- Im anschließenden Dialog können Sie den vorgeschlagenen Installationspfad Ihren Wünschen anpassen oder ohne Änderung übernehmen. Klicken Sie danach auf den Button 'Next'.
- Danach wird überprüft, ob sich das Windows-Installationsprogramm bereits auf Ihrem Rechner befindet. Mit Betätigen des 'Next'-Buttons werden Ihre Einstellungen wirksam und die Installation beginnt.
- Bitte beachten Sie die Informationen in der angezeigten Readme-Datei. Sollten Sie einen VCI-Treiber der Firma IXXAT > Version 1.12 benutzen, so kopieren Sie bitte die VCI_WS32.dll aus dem VCI114-Verzeichnis in das Installationsverzeichnis von PowerConfig, um einen einwandfreien Betrieb des Programms zu gewährleisten.
- Die Installation ist nun abgeschlossen. Klicken Sie den 'Finish'-Button.
- Im Startverzeichnis von Windows wird ein Eintrag „PowerConfig“ vorgenommen. Um das Programm starten zu können, klicken Sie bitte diesen Eintrag. Sie können auch einen Shortcut auf dem Desktop erstellen (einfach das Programmicon mit Drag and Drop aus dem Fenster des geöffneten Programmverzeichnisses auf den Desktop ziehen und loslassen).

6.3 Das Hauptfenster

Das folgende Bild zeigt das Hauptfenster von PowerConfig



Das Hauptfenster von PowerConfig ist in drei Bereiche unterteilt.

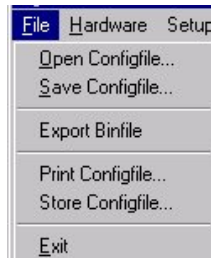
1. Die Menüleiste der Anwendung. Der Inhalt der Menüleiste wird im nächsten Abschnitt beschrieben.
2. Die PowerCube™-Modulliste. In dieser Liste finden Sie alle angeschlossenen Module. Hier finden Sie die Seriennummer und die CAN-Bus-Adresse der Module. Alle Änderungen werden für das Modul durchgeführt, welches in der Liste selektiert ist (gilt nur für PW- und PLT-PowerCube™-Module). Durch Doppelklicken auf einen Listeneintrag wird der Dialog zum Ändern der Parameter für das betreffende Modul geöffnet.
3. Die Buttons. Der Scan Button wird verwendet, um die angeschlossenen Module zu finden und in der Modulliste anzuzeigen. Durch erneutes Betätigen des Scan Buttons kann, unter Berücksichtigung eines eventuell zwischenzeitlich geänderten Initstrings, erneut nach Modulen gesucht werden. Der Rescan Button kann verwendet werden, um die Liste zu aktualisieren, falls Sie zur Laufzeit des Programms neue Module angeschlossen haben. Alle Änderungen die Sie bisher getätigt haben gehen verloren, falls Sie zuvor Ihre Änderung noch nicht gebrannt haben. Ein zwischenzeitlich geänderter Initstring wird nicht berücksichtigt. Mittels des Burn Buttons gelangt man in den Brenn-Dialog, in dem die geänderten Parameter dauerhaft in das Modul gebrannt werden können. In der Mitte zwischen den Buttons klärt eine Anzeige über die Version der vom Programm genutzten m5dll auf (nur, falls Scanvorgang erfolgreich war). Während des Scanvorgangs wird an dieser Stelle eine Fortschrittsanzeige eingeblendet, die den ungefähren Fortgang des Scanprozesses wiedergibt.

Bitte beachten Sie, daß beim Starten des Programms noch nicht auf die Hardware zugegriffen wird.

6.3.1 Die Menüleiste

6.3.1.1 Der Menüpunkt Datei

Die erste Spalte in der Menüleiste ist das Dateimenü.



Mit dem Dateimenü haben Sie die Möglichkeit, Modulkonfigurationen auf einem Speichermedium zu sichern oder von dort zu laden.

Open Configfile

Um eine Konfigurationsdatei zu öffnen, muß das passende Betriebssystem ausgewählt worden sein. Dies geschieht entweder automatisch beim erfolgreichen Scannen nach Modulen oder kann manuell durch entsprechende Auswahl im Menü Hardware (siehe dort) selektiert werden. Bei manueller Auswahl muß der Benutzer das für die Konfigurationsdatei (*.cnf) passende Betriebssystem kennen. Zwischen Konfigurationsdateien für alte Betriebssysteme (250x und 251x) und neueren Betriebssystemen (ab 350x) kann aufgrund der unterschiedlichen Dateilänge (ersichtlich z.B. im Windows Explorer) leicht unterschieden werden. Klicken Sie nun einfach auf den Menüeintrag "Open Configfile" und wählen eine Datei in der darauffolgenden Auswahlbox aus. Die geladenen Parameter überschreiben die aktuellen Parameter des dabei selektierten Moduls (nicht dauerhaft).

Beispiel:

Sie haben ein Modul am CAN-Bus angeschlossen und erfolgreich per Sacn erkannt. Wählen Sie anschließend im Dateimenü den Punkt "Open Configfile" und laden eine Konfigurationsdatei in den Speicher des Computers. Die Parameter des Moduls sind nun überschrieben (nur im PC). Sie können Ihre alte Konfiguration wiederherstellen, indem Sie den Button "Scan" bzw. "Rescan for Modules" drücken.

Save Configfile

Dieser Menüpunkt wird anwählbar, wenn erfolgreich nach Modulen gescannt wurde oder eine Konfigurationsdatei mit dem Menüpunkt "Open Configfile" ausgewählt wurde. Wenn Sie die Konfiguration eines ausgewählten Moduls oder eine geladene Konfigurationsdatei speichern möchten (mit der Extension .cnf), wählen Sie diesen Menüpunkt. Sie können eine gespeicherte Konfigurationsdatei später auch einem anderen Modul (oder demselben Modul) zuweisen. Beachten Sie, daß die Seriennummer dabei nicht geändert werden kann. Markieren Sie einfach ein Modul und klicken dann auf "Open Configfile".

Export Binfile

Dieser Menüpunkt wird anwählbar, wenn erfolgreich nach Modulen gescannt wurde. Bei dieser Option haben Sie die Möglichkeit, alle Parameter und das Betriebssystem zu sichern. Diese Datei kann später verwendet werden, um Module nicht nur mit Parametern, sondern auch mit dem Betriebssystem zu brennen.

Print Configfile

Für Ihre Dokumentation können Sie sich die Parameter eines jeden angeschlossenen Moduls ausdrucken lassen.

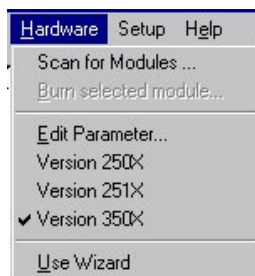
Store Configfile

Neu ist die Möglichkeit, die Parameter einer Konfigurationsdatei als ASCII-Datei mit der Endung .out zu speichern.

Exit

Wenn Sie diesen Menüpunkt klicken, beenden Sie das PowerConfig Programm ohne weitere Meldungen.

6.3.1.2 Der Menüpunkt Hardware



Scan for Modules

Der Menüpunkt wird verwendet, um die angeschlossenen Module zu finden und in der Modulliste anzuzeigen.

Burn selected module

Dieser Menüpunkt ist nur zugänglich, wenn zuvor erfolgreich nach Modulen gescannt oder eine Datei geladen wurde. Wenn Sie diesen Menüpunkt auswählen, öffnet sich der Dialog "Burning Module". (siehe Abschnitt "Dialog Burning Module"). In das selektierte Modul können die Parameter (und, falls gewünscht, das Betriebssystem) eingebrannt werden.

Edit Parameter

Dieser Menüpunkt ist nur zugänglich, wenn zuvor erfolgreich nach Modulen gescannt wurde bzw. wenn eine Konfigurationsdatei mit "Open Configfile" geladen wurde. Er öffnet den Hauptkonfigurationsdialog, mit dem Sie einzelne Parameter des selektierten Moduls setzen können. Alle Änderungen werden nur im Speicher des Computers gehalten und nicht im ROM des Moduls. Sie müssen Ihre Änderungen brennen, um sie als ROM-Defaults sichern zu können. Ein Sonderfall ist die Auswahl des Menüpunkts "Use Wizard" (Häkchen gesetzt). Dies wird an entsprechender Stelle erläutert.

Der Hauptkonfigurationsdialog ist unter ["Die Property Pages - Modulparameter editieren"](#) beschrieben.

Versionserkennung

Zu jedem Modul wird Ihnen automatisch angezeigt, mit welcher Betriebssystemversion dieses Modul ausgestattet ist. Sie haben die Möglichkeit, ein Update des Modulbetriebssystems von 250X auf 251X durchzuführen. Einige wenige Parameter haben sich zwischen den Versionen 250X und 251X geändert. Ein Update der Betriebssystemversionen 25xx auf 350X ist nicht möglich und wird von PowerConfig auch nicht zugelassen. Die Auswahl des Betriebssystems durch Selektieren des betreffenden Menüpunkts bestimmt das Betriebssystem, das im Dialog "Burning Module" zum Flash-Down-

load angezeigt wird. Folgende Auswahl steht Ihnen zur Verfügung:

- Version 250X
- Version 251X
- Version 350X

(Zusammensetzung der Versionsnummern)

Use Wizard

Wird dieser Menüpunkt aktiviert (Häkchen gesetzt), so kommt man beim Aufruf des Menüpunkts "Edit Parameter" in den neuen Dialog "Select Application" (siehe Benutzung des Konfigurations-wizards).

6.3.1.3 Der Menüpunkt Setup

Der Menüpunkt "Setup" sieht wie folgt aus:



Config

Durch Aufruf dieses Untermenüpunkts gelangt man in den Dialog "Initialization". Dieser Dialog sollte nach Installation des Programms als erstes aufgerufen werden, da hier z.B. der Initstring und die Kommunikationsschnittstelle gesetzt werden. Einzelheiten siehe unter "Der Dialog Config".

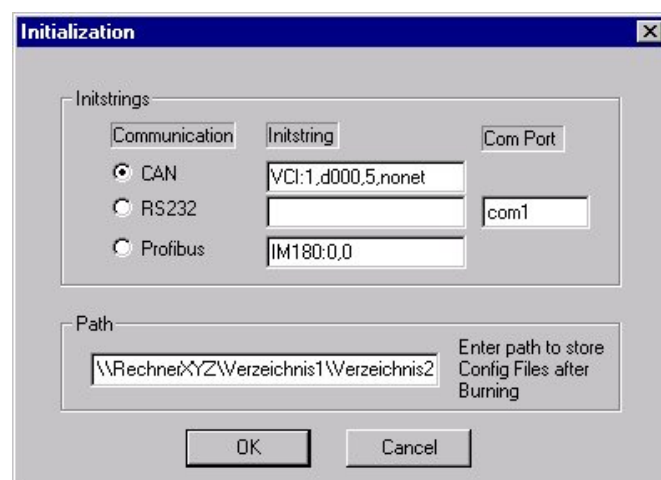
Dialog Language

Bislang noch nicht unterstützt, ist dieser Untermenüpunkt dafür vorgesehen, die Spracheinstellungen des Programms zwischen deutsch und englisch (später auch französisch) umschalten zu können.

6.4 Die Dialoge

6.4.1 Der Dialog "Initialization"

Dieser Dialog hat folgendes Aussehen:



Er wird aufgerufen, um festzulegen, über welche Kommunikationsschnittstelle und mit welchem Initstring die Kommunikation mit dem Modul aufgebaut wird. Die Angabe des Com Port ist für die Kom-

munikation über die RS232-Schnittstelle wichtig, so also auch, wenn z.B. über RS232 im sogenannten Bootstrapmode gebrannt werden soll. Die Pfadangabe bezieht sich auf den Dateipfad, in dem nach dem Brennen automatisch die Konfigurationsdatei des gebrannten Moduls abgelegt wird. Um dieses Feld wieder verlassen zu können, müssen Sie einen gültigen Pfad angeben, auf dem Sie Schreibrechte haben. Die Angabe eines Netzwerkpfades ist möglich. Der Dateiname der abgespeicherten Konfigurationsdatei setzt sich aus der Seriennummer und der Endung .cnf zusammen. Beim Betätigen des OK-Buttons werden die eingetragenen Einstellungen in der Ini-Datei PowerCfg.ini im Windows-Verzeichnis abgespeichert (siehe unter "Die Ini-Datei PowerCfg.ini") und sind sofort wirksam (z.B. beim Scannen nach Modulen). Mit Cancel werden alle Änderungen verworfen (Ausnahme: ungültige Pfadeingabe).

6.4.2 Die Benutzung des Konfigurationswizards

Ist der Menüpunkt "Use Wizard" aktiv (Häkchen gesetzt), so wird beim Aufruf des Menüpunkts "Edit Parameter" sowie nach erfolgreichem Scannen beim Doppelklick auf ein selektiertes Modul der Dialog "Select Application" geöffnet:

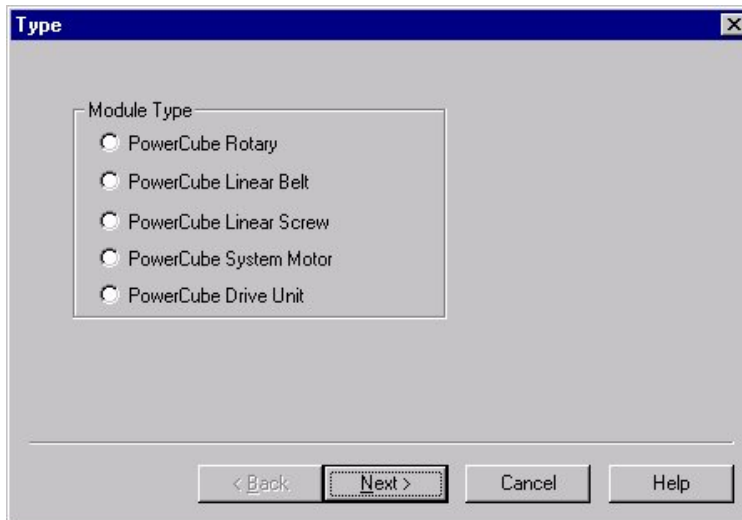
Der Dialog "Select Application"



Hier können Sie auswählen, ob Sie direkt weiter in den Hauptkonfigurationsdialog (Read and Change Settings) oder in einen speziellen Wizardmodus (Configure new Module) wechseln wollen. Dieser Wizardmodus ist nur für Module mit neuer Elektronik mit den Betriebssystemversionen 350X vorgesehen und kann daher nur bei Auswahl des Menüpunkts "Version 350X" genutzt werden! Bei Auswahl der Option "Configure new Module" und Betätigen des OK-Buttons gelangt man in einen Konfigurationsdialog, der es mit wenigen Mausklicks gestattet, die typischen Parameter für ausgewählte Modulklassen zu setzen.

Der erste Dialog dient zur Auswahl des Modultyps:

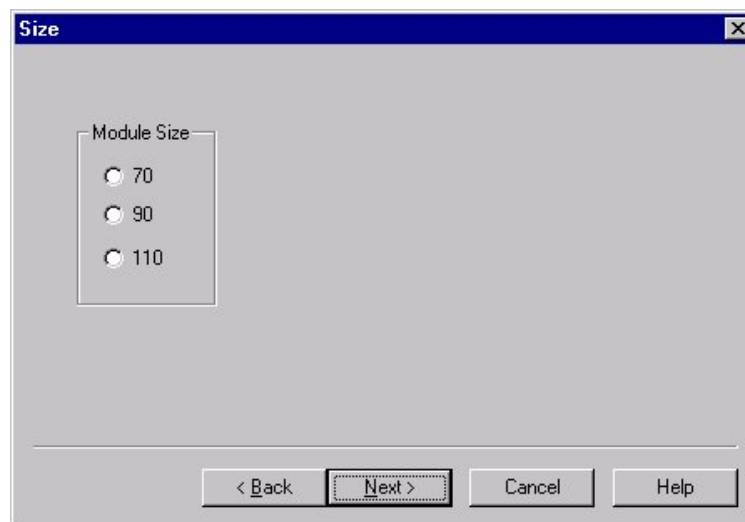
Der Wizard-Dialog "Type"



Nach Auswahl eines Modultyps kann mit Betätigen des Next Buttons in den nächsten Wizard-Dialog gewechselt werden.

Im zweiten Wizard-Dialog kann die Dimensionierung des ausgewählten Modultyps selektiert werden:

Der Wizard-Dialog "Size"

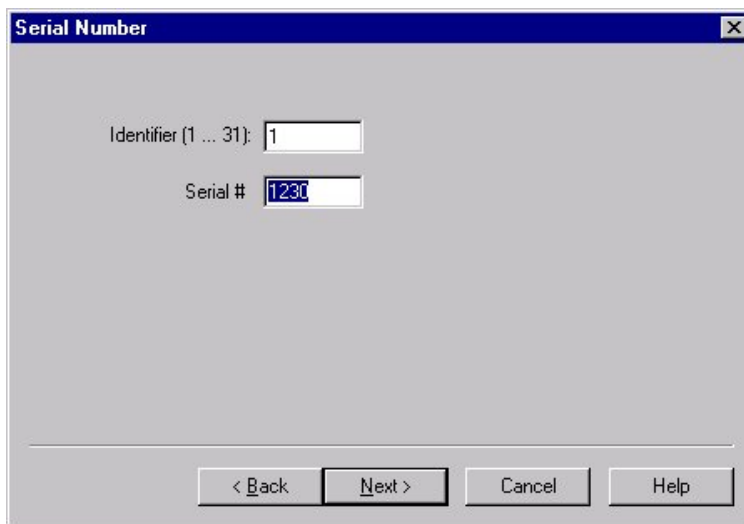


Nach Auswahl der Baugröße kann mit dem Next Button zum nächsten Wizard-Dialog gewechselt werden. Bitte beachten Sie, daß zur Zeit die Modul-Konfigurierung mit Hilfe des Wizards nur für PowerCube™-Drehmodule der Baugrößen 70 und 90 (benötigte Dateien: PR70_35.cnf und PR90_35.cnf) sowie für PowerCube™-Linearmodule mit Zahnriemen der Baugröße 70 (benötigte Datei: PLB70_35.cnf) unterstützt wird. Sollten diese Dateien nicht in Ihrem Lieferumfang enthalten sein, so können sie bei Bedarf via E-Mail von AMTEC bestellt werden. Zukünftige Versionen von PowerConfig werden gegebenenfalls auch weitere Modultypen und -baugrößen unterstützen. Über neue PowerConfig-Versionen können Sie sich auf der Support-Seite der AMTEC-Homepage, http://www.powercube.de/index_support.html, informieren.

Nach Auswahl eines unterstützten Modultyps und einer unterstützten Baugröße gelangen sie in den nächsten Wizard Dialog, der Ihnen die Möglichkeit der Auswahl einer Seriennummer und der Modul-

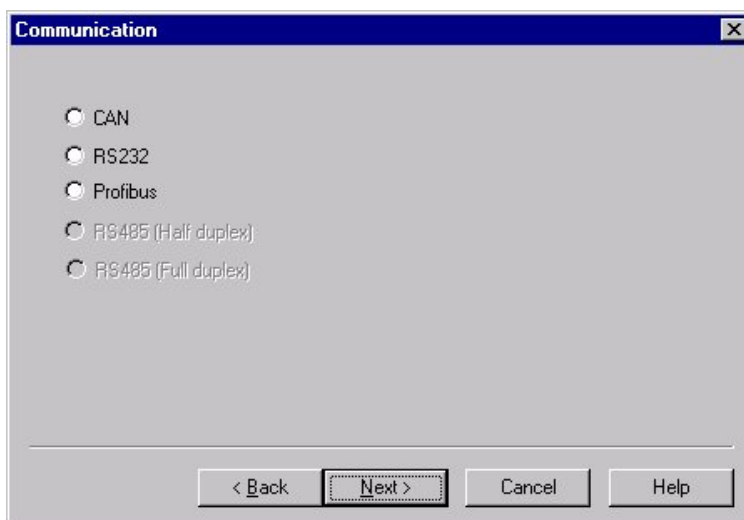
adresse (Modul-ID) bietet:

Der Wizard-Dialog "Serial Number"



Bitte berücksichtigen Sie, daß hierbei auch die Vergabe einer neuen Seriennummer möglich ist. Wir bitten Sie jedoch, bei der Rekonfiguration eines Moduls die auf dem Modul angegebene Seriennummer zu verwenden, da der Support von Seiten AMTECs durch falsche Seriennummern extrem erschwert wird und wir für Schäden, die aufgrund falsch eingebrannter Seriennummern entstehen, keine Haftung übernehmen. Nach Eingabe der korrekten Daten kann mit dem Next Button zum nächsten Wizard-Dialog gewechselt werden:

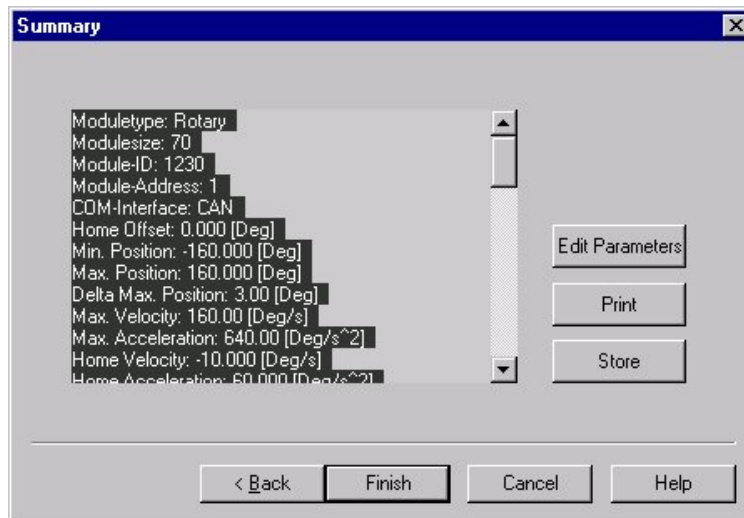
Der Wizard-Dialog "Communication"



In diesem Dialog kann die Kommunikationsschnittstelle, über die Sie mit Ihrem Modul kommunizieren, festgelegt werden. Hierbei ist zu beachten, daß das Kommunikationsinterface nicht nur über die Software rekonfiguriert werden kann, sondern auch Hardwareänderungen am Modul vorgenommen werden müssen. Daher wird **dringend abgeraten**, dem Modul eine andere Kommunikationsschnittstelle zuzuordnen, als es vor dem Brennen hatte, da die Kommunikation sonst nicht mehr hergestellt werden kann. Die Kommunikationsschnittstelle RS485 (Halbduplex oder Vollduplex) wird zur Zeit noch nicht unterstützt.

Nach dem Betätigen des Next Buttons gelangen Sie in den letzten Wizard-Dialog, in dem eine Zusammenfassung der neu zu brennenden Parameter angezeigt wird, die sich aus Ihrer Auswahl und der im Hintergrund geladenen Parameterdatei zusammensetzen:

Der Wizard-Dialog "Summary"



Bei Bedarf können hier Parameter noch verändert werden, indem mit dem Back Button zu den gewünschten Wizard-Dialogseiten zurückgesprungen wird und dort neue Werte gesetzt werden. Weiterhin ist es durch Betätigen der Taste „Edit Parameter“ auch möglich, in den Hauptkonfigurationsdialog zu springen und dort einzelne Parameter zu ändern. Mit den Tasten Print bzw. Store kann die so ausgewählte Modulkonfiguration ausgedruckt bzw. als ASCII-Datei gespeichert werden. Mit Cancel verlässt man den Konfigurationswizard und verwirft alle Einstellungen. Die Help-Funktion wird zur Zeit nicht unterstützt. Das Betätigen des Finish Buttons ruft den "Burning Module"-Dialog auf (siehe dort). An diesem Punkt können keine Parameteränderungen mehr vorgenommen werden. Mit Betätigen des Close Buttons werden alle Einstellungen verworfen.

6.4.3 Die Property Pages - Modulparameter editieren

Der Hauptkonfigurationsdialog kann durch Anwahl des Untermenüpunkts "Edit Parameter" nach Laden einer Konfigurationsdatei oder nach erfolgreichem Scannen nach Modulen auch durch Doppelklicken auf das gewünschte Modul aufgerufen werden.

Dieser Dialog unterteilt sich in drei Bereiche:

1. Register für unterschiedliche Parametergruppen. Hinter jedem Register befindet sich ein Eigenschaftsfenster. Sie können zwischen den Registern hin und her schalten. Sie brauchen zuvor nicht den OK-Button klicken. Ihre Änderungen in den jeweiligen Registern bleiben erhalten.
2. Der Bereich, in dem Sie die Eigenschaftenseite verlassen können. Bei einem Klick auf den Cancel-Button gehen Ihre Einstellungen verloren. Klicken Sie hingegen den OK-Button, so bleiben Ihre Einstellungen erhalten.
3. Der dritte Bereich ist der wichtigste Bereich. Er sieht je nach Wahl des Registers anders aus. Es gibt insgesamt fünf Registerblätter.

Die erste, nach Aktivierung sichtbare Seite des Hauptkonfigurationsdialogs ist das Registerblatt "Identification".

6.4.3.1 Der Konfigurationsdialog Identification

The screenshot shows the 'Property Sheet' dialog box with the 'Identification' tab selected. The 'Type' section has 'Linear Belt' selected, with 'tooth wheel diameter' set to 46.69 mm. The 'Motor' section has 'max. rpm' at 3272.41, 'Nominal Current' at 5 A, 'Performance' at 125 W, 'Max. Velocity (calculated)' at 80 mm/s, 'Acceleration Factor' at 4, 'Gear Ratio' at 100, and 'Torque Ratio' at 1. The 'Position Parameters' section has 'Homeoffset' at 0 mm, 'Minimum Position' at -160 mm, 'Maximum Position' at 160 mm, and 'Max Delta Position' at 1 mm. The 'Identification' section has 'Serial Number (ID)' at 1055, 'Module Address' at 1, 'Cube Version' at 2511, and 'Burn Counter' at 1.

Folgende Parameter können hier gesetzt werden:

Type

Mit den Optionsbuttons können Sie den Modul-Typ angeben, der korrespondierende Modulparameter wird in den nebenstehenden Eingabefeldern gesetzt.

- Lead: dieser Parameter wird für Linearmodule mit Spindeltrieb verwendet. Geben Sie die Spindelsteigung in Millimeter ein.
- Tooth wheel diameter: dieser Parameter wird nur für lineare PowerCube™-Module mit Zahnriemenantrieb verwendet. Geben Sie den Wirkdurchmesser der antreibenden Zahnscheibe in Millimeter ein.
- Rotary: der Faktor für Drehmodule ist immer gleich eins, daher ist kein Eingabefeld vorhanden.

Abhängig von Ihrer Auswahl werden alle Einheiten sofort umgestellt. Alle Einheiten für Linearmodule werden auf Millimeter umgestellt. Die Einheiten für Drehmodule werden auf Grad umgestellt. Bitte beachten Sie, daß keine Konvertierung der Werte stattfindet.

max. rpm

Die maximal mögliche Drehzahl Ihres Motors. Diesen Wert entnehmen Sie bitte der Produktspezifikation. Dies ist eines der Felder, das zur Berechnung der empfohlenen Höchstgeschwindigkeit herangezogen wird. Die Eingabe eines neuen Wertes verändert die empfohlene Maximalgeschwindigkeit sofort. Hierbei ist zu beachten, daß die Eingabe eines unganzzahligen Wertes nur möglich ist, wenn erst die vollständige Zahl eingegeben wird und der Dezimalpunkt zum Schluß gesetzt wird.

Nominal Current

Der erforderliche Nominalstrom der Module beträgt 5A (Baugröße:70) bzw. 10A (Baugrößen 90 und 110). Dies entspricht den Einstellungen 15A bzw. 30A im Eingabefeld Max. Current der Vorgängersoftware *easyConfig*.

Performance

Die Motorleistung. Die Eingabefelder für Nominalstrom und Motorleistung sind gekoppelt. Soll eine Motorleistung von 250/500W auf 125W geändert werden, dann muß der Nominalstrom vorher auf kleiner oder gleich 5A zurückgesetzt werden.

Max. Velocity (calculated)

Rechenfeld, das unter Berücksichtigung der max. Motordrehzahl (max. rpm), der Getriebeunter/übersetzung (Gear Ratio) und der Kenngröße für Linearmodule (Lead bzw. Tooth wheel diameter) die empfohlene Höchstgeschwindigkeit der Module berechnet. Die Berechnung erfolgt bei Änderungen im max. rpm-Feld sofort; bei Änderungen in den Feldern Gear Ratio, Lead und Tooth wheel diameter muß zur Neuberechnung in ein anderes Feld geklickt werden. Der berechnete Geschwindigkeitswert gilt als maximaler Grenzwert und sollte keinesfalls überschritten werden. Das Ändern der Modulhöchstgeschwindigkeit darüber hinaus geschieht auf eigene Verantwortung!

Gear Ratio

Dieser Wert gibt die Getriebeübersetzung Ihres Antriebs an (z.B. 160 für eine Untersetzung von 160:1 oder 0,5 für eine Übersetzung von 1:2). Daher sollte keinesfalls der werkseitig voreingestellte Wert verändert werden. Ausnahme: Motion Controller und Systemmotoren, in die kundenseitig ein Getriebe eingebaut wird.

Torque Ratio

Dieser Wert ist immer auf "1" voreingestellt und kann derzeit nicht verändert werden.

Invert Motor Rotation Direction

Dreht die Motorlaufrichtung um. Dieser Wert ist ab Werk korrekt voreingestellt und darf nicht verändert werden. Nur unter der Modulbetriebsversion 350X zugänglich.

Homeoffset

Mit diesem Wert können Sie die "Position 0" des Moduls nach Ihren Wünschen konfigurieren, falls die Home position einmal nicht exakt mit dem von Ihnen gewünschtem Nullpunkt übereinstimmt.

Max./Min. Position

Diese beiden Parameter setzen die Bereichsgrenzen für auszuführende Bewegungen. Die Module akzeptieren im späteren Betrieb keine Werte, die außerhalb dieser Grenzen liegen.

Max Delta Position

Mit diesem Parameter setzen Sie die maximal erlaubte Abweichung der Ist-Position von der vom Modul berechneten Soll-Position. Befindet sich das Modul in Bewegung und wird diese Abweichung überschritten, so wird das Fehlerwort „TOW“ zurückgeliefert und ein „HALT“ ausgelöst.

Serial Number

Die Seriennummer des Moduls. Diese wird von AMTEC vergeben und darf nicht verändert werden (siehe auch "Der Wizard-Dialog Serial Number").

Module Address

Dieser Parameter setzt die physikalische Adresse des Moduls und kann zwischen 1 und 31 liegen. Bitte beachten Sie, daß Verwendung gleicher Moduladressen in einem Verbund mehrerer Module zu Kommunikationsproblemen führt!

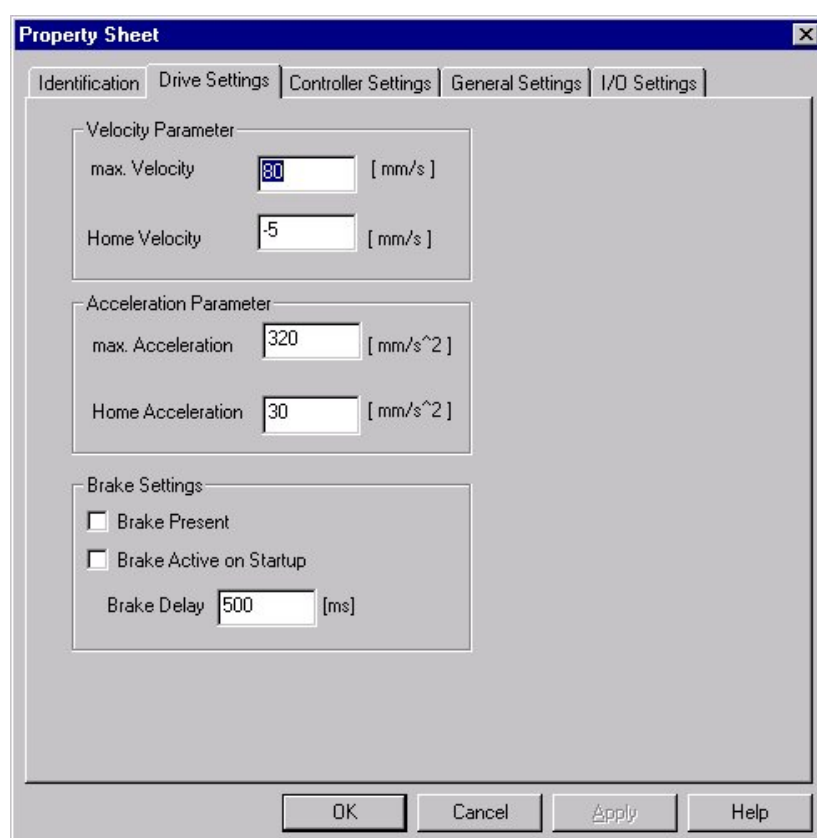
Cube Version

Dieser Wert zeigt die Version des Betriebssystems und kann nicht editiert werden.

Burn Counter

Zähler, der die Anzahl der am Modul durchgeführten Brennvorgänge protokolliert. Kann nicht editiert werden und wird zur Zeit nicht unterstützt.

6.4.3.2 Der Konfigurationsdialog Drive Settings



max. Velocity

Dieser Parameter setzt die maximale Geschwindigkeit der PowerCube™-Module. Die maximale Geschwindigkeit leitet sich aus der max. Drehzahl des antreibenden Motors und den Übersetzungsverhältnissen ab. Sie sollte den im Feld "[Max. Velocity \(calculated\)](#)" berechneten Wert nicht übersteigen. Die PowerCube™-Module akzeptieren im späteren Betrieb keine Werte, die größer sind als der hier eingestellte Wert.

Home Velocity

Mit der hier eingestellten Geschwindigkeit wird die "Home position" angefahren. Dieser Wert kann auch vorzeichenbehaftet angegeben werden, um die Richtung der Referenzfahrt vorzugeben.

max. Acceleration

Dieser Parameter setzt die maximale Beschleunigung der PowerCube™-Module. Die maximale Beschleunigung wird durch die Nennbeschleunigung des antreibenden Motors bestimmt. Die PowerCube™-Module akzeptieren im späteren Betrieb keine Werte, die größer sind als der hier eingestellte Wert.

Home Acceleration

Mit der hier eingestellten Beschleunigung wird die "Home position" angefahren.

Brake Present

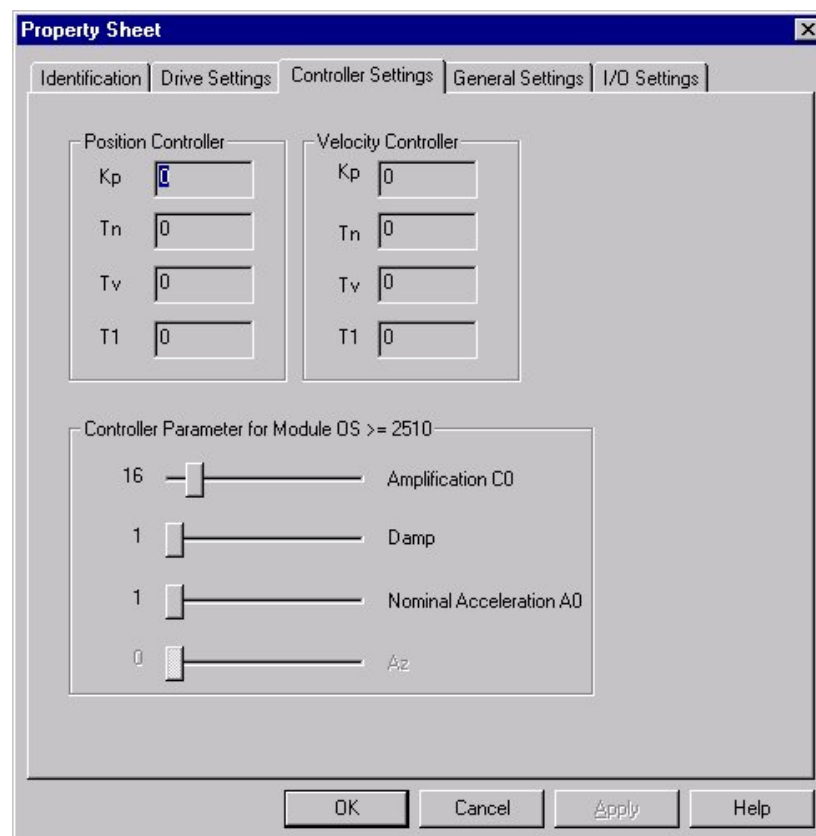
Bei Aktivierung dieser Eigenschaft schaltet der Antriebscontroller nach Abschluß der Bewegung automatisch die Bremse ein. Die Dauer zwischen Halt und Aktivierung der Bremse kann mit „Brake Delay“ bestimmt werden (s.d.).

Brake Active on Startup

Ist dieser Punkt aktiviert, so bleibt die Bremse nach dem Einschalten aktiv. Sie wird erst gelöst, wenn ein Fahrbefehl zur Ausführung kommt.

Brake Delay

Dieser Parameter bestimmt die Dauer für die Zuschaltung der Haltebremse nach Beendigung des letzten Bewegungskommandos. Er sollte nicht kleiner als 500 ms und nicht größer als 5000 ms gewählt werden.

6.4.3.3 Der Konfigurationsdialog Controller Settings

Diese Seite ist nur für technisch versierte Personen gedacht. Hier können Sie die Regelkreisparameter im Mikroprozessor ändern. Je nach Betriebssystem steht Ihnen nur der obere Bereich (Version 250X) oder die unteren Schieberegler (Versionen 251X und 350X) zur Verfügung.

Diese Werte sind werksseitig von AMTEC mit sinnvollen Werten voreingestellt und geprüft worden. Sie sollten ohne ernsthafte Notwendigkeit nicht verändert werden. Mit dem Programm "m5apitst.exe", das mit der m5dll mitgeliefert wird, können sie zur Laufzeit des Programms die drei Reglerparameter CO, Damp und AO neu setzen (mit der Kommandomenüfolge (D)rive, (S)et, (N)ew) und das Modulverhalten mit Move Loop-Befehlen bei Ihren Lastfall testen. Haben Sie die passenden Werte gefunden, so beenden Sie m5apitst und starten PowerConfig, ohne die Stromversorgung des Moduls abzuschalten. Die neuen Reglerparameter werden nun direkt von PowerConfig übernommen und können sofort gebrannt werden. Werden die neuen Controllerparameter nicht in das Modul eingebannt, gehen Sie beim Abschalten der Stromversorgung des Moduls verloren.

Amplification CO

Dieser Wert stellt einen Verstärkungsfaktor dar. Je größer CO gesetzt wird, desto geringer wird die Verstärkung.

Damp

Dieser Wert stellt einen Dämpfungsfaktor dar. Je größer Damp gesetzt wird, desto größer wird die Dämpfung.

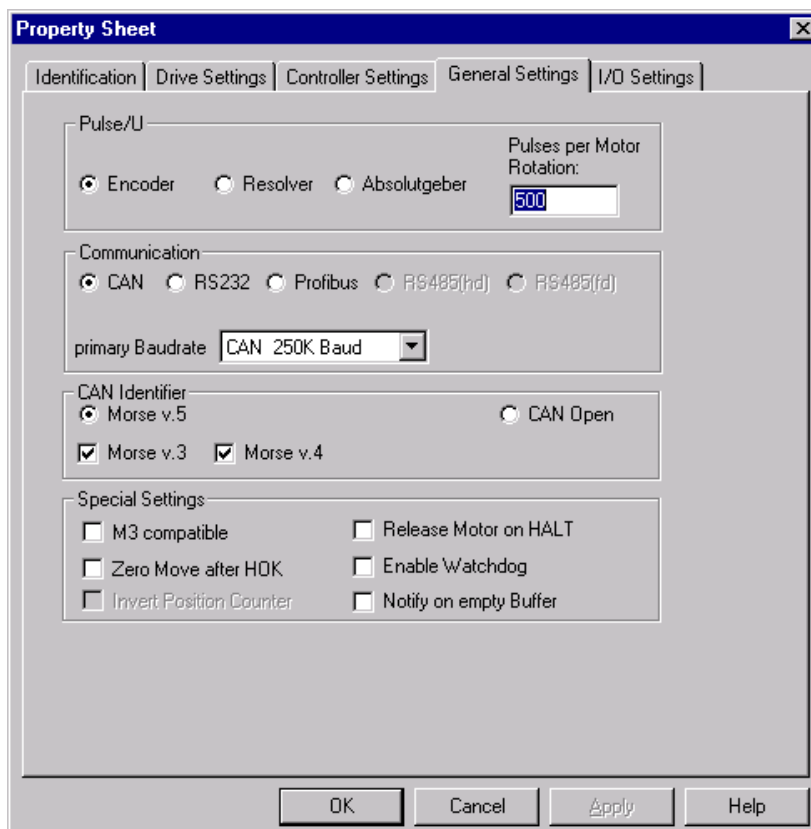
Nominal Acceleration AO

Hier wird die Beschleunigungsfähigkeit des Antriebs eingestellt. Je größer AO gesetzt wird, desto größere Beschleunigungsfähigkeit wird unterstellt.

Az

Die Änderung dieses Wertes wird nicht unterstützt.

6.4.3.4 Der Konfigurationsdialog General Settings



Feedback

Die Rückführung der Lageinformation kann über:

- Encoder Feedback
- Resolver Feedback
- Absolute Feedback

realisiert werden. Auch diese Werte haben direkten Einfluß auf die Hardware der PowerCube™-Module. Sie sollten die Werte nicht ändern, wenn nicht auch die Hardware modifiziert worden ist.

Pulses Per Motor Rotation

Dieser Parameter beschreibt die Anzahl der Impulse je Umdrehung der Encoder-Scheibe im Modul. Bei der Änderung dieses Parameters sollten Sie Vorsicht walten lassen, da er mit der Hardware im PowerCube™-Modul im Zusammenhang steht.

Communication

Die Kommunikation kann über folgende Schnittstellen erfolgen:

- CAN
- RS232
- Profibus (nur Modulbetriebsversion 350X)

Das Modul muß hardwareseitig für die unterschiedlichen Schnittstellen vorbereitet sein. Die werksseitig vorgenommene Voreinstellung sollte daher nicht verändert werden, ohne daß auch entsprechende Veränderungen an der Hardware vorgenommen wurden, da sonst die Kommunikation mit dem Modul nicht mehr möglich ist! Die Schnittstelle RS485 (Halbduplex sowie Vollduplex) wird zur

Zeit nicht unterstützt.

Primary Baudrate

In diesem Popup-Menü können je nach Auswahl der Kommunikationsschnittstelle die Übertragungsfrequenz der CAN- bzw. der RS232-Kommunikation eingestellt werden. Für die Profibuschnittstelle sind keine Einstellungen erforderlich.

CAN Identifier

Diese Optionen sind nur unter der Modulbetriebsversion 350X zugänglich. Hier erfolgt eine Auswahl der CAN-Identifizier, auf die das Modul antworten soll. Bei Auswahl des Morse v.5-Modus kann darüber hinaus festgelegt werden, ob das Modul Morse v.3 und/oder Morse v.4 kompatible Befehle verstehen soll.

M3 Compatible

Das Modul wird in den vollständigen Morse v.3-Kompatibilitätsmodus geschaltet. Im Gegensatz zur Einstellung Morse v.3 bei den CAN-Identifiern versteht das Modul nicht nur Morse v.3-Befehle, sondern verhält sich bei Initialisierung mit einem entsprechenden Initstring vollständig wie ein Morse v.3-Modul. Dies bedeutet:

- automatisches Senden des Modulstatus nach Aktivierung durch CANID-Modulrequest.
- Verweigerung von Bewegungsbefehlen, solange keine Referenzpunktfahrt erfolgt ist.

Zero Move after HOK

Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Das Modul wird direkt nach der Referenzfahrt auf die benutzerdefinierte Nullposition bewegt (siehe [Homeoffset](#)).

Invert Position Counter

Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Die Vorzeichen der min./max. Position können vertauscht werden. Wird zur Zeit nicht unterstützt.

Release Motor on HALT

Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Wird per Nothalt oder in Kombination mit einem Fehlerstatus das HALT-Flag gesetzt, so wird die Motorregelung abgeschaltet.

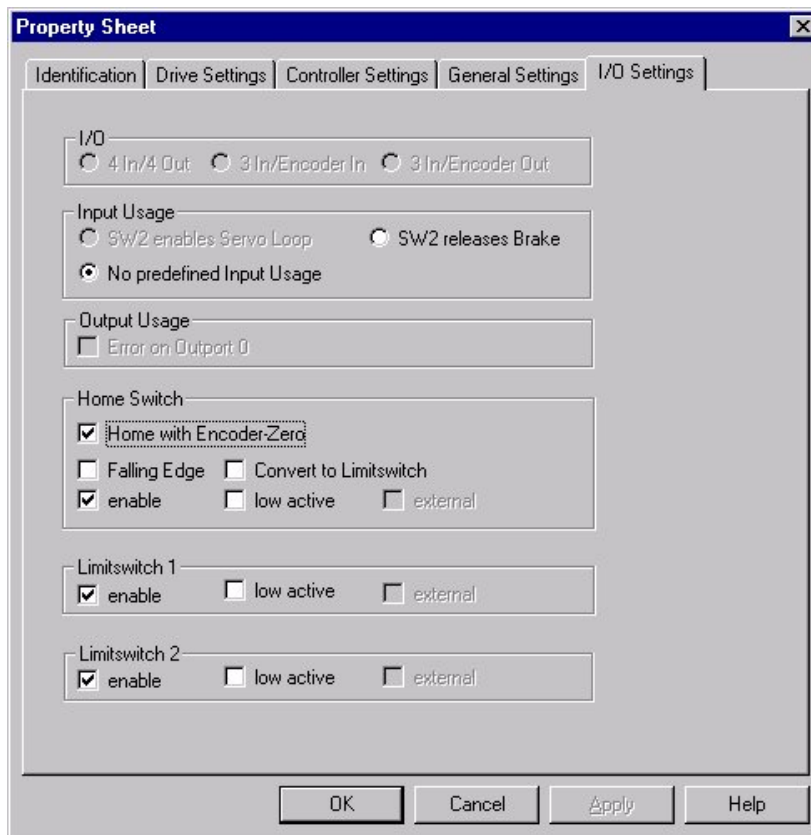
Enable Watchdog

Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Die im Modul implementierte Watchdog-Funktion wird freigeschaltet. Vom Anwendungsprogramm kann zugleich ein „Lebenszeichen“ unter Nutzung eines CANID_CMDALL gesendet werden. Wurde das „Lebenszeichen“ einmal gesendet, so muß es spätestens nach 100 ms wiederholt werden, anderenfalls löst das Modul selbständig einen Nothalt aus. Damit wird vom Modul überprüft, ob sich der Steuerrechner noch im normalen Arbeitsmodus befindet oder eine Fehlfunktion hat. In letzterem Fall schaltet sich das Modul ab.

Notify on empty Buffer

Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Eine Benachrichtigung wird geschickt, wenn im Step-Bewegungsmodus der Befehlspeicher leer ist.

6.4.3.5 Der Konfigurationsdialog I/O Settings



I/O

Diese Einstellungen sind nur unter der Modulbetriebsversion 350X zugänglich. Hier wird die Bedeutung der Input/Output-Signale, die am 15-poligen I/O-Interface (siehe [Pinning des zusätzlichen 15-poligen I/O-Interfaces](#)) anliegen, festgelegt. Bitte bedenken Sie, daß die Veränderung dieser Einstellung mit H/W-Einstellungen im Modul zusammenläuft, die auch die Bedeutung der Pins beeinflussen. Einer Änderung der I/O-Settings muß eine Umschaltung der H/W vorausgegangen sein.

Input Usage

Diese Einstellung bestimmt, welches Modulverhalten ein Eingangssignal auf dem für den externen Endschalter 2 bestimmten Datenkanal (siehe [Pinning des zusätzlichen 15-poligen I/O-Interfaces](#)) auslöst.

- **SW2 enables Servo Loop:** Diese Option ist nur unter der Modulbetriebsversion 350X zugänglich. Ein High auf der SW2-Datenleitung aktiviert die Motorregelung. Ein Low deaktiviert die Motorregelung.
- **SW2 releases Brake:** Ein High auf der SW2-Datenleitung löst die Bremse. Ein Low verriegelt die Bremse.
- **No predefined Input Usage:** Ein Eingangssignal auf der SW2-Datenleitung löst kein vordefiniertes Modulverhalten aus.

Diese Einstellungen können nur getroffen werden, wenn kein externer Endlagenschalter 2 an das Modul angeschlossen ist! Daher sind die betreffenden Optionsfelder bei aktiviertem externen Limitswitch 2 (s.u.) nicht zugänglich.

Output Usage: Error on Output 0

Bei Aktivierung dieser Option wird im Fehlerfall vom Modul ein Signal über den Datenausgangskanal 0 gesendet (siehe [Pinning des zusätzlichen 15-poligen I/O-Interfaces](#)).

Home Switch:

Der Referenzschalter wird vom Betriebssystem der PowerCube™-Module verwendet, um während einer Homing-Prozedur eine absolute Nullage zu finden.

Home with Encoder-Zero

Um die Initialisierungsprozedur mit dem Referenzschalter zu triggern und sie mit dem "Zero-Impuls" des Encoders zu beenden, müssen Sie diesen Punkt markieren. Anderenfalls wird die Schaltfläche des Referenzschalters allein für die Referenzfahrt genutzt.

Falling Edge

Der Schalter arbeitet auf der fallenden Flanke, bei Nichtmarkierung auf der steigenden Flanke.

Convert to Limit Switch

Nach Durchführung einer Homing-Prozedur ändert sich das Verhalten des Referenzschalters in das eines Endlagenschalters.

enable

Aktivierung der Referenzschalter. Wenn dieser Punkt nicht aktiviert ist, werden die übrigen Einstellungen der Referenzschalter ignoriert.

low active

Der Schalter ist low-aktiv, bei Nichtmarkierung high-aktiv.

external

Es wird der externe Referenzschalter-Eingang ausgewertet.

Limitswitch 1 und Limitswitch 2**enable**

Schalter aktiv.

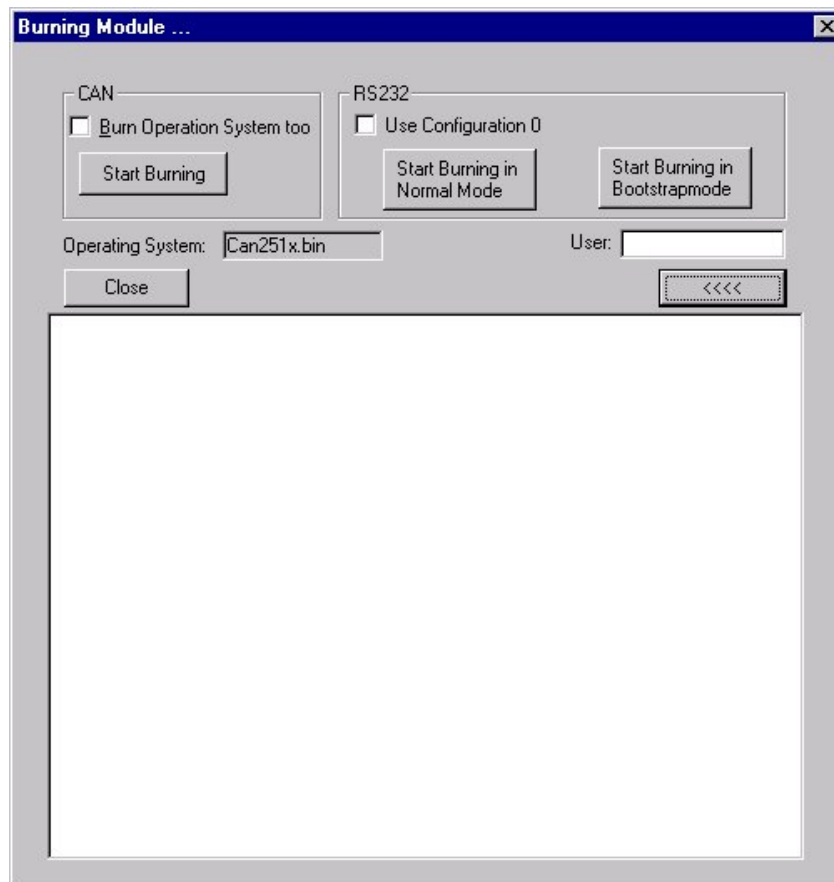
Low active

Der Schalter ist low-aktiv, im anderen Fall high-aktiv.

external

Es werden die externen Endschaltereingänge ausgewertet (siehe [Pinning des zusätzlichen 15-poligen I/O-Interfaces](#)). Im Falle des Limitswitch 2 (SW2) wird mit Setzen dieser Option die Sektion "Input Usage" deaktiviert (s.o.).

6.4.4 Der Dialog "Burning Module"



In diesem Dialog-Fenster werden schließlich die ausgewählten Einstellungen durch Flash-Download der Daten in ein EPROM im Modul fixiert. Es gibt zwei Schnittstellen, über die Daten in das Modul gebrannt werden können:

1. CAN.

Der Button „Start Burning“ wird für den Fall verwendet, daß die PowerCube™-Module per CAN-Interface angeschlossen sind. Sie haben nun zwei Möglichkeiten. Der erste Fall ist der Normalfall, indem nur die Parameter in das Modul gebrannt werden. Treffen Sie hingegen die Auswahl "Burn Operation System too" und anschließend "Start Burning", werden nicht nur die Parameter in das Modul gebrannt, sondern auch das Betriebssystem. Die Auswahl des Betriebssystems findet automatisch durch die Selektion der Version im Menü "Hardware" sowie der Kommunikationsschnittstelle im Konfigurationsdialog "General Settings" statt. Ein Upgrading vom Betriebssystem 250X auf 251X kann durch Selektieren von „Version 251X“ im Menü „Hardware“ nach dem Scannen des betreffenden Moduls erreicht werden. Bitte beachten Sie, daß dabei Controller-Parameter neu gesetzt werden müssen. Ein umgekehrtes Downgrading wird nicht empfohlen! Sie finden die notwendigen Betriebssystemdateien im Verzeichnis „OS Directory“ von PowerConfig. Im Falle der CAN-Kommunikation sind das "can250x.bin", "can251x.bin" sowie "can350x.bin"; für die RS232-Schnittstelle existieren "Rs250x.bin", "Rs251x.bin" und "Rs350x.bin". Die Betriebssystemversionen für die Profibuschnittstelle sind "Pbus250x.bin", "Pbus251x.bin" sowie "Pbus350x.bin". In der PowerConfig.ini Datei sollte der Pfad, in dem die Betriebssystemdateien liegen, unter dem Punkt „OS Directory“ in der Sektion [Software] eingetragen oder die Dateien in das Arbeitsverzeichnis von PowerConfig kopiert werden, da das Programm diese Dateien sonst nicht findet.

2. RS232.

Der Button "Start Burning in Bootstrapmode" in diesem Bereich wird verwendet, um das Brennen über ein spezielles Bootstrap-Kabel (gesondert bei AMTEC zu ordern) zu ermöglichen. Dabei wird das Betriebssystem des Moduls immer neu gebrannt. Wird die Option "Use Configuration 0" selektiert, kann zusätzlich ein geladener Parametersatz in das Modul gebrannt werden. Diese Möglichkeit ist z.B. dann hilfreich, wenn keine Kommunikation im normalen CAN-Betrieb mehr möglich ist. Hierzu müssen Sie das Bootstrap-Kabel mit einem speziellen Stecker im Modul verbinden, der nach Abschrauben der 15-poligen I/O-Buchse zugänglich wird. Bitte wenden Sie sich an Ihren Servicepartner, der Ihnen erklärt, wie Sie diese Option nutzen können.

Die Benutzung des Buttons "Start Burning in Normal Mode", der das Brennen über ein normales RS232-Kabel ermöglichen soll, wird zur Zeit noch nicht unterstützt.

Bitte berücksichtigen Sie, daß sich höchstens zwei konfigurierbare Bewegungsachsen an der Datenleitung befinden dürfen, damit der Brennvorgang in Gang gesetzt werden kann (PowerCube™ Handgelenksmodule (PW) sowie Lift'n Turn-Module (PLT)). Anderenfalls wird eine Fehlermeldung ausgegeben, ohne das eine Brennvorgang durchgeführt wurde.

Das Feld "Operating System" zeigt die Betriebssystemversion an, die bei Bedarf in das Modul-EPROM gebrannt wird. Es kann nicht editiert werden.

Das Feld "User" muß vor Beginn des Brennvorgangs mit mindestens einem Zeichen gefüllt sein. Der Mitarbeitername bzw. das -kürzel werden beim Ausdrucken und Speichern der Konfigurationsdatei als ASCII-Datei mit ausgegeben.

Wenn Sie den Brennprozeß starten, werden Sie um eine Bestätigung gebeten. Sollte Ihre Bestätigung positiv ausfallen, wird sofort der Brennvorgang eingeleitet. Am Ende dieses Vorganges erhalten Sie eine Nachricht über Erfolg oder Mißerfolg. Im Falle eines Mißerfolges können Sie versuchen den Vorgang erneut zu starten, was aber im Normalfall nicht gelingt, da das Modul zu keiner Kommunikation mehr fähig ist. Schalten Sie die Versorgungsspannung einmal aus und wieder ein und versuchen Sie es erneut.

Wenn Ihr Brennvorgang erfolgreich war, müssen Sie die Versorgungsspannung einmal aus und wieder einschalten, damit das Modul die neuen Vorgabewerte in das RAM lädt.

Beim Schließen des Dialogs via Close-Button bleiben alle geänderten Parameter dem selektierten Modul zugeordnet, sind aber nicht physikalisch ins Modul gebrannt. Beim Schließen des Programms sind dann alle Änderungen verworfen.

6.5 Fehlerbehandlung

ACHTUNG: Sie sollten PowerConfig nur auf Rechnern mit CPU's ab Pentium 60 zum Brennen von Modulen benutzen. Die Benutzung von Rechnern mit 486er-CPU zum Konfigurieren von Modulen zieht mit hoher Wahrscheinlichkeit Brennfehler nach sich, die ein erneutes Brennen im RS232-Bootstrapmode nötig machen. Für Schäden, die aufgrund falscher Rechnerkonfiguration beim Brennen am Modul entstehen, übernimmt AMTEC keine Haftung.

Bitte unbedingt beachten: Beim Brennen mit PowerConfig muß in jedem Fall der korrekte Interrupt angegeben werden.

- ISA-CAN-Karten der Firma IXXAT: Der 3. Parameter des Initstring enthält den korrekten Interrupt. Beispiel: VCI:1,d000,**5**
- PCI-CAN-Karten der Firma IXXAT: Der 3. Parameter des Initstring muß eine Zahl größer Null enthalten. Dann wird der vom Betriebssystem für diese Karte reservierte Interrupt verwendet. Beispiel: VCI:6,0,**1**
- CANdy-Karte: Unter Win95/98 Anschluß der Karte an den LPT-Port. Im Initialisierungsstring muß der IRQ des LPT-Ports angegeben werden, da der Flashdownload sonst fehlschlägt! Unter WinNT/2000 gibt es derzeit keine Möglichkeit, über die CANdy-Karte Module zu brennen.

Folgende bekannte Schwierigkeiten können bei der Installation bzw. der Benutzung des Programms auftreten:

- Unter Windows NT wird die Installation mit der Meldung „Installation incomplete“ abgebrochen. Lösung: Bitte führen Sie die Installation als Administrator durch!
- Unter Windows 9x stürzt das Programm nach Eingabe des Initstrings und Scannen nach Modulen mit einer Schutzverletzung der Kernel32.dll ab. Lösung: Bitte geben Sie in Ihrem Initstring immer die Option nonet an (siehe: ["Der Aufbau des Basis-Initialisierungsstrings"](#)).
- Die Seite „Controller Settings“ kann nicht angezeigt werden. Lösung: Sie müssen den Internetexplorer ab Version 4.01 installieren => Update des Betriebssystems.
- Beim Versuch, ein Modul zu brennen, kommt in der Statusanzeige des Brenndialogs die Fehlermeldung: „Error sending Flash Mode Command! [207]“
Lösung: Im Initstring muß explizit der Interrupt der Schnittstelle angegeben werden (bei PCI-Karten: eine Zahl ungleich Null; zeigt an, daß Interrupt benutzt werden soll). Das Modul hat dabei keinen Schaden genommen; nach Aus- und Einschalten des Stroms ist es wieder normal ansprechbar.
- Beim Versuch, ein Modul zu brennen, kommen in der Statusanzeige des Brenndialogs die Fehlermeldungen: „Cannot erase bank ...! But I try to burn it...“
und „CheckClear failed“
Lösung: Im Initstring muß explizit der Interrupt der Schnittstelle angegeben werden (bei PCI-Karten: eine Zahl ungleich Null; zeigt an, daß Interrupt benutzt werden soll). Diese Meldung kann auch erscheinen, wenn das Programm auf einem Rechner mit 486-CPU ausgeführt wurde.
ACHTUNG: Beim Auftreten dieser Fehlermeldungen sind Speicherbänke des Moduls überschrieben worden. Es ist nun nicht mehr ansprechbar und muß über den RS232-Bootstrap-Mode mit einer neuen Ur-Konfiguration versehen werden. Bitte konsultieren Sie unbedingt den AMTEC-Kundenservice!

7. PowerCube™ Betriebssystem

Das Betriebssystem jedes PowerCube™-Antriebs erfüllt folgende Aufgaben:

- Echtzeitsteuerung und -regelung des Motors auf Grundlage der Lageerfassung.
- Überwachung der Stromaufnahme des Motors und der Temperatur der Motorwicklung und Endstufentransistoren sowie der Endlagen des Systems.
- Ausführung der Bewegungsfunktionen nach den Kommandos der übergeordneten Steuerung.
- Steuerung der Bremse (sofern eine Bremse im Antrieb integriert ist).

In den folgenden Abschnitten sind die Funktionen des Betriebssystems näher erläutert. Dazu gehören Informationen zum Datenaustausch zwischen Master und Slave sowie über Diagnostizierung und Behebung von Fehlermeldungen des PowerCube™.

7.1 Kommunikationsschnittstellen

Die PowerCube™-Antriebsmodule der amtec GmbH werden mittels einer seriellen Kommunikationsschnittstelle angesteuert, über die alle Kommandos zur Bewegungssteuerung, Parametrierung und Überwachung zwischen den Modulen und der übergeordneten Steuerung ausgetauscht werden.

Die Kommunikationsschnittstelle ist gegenwärtig in Form eines CAN-Feldbusprotokolls (nach ISO/DIS 11898), RS232, Profibus sowie auf RS-485 realisiert.

7.2 PowerCube™ Kommunikationsprotokoll

Für die Übertragung von Daten zwischen Master und PowerCube™ wird unabhängig von der verwendeten Busschnittstelle ein einheitliches Datenprotokoll eingesetzt. Die in serieller Weise übertragenen Daten werden vom Antrieb empfangen, interpretiert und beantwortet. Die Dauer für diesen Prozeß ist von der eigentlichen Busschnittstelle und den gewählten Übertragungsparametern abhängig.

Ein PowerCube™-Kommando hat folgenden grundsätzlichen Aufbau:



Die übertragenen Command ID's, Parameter ID's und Folgebytes sind für alle für Kommunikationsarten (CAN, RS232, Profi, RS485) der PowerCube™-Module identisch. Die Identifizier variieren entsprechend der Vorgaben des eingesetzten Busprotokolls.

7.2.1 PowerCube™-Identifizier für den CAN-Bus

Die im PowerCube™ verwendeten CAN-Identifizier entsprechen der CAN-Spezifikation 2.0 Part A mit 11 Identifizier-Bits

Bezeichnung	0 LSB	1	2	3	4	5	6	7	8	9	10	Telegramm- Länge [Byte]	Hexadezi- malwert
CANID_CMDACK	m	m	m	m	m	1	0	1	X	X	X	1/8	0x0a0 + Modul-ID
CANID_CMDGET	m	m	m	m	m	0	1	1	X	X	X	1	0x0c0 + Modul-ID
CANID_CMDPUT	m	m	m	m	m	1	1	1	X	X	X	8	0x0e0 + Modul-ID

m = Module-ID, X = reserviert für interne Verwendung

Ein spezieller CAN-Identifizier ermöglicht die Kommunikation mit allen angeschlossenen Modulen über eine einzige Kommandozeile.:

Name	0 LSB	1	2	3	4	5	6	7	8	9	10	Data Length [Bytes]	Hex value
CANID_CMDALL	0	0	0	0	0	0	0	0	1	0	0	1/8	0x100

Die CAN-Identifizier 0x3e9 bis 0x7e5 sind für die Anwendung von bis zu 51 I/O-Modulen der Firma Thomas Wünsche reserviert.

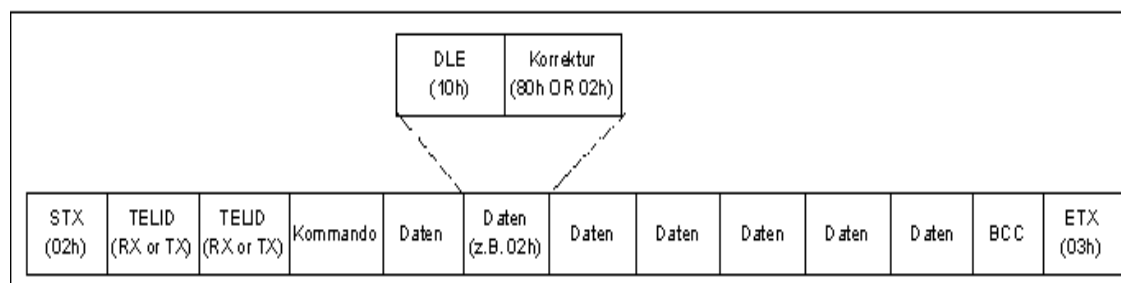
7.2.2 PowerCube™-Identifizier für den Profibus

Die Adressierung der Teilnehmer im Profibus-Protokoll ist Bestandteil der physikalischen Schicht. Es ist daher kein Identifizier für Profibus-Anwender erforderlich.

7.2.3 PowerCube™-Identifizier für die RS-Kommunikation (RS232, RS485)

Der Datenverkehr via RS-Kommunikation wird nicht wie bei den industriellen Feldbussystemen (CAN, Profibus) durch eine Hardware-Schicht unterstützt, die eine fehlerfreie Datenübertragung garantiert. Aus diesem Grunde hat AMTEC ein eigenes Datenprotokoll zur sicheren Übertragung von Daten über die RS-Schnittstellen entworfen.

Die Telegrammstruktur unter RS232 (RS485) hat folgendes Aussehen:



Ein Frame wird also immer von dem Sonderzeichen STX (02h) eingeleitet und mit dem Sonderzeichen ETX (03h) abgeschlossen. Kommen innerhalb des Telegrammes diese Sonderzeichen vor, so werden

Sie durch die Zeichenkombination DLE (10h) und (80h OR Zeichen) ersetzt. Diese DLE-Korrektur gilt für den gesamten Datenbereich. Die Telegrammlänge kann also maximal 24 Byte betragen.

Als BCC wird der Block Check Character bezeichnet, eine Prüfsumme berechnet aus der Summe über alle Nettodaten (ohne Berücksichtigung der DLE-Korrektur). Die Prüfsicherheit beträgt mit dieser Methode ca. 95%.

$BCC = (TELID + Command + Data);$

$BCC = BCC + (BCC \gg 8);$

Der 16-Bit Identifier TELID unterscheidet Befehle an das Modul (TELID_SENDDAT) und Antworten vom Modul (TELID_RECVDAT). Sie haben folgenden Aufbau:

Bit-Zuordnung	1. Byte								2. Byte							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
TELID_SENDDAT	0	0	0	0	0	1	m	m	m	m	m	0	n	n	n	n
TELID_RECVDAT	0	0	0	0	1	0	m	m	m	m	m	0	n	n	n	n

m: Modul ID 0 ... 31 n:)Anzahl Folgebytes (netto)

Eine Nachricht wird dann vom Modul interpretiert, wenn

- das SendelD in TELID steckt,
- die Moduladresse stimmt und
- die Anzahl der empfangenen Netto-Daten mit der kodierten Länge übereinstimmt.

Die Prüfsumme wird aus Kompatibilitätsgründen derzeit nicht ausgewertet.

7.2.4 Übersicht über die PowerCube™-Command ID's

Im folgenden erhalten Sie einen Überblick darüber, in welcher Reihenfolge die notwendigen Daten an den PowerCube™ zu senden sind. Anhand der übertragenen Daten wird zunächst festgelegt, welches Kommando abgearbeitet werden soll. Dabei sind folgende Kommandos verfügbar:

Kommando	CommandID (Dec/Hex)	Bedeutung	Länge
Reset	0/0x00	Fehlerstatus löschen	1 Byte
Home	1/0x01	Referenzfahrt auslösen	1 Byte
Halt	2/0x02	Nothalt auslösen	1 Byte
RecalcPIDParam	9/0x09	Neuberechnen der Regelschleife	1 Byte
SetExtended	8/0x08	Parameter schreiben	3 bis 6 Byte
GetExtended	10/0x0a	Parameter lesen	2 Byte
SetMotion	11/0x0b	Bewegung ausführen	6 bis 8 Byte
SetIStep	13/0x0d	Step-Bewegungskommando	7 Byte

Die gesendeten Daten beantwortet das Modul, sofern dies im [Konfigurationswort Config](#) festgelegt ist, mit einem Acknowledge, so daß der Datenfluß der Informationsbytes wie nachfolgend beschrieben zusammengefasst werden kann (siehe auch Kap. 7.2.6 [Aufbau der Datenströme](#)):

Kommando	Send	Acknowledge
Reset	[0x00]	[0x00]
Home	[0x01]	[0x01]
Halt	[0x02]	[0x02]
RecalcPIDParam	[0x09]	[0x09]
SetExtended	[0x08] [param] [data] ...	[0x08] [param] [0x64]
GetExtended	[0x0a] [param]	[0x0a] [param] [data] ...
SetMotion	[0x0b] [mode] [data] ...	[0x0b] [mode] [0x64]
SetStep	[0x0d] [data] ...	[0x0d]

Für die Kommandos „Parameter lesen“ (GetExtended) und „Parameter schreiben“ (SetExtended) ist im nachfolgenden Datenbyte festgelegt, auf welchen Parameter sich die zu übertragenden Daten beziehen. Die Anzahl der sich daran anschließenden Datenbytes ist abhängig vom Datentyp des betreffenden Parameters. Die Liste der Parameter ist sehr umfangreich und wird im folgenden näher erläutert.

Das Kommando „Bewegung ausführen“ bedingt als Folgewert den Bewegungsmodus. Die Anzahl der darauf folgenden Datenbytes richtet sich nach dem gewählten Bewegungsmodus. Eine Tabelle im Abschnitt 7.2.8 zeigt die verschiedenen Bewegungsarten und die zugehörigen Parameter.

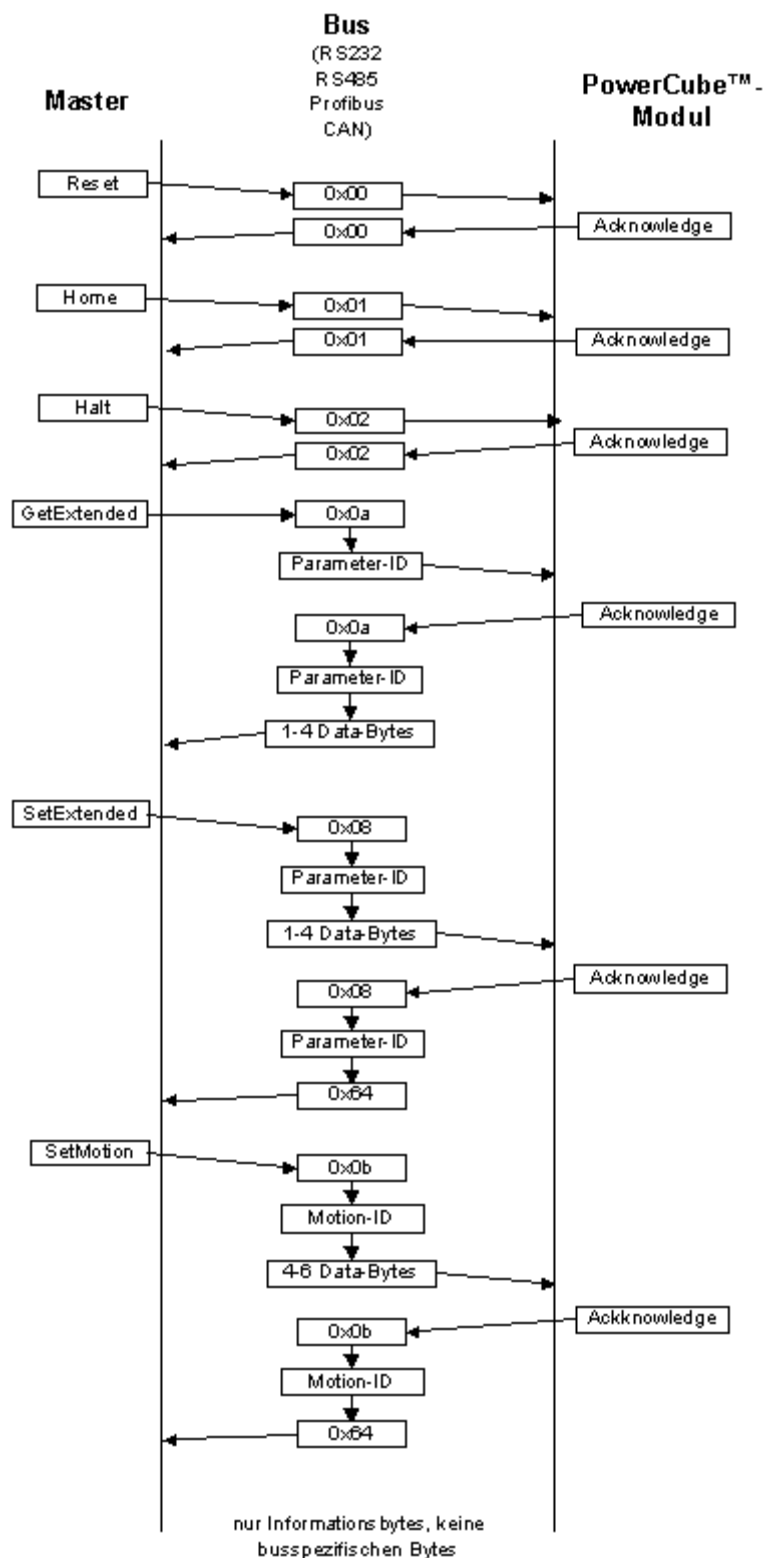
7.2.5 Datentypen

Für die Realisierung des Datenprotokolls ist die Kenntnis der im PowerCube™ verwendeten Datentypen erforderlich:

Datentyp	Bedeutung	Datenbreite	Wertebereich dezimal
Int8	Byte	8 Bit	- 128 bis 127
UInt8	Byte, vorzeichenlos	8 Bit	0 bis 255
Int16	Wort	16 Bit	-32768 bis 32767
UInt16	Wort, vorzeichenlos	16 Bit	0 bis 65535
Int32	Doppelwort	32 Bit	-2147483848 bis 2147483847
UInt32	Doppelwort, vorzeichenlos	32 Bit	0 bis 4294967295
float	Gleitkommawert	32 Bit	3.4 e-038 bis 3.4 e+038

7.2.6 Aufbau der Datenströme

Der Aufbau der Datenströme (Informationsbytes) zur Steuerung des PowerCube™ ist für alle Kommunikationsarten prinzipiell identisch. Im folgenden wird gezeigt, wie die einzelnen Kommandos aufgebaut werden. Das Diagramm zeigt nur Informationsbytes und keine busspezifischen Bytes.



7.2.7 Übersicht über die PowerCube™-Parameter ID's

Die folgende Tabelle enthält die Informationen über modulinterne Parameter, die über die Kommunikation gelesen und geschrieben werden können.

Parameter	ParameterID (Dec/Hex)	Bedeutung	Datentyp	Read	Write
DefHomeOffset	0/0x00	Offset zur Referenzlage (Defaultwert). Mit Hilfe dieses Wertes kann ein Offset zur physikalischen Nulllage (Referenzpunkt) des Antriebs eingestellt werden.	float	x	
DefGearRatio	1/0x01	Getriebeübersetzung (Defaultwert)	float	x	
DefLinRatio	2/0x02	Faktor zur Umrechnung der Dreh- in Linearbewegung (Defaultwert)	float	x	
DefMinPos	3/0x03	Minimalposition des Antriebs (Defaultwert)	float	x	
DefMaxPos	4/0x04	Maximalposition des Antriebs (Defaultwert)	float	x	
DefMaxDeltaPos	5/0x05	Maximal zulässige Regelabweichung des Lagereglers: "Schleppabstand" (Defaultwert)	float	x	
DefMaxDeltaVel	6/0x06	Maximal zulässige Regelabweichung des Geschwindigkeitsreglers (Defaultwert)	float	x	
DefTorqueRatio	7/0x07	Faktor zur Umrechnung zwischen Stromaufnahme und ausgeübtem Moment (Defaultwert)	float	x	
DefCurRatio	8/0x08	Faktor zur Umrechnung der Rohdaten des Stromreglers in Ampere (Defaultwert)	float	x	
DefMinVel	9/0x09	Minimalgeschwindigkeit des Antriebs (Defaultwert)	float	x	
DefMaxVel	10/0x0a	Maximalgeschwindigkeit des Antriebs (Defaultwert)	float	x	
DefMinAcc	11/0x0b	Minimalbeschleunigung des Antriebs (Defaultwert)	float	x	
DefMaxAcc	12/0x0c	Maximalbeschleunigung des Antriebs (Defaultwert)	float	x	
DefMinCur	13/0x0d	Minimaler Sollwert für den Stromregler (Defaultwert)	float	x	
DefMaxCur	14/0x0e	Maximaler Sollwert für den Stromregler (Defaultwert)	float	x	
DefHomeVel	15/0x0f	Geschwindigkeit für die Referenzpunktfahrt. Dieser Wert ist vorzeichenbehaftet und legt auf diese Weise die Richtung fest. (Defaultwert)	float	x	
DefHomeAcc	16/0x10	Beschleunigung für die Referenzpunktfahrt (Defaultwert)	float	x	
DefCubeSerial	26/0x1a	Seriennummer des PowerCube™-Antriebs (Defaultwert)	UInt32	x	
DefConfig	27/0x1b	Konfigurationsdaten des Antriebs (Defaultwert)	UInt32	x	

DefPulsesPerTurn	28/0x1c	Anzahl der ausgewerteten Geberimpulse pro Motorumdrehung (Defaultwert)	UInt32	x	
DefCubeVersion	29/0x1d	Versionsinformation der Firmware (Defaultwert)	UInt16	x	
DefServiceInterval	30/0x1e	Wartungsintervall angegeben in Betriebsstunden (Defaultwert)	UInt16	x	
DefBrakeTimeOut	31/0x1f	Verzögerungszeit für das Zuschalten der Bremse nach Stillstand des Motors angegeben in ms (Defaultwert)	UInt16	x	
DefAddress	32/0x20	Busadresse des Moduls [1...31] (Defaultwert)	UInt8	x	
DefPrimBaud	34/0x22	Kennziffer für die voreingestellte Baudrate der CAN-Kommunikation (Defaultwert)	UInt8	x	
DefScndBaud	35/0x23	Kennziffer für die voreingestellte Baudrate der RS232/RS485-Kommunikation (Defaultwert)	UInt8	x	
PosCount	36/0x24	Absoluter Zählerstand des Positionsgegers (Istwert)	Int32	x	
RefPosCount	37/0x25	Zählerstand des Positionsgegers bei Erkennung des Referenzschalters (Istwert)	Int32	x	
DioSetup	38/0x26	Register der aktuell gesetzten Bits der internen DIOs (siehe PCube_getDioData)	UInt32	x	x
CubeState	39/0x27	Statuswort des Antriebs (Istwert)	UInt32	x	
TargetPosInc	40/0x28	Sollwertvorgabe in Geberimpulsen für die Position (Zielwert)	UInt32	x	x
TargetVelInc	41/0x29	Sollwertvorgabe in Geberimpulsen/ms für die Geschwindigkeit (Zielwert)	UInt32	x	x
TargetAccInc	42/0x2a	Sollwertvorgabe in Geberimpulsen/ms ² für die Beschleunigung (Zielwert)	UInt32	x	x
StepInc	43/0x2b	Schrittweite in Geberimpulsen des Antriebs im Stepbetrieb (Istwert)	UInt32	x	
HomeOffsetInc	44/0x2c	Nullpunktoffset des Antriebs in Geberimpulsen (Istwert)	Int32	x	
RawCur	53/0x35	Sollwert des digitalen Stromreglers in Digits [-511...512] (Istwert/Vorgabe)	Int16	x	x
HomeToZeroInc	54/0x36	Anzahl der Geberimpulse zwischen Referenzschalter und Nullimpuls des Gebers (Istwert)	Int32	x	
Config	57/0x39	Konfigurationswort für den Antrieb (Vorgabe)	UInt32	x	x
MoveMode	58/0x3a	Betriebsart des Antriebs (Istwert)	UInt8	x	
IncRatio	59/0x3b	Verhältnis aus Geberimpulsen und Wegeinheit (Istwert)	float	x	
ActPos	60/0x3c	Aktuelle Position, vorberechnet in Wegeinheiten (Istwert)	float	x	

ActPos_	61/0x3d	Aktuelle Position im vergangenen Abtastzeitpunkt, vorberechnet in Weeinheiten (Istwert)	float	x	
IPolPos	62/0x3e	Aktuelle Sollposition, vorberechnet in Weeinheiten (Istwert)	float	x	
DeltaPos	63/0x3f	Differenz zwischen Soll- und Istposition, vorberechnet in Weeinheiten = Regelabweichung (Istwert)	float	x	
MaxDeltaPos	64/0x40	Maximal zulässige Differenz zwischen Soll- und Istposition, vorberechnet in Weeinheiten = max. zulässige Regelabweichung (Vorgabe)	float	x	x
ActVel	65/0x41	Aktuelle Geschwindigkeit des Antriebs, vorberechnet in Weeinheiten/s (Istwert)	float	x	
IPoVel	66/0x42	Aktuelle Sollgeschwindigkeit, vorberechnet in Weeinheiten/s (Istwert)	float	x	
MinPos	69/0x45	Endlage des Antriebs in Weeinheiten = Minimalposition (Vorgabe)	float	x	x
MaxPos	70/0x46	Endlage des Antriebs in Weeinheiten = Maximalposition (Vorgabe)	float	x	x
MaxVel	72/0x48	Maximal zulässige Geschwindigkeit des Antriebs in Weeinheiten/s (Vorgabe)	float	x	x
MaxAcc	74/0x4a	Maximal zulässige Beschleunigung des Antriebs in Weeinheiten/s ² ; (Vorgabe)	float	x	x
MaxCur	76/0x4c	Maximal zulässiger Stromsollwert in Ampere für den digitalen Stromregler (Vorgabe)	float	x	x
Cur	77/0x4d	Aktueller Stromsollwert in Ampere des digitalen Stromreglers (Vorgabe)	float	x	x
TargetPos	78/0x4e	Sollwert für die Positionsvorgabe in Weeinheiten (Zielwert)	float		x
TargetVel	79/0x4f	Sollwert für die Geschwindigkeitsvorgabe in Weeinheiten/s (Zielwert)	float		x
TargetAcc	80/0x50	Sollwert für die Beschleunigungsvorgabe in Weeinheiten/s ² ; (Zielwert)	float		x
DefC0	81/0x51	Verstärkungsfaktor C0 (Defaultwert)	Int 16	x	
DefDamp	82/0x52	Dämpfungsfaktor Damp (Defaultwert)	Int 16	x	
DefA0	83/0x53	Nominalbeschleunigung des Motors (Defaultwert)	Int 16		
ActC0	84/0x54	Verstärkungsfaktor C0 (Vorgabe)	Int 16	x	x
ActDamp	85/0x85	Dämpfungsfaktor Damp (Vorgabe)	Int 16	x	x

ActAO	86/0x86	Nominalbeschleunigung des Motors (Vorgabe)	Int16	x	x
DefBurnCount	87/0x87	Anzahl der bisher am Modul durchgeführten Brennvorgänge (Defaultwert)	UInt8	x	
Setup	88/0x88	Neu ab Version 3.5.00: 32-Bit-Wort, in dem u.a. Feedback-, DIO- und Kommunikationssetup hinterlegt sind (Default)	UInt32	x	
HomeOffset	89/0x89	Offset zur Referenzlage (Vorgabe)	float	x	x

Defaultwert: Diese Daten sind im ROM (Read Only Memory) des PowerCube™ gespeichert und werden nach dem Zuschalten der Betriebsspannung als Vorgabewerte aktiv.

Vorgabe: Diese Daten können für die Dauer des Betriebs vom Anwender verändert werden. Nach dem Abschalten der Betriebsspannung gehen die Einstellungen verloren und müssen nach dem erneuten Zuschalten wiederholt werden.

Zielwert: Dies sind die kurzzeitigen Zielvorgaben für Fahrkommandos. Sie beziehen sich ausschließlich auf Zielpositionen, -geschwindigkeiten und -beschleunigungen.

Die 32-Bit Konfigurationsworte Config und Setup (neu ab Modulversion 3.5.00) enthalten u.a. Einstellungen zur Bremsen-, Kommunikations-, Feedback- und Schalterkonfiguration. Setup kann zur Laufzeit nur gelesen, nicht jedoch verändert werden, während die Konfigurationsbits von Config zur Laufzeit neu gesetzt werden können. Nachfolgend ist die Bitbelegung ab Modulversion 3.5.00 wiedergegeben.

7.2.7.1 Konfigurationsbits im 32-Bit Wort Setup (Read Only)

Bezeichner	Bit-ID (Hex)	Bedeutung
SETUP_ENCODER_FEEDBACK	0x00000001	Lageinformation über Encoder Feedback
SETUP_RESOLVER_FEEDBACK	0x00000002	Lageinformation über Resolver Feedback
SETUP_ABSOLUTE_FEEDBACK	0x00000004	Lageinformation über Absolutgeber-Feedback
SETUP_4IN_4OUT	0x00000008	Beschaltung des 15-poligen I/O-Interfaces Typ I
SETUP_3IN_ENCODER_IN	0x00000010	Beschaltung des 15-poligen I/O-Interfaces TypII
SETUP_3IN_ENCODER_OUT	0x00000020	Beschaltung des 15-poligen I/O-Interfaces TypIII
SETUP_RS232	0x00000040	Modulkommunikation über RS232-Schnittstelle
SETUP_CAN	0x00000200	Modulkommunikation über CAN-Schnittstelle
SETUP_PROFIBUS	0x00000400	Modulkommunikation über Profibus-Schnittstelle
SETUP_USE_M3ID	0x00000800	CAN-Identifizier für M3-Befehlssatz reservieren
SETUP_USE_M4ID	0x00001000	CAN-Identifizier für M4-Befehlssatz reservieren
SETUP_INVERT_MOTORDIR	0x00004000	Umkehr der Motorlaufrichtung
SETUP_USE_SW2_AS_ENABLE	0x00008000	High auf SW2-Datenleitung aktiviert Motorregelung
SETUP_USE_SW2_AS_BRAKE	0x00010000	High auf SW2-Datenleitung löst Bremse
SETUP_ERROR_TO_OUT0	0x00020000	Im Fehlerfall sendet Modul ein Signal über Datenausgangskanal 0

7.2.7.2 Konfigurationsbits im 32-Bit Wort Config

Bezeichner	Bit-ID (Hex)	Bedeutung
CONFIG_BRAKE_PRESENT	0x00000008	Nach Abschluß der Bewegung wird Bremse automatisch aktiviert
CONFIG_BRAKE_AT_POWERON	0x00000010	Nach Einschalten bleibt Bremse aktiv
CONFIG_SWR_WITH_ENCODERZERO	0x00000020	Homing-Prozedur wird mit „Zero-Impuls“ des Encoders beendet
CONFIG_SWR_AT_FALLING_EDGE	0x00000040	Referenzschalter arbeitet auf fallender Flanke
CONFIG_CHANGE_SWR_TO_LIMIT	0x00000080	Nach Homing-Prozedur wird Referenzschalter zu Endlagenschalter
CONFIG_SWR_ENABLED	0x00000100	Referenzschalter wird aktiviert
CONFIG_SWR_LOW_ACTIVE	0x00000200	Referenzschalter ist low-aktiv
CONFIG_SWR_USE_EXTERNAL	0x00000400	Der externe Referenzschalter-Eingang wird ausgewertet
CONFIG_SW1_ENABLED	0x00000800	Endlagenschalter 1 wird aktiviert
CONFIG_SW1_LOW_ACTIVE	0x00001000	Endlagenschalter 1 low-aktiv
CONFIG_SW1_USE_EXTERNAL	0x00002000	Externer Endlagenschalter-Eingang 1 wird ausgewertet
CONFIG_SW2_ENABLED	0x00004000	Endlagenschalter 2 wird aktiviert
CONFIG_SW2_LOW_ACTIVE	0x00008000	Endlagenschalter 2 low-aktiv
CONFIG_SW2_USE_EXTERNAL	0x00010000	Externer Endlagenschalter-Eingang 2 wird ausgewertet
CONFIG_LINEAR_DRIVE	0x00020000	Modultyp ist Linearmodul
CONFIG_M3_COMPATIBLE	0x00100000	Befehlssatz und Verhalten des Moduls ist kompatibel zu MORSE3
CONFIG_LINEAR_SCREW	0x00200000	Modultyp ist Linearmodul mit Spindeltrieb
CONFIG_NOTIFYEMPTYBUFFER	0x00400000	Benachrichtigung bei leerem Befehlspeicher im Step-Bewegungsmodus
CONFIG_DISABLE_ON_HALT	0x00800000	Bei Setzen des HALT-Flags wird Motorregelung abgeschaltet
CONFIG_WATCHDOG_ENABLE	0x01000000	Watchdog-Funktion im Modul wird freigeschaltet
CONFIG_ZERO_MOVE_AFTER_HOK	0x02000000	Modul wird nach Referenzfahrt auf benutzerdefinierte Nullposition bewegt
CONFIG_DISABLE_ACK	0x04000000	Keine Bestätigung vom Modul nach SET-Kommando

7.2.7.3 Bedeutung der Bits im 32-Bit Wort DioSetup

Bezeichner	Bit-ID (Hex)	Bedeutung des gesetzten Bits
INBIT0	0x00000001	Digital-Eingang 1 aktiv
INBIT1	0x00000002	Digital-Eingang 2 aktiv
INBIT2	0x00000004	Digital-Eingang 3 aktiv
INBIT3	0x00000008	Digital-Eingang 4 aktiv
OUTBIT0	0x00000010	Digital-Ausgang 1 aktiv
OUTBIT1	0x00000020	Digital-Ausgang 2 aktiv
OUTBIT2	0x00000040	Digital-Ausgang 3 aktiv
OUTBIT3	0x00000080	Digital-Ausgang 4 aktiv
INSWR (*)	0x00000100	Referenzschalter aktiv
INSW1 (*)	0x00000200	Endlagenschalter 1 aktiv
INSW2 (*)	0x00000400	Endlagenschalter 2 aktiv
alle übrigen	0x00000800	unbenutzt
	- 0x80000000	

(*) Gültig ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13. Darstellung stets in positiver Logik (high = aktiv).

7.2.8 Ausführung von Bewegungen

Parameter	ParameterID (Dec/Hex)	Bedeutung	Folgedaten	Länge
FRAMP_MODE	4/0x04	Einzelstreckenfahrt	Sollposition (float)	6 Byte
FSTEP_MODE	6/0x06	Inkrementenfahrt	Sollposition (float) und Zeit (UInt 16)	8 Byte
FVEL_MODE	7/0x07	Geschwindigkeitsregelung	Sollgeschwindigkeit (float)	6 Byte
FCUR_MODE	8/0x08	Stromregelung	Sollstrom (float)	6 Byte
IRAMP_MODE	9/0x09	Einzelstreckenfahrt im Inkrementenmodus	Sollposition (Int32) *	4 Byte
ISTEP_MODE	11/0x0b	Einzelschrittfahrt im Inkrementenmodus	Sollposition (Int32) und Zeit (UInt 16)	6 Bytes
IVEL_MODE	12/0x0c	Geschwindigkeitsregelung im Inkrementenmodus	Sollgeschwindigkeit (Int32)	4 Bytes
ICUR_MODE	13/0x0d	Stromregelung im Inkrementenmodus	Sollstrom (Int32)	4 Bytes
FRAMP_ACK (*)	14/0x0e	Einzelstreckenfahrt mit spez. Antworttelegramm (s.u.)	Sollposition (float)	6 Byte
FSTEP_ACK (*)	16/0x10	Inkrementenfahrt mit spez. Antworttelegramm (s.u.)	Sollposition (float) und Zeit (UInt 16)	8 Byte
FVEL_ACK (*)	17/0x11	Geschwindigkeitsregelung mit spez. Antworttelegramm (s.u.)	Sollgeschwindigkeit (float)	6 Byte
FCUR_ACK (*)	18/0x12	Stromregelung mit spez. Antworttelegramm (s.u.)	Sollstrom (float)	6 Byte

(*) Gültig ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13. Das Antworttelegramm (Acknowledge) auf diese Kommandos enthält die aktuelle Position des Moduls (4Byte float-Wert), einen Kurzstatus (1 Byte) und den Status der internen digitalen IOs aus dem Statuswort DioSetup (die ersten 8 Bit aus DioSetup = 1 Byte). Die Wahl dieses Bewegungsmodus ist insbesondere bei der Inkrementenfahrt (FSTEP_ACK) wegen der komprimierten Informationsrückgabe von Interesse. Beschreibung und Benutzung werden im nachfolgenden Befehlsbeispiel näher erläutert.

Maßeinheiten der PowerCube™-Module

Linearmodule:

Lage	Geschwindigkeit	Beschleunigung	Strom
m	m/s	m/s ²	A

Drehmodule:

Lage	Geschwindigkeit	Beschleunigung	Strom
rad	rad/s	rad/s ²	A

Im Inkrementmodus für Linear- und Drehmodule:

Lage	Geschwindigkeit	Beschleunigung	Strom
inc	inc/s	inc/s ²	digits (-500 --- 500)

Befehlsbeispiele

Gegeben sei:

CubeID = 17,

TargetAcc = 0.5 m/s²,

TargetVel = 0.08 m/s,

Pos = 0.75 m

Die vom Anwender zu sendenden Daten befinden sich in der Reihe „Send“, das vom Modul zurückgegebene Acknowledge befindet sich in der direkt darunter gelegenen Zeile „Ack“.

CAN

	CAN-ID*	Byte 0 ComandID	Byte 1 Param.ID	Byte 2 Param.	Byte 3 Param.	Byte 4 Param.	Byte 5 Param.	Byte 6 Param.	Byte 7 Param.
Send	0x0F1	0x01 Home	-	-	-	-	-	-	-
Ack	0x0F1	0x01	-	-	-	-	-	-	-
Send	0x0F1	0x00 Reset	-	-	-	-	-	-	-
Ack	0x0F1	0x00	-	-	-	-	-	-	-
Send	0x0F1	0x08 SetExtended	0x50 TargetAcc	0x00 Acc	0x00 Acc	0x00 Acc	0x3f Acc	-	-
Ack	0x0F1	0x08	0x50	0x64	-	-	-	-	-
Send	0x0F1	0x08 SetExtended	0x4f TargetVel	0x0a Vel	0xd7 Vel	0xa3 Vel	0x3d Vel	-	-
Ack	0x0F1	0x08	0x4f	0x64	-	-	-	-	-
Send	0x0F1	0x0b SetMotion	0x04 FRAMPMODE	0x00 Pos	0x00 Pos	0x40 Pos	0x3f Pos	-	-
Ack	0x0F1	0x0b	0x04	0x64	-	-	-	-	-

* CAN-ID = CANID_CMDPUT + CubeID = 0x0e0 + 0x011 = 0x0F1

Ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13 ist es möglich, FRAMP_ACK anstelle von FRAMP_MODE zu verwenden. In diesem Fall enthält das Acknowledge des Moduls Statusinformationen wie aktuelle Position, Kurzstatus und DIO-Status in komprimierter Form.

Die 8 Bit des Kurzstatus sind wie folgt definiert::

Bit-Zuordnung	7	6	5	4	3	2	1	0
Bedeutung	FULL	INPROG	END	MOTION	SW2	SW1	SWR	NOK *
Entspricht dem CubeState Bit	FULL- BUFFER	INPRO- GRESS	RAMP_ END	MOTION	SW2	SW1	SWR	---

* NOK (NOT_OK) setzt sich aus einer logischen ODER-Verknüpfung der Flags STATE_ERROR, ISTATE_HOK und STATE_HALTED zusammen.

Die letzten beiden Zeilen des obigen Beispiels würden dann also wie folgt aussehen::

Send	0x0f1	0x0b SetMotion	0x04 FRAMP_ACK	0x00 Pos	0x00 Pos	0x40 Pos	0x3f Pos	-	-
Ack	0x0f1	0x0b	0x0e	ActPos	ActPos	ActPos	ActPos	Short- state	Dio

Profibus

	Byte 0 ComandID	Byte 1 Param.ID	Byte 2 Param.	Byte 3 Param.	Byte 4 Param.	Byte 5 Param.	Byte 6 Param.	Byte 7 Param.
Send	0x01 Home	-	-	-	-	-	-	X
Ack	0x01	-	-	-	-	-	-	-
Send	0x00 Reset	-	-	-	-	-	-	X
Ack	0x00	-	-	-	-	-	-	-
Send	0x08 SetExtended	0x50 Target-Acc	float Acc	float Acc	float Acc	float Acc	-	X
Ack	0x08	0x50	0x64	-	-	-	-	-
Send	0x08 SetExtended	0x4f TargetVel	float Vel	float Vel	float Vel	float Vel	-	X
Ack	0x08	0x4f	0x64	-	-	-	-	-
Send	0x0b SetMotion	0x04 FRAMPMODE	float Pos	float Pos	float Pos	float Pos	-	X
Ack	0x0b	0x04	064	-	-	-	-	-

X: Bei der Übertragung von Daten über den Profibus ist darauf zu achten, daß im Byte 7 ein veränderlicher Wert, z.B. ein inkrementeller Counter, gesetzt wird, da beim Senden identischer Telegramme (beispielsweise mehrere Bewegungsrampen mit gleichen Parametern) nur das erste ausgewertet wird!

7.2.9 Kommandos an alle verbundenen Module in einer Kommandozeile senden

Mit Hilfe des schon erwähnten CAN-Identifiers CANID_CMDALL ist es möglich, in einer einzigen Kommandozeile folgende Befehle an alle verbundenen Module zu schicken. Diese Befehle werden besonders schnell verarbeitet; es gibt allerdings von Seiten der Module kein Acknowledge.

	Byte 0: ComandID	Byte 1: Param.ID	Bedeutung
CAN-ID = CANID_CMDALL (0x100)	0x00		Reset
	0x02		Halt
	0x07		Watchdog-Refresh
	0x09	0x00 ... 0x03 (*)	Change Baudrate

(*) Baudrate: 0 = 75K, 1 = 250K, 2 = 500K, 3 = 1000K.

Eine ausführliche Erläuterung der Befehle ist im Abschnitt "[Die API-Schnittstelle zur PowerCube™-Steuerung](#)" vorhanden.

7.3 Betriebsverhalten des PowerCube™

Die umfangreichen Funktionen des PowerCube™-Betriebssystems bedingen für den störungsfreien Betrieb des Antriebs einige wichtige Regeln, die unbedingt Beachtung finden sollten. Dies bezieht sich auf den Ablauf bei der Vorgabe von Bewegungssequenzen und Maßnahmen zur Überwachung des Systems. Die Auslegung der Mastersteuerung ist meist abhängig vom konkreten Anwendungsfall. Bei der Auslegung der Steuerung sollten folgende Faktoren betrachtet werden:

- Echtzeitverhalten der Anlage und zeitliche Anforderungen für den Betrieb
- Verkopplung von mehreren Antrieben in einer kinematischen Kette
- Sicherheitsanforderung beim Betrieb der Gesamtanlage.

In jedem Fall kann die Nutzung des PowerCube™ für die besonderen Anforderungen der Anwendung optimiert werden. Dazu erhalten Sie in den folgenden Abschnitten nähere Informationen.

7.3.1 Betriebsweise im Normalfall

Nach Zuschalten der Betriebsspannung ist der PowerCube™ sofort betriebsbereit. Die rückgemeldete Position entspricht dem voreingestellten Homeoffset. Der Antrieb kann mit den Bewegungskommandos verfahren werden.

Bei den Modulen handelt es sich um mechatronische Elemente, deren Bauteile fettgeschmiert sind. Die Maximaldaten der Module sollten also erst nach einer Warmlaufphase abgefordert werden.

Üblicherweise ist die erste Bewegung des Antriebs nach dem Zuschalten der Betriebsspannung die Referenzpunktfahrt (Homing). Dies ist notwendig, weil die Einstellungen des Antriebs hinsichtlich Maximal- und Minimalposition auf die Nulllage des Antriebs bezogen sind (Nulllage ist Referenzpunkt abzüglich Homeoffset). Die Begrenzung der Lagesollwerte auf die voreingestellten Daten funktioniert demzufolge nur nach erfolgreicher Referenzpunktfahrt. Sie sollten aus diesem Grund folgende Hinweise berücksichtigen:

- Vorgabe von Bewegungskommandos erst nach erfolgreicher Referenzpunktfahrt,
- Ist eine Referenzpunktfahrt nicht möglich, so sollten Sie nur langsame Bewegungen auslösen, die Sie kontrollieren und im Notfall abschalten können,
- Fahren Sie den Antrieb vor dem Abschalten der Betriebsspannung stets zurück in die Referenzlage.
- Im Statuswort des Antriebs wird durch das Bit STATE_HOME_OK signalisiert, daß eine Referenzfahrt erfolgreich war.

Die Vorgabe von Daten an den Antrieb wird vom Betriebssystem des PowerCube™ überwacht. Im einzelnen bedeutet dies:

Überwachung der Vorgaben von Zielposition, Geschwindigkeit, Beschleunigung und Strom bei Fahrkommandos. Die Vorgaben werden auf die voreingestellten Maximalwerte begrenzt. Vorgaben kleiner als die voreingestellten Minimalwerte werden ebenfalls begrenzt (siehe unter "[Lesen und Schreiben von Parametern](#)": Zielwerte).

Überwachung der Vorgaben beim Setzen von Parametern. Dies bezieht sich auf alle Werte, die über die Kommunikation für die Dauer des Betriebs verändert werden können (siehe unter "[Lesen und Schreiben von Parametern](#)": Vorgabe).

7.3.2 Die Benutzung der Modul-Watchdogfunktion

- Für jedes angeschlossene Modul muß sichergestellt sein, daß im Konfigurationswort **Config** der Watchdog verfügbar gemacht worden ist (Flag `CONFIG_WATCHDOG_ENABLE`).
- An jedes angeschlossene Modul muß die Kommandosequenz `CANID_CMDPUT (0x0e0 + Modul-ID) + CommandID 0x07` gesendet werden, um den modulinternen Watchdog online zu schalten. Ein erneutes Senden dieser Sequenz schaltet den Watchdog zurück in den offline-Zustand.
- In einem Intervall kleiner 50 ms muß die Kommandosequenz `CANID_CMDALL (0x100) + CommandID 0x07` gesendet werden, die bei allen angeschlossenen Modulen das Watchdog-Refresh auslöst.
- Falls das Watchdog-Refresh nicht zyklisch innerhalb dieser Zeitspanne ausgelöst wird, werden alle Module in einen Haltezustand versetzt, wobei sie die Statusflags `STATE_COMM_ERROR` und `STATE_HALTED` anzeigen.
- Um die Module wieder zu aktivieren, kann man nun entweder den Watchdog jedes einzelnen Moduls offline schalten und die Module anschließend mit einem RESET zurücksetzen, oder es wird erneut zyklisch ein Watchdog-Refresh und mittels der Kommandosequenz `CANID_CMDALL (0x100) + CommandID 0x00` ein RESET gleichzeitig an alle Module gesendet.

7.3.3 Der Modulstatus: Das Statuswort CubeState

Der Zustand des Antriebs kann jederzeit mit Hilfe des Statusworts "CubeState" ermittelt werden. Anhand des Modulstatus erkennt der Anwender den augenblicklichen Ausführungszustand von Befehlen, kann Fehlermeldungen und Ursachen für Fehler ermitteln. Es ist in jedem Fall ratsam, das Statuswort in regelmäßigen Abständen zu pollen. Die Häufigkeit wird hierbei von der Anwendung bestimmt. Das Statuswort ist ein vorzeichenloser 32-Bit Integer-Wert, dessen Information bitweise organisiert ist. Jedem Bit ist eine Statusinformation zugeordnet. Im Modulstatus können mehrere Bits gleichzeitig gesetzt sein. Dies ist der Aufbau des CubeState:

Name	Bit	Wert	Bedeutung
STATE_ERROR	0	0x00000001	Es liegt ein Fehler im Modul vor. Der Antrieb löst einen Nothalt aus und nimmt keine weiteren Bewegungskommandos an. Die Fehlerursache kann anhand der Fehlerbits ermittelt werden. In vielen Fällen kann der Anwender einen Fehlerzustand durch ein RESET-Kommando quittieren und mit der Arbeit fortfahren.
STATE_HOME_OK	1	0x00000002	Dieses Bit wird nach erfolgreicher Referenzpunktfahrt gesetzt. Es bedeutet, daß der Antrieb den Nullpunkt in seinem Koordinatensystem eindeutig zugeordnet hat und alle voreingestellten Werte, die die Positionsbegrenzung betreffen Gültigkeit haben. Dieses Bit wird bei Beginn jeder Referenzpunktfahrt (Kommando HOME) gelöscht.
STATE_HALTED	2	0x00000004	Dieses Bit wird im Zusammenhang mit einem Nothalt gesetzt. Es besagt, daß der Antrieb steht und sich in einem Sicherheitszustand befindet. Bevor der Antrieb wieder Bewegungskommandos ausführt, muß er ein RESET-Kommando erhalten haben und eventuelle Fehlerbits gelöscht haben. Nothalt kann durch die interne Fehlerüberwachung oder durch das HALT-Kommando ausgelöst werden.
STATE_POWERFAULT	3	0x00000008	Dieses Bit gibt einen Fehler der PowerCube™-Endstufe an. In den meisten Fällen wird nach Auftreten dieses Fehlers ein Abschalten der Betriebsspannung notwendig sein. Der Fehler wird genauer durch die Bits 18 bis 23 beschrieben.
STATE_TOW_ERROR	4	0x00000010	Schleppfehler. Der Lageregler konnte der Sollwertvorgabe nicht folgen und hat eine nicht zulässige Regelabweichung detektiert. Die maximal zulässige Regelabweichung kann anhand des Wertes „MaxDeltaPos“ eingestellt werden.
STATE_COMM_ERROR	5	0x00000020	Ein Fehler bei der Datenübertragung ist aufgetreten. Sollte dieser Fehler häufig auftreten, ist eine Untersuchung der Verbindungskabel sinnvoll. Prüfen Sie, ob der Bus mit einem Widerstand abgeschlossen wurde.
STATE_SWR	6	0x00000040	Dieses Bit zeigt den Zustand des Referenzschalters an. Ist es gesetzt, so ist der Referenzschalter aktiv (positive Logik).
STATE_SW1	7	0x00000080	Dieses Bit zeigt den Zustand des Endlagenschalters "1" an. Ist es gesetzt, so ist der Endlagenschalter "1" aktiv (positive Logik), die Endlage wurde überschritten.
STATE_SW2	8	0x00000100	Dieses Bit zeigt den Zustand des Endlagenschalters "2" an. Ist es gesetzt, so ist der Endlagenschalter "2" aktiv (positive Logik), die Endlage wurde überschritten.

STATE_BRAKEACTIVE	9	0x00000200	Dieses Bit zeigt den Zustand der Bremse an. Es wird nur verwendet, wenn eine Bremse konfiguriert wurde. Ist das Bit gesetzt, so ist die Bremse aktiviert und übt ihre Wirkung aus.
STATE_CURLIMIT	10	0x00000400	Dieses Bit ist eine Warnung des Lagereglers, dessen Stellwertbegrenzung aktiviert wurde. Der Antrieb befindet sich mit den aktuellen Einstellungen im Grenzlasterbereich und dem Regler fehlt die notwendige Regelreserve. Das Bit kann durch RESET gelöscht werden.
STATE_MOTION	11	0x00000800	Dieses Bit zeigt an, daß sich der Antrieb in Bewegung befindet. Nach Stillstand wird es automatisch gelöscht.
STATE_RAMP_ACC	12	0x00001000	Dieses Bit zeigt an, daß der Antrieb in die Beschleunigungsphase beim Rampenbetrieb eingetreten ist. Es wird erst nach Beendigung der Rampenfahrt gelöscht.
STATE_RAMP_STEADY	13	0x00002000	Dieses Bit zeigt an, daß der Antrieb in die Phase der gleichförmigen Bewegung beim Rampenbetrieb eingetreten ist. Es wird erst nach Beendigung der Rampenfahrt gelöscht.
STATE_RAMP_DEC	14	0x00004000	Dieses Bit zeigt an, daß der Antrieb in die Verzögerungsphase beim Rampenbetrieb eingetreten ist. Es wird erst nach Beendigung der Rampenfahrt gelöscht.
STATE_RAMP_END	15	0x00008000	Mit diesem Bit signalisiert der Antrieb, daß die Rampenfahrt beendet wurde und sich der Motor im Stillstand befindet.
STATE_INPROGRESS	16	0x00010000	Dieses Bit hat nur im Stepbetrieb Bedeutung. Ein STEP-Kommando befindet sich in Ausführung.
STATE_FULLBUFFER	17	0x00020000	Dieses Bit hat nur im Stepbetrieb Bedeutung. Es wird gesetzt, wenn STATE_INPROGRESS gesetzt ist und ein weiteres STEP-Kommando empfangen wurde. Das zuletzt empfangene STEP-Kommando wird in den Puffer übernommen und nach Ausführung des vorhergehenden STEP-Kommandos automatisch "angehängt".
STATE_POW_VOLT_ERR	18	0x00040000	Die Zwischenkreisspannung des Motors ist unter den zulässigen Minimalwert gesunken. Die Endstufe wird abgeschaltet. Dieser Fehler kann durch RESET zurückgesetzt werden.
STATE_POW_FET_TEMP	19	0x00080000	Die Temperatur der Endstufentransistoren hat das zulässige Maximum überschritten. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden.
STATE_POW_WDG_TEMP	20	0x00100000	Die Temperatur der Motorwicklungen hat das zulässige Maximum überschritten. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden.
STATE_POW_SHORTCUR	21	0x00200000	Es ist ein Kurzschluß in der Endstufe aufgetreten. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden. Konsultieren Sie Ihren PowerCube™-Servicepartner.
STATE_POW_HALLERR	22	0x00400000	Es ist ein Fehler bei der Erfassung der Hallimpulse aufgetreten. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden.
STATE_POW_INTEGRALERR	23	0x00800000	Die Dauerbelastung des Motors hat die Grenze der Zulässigkeit überschritten. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden. Prüfen Sie die Belastung des Antriebs und die Vorgaben für Geschwindigkeit und Beschleunigung.

STATE_CPU_OVERLOAD	24	0x01000000	Die CPU ist überlastet. Die Endstufe wird abgeschaltet. Die Betriebsspannung muß abgeschaltet werden. Wenden Sie sich an Ihren PowerCube™-Servicepartner.
STATE_POW_TIMEOUT *	24	0x01000000	Kommunikation zwischen CPU und Stromregler ist unterbrochen. Die Betriebsspannung muß abgeschaltet werden. Wenden Sie sich an Ihren PowerCube™-Servicepartner. *Ab Modulversion 3.5.14 verfügbar. Ersetzt STATE_CPU_OVERLOAD.
STATE_BEYOND_HARD	25	0x02000000	Der PowerCube™ ist außerhalb des zulässigen Bewegungsbereichs. Um den Antrieb mit Motorkraft wieder aus der Endlage zu befreien, ist eine vorgeschriebene Prozedur einzuhalten. Siehe dazu " Betriebsweise bei Störungen ".
STATE_BEYOND_SOFT	26	0x04000000	Der PowerCube™ hat die Software-Endlagen überschritten. Es wird ein Nothalt ausgeführt. Dieser Fehler kann durch RESET gelöscht werden.
STATE_POW_SETUP_ERR *	27	0x08000000	Fehler bei der Initialisierung des Stromreglers, Moduleinstellung stimmt nicht mit Reglerkonfiguration überein (5A/10A Typen). Die Betriebsspannung muß abgeschaltet werden. Wenden Sie sich an Ihren PowerCube™-Servicepartner. *Ab Modulversion 3.5.14 verfügbar.

7.3.4 Betriebsweise bei Störungen

Die Vorgabe von Daten an den Antrieb wird vom Betriebssystem des PowerCube™ überwacht. Im einzelnen bedeutet dies:

Fehler, die während des Betriebes auftreten, werden Ihnen im Statusworts "[CubeState](#)" mitgeteilt. Bei den Fehlermeldungen sollte man zwischen den zurücksetzbaren Fehlern und den Fehlern unterscheiden, die ein Abschalten des Moduls erfordern .

Folgende Fehler erfordern ein sofortiges Abschalten des Moduls:

- Temperaturfehler in der Endstufe oder in der Motorwicklung.
Wenn Sie Ihre Module im Dauerbetrieb überlasten, kann es zu diesem Fehler kommen. Bitte überdenken Sie den Einsatz der Module.
- Kurzschluß in der Endstufe, Hallgeberfehler
Diese Fehler können eigentlich nur durch äußere Gewalteinwirkung auftreten. Bitte wenden Sie sich an Ihren Service-Partner.

Die folgenden Fehler können zurückgesetzt werden:

- [Schleppfehler](#)
- [Fehler durch Überfahren der Endlagenschalter](#).
- [Fehler in der Datenübertragung](#)
- [Zusammenbruch der Zwischenkreisspannung](#)

8. PowerCube™ DriveLib (M5API) - Programmierhandbuch

Die von der AMTEC GmbH entwickelten intelligenten Antriebsmodule des PowerCube™-Systems werden über eine serielle Kommunikationsschnittstelle angesteuert. Das Prinzip der Kommunikation mit den Antriebsmodulen und die dafür verfügbare Systemsoftware werden in dem vorliegenden Programmierhandbuch beschrieben.

In diesem Programmierhandbuch sind alle verfügbaren Funktionsbibliotheken des PowerCube™-Systems beschrieben. Bitte beachten Sie, daß nur die Abschnitte für Ihre Anwendung Gültigkeit haben, für die Sie die Funktionsbibliotheken erworben haben.

Die für alle Bibliotheken verwendete, einheitliche *DriveLib*-Programmierschnittstelle stellt eine Treiber-Schnittstelle zur Ansteuerung von intelligenten Positionierantrieben des PowerCube™-Systems dar. Auf der Basis dieser Programmierschnittstelle entwickelte Applikationen erlauben den Einsatz aller Funktionsbibliotheken des PowerCube™-Systems.

Application Program Interface = API

Diese Programmierschnittstelle (im folgenden API genannt) ist von der konkreten Implementierung unabhängig, das heißt daß unterschiedliche Treibervarianten über die gleiche Programmierschnittstelle angesteuert werden.

8.1 Leistungsumfang der PowerCube™-Systemsoftwaremodule

Im einzelnen bietet das API mit seinen einfach zu handhabenden Funktionen folgende Leistungen:

- Ansteuerung der PowerCube™-Antriebsmodule (m5dll)

Demnächst optional gegen Aufpreis erhältlich:

- ActiveX-Komponente (xm5device) zur Einbettung in unterschiedliche Entwicklungsumgebungen.

Die konkreten Treiber des API stehen Ihnen in Form vorcompilierter dynamischer Funktionsbibliotheken (DLL) zur Einbindung in Ihre Anwendersoftware zur Verfügung.

8.2 Systemvoraussetzungen

Überzeugen Sie sich bitte zunächst davon, daß Ihr Rechnersystem die nötigen Voraussetzungen zur Anwendung der PowerCube™-Systemsoftware erfüllt.

- Der verwendete PC muß ein IBM-kompatibler PC mit mindestens einem Pentium-133 Prozessor sein.
- Es muß Windows 95, 98 oder NT installiert sein.
- Die gegebenenfalls notwendige Hardware-Schnittstelle muß installiert sein und der Treiber dieser Hardwareschnittstelle (z.B. für CAN) muß geladen sein.

8.3 Allgemeine Hinweise zur Installation

Die für die Anwendung der einzelnen Funktionsbibliotheken notwendigen Dateien (C-Headerdateien, LIB-Dateien) sind in einem Verzeichnis zusammengefaßt. Im Lieferumfang enthalten ist generell auch ein einfaches Testprogramm, das Ihnen den Einstieg in die Anwendung der PowerCube™-Systemsoftware erleichtern soll.

Die Kommunikation der Softwareschnittstelle zu den Modulen erfordert es, daß entsprechende

Treiber im System installiert worden sind, die es ermöglichen, eine Kommunikation zwischen m5dll und dem PowerCube™-Modul herzustellen. Diese Treiber sind nicht Bestandteil der m5dll und deren Installation ist herstellerspezifisch und muß gegebenenfalls im Handbuch des Hardwareanbieters nachgelesen werden.

Von der aktuell vorliegenden Version der m5dll werden folgende Schnittstellen unterstützt:

- CAN Interfaces der Firma IXXAT
- RS232 Standard RS232-Schnittstelle des Rechners.
- NET Kommunikation über TCP/IP-Netze

Die Kommunikation über die RS232-Schnittstelle erfolgt ohne die Notwendigkeit einer Treiberinstallation, da diese bereits im Betriebssystem enthalten sind.

Zur Kommunikation über die VCI (CAN) Schnittstelle ist es erforderlich, die VCI-Treiber der Firma IXXAT zu installieren.

Sollte die Netzwerkschnittstelle verwendet werden, so muß auf dem Clientrechner die WINSOCK2 Schnittstelle im Betriebssystem installiert sein. Dies ist unter Windows NT 4.0 SP3 und Windows 98 standardmäßig der Fall. Des weiteren wird das Netzwerkprotokoll TCP/IP benötigt, welches auch korrekt installiert sein muß.

8.4 Typographie

Zur Hervorhebung und übersichtlicheren Darstellung bestimmter Sachverhalte werden innerhalb dieses Handbuchs verschiedene Schriftarten verwendet:

- reserved* reservierte Wörter, Namen von Windows-API-Funktionen etc.
- listing* Programmlistings von Beispielprogrammen, Namen von API-Funktionen, Elemente der Programmiersprache
- parameter* Parameter und Rückgabewerte der API-Funktionen.

8.5 Grundlagen der Programmierung mit PowerCube™-Systembibliotheken

Die PowerCube™-Systembibliotheken lassen sich - unabhängig von der unterstützten Thematik - nach einem bestimmten Schema anwenden. Wie bereits eingangs geschildert, ist die Schnittstellendefinition unabhängig von der konkreten Implementierung. Darüberhinaus sind einige allgemeine Funktionen bei allen Schnittstellendefinitionen zu finden.

Im folgenden werden die grundlegenden Mechanismen der verschiedenen allgemeinen Dienste beschrieben.

8.5.1 Aufbau der Systembibliotheken

Treiber sind Software-Bibliotheken zur Ansteuerung von Hardware-Schnittstellen. Die Treiber werden in der Regel in Form von dynamischen Bibliotheken (DLL) bereitgestellt.

8.5.1.1 Format der M5dll-Funktionen

Der Funktionsprototyp aller M5dll-Funktionen ist ähnlich aufgebaut:

- Funktionsname
Der Name jeder Funktion - ausgenommen wenige allgemeine Funktionen, die in jedem Treiber vorhanden sind - beinhaltet die Thematik, der diese Funktion zuzuordnen ist.
- Parameter und gelieferte Daten
Von der Funktion zu liefernde Daten werden über Referenzparameter an das Anwenderprogramm übermittelt. Dadurch können Funktionen auch mehrere Informationen gleichzeitig bereitstellen.
- Fehlercodes
Der eigentliche Rückgabewert der Funktion stellt einen Fehlercode dar. Dadurch läßt sich der Erfolg des Funktionsaufrufs oder die genaue Ursache eines Fehlers feststellen. Es gilt, daß alle Werte ungleich 0 einen Fehlerzustand darstellen. Liefern Funktionen 0 zurück, so ist davon auszugehen, daß die Funktion erfolgreich ausgeführt wurde.

8.6 Programmierung mit der m5dll-Bibliothek

In diesem Abschnitt werden allgemein im Umgang mit der m5dll Bibliothek häufig benötigte Datentypen und Strukturen erläutert.

8.6.1 Allgemeines

Die m5dll-Bibliothek wird in der Regel in Form einer DLL ausgeliefert.

Folgende Dateien werden im Zusammenhang mit der Programmierung benötigt:

- die Funktionsbibliothek (DLL) der m5dll,
- *C-Header*-Dateien mit allgemeinen Definitionen, Konstanten und Funktionsprototypen.

Da meist C- oder C++-Programmierungsumgebungen benutzt werden, bezieht sich die Beschreibung der Datentypen und Strukturen auf die Programmiersprache C.

8.6.2 Dateien

Wie bereits oben genannt, werden neben der eigentlichen Bibliothek-DLL die C-Header-Dateien zur Programmentwicklung benötigt.

Diese Header-Dateien (*include*-Dateien) enthalten folgende Informationen:

- Definition der Standard-Datentypen
- Definition der verwendeten Strukturen
- Definition von Konstanten und Fehlercodes
- Liste der Funktionsprototypen

Die Header-Dateien beschreiben in der Regel keine Eigenschaften der konkreten Bibliothek, sondern die für mehrere Implementierungen gültige Schnittstelle. Daher werden für die konkrete Version der Bibliothek keine weiteren Informationen benötigt.

8.6.3 Programmierkonventionen

Zu den im folgenden genannten Konventionen zur Programmierung gehören die Datentypen und die Definition der Fehlercodes.

8.6.4 Datentypen

Die bei den diversen Funktionen verwendeten Standard-Datentypen sind:

Datentyp	Datentypbedeutung
Void	void
Char, Int8, Bool	8 Bit ganzzahlig mit Vorzeichen
UChar, Byte, UInt8	8 Bit ganzzahlig ohne Vorzeichen
Short, Int16	16 Bit ganzzahlig mit Vorzeichen
UShort, SWord, UInt16	16 Bit ganzzahlig ohne Vorzeichen
Int, Int32	32 Bit ganzzahlig mit Vorzeichen
UInt, Word, UInt32	32 Bit ganzzahlig ohne Vorzeichen
Float	32 Bit Gleitkomma
Double	64 Bit Gleitkomma
LongDouble	80 Bit Gleitkomma

Alle oben genannten Datentypen stellen lediglich speziell vereinbarte Namen für die in der Regel bereits in der Programmierumgebung vorhandenen Standarddatentypen dar. Diese Vereinbarungen befinden sich in der Datei TYPES.H. Darüberhinaus sind noch weitere Datentypen in dieser Datei vereinbart, die eine spezielle Verwendung bezeichnen.

Der von allen M5-Funktionen gelieferte Rückgabewert ist vom Typ:

- int 32 Bit ganzzahlig mit Vorzeichen - wird verwendet, um den Erfolg eines Funktionsaufrufes anzuzeigen.

8.6.5 Rückgabewert von Funktionen

Rückgabewert	Rückgabewert-Bedeutung
<i>CLD_OK</i>	Erfolg - kein Fehler
<i>CLERR_INITIALIZATIONERROR</i>	Fehler bei der Initialisierung
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_BADPARAM</i>	falsche Parameter wurden an die Funktion übergeben
<i>CLERR_NOINITSTRING</i>	es wurde kein Initialisierungsstring angegeben
<i>CLERR_NODEVICENAME</i>	Schnittstelle wurde nicht angegeben
<i>CLERR_BADDEVICEINITSTRING</i>	Der angegebene Initialisierungsstring ist nicht gültig
<i>CLERR_HARDWARENOTFOUND</i>	Treiber findet keine Interfacekarte
<i>CLERR_DEVICENOTOPEN</i>	Schnittstelle konnte nicht geöffnet werden
<i>CLDERR_THREADNOTCREATED</i>	interner Thread konnte nicht erzeugt werden
<i>CLDERR_WRONGHANDLE</i>	Sie verwenden einen bereits gelöschten, oder noch nicht angelegten Zeiger auf ein Device
<i>CLDERR_RECEIVEERROR</i>	Angeforderter Parameter konnte nicht empfangen werden
<i>CLDERR_DRIVE_NOMODULEFOUND</i>	Es wurden keine ansprechbaren Module gefunden
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Modul(e) antworten nicht

Die spezieller Thematik zugeordneten Fehlercodes sind im Referenzteil des vorliegenden Handbuches zu jeder Funktion erklärt.

8.6.6 Vorgehensweise bei Fehlern

Falls eine PCube-Funktion einen Rückgabewert ungleich 0 liefert, wird damit ein Fehler signalisiert. Der Rückgabewert stellt eine 32-Bit-Konstante dar, die zur Fehleranalyse ausgewertet werden kann.

8.7 Grundlagen der m5dll-Schnittstelle

In diesem Abschnitt wird die Programmier-Schnittstelle *m5dll* zur Ansteuerung von intelligenten Antriebsmodulen beschrieben. Dazu zählen Funktionen zur Initialisierung der Schnittstelle zu den Antrieben, Funktionen zur Parametrierung der Antriebe und Bewegungsfunktionen.

8.7.1 Allgemeines

Die Funktionen der Programmierschnittstelle beziehen sich auf einen einzelnen Antrieb (*Drive*) der zu einer definierten Schnittstelle (Device) gehört.

8.7.1.1 Bewegungsart

Die modulspezifischen Funktionen zur Parametrierung und Bewegung erhalten in der Regel Werte im Gleitkommaformat (32 Bit), die die Längenangaben (bei Linearmodulen) oder Winkelangaben (bei Drehmodulen) in Einheiten des SI-Systems darstellen (m oder rad). Dadurch wird der Anwender von der sonst erforderlichen Umrechnung in sogenannte Inkremente (Geberimpulse der Antriebseinheiten) entlastet.

8.7.2 Vorbereitung

Der folgende Abschnitt beschreibt die erforderlichen Maßnahmen, um die Module zu initialisieren.

8.7.2.1 Initialisierung der verwendeten Schnittstelle (Device)

Für das Versenden und Empfangen von Nachrichten zwischen den intelligenten Antriebsmodulen und dem PC ist in der Regel eine Einsteckkarte verantwortlich. Sie muß zu Beginn der Arbeiten initialisiert und nach Abschluß entsprechend deinitialisiert werden. Dazu dienen die Funktionen [PCube_openDevice](#) und [PCube_closeDevice](#). Mit dem zu übergebenden Initialisierungsstring erhält die m5dll die Information, auf welcher Schnittstelle die Kommunikation mit den Modulen stattfinden soll:

Zur Zeit werden folgende Schnittstellen unterstützt:

- VCI Interface der Firma IXXAT, CAN-Controller
- RS232

Der Initialisierungsstring besteht im groben aus zwei Teilen:

1. der Schnittstellenkennung und den
2. Parameterlisten.

Einige Parameter sind für alle Schnittstellen identisch, andere jedoch spezifisch für die verwendete Schnittstelle.

Vergewissern Sie sich, daß alle relevanten Informationen Ihrer Schnittstellenkarte bekannt sind. Dies wären bei CAN-Karten z.B. die IO-Adresse, Interrupt, Kartentyp (PCMCIA, ISA, PCI) oder bei RS232 die Comport-Nummer und die Baudrate der Module (in der Regel 9600 Baud).

8.7.3 Der Initialisierungsstring

Der Initialisierungsstring bestimmt die verwendete Hardware, die vom Anwendungsprogramm angesteuert werden soll. Der Aufbau des Initialisierungsstrings ist aufgrund der Vielfalt der möglichen Hardwareeigenarten und variantenreichen Sonderfunktionen der m5dll recht komplex. Die Hardware unterstützt die Schnittstellen RS232, RS485, Profibus und CAN. Die Software unterstützt derzeit nur die Schnittstellen RS232 und CAN. Außerdem wird eine Softwareanbindung der Vorgänger-Baureihe (MORSE / M3DLL) ermöglicht.

8.7.3.1 Der Aufbau des Basis-Initialisierungsstrings

Grundsätzlich ist die Form des Initialisierungsstring folgende:

SCHNITTSTELLENKENNUNG:par1,par2,par3,option1=on,option2=off"

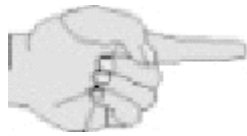
Der Basis-Initialisierungsstring enthält die Schnittstellenkennung und bis zu drei Parameter. Weitere Optionen werden entsprechend angefügt.

Mögliche Schnittstellenkennungen und deren zugehörige Parameter sind:

Schnittstelle	Parameter			
VCI: oder M3VCI:	TYPE	Name	PORT bzw. PCMCIA Slot bzw. Board-Zähler, bzw. Basisadresse	Interrupt oder Baudrate (*)
	0	iPC-I 165	Basisadr.:z.B. d000	IRQ:1 - 15
	1	iPC-I 320	Basisadr.:z.B. d000	IRQ:1 - 15
	2	CANdy 320	LPT-Port z.B. 1	IRQ: 5 oder 7 bzw. der vom System für den LPT vergebene IRQ
	3	tinCAN V2	PCMCIA-Slot: z.B. 0	IRQ:1 - 15
	4	keine Unterstützung		
	5	keine Unterstützung		
	6	iPC-I 165 PCI	Board-Nr.: z.B. 0	1
7	iPC-I 320 PCI	Board-Nr.: z.B. 0	1	
NET	HOST-IP-Adresse oder HOSTNAME : z.B. localhost oder 192.168.1.1			
RS232		COM Port: 1-4	BAUD Rate: z.B. 9600	

(*) Bei Verwendung von Karten, die einen Interrupt (IRQ) unterstützen, sollte dieser in jedem Fall auch benutzt werden. Dies ist insbesondere bei der Verwendung von PowerConfig (Modulkonfiguration) zwingend notwendig!

Schnittstelle	Beispiel INIT-Strings	
VCI: oder M3VCI:	TYPE	INIT-String
	0	VCI:0,D000,5 (Win 95/98 VCI:0,D000,5,nonet)
	1	VCI:1,D800,7 (Win 95/98 VCI:1,D800,7,nonet)
	2	VCI:2,1,7 (Win 95/98 VCI:2,1,7,nonet)
	3	VCI:3,0,5 (Win 95/98 VCI:3,0,5,nonet)
	4	keine Unterstützung
	5	keine Unterstützung
	6	VCI:6,0,1 (Win 95/98 VCI:6,0,1,nonet)
7	VCI:7,0,1 (Win 95/98 VCI:7,0,1,nonet)	
NET	NET: LOCALHOST NET: 192.168.1.1	
RS232	RS232:1,9600 (Win 95/98 RS232:1,9600,nonet)	
Profibus IM180	IM180:0,0 (Win 95/98 IM180:0,0,nonet)	



```

Char* pInitString = "VCI:1,d000,11";
// Startadresse d000H
// Interrupt 11
// VCI-CAN Interface

// Type iPC320
int xtrnlID;

int retVal = Pcube_openDevice( &xtrnlID, pInitString );
if( retVal != CLD_OK ) // Initialisierung fehlerhaft?
... // ja - Fehlerauswertung
else
{ ... // weitere Funktionen
}

Pcube_closeDevice( xtrnlID ); // letzter Funktionsaufruf...

```

8.7.3.2 Optionen im Initialisierungsstring

Die im folgenden beschriebenen Optionen werden durch Komma getrennt nach dem Basisinitialisierungsstring angefügt. Folgende Optionen bestehen, um die m5dll zusätzlich zu parametrieren:

8.7.3.2.1 Debugmodus

Durch Anfügen der Option D direkt nach dem Basis-Initialisierungsstring wird die Kommunikation der m5dll mit den PowerCube™-Modulen in eine Datei debug.txt protokolliert, welche im Verzeichnis der Anwendung liegt, die die m5dll verwendet.

8.7.3.2.2 Netzwerkspezifische Optionen

Die Optionen der Schnittstellenkennung NET: lauten:

port=Portadresse Falls ein Server gestartet wurde, der eine Netzwerkfunktionalität besitzt, so wird mittels dieser Option dessen Portadresse gesetzt. Diese muß der des Servers entsprechen.

passwd=PASSWORT Falls ein Server gestartet wurde der eine Netzwerkfunktionalität besitzt, so wird mittels dieser Option ein Authentifizierungspasswort gesetzt. Dieses muß dem des Servers entsprechen.

8.7.3.2.3 CAN-spezifische Optionen

Die Optionen der Schnittstellenkennung VCI: und M3VCI: lauten:

nonet Es soll kein Netzwerkserver gestartet werden.

port=Portadresse Falls ein Netzwerkserver gestartet werden soll wird mittels dieser Option dessen Portadresse gesetzt

passwd=PASSWORT Falls ein Server gestartet werden soll, so wird mittels dieser Option ein Authentifizierungspasswort gesetzt.

8.7.3.2.4 RS232-spezifische Optionen

nonet Es soll kein Netzwerkserver gestartet werden.

port=Portadresse Falls ein Netzwerkserver gestartet werden soll wird mittels dieser Option dessen Portadresse gesetzt

passwd=PASSWORT Falls ein Server gestartet werden soll, so wird mittels dieser Option ein Authentifizierungspasswort gesetzt.

8.7.4 Einige allgemeine Sonderfunktionen

8.7.4.1 Abfragen der Anzahl der Module

Mittels der Funktion `PCube_getModuleCount` kann ermittelt werden, wie viele Module an der spezifizierten Schnittstelle angeschlossen sind.

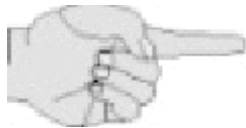
Mit Hilfe der Funktion `PCube_getModulesIdMap` kann eine Liste der gefundenen Module mit den physikalischen ID's erzeugt werden. Diese Funktion liefert darüber hinaus noch die Anzahl der gefundenen Module zurück.

```
{
    int idMap[30];
    numDrives=Pcube_getModulesIdMap(xtrnlId, idMap);
}
```

8.7.4.2 Modultyp erfragen

Zur Feststellung der Bewegungsform des jeweiligen Antriebsmoduls und anderer wesentlicher Eigenschaften dient die Funktion `PCube_getModuleType`. Die Funktion liefert den Typ des Moduls zurück und entscheidet über die Interpretation der Funktionsparameter. Folgende Modultypen sind definiert:

- `ROTARY_DRIVE` Antriebsmodul ist ein Drehmodul, die Parameter werden in der Einheit **rad** (Radiant) angegeben.
- `LINEAR_DRIVE` Antriebsmodul ist ein Linearmodul, die Parameter werden in der Einheit **m** (Meter) angegeben.



Beispiel:

```
int type;
CLDRet retVal = Pcube_getModuleType( xtrnlId, &type );
if(type == LINEAR_DRIVE )
... // Angaben in Meter

else if( type == ROTARY_DRIVE )
... // Angaben in Radiant
... // weitere Funktionen
```


8.7.4.3 Programmierung von Bewegungen

Im folgenden werden die Einheiten der Bewegungsparameter sowie die Bedeutung einzelner Funktionskategorien erläutert. Zur Verwendung der Funktionen siehe auch Abschnitt 15.8, 'Referenz'.

8.7.4.3.1 Einheiten

Bei der Programmierung von Bewegungen und der Einstellung von Bewegungsparametern ist zu beachten, daß alle Angaben in SI-Einheiten erfolgen, um weiterführende Berechnungen zu vereinfachen.

Die verwendeten Einheiten sind:

Einheiten	Linearantrieb	Drehantrieb
Position	m	rad
Geschwindigkeit	m/s	rad/s
Beschleunigung	m/s ²	rad/s ²
Motorstrom	A	A
Temperatur	K	K

8.7.4.3.2 Verfahrbereich

Das Betriebssystem des Antriebsmoduls schränkt normalerweise den möglichen Verfahrbereich ein. Dieser *Verfahrbereich* bezeichnet den Abstand zwischen den im Modul eingestellten Softwareendlagen.

8.7.4.3.3 Dynamik

Bei allen Bewegungen des Antriebs werden voreingestellte oder von der übergeordneten Steuerung vorgegebene Werte für Fahrgeschwindigkeit und -beschleunigung verwendet. Dabei werden zwei auch als *Dynamik* bezeichnete Wertepaare benutzt. Die *aktuelle* Dynamik wird bei allen von der übergeordneten Steuerung (dem PC-Anwenderprogramm) ausgelösten Bewegungen verwendet. Die *maximale* Dynamik dient dem Modul-Betriebssystem zur Begrenzung aller Bewegungen. Diese Grenzwerte sind standardmäßig im Modul voreingestellt, sie können jedoch von der übergeordneten Steuerung weiter eingeschränkt werden. Diese Parametereinstellungen sind online möglich. Nach Abschalten sind die Werte jedoch verloren.

8.7.4.3.4 Status des Antriebsmoduls abfragen

Bit im Statuswort	Bedeutung
STATE_HOME_OK	Die Homing-Fahrt des Moduls wurde bereits erfolgreich durchgeführt.
STATE_HALTED	Der HALT-Zustand wurde aufgrund eines weiteren Fehlers oder durch das Kommando Pcube_haltModule() hervorgerufen.
STATE_SWR	Das Modul befindet sich im Bereich des Referenzschalters.
STATE_SW1	Das Modul befindet sich im Bereich des Endlagenschalters 1.
STATE_SW2	Das Modul befindet sich im Bereich des Endlagenschalters 2
STATE_MOTION	Das Modul befindet sich in einer Bewegung
STATE_RAMP_ACC	Das Modul beschleunigt gerade.
STATE_RAMP_STEADY	Das Modul befindet sich in einer Rampenfahrt im Bereich mit konstanter Geschwindigkeit.
STATE_RAMP_DEC	Das Modul bremst ab (absteigende Rampenfahrt).
STATE_RAMP_END	Die Rampenfahrt wurde beendet.
STATE_INPROGRESS	Ein Step-Kommando ist in Ausführung. Es kann noch eine weitere Position mittels Pcube_moveStep() übergeben werden (nicht bei M3-Kompatibilität).
STATE_FULLBUFFER	Das zuletzt empfangene STEP-Kommando wird in den Puffer übernommen und nach Ausführung des vorhergehenden STEP-Kommandos automatisch "angehängt" (nicht bei M3-Kompatibilität).
STATE_BRAKEACTIVE	Die Bremse ist aktiviert.
STATE_CURLIMIT	Der Antrieb befindet sich mit den aktuellen Einstellungen im Grenzlastbereich
STATE_ERROR	Kennzeichnet, daß ein Fehler vorliegt. Die Fehlerursache kann durch Auswertung der restlichen Flags bestimmt werden.
STATE_POWERFAULT	Ein Fehler in der Motorendstufe des Moduls wurde entdeckt.
STATE_TOW_ERROR	Die maximale Abweichung von der Sollposition wurde überschritten.
STATE_COM_ERROR	Ein Fehler in der Kommunikation wurde bemerkt.
STATE_POW_VOLT_ERR	Die Zwischenkreisspannung des Motors ist unter den zulässigen Minimalwert gesunken. Die Endstufe wird abgeschaltet.
STATE_POW_FET_TEMP	Die Temperatur der Endstufentransistoren hat das zulässige Maximum überschritten. Die Endstufe wird abgeschaltet.
STATE_POW_WDG_TEMP	Die Temperatur der Motorwicklungen hat das zulässige Maximum überschritten. Die Endstufe wird abgeschaltet.
STATE_POW_SHORTCUR	Es ist ein Kurzschluß in der Endstufe aufgetreten. Die Endstufe wird abgeschaltet

STATE_POW_HALLERR	Es ist ein Fehler bei der Erfassung der Hallimpulse aufgetreten. Die Endstufe wird abgeschaltet.
STATE_POW_INTEGRALERR	Die Dauerbelastung des Motors hat die Grenze der Zulässigkeit überschritten. Die Endstufe wird abgeschaltet.
STATE_CPU_OVERLOAD	Die CPU ist überlastet Die Endstufe wird abgeschaltet.

Für die genaue Beschreibung der Statusbits sowie der notwendigen Reaktionen darauf ist unbedingt das Kapitel "Beschreibung des PowerCube™ -Kommunikationsprotokolls", Abschnitt "[Der Modulstatus: Das Statuswort CubeState](#)" zu lesen.

8.7.4.3.5 Dateien

Zur Vereinfachung der Programmierung genügt es, wenn Sie die Datei m5.H mittels der Anweisung `#include "m5.h"` einfügen. Die Include-Datei muß sich in einem dem Projekt zugänglichen Verzeichnis befinden.

8.8 Referenz

Im folgenden Referenzteil werden alle Funktionen erklärt. Ihre Parameter und Rückgabewerte sowie die Bedeutung des Funktionsaufrufes werden beschrieben.

8.8.1 Übersicht

Initialisierungs- und Verwaltungsfunktionen

- [PCube_openDevice](#) Initialisierung der Treiber-DLL
- [PCube_closeDevice](#) Beenden der Treiber-DLL
- [PCube_getModulesCount](#) Abfrage der Anzahl der Drives
- [PCube_getModulesIdMap](#) Abfrage der ID-Zuordnung
- [PCube_getModuleType](#) Abfrage des Modultyps

Kommandofunktionen

- [PCube_syncModule](#) Referenzpunkt anfahren
- [PCube_haltModule](#) Antriebsmodul anhalten
- [PCube_resetModule](#) Rücksetzen des Modulstatus

Statusfunktionen

- [PCube_getCubeState](#) allgemeine Funktion zur Statusabfrage
- [PCube_querySyncEnd](#) Referenzpunkt erreicht?
- [PCube_queryEndPos](#) Endpunkt der Bewegung erreicht?

Bewegungsfunktionen

- [PCube_moveRamp](#) Bewegungskommando im Rampenmodus
- [PCube_moveRampInc](#) Bewegungskommando im Rampenmodus (Parameter in Inkrementen)
- [PCube_moveRampExtended](#) Bewegungskommando im Rampenmodus mit Statusrückgabe
- [PCube_moveStep](#) Bewegungskommando im Schrittmodus
- [PCube_moveStepInc](#) Bewegungskommando im Schrittmodus (Parameter in Inkrementen)
- [PCube_moveStepExtended](#) Bewegungskommando im Schrittmodus mit Statusrückgabe
- [PCube_moveVel](#) Bewegungskommando im Geschwindigkeitsmodus
- [PCube_moveVelInc](#) Bewegungskommando im Geschwindigkeitsmodus (Parameter in Inkrementen)
- [PCube_moveCurrent](#) Bewegungskommando im Strommodus
- [PCube_moveCurrentInc](#) Bewegungskommando im Strommodus (Parameter in Inkrementen)
- [PCube_moveCurrentExtended](#) Bewegungskommando im Strommodus mit Statusrückgabe

Abfragen des aktuellen Modulzustands

- [PCube_getActPos](#) aktuelle Position abfragen
- [PCube_getActVel](#) aktuelle Geschwindigkeit abfragen
- [PCube_getDeltaPos](#) aktuellen Schleppfehler der Position abfragen
- [PCube_getCur](#) aktuelle Stromaufnahme abfragen

Abfragen und Setzen von Bewegungsgrenzwerten

- [PCube_getMaxPos](#) maximale Position abfragen
- [PCube_setMaxPos](#) maximale Position setzen
- [PCube_getMinPos](#) minimale Position abfragen
- [PCube_setMinPos](#) minimale Position setzen
- [PCube_getHomeOffset](#) benutzerdefinierte Nullposition abfragen
- [PCube_setHomeOffset](#) benutzerdefinierte Nullposition setzen
- [PCube_getMaxVel](#) maximale Geschwindigkeit abfragen
- [PCube_setMaxVel](#) maximale Geschwindigkeit setzen
- [PCube_getMinVel](#) minimale Geschwindigkeit abfragen
- [PCube_setMinVel](#) minimale Geschwindigkeit setzen
- [PCube_getMaxAcc](#) maximale Beschleunigung abfragen
- [PCube_setMaxAcc](#) maximale Beschleunigung setzen
- [PCube_getMinAcc](#) minimale Beschleunigung abfragen
- [PCube_setMinAcc](#) minimale Beschleunigung setzen
- [PCube_getMaxCur](#) maximalen Strom abfragen
- [PCube_setMaxCur](#) maximalen Strom setzen
- [PCube_getMinCur](#) minimalen Strom abfragen
- [PCube_setMinCur](#) minimalen Strom setzen
- [PCube_getMaxDeltaPos](#) maximale Positionsabweichung abfragen
- [PCube_setMaxDeltaPos](#) maximale Positionsabweichung setzen

Abfragen und Setzen der Koeffizienten des Lageregelkreises

Diese Funktionen stehen erst ab der Version 2.5.1 zur Verfügung

- [PCube_getCO](#) Koeffizient CO abfragen
- [PCube_getDamp](#) Dämpfungswert abfragen
- [PCube_getAO](#) Koeffizient AO abfragen
- [PCube_setCO](#) Koeffizient CO setzen
- [PCube_setDamp](#) Dämpfungswert setzen
- [PCube_setAO](#) Koeffizient AO setzen
- [PCube_recalcPIDParams](#) Neuberechnen der Regelschleife

Abfragen und Setzen der internen Digital-Ein- und Ausgänge

- [PCube_getDioData](#) Bits der internen DIOs abfragen
- [PCube_setDioData](#) Bits der internen DIOs setzen

Abfrage von Vorgabewerten

- [PCube_getDefMaxPos](#) maximale Position abfragen
- [PCube_getDefMinPos](#) minimale Position abfragen
- [PCube_getDefMaxVel](#) maximale Geschwindigkeit abfragen
- [PCube_getDefMinVel](#) minimale Geschwindigkeit abfragen

- PCube_getDefMaxAcc maximale Beschleunigung abfragen
- PCube_getDefMinAcc minimale Beschleunigung abfragen
- PCube_getDefMaxCur maximalen Strom abfragen
- PCube_getDefMinCur minimalen Strom abfragen
- PCube_getDefDeltaPos maximale Positionsabweichung abfragen
- PCube_getDefCO Koeffizient CO abfragen
- PCube_getDefDamp Dämpfung abfragen
- PCube_getDefAO Koeffizient AO abfragen

8.8.2 Initialisierungs- und Verwaltungsfunktionen

Die im folgenden beschriebenen Funktionen dienen der Initialisierung der Schnittstelle und des angeschlossenen Manipulators mit seinen Antriebsmodulen.

8.8.2.1 PCube_openDevice

Aufgabe

Initialisierung der Schnittstelle zum PowerCube Drive

Prototyp

```
int PCube_openDevice( int* pDev, const char* pInitString );
```

Beschreibung

Diese Funktion initialisiert die Schnittstelle die anhand der in der Zeichenkette *pInitString* übergebenen Parameter. Der [Aufbau des Initialisierungsstrings](#) hängt von der Installation Schnittstelle ab. Der Parameter pDev ist bei allen weiteren Pcube Funktionen anzugeben. Es ist möglich, mehrere verschiedene Schnittstellen innerhalb einer Anwendung zu öffnen. Die Identifikation, welche Schnittstelle verwendet wird, ist durch das zurückgelieferte Handle pDev bekanntgegeben.

Rückgabewert

<i>CLD_OK</i>	Schnittstelle konnte erfolgreich initialisiert werden.
<i>CLDERR_INITIALIZATIONERROR</i>	Fehler bei der Initialisierung
<i>CLDERR_BADPARAM</i>	Falsche Parameter im Initialisierungsstring oder falsch formatiert (siehe entsprechenden Abschnitt im Anhang).
<i>CLDERR_NOINITSTRING</i>	es wurde kein Initialisierungsstring angegeben
<i>CLDERR_NODEVICENAME</i>	Schnittstelle wurde nicht angegeben
<i>CLDERR_BADDEVICEINITSTRING</i>	Der angegebene Initialisierungsstring ist nicht gültig
<i>CLDERR_HARDWARENOTFOUND</i>	VCI-Treiber findet keine Interfacekarte
<i>CLDERR_DEVICENOTOPEN</i>	Schnittstelle konnte nicht geöffnet werden
<i>CLDERR_THREADNOTCREATED</i>	interner Thread konnte nicht erzeugt werden
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

8.8.2.2 PCube_closeDevice

Aufgabe

Initialisierung der Schnittstelle zum PowerCube Drive

Prototyp

```
int PCube_closeDevice( int dev);
```

Beschreibung

Diese Funktion gibt die von der geöffneten Schnittstelle dev belegten Ressourcen frei. Des weiteren wird ein HALT-Kommando an alle an der definierten Schnittstelle vorhandenen Module geschickt.

Rückgabewert

<i>CLD_OK</i>	Device wurde erfolgreich freigegeben.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.2.3 PCube_getModulesCount

Aufgabe

Abfrage der Anzahl Module

Prototyp

```
int PCube_getModulesCount(int dev, int* count);
```

Beschreibung

Über den Zeiger *count* wird die Zahl der erkannten Antriebsmodule an das Anwenderprogramm zurückgeliefert.

Rückgabewert

<i>CLD_OK</i>	Es wurde erfolgreich eine Kommunikation durchgeführt...
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.2.4 PCube_getModulesIdMap

Aufgabe

Abfrage der Drive ID's eines Devices

Prototyp

```
int PCube_getModulesIdMap(int dev, int *idMap);
```

Beschreibung

Diese Funktion füllt eine Liste mit den gefundenen ID's der angeschlossenen Module. Somit ist ein Ansprechen der Module mittels logischer Nummern möglich, ohne die ID's der einzelnen Drives explizit zu kennen.

Rückgabewert

<i>retVal</i>	Anzahl der gefundenen Module
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.2.5 PCube_getModuleType

Aufgabe

Abfrage der Drive ID's eines Devices

Prototyp

```
int PCube_getModuleType(int dev, int cube, UInt8 *type);
```

Beschreibung

Die Funktion fordert den Modultyp des Antriebsmoduls *cube* ab. Dies ist ein 8-Bit-Wort, das in der Variablen, auf die der Zeiger *type* zeigt, abgelegt wird.

Es sind zwei Grundtypen definiert:

ROTARY_DRIVE 0x0f	Angaben in Radiant
LINEAR_DRIVE 0xf0	Angaben in m

Rückgabewert:

<i>CLD_OK</i>	Es wurde erfolgreich eine Kommunikation durchgeführt...
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.3 Kommandofunktionen

Die folgenden Funktionen stellen komplexe Kommandos dar, die das Antriebsmodul selbständig ausführt.

8.8.3.1 PCube_syncModule

Aufgabe

Referenzpunkt anfahren

Prototyp

```
int PCube_syncModule( int dev,int cube);
```

Beschreibung

Mit Aufruf dieser Funktion fährt das Antriebsmodul *cube* automatisch seinen Referenzpunkt an.

Vor dem ersten Fahrkommando muß das Antriebsmodul synchronisiert werden, das heißt es muß seinen Referenzpunkt anfahren, um einen absoluten Bezug zum umgebenden Koordinatensystem zu erhalten.

Die Funktion `PCube_syncModule` startet den Vorgang des selbständigen Anfahrens des Referenzpunktes. Mit Hilfe der Funktion `PCube_querySyncEnd` kann vom Anwenderprogramm aus der Abschluß des Vorgangs festgestellt werden.

Nach dem erfolgreichen Anfahren des Referenzpunktes werden Bewegungskommandos erst wirksam, wenn der Sicherheitszustand im Modul aufgehoben wurde (siehe `PCube_resetModule`).

Rückgabewert

<i>CLD_OK</i>	Anfahren des Referenzpunktes konnte erfolgreich begonnen werden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Beispiel 1

Das folgende vereinfachte Beispiel demonstriert das Anfahren des Referenzpunktes unter Nutzung einer Programmschleife

Das Anfahren des Referenzpunktes kann mehrere Sekunden dauern. Sie sollten in solch einem Anwendungsprogramm einen zusätzlichen Thread erzeugen, der das korrekte Referenzpunktfahren übernimmt und die Anwendung nicht behindert.



```
CLDRet retVal = PCube_syncModule( dev, i );
if( retVal == CLD_OK )
{ Bool sync = FALSE;
...
while( (retVal = PCube_querySyncEnd( dev, i, &sync )) ==
CLD_OK && !sync )
Sleep(100);
...
}
```

Die Endlosschleife wird abgebrochen, wenn ein Fehler auftritt oder der Referenzpunkt erreicht wurde.

Beispiel 2

Folgendes Beispiel zeigt das Anfahren des Referenzpunktes mit Hilfe von WM_TIMER-Nachrichten unter Windows.



```
CLDRet retVal;
Bool sync;
...
case WM_COMMAND:
... // Referenzpunkt anfahren
retVal = PCube_syncModule( dev, i );
...
break;
case WM_TIMER:
... // Referenzpunkt erreicht?
sync = FALSE;
retVal = PCube_querySyncEnd( dev, i, &sync );
if( retVal != CLD_OK )
... // FEHLER!
else if( sync )
... // OK, Modul synchronisiert.
else
... // Vorgang dauert noch an ...
```

Die Nachricht WM_TIMER sollte solange den Vorgang überwachen - beispielsweise im zeitlichen Abstand von 200 Millisekunden - bis ein Fehler festgestellt oder der Referenzpunkt erreicht wurde. Durch diese Trennung des Vorgangs in *Starten* und *zyklisches Überprüfen* wird das System nicht unnötig lange blockiert.

Siehe auch [PCube_querySyncEnd](#), [PCube_resetModule](#)

8.8.3.2 PCube_haltModule

Aufgabe

Antriebsmodul anhalten

Prototyp

```
int PCube_haltModule( int dev,int cube);
```

Beschreibung

Diese Funktion bewirkt, daß das Antriebsmodul *cube* automatisch anhält.

Nach der Ausführung dieser Funktion geht das Antriebsmodul automatisch in einen Sicherheitszustand über. In diesem Zustand werden vom Modul keine Bewegungskommandos angenommen. Zur Aufhebung dieses Zustands muß das Kommando [PCube_resetModule](#) gesendet werden.

Die Funktion [PCube_haltModule](#) startet den Vorgang des selbständigen Anhaltens des Antriebsmoduls. Mit Hilfe der Funktion [PCube_queryEndPos](#) kann vom Anwenderprogramm aus der Stillstand des Moduls festgestellt werden.

Rückgabewert

<i>CLD_OK</i>	Anhalten des Moduls konnte erfolgreich begonnen werden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_queryEndPos](#), [PCube_resetModule](#)

8.8.3.3 PCube_resetModule

Aufgabe

Rücksetzen des Modulstatus

Prototyp

```
int PCube_resetModule( int dev,int cube);
```

Beschreibung

Diese Funktion bewirkt, daß das Antriebsmodul *cube* den Sicherheitszustand aufhebt und damit Bewegungskommandos erlaubt werden.

Folgende Ereignisse führen zur Einnahme des Sicherheitszustands im Antriebsmodul:

- Feststellung eines Fehlers,
- Auslösung eines HALT-Kommandos durch das Anwenderprogramm (durch Aufruf der Funktion [PCube_haltModule](#))

Erst nach Ausführung der Funktion [PCube_resetModule](#) werden vom Antriebsmodul Bewegungskommandos angenommen.

Die Funktion [PCube_resetModule](#) scheitert, wenn die Bedingungen, die zu einem Fehler führten, noch vorliegen. Mit Hilfe der Funktion [PCube_getCubeState](#) kann vom Anwenderprogramm aus der Status des Moduls festgestellt werden.

Mögliche Fehler sind Im Kapitel "Beschreibung des PowerCube™ -Kommunikationsprotokolls", Abschnitt "[Der Modulstatus: Das Statuswort CubeState](#)" aufgeführt.

Rückgabewert

<i>CLD_OK</i>	RESET-Kommando wurde erfolgreich an das Modul gesendet
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '[Vorgehensweise bei Fehlern](#)'.

Siehe auch [PCube_syncModule](#), [PCube_haltModule](#), [PCube_getCubeState](#)

8.8.4 Statusfunktionen

Die im folgenden genannten Funktionen beschreiben den Status des Antriebsmoduls.

8.8.4.1 PCube_getCubeState

Aufgabe

Allgemeine Funktion zur Statusabfrage

Prototyp

```
int PCube_getCubeState( int dev, int cube, UInt32* pValue );
```

Beschreibung

Die Funktion fordert das Statuswort des Antriebsmoduls *cube* ab. Dies ist ein 32-Bit-Wort, das in der Variablen, auf die der Zeiger *pValue* zeigt, abgelegt wird. Jedes Bit in diesem Statuswort hat eine bestimmte Bedeutung, die weiter unten erklärt wird.

Hinweis: Das Statuswort ist im Abschnitt [Status des Antriebsmoduls](#) abfragen beschrieben

Rückgabewert

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

<i>CLD_OK</i>	Status des Moduls wurde abgefordert.
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Siehe auch [PCube_querySyncEnd](#), [PCube_queryEndPos](#)

8.8.4.2 PCube_querySyncEnd

Aufgabe

Prüfung, ob Referenzpunkt erreicht

Prototyp

```
int PCube_querySyncEnd( int dev, int driveId, Bool* pBool );
```

Beschreibung

Die Funktion vereinfacht die Prüfung auf Erreichen des Referenzpunktes. Sie ist nur sinnvoll nach Ausführung der Funktion [PCube_syncModule](#). Das Ergebnis der Prüfung (TRUE == Referenzpunkt erreicht) wird über *pBool* zurückgeliefert.

Rückgabewert

<i>CLD_OK</i>	Prüfung ist erfolgt - in der Variablen, auf die der Zeiger <i>pBool</i> zeigt, befindet sich das Ergebnis der Prüfung.
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Beispiel

Ein ausführliches Beispiel ist zur Funktion [PCube_syncModule](#) zu finden.

Siehe auch [PCube_syncModule](#)

8.8.4.3 PCube_queryEndPos

Aufgabe

Prüfung, ob Endpunkt der Bewegung erreicht

Prototyp

```
int PCube_queryEndPos( int dev, int driveId, Bool* pBool );
```

Beschreibung

Die Funktion vereinfacht die Prüfung auf Erreichen des Endpunktes einer Rampenbewegung. Sie ist nur sinnvoll nach Ausführung der Funktion [PCube_moveRamp](#). Darüberhinaus wird nach Auslösung des HALT-Kommandos oder nach Auftreten eines Fehlers im Modul angezeigt, ob das Modul seinen Stillstand erreicht hat. Das Ergebnis der Prüfung (TRUE == Endpunkt erreicht) wird über *pBool* zurückgeliefert.

Rückgabewert

<i>CLD_OK</i>	Prüfung ist erfolgt - in der Variablen, auf die der Zeiger <i>pBool</i> zeigt, befindet sich das Ergebnis der Prüfung.
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Beispiel Ein ausführliches Beispiel ist zur Funktion [PCube_moveRamp](#) zu finden.

Siehe auch [PCube_moveRamp](#)

8.8.5 Bewegungsfunktionen

Bei Verwendung von Greifer-Modulen (PG70, PG90) ist auf folgende Besonderheit zu achten. Wenn Sie dieses Modul nicht im Current-Modus bewegen (PCube_moveCurrent) müssen Sie unbedingt darauf achten, daß die von Ihnen eingegebene Endposition mit der Außenposition des zu greifenden Gegenstandes übereinstimmt. Ansonsten erhalten Sie Schleppfehlermeldungen.

8.8.5.1 PCube_moveRamp

Aufgabe

Bewegungskommando im Rampenmodus

Prototyp

```
int PCube_moveRamp( int dev, int cube, float pos, float vel,
float acc);
```

Beschreibung

Die Funktion löst eine Bewegung des Antriebsmoduls *cube* im Rampenmodus aus. Dazu wird die Zielposition *pos* mit der maximalen Geschwindigkeit *vel* selbständig angefahren. Die Beschleunigung erfolgt mit dem in *acc* angegebenen Wert. Falls sich das Modul zum Zeitpunkt des Funktionsaufrufes bereits in einer Bewegung befand, wird mit der angegebenen Dynamik unmittelbar die neue Zielposition angefahren.

Die Zielposition *pos* bezeichnet den Abstand zum Referenzpunkt. Die Werte *pos*, *vel* und *acc* werden in Einheiten des SI-Systems angegeben (siehe 'Einheiten'). Falls höhere als die aktuell zulässigen Maximalwerte angegeben werden, werden diese automatisch auf die Maximalwerte begrenzt. Ein Überfahren der Endlagen ist nicht möglich.

Die Funktion schlägt fehl, wenn

- das Modul noch nicht seinen Referenzpunkt angefahren hatte und zurückgesetzt wurde (siehe [PCube_syncModule](#) und [PCube_resetModule](#)),
- sich das Modul nach einem Fehler im Sicherheitszustand befindet (siehe [PCube_getCubeState](#))

Das bedeutet, daß die Funktion scheinbar korrekt ausgeführt wurde, sich das Antriebsmodul jedoch nicht bis zur Zielposition bewegt. Aus diesem Grund ist es ratsam, vor Ausführung der Funktion den Modulstatus zu überprüfen (Funktion [PCube_getCubeState](#)). Bei folgendem Bitmuster ist die Funktion korrekt ausführbar, das heißt die Bewegung wird (zumindest bis zur Endlagenposition) stattfinden:

Bit im Statuswort	erforderlicher Zustand
•STATE_HALTED	darf nicht gesetzt sein
•STATE_HOME_OK	muß gesetzt sein
•STATE_ERROR	darf nicht gesetzt sein

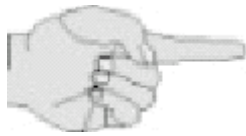
Falls sich die geforderte Zielposition außerhalb der Softwareendlagen befindet, wird das Modul entsprechend der Funktionsparameter bewegt, soweit es die Softwareendlagen zulassen. Unmittelbar vor Erreichen der Softwareendlagen wird das Modul mit der programmierten Beschleunigung abgebremst, so daß die vorgegebene Endlage nicht überschritten wird.

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Beispiel



```
Float pos = 0.120, vel = 0.045, acc = 0.100;
Bool ready;
...
case WM_COMMAND:
... // Zielposition anfahren
retVal = PCube_moveRamp( dev, driveId, pos, vel, acc );
...
break;
case WM_TIMER:
... // Endposition erreicht?
ready = FALSE;
retVal = PCube_queryEndPos( dev, driveId, &ready );
if( retVal != CLD_OK )
... // FEHLER!
else if( ready )
... // OK, Zielposition erreicht.
else
... // Vorgang dauert noch an oder
// prüfen, ob Fehler oder
// Softwareendlage ...
```

Siehe auch [PCube_queryEndPos](#), [PCube_move...](#)

8.8.5.1a PCube_moveRampInc

Aufgabe

Bewegungskommando im Rampenmodus

Prototyp

```
int PCube_moveRampInc( int dev, int cube, Int32 pos,
Int32 vel, Int32 acc);
```

Beschreibung

Die Funktion wird wie [PCube_moveRamp](#) eingesetzt mit dem Unterschied, daß hier Integer-Werte statt Float-Parameter übergeben werden können. Zur Dimensionierung der Parameter siehe unter [Maßeinheiten der PowerCube™-Module](#).

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveStepInc](#), [PCube_moveVelInc](#), [PCube_moveCurrentInc](#)

8.8.5.1b PCube_moveRampExtended

Aufgabe

Bewegungskommando im Rampenmodus mit erweiterter Statusrückgabe. Verfügbar ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13

Prototyp

```
int PCube_moveRampExtended( int dev, int cube, float pos,
float vel, float acc, unsigned long *stateRet, unsigned char
*dioRet, float *posRet);
```

Beschreibung

Die Funktion wird wie [PCube_moveRamp](#) eingesetzt mit dem Unterschied, daß über die Zeiger *stateRet*, *dioRet* und *posRet* ein Kurzstatus, der Zustand der internen digitalen IOs und die aktuelle Position des Moduls zurückgeliefert werden und für eine sofortige Auswertung zur Verfügung stehen (siehe auch Kapitel 7.2.8, [Befehlsbeispiele](#))

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveStepExtended](#), [PCube_moveCurrentExtended](#)

8.8.5.2 PCube_moveStep

Aufgabe

Bewegungskommando im Schrittmodus.

Prototyp

```
int PCube_moveStep( int dev, int cube, float pos, UInt16
step );
```

Beschreibung

Die Funktion bewegt das Antriebsmodul *cube* im Schrittmodus. Dabei wird versucht, die angegebene Zielposition *pos* innerhalb von *step* Millisekunden anzufahren. Die berechnete Geschwindigkeit ist

$$v = (pos - \langle \text{aktuelle Position} \rangle) / step.$$

Die im Antriebsmodul integrierte Bewegungssteuerung nimmt selbständig eine eventuell erforderliche Beschleunigung auf die errechnete Geschwindigkeit vor. Hierfür werden die vorgegebenen maximalen Dynamik-Werte verwendet. Sofern die maximalen Dynamikwerte nicht überschritten werden, sorgt die Bewegungssteuerung dafür, daß die Zielposition zum vorgegebenen Zeitpunkt erreicht wird, selbst wenn der Wert der resultierenden Geschwindigkeit kurzzeitig höher als der der berechneten ist.

Falls spätestens bis Ablauf von *step* Millisekunden kein neues Schrittkommando oder ein Kommando einer anderen Betriebsart gesendet wurde, hält das Modul selbständig mit der aktuellen (Brems-) Beschleunigung an.

Die unter [PCube_moveRamp](#) vorgestellten Informationen über das Fehlschlagen der Funktion, über den erforderlichen Modulstatus und zum Verhalten vor Softwareendlagen gelten auch hier.

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_move...](#)

8.8.5.2a PCube_moveStepInc

Aufgabe

Bewegungskommando im Schrittmodus

Prototyp

```
int PCube_moveStepInc( int dev, int cube, Int32 pos,
  UInt16 step);
```

Beschreibung

Die Funktion wird wie [PCube_moveStep](#) eingesetzt mit dem Unterschied, daß hier Integer-Werte statt Float-Parameter übergeben werden können. Zur Dimensionierung der Parameter siehe unter [Maßeinheiten der PowerCube™-Module](#).

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveRampInc](#), [PCube_moveVelInc](#), [PCube_moveCurrentInc](#)

8.8.5.2b PCube_moveStepExtended

Aufgabe

Bewegungskommando im Schrittmodus mit erweiterter Statusrückgabe. Verfügbar ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13

Prototyp

```
int PCube_moveStepExtended( int dev, int cube, float pos,
    unsigned short itime, unsigned long *stateRet, unsigned char
    *dioRet, float *posRet);
```

Beschreibung

Die Funktion wird wie [PCube_moveStep](#) eingesetzt mit dem Unterschied, daß über die Zeiger *stateRet*, *dioRet* und *posRet* ein Kurzstatus, der Zustand der internen digitalen IOs und die aktuelle Position des Moduls zurückgeliefert werden und für eine sofortige Auswertung zur Verfügung stehen (siehe auch Kapitel 7.2.8, [Befehlsbeispiele](#))

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveRampExtended](#), [PCube_moveCurrentExtended](#)

8.8.5.3 PCube_moveVel

Aufgabe

Bewegungskommando im Geschwindigkeitsmodus

Prototyp

```
int PCube_moveVel( int dev, int cube, float vel);
```

Beschreibung

Das Antriebsmodul *cube* wird in den Geschwindigkeitsmodus überführt. Das Modul beschleunigt mit der maximal zulässigen Beschleunigung selbständig auf die neue Geschwindigkeit *vel*. Die angegebene Geschwindigkeit *vel* darf nicht größer als die programmierte maximale Geschwindigkeit sein, ansonsten wird *vel* begrenzt.

Die Geschwindigkeit *vel* wird je nach Modultyp in **m/s** oder **rad/s** angegeben.

Das Modul bewegt sich kontinuierlich mit der angegebenen Geschwindigkeit, bis ein anderes Bewegungskommando oder ein HALT-Kommando gesendet wird oder bis die Softwareendlagen erreicht werden.

Die unter [PCube_moveRamp](#) vorgestellten Informationen über das Fehlschlagen der Funktion, über den erforderlichen Modulstatus und zum Verhalten vor Softwareendlagen gelten auch hier.

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch PCube_move...

8.8.5.3a PCube_moveVelInc

Aufgabe

Bewegungskommando im Geschwindigkeitsmodus

Prototyp

```
int PCube_moveVelInc( int dev, int cube, Int32 vel);
```

Beschreibung

Die Funktion wird wie [PCube_moveVel](#) eingesetzt mit dem Unterschied, daß hier Integer-Werte statt Float-Parameter übergeben werden können. Zur Dimensionierung der Parameter siehe unter [Maßeinheiten der PowerCube™-Module](#).

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveRampInc](#), [PCube_moveStepInc](#), [PCube_moveCurrentInc](#)

8.8.5.4 PCube_moveCurrent

Aufgabe

Bewegungskommando im Strommodus

Prototyp

```
int PCube_moveCurrent( int dev, int cube, float cur);
```

Beschreibung

Das Antriebsmodul *cube* wird in den Strom-Modus überführt.

Das Modul bewegt sich kontinuierlich mit der angegebenen Stromstärke, bis ein anderes Bewegungskommando oder ein HALT-Kommando gesendet wird oder bis die Softwareendlagen erreicht werden.

Dieses Kommando ist insbesondere dann zu verwenden, wenn das Modul durch Fremdeinwirkung oder durch falsche Bewegungskommandos über eine Hardwareendlage hinausgefahren wurde. Eine Möglichkeit, das Modul manuell aus dieser Position zu bewegen, ist der Aufruf von PCube_moveCurrent mit dem Parameter 0, der die Stromzuführung zum Antrieb unterbindet und eine eventuell vorhandene Magnetbremse löst. Andererseits ist es auch möglich, durch Angabe kleiner Stromstärken (1 bis 2 Ampere) softwaregesteuert aus der Endlagenposition herauszufahren (Richtung der Bewegung wird mit Vorzeichen des Stromstärkeparameters beeinflusst). Die Wahl größerer Stromstärken kann bei falscher Bewegungsrichtung das Modul beschädigen!

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Siehe auch PCube_move...

8.8.5.4a PCube_moveCurrentInc

Aufgabe

Bewegungskommando im Strommodus

Prototyp

```
int PCube_moveCurrentInc( int dev, int cube, Int16 cur);
```

Beschreibung

Die Funktion wird wie [PCube_moveCurrent](#) eingesetzt mit dem Unterschied, daß hier Integer-Werte statt Float-Parameter übergeben werden können. Zur Dimensionierung der Parameter siehe unter [Maßeinheiten der PowerCube™-Module](#).

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_moveRampInc](#), [PCube_moveStepInc](#), [PCube_moveVellInc](#)

8.8.5.4b PCube_moveCurrentExtended

Aufgabe

Bewegungskommando im Strommodus mit erweiterter Statusrückgabe. Verfügbar ab PowerCube™-Betriebssystemversion 2.5.16 bzw. 3.5.13

Prototyp

```
int PCube_moveCurrentExtended( int dev, int cube, float cur,
unsigned long *stateRet, unsigned char *dioRet, float *posRet);
```

Beschreibung

Die Funktion wird wie [PCube_moveCurrent](#) eingesetzt mit dem Unterschied, daß über die Zeiger *stateRet*, *dioRet* und *posRet* ein Kurzstatus, der Zustand der internen digitalen IOs und die aktuelle Position des Moduls zurückgeliefert werden und für eine sofortige Auswertung zur Verfügung stehen (siehe auch Kapitel 7.2.8, [Befehlsbeispiele](#))

Rückgabewert

<i>CLD_OK</i>	Bewegung wurde ausgelöst.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '[Vorgehensweise bei Fehlern](#)'.

Siehe auch [PCube_moveRampExtended](#), [PCube_moveStepExtended](#)

8.8.6 Abfragen des aktuellen Modulzustands

8.8.6.1 PCube_getActPos

Aufgabe

Holt die aktuelle Position des Moduls

Prototyp

```
int PCube_getActPos( int dev, int cube, float* pValue );
```

Beschreibung

Die aktuelle Position des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Hinweis: Der zulässige Bereich der Modul erreichbaren Positionen ist mit Hilfe der Funktion [PCube_setMinPos](#) und [PCube_setMaxPos](#) programmierbar.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Aktuelle Position wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Siehe auch [PCube_getActVel](#), [PCube_getDeltaPos](#)

8.8.6.2 PCube_getActVel

Aufgabe

Holt die aktuelle Geschwindigkeit des Moduls

Prototyp

```
PCube_getActVel( int dev, int cube, float* pValue );
```

Beschreibung

Die aktuelle Geschwindigkeit des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Dies ist die aus den Vorgaben der Bewegungssteuerung resultierende Geschwindigkeit des Moduls.

Hinweis: Die zulässige Maximal-Geschwindigkeit des Moduls ist mit Hilfe der Funktion [PCube_setMaxVel](#) programmierbar.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**.

Rückgabewert

<i>CLD_OK</i>	Aktuelle Geschwindigkeit wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getActPos](#)

8.8.6.3 PCube_getDeltaPos

Aufgabe

Holt den aktuellen Schleppfehler des Moduls

Prototyp

```
int PCube_getDeltaPos( int dev, int cube, float* pValue );
```

Beschreibung

Der aktuelle Schleppfehler des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Der Schleppfehler ist die Differenz zwischen der von der Bewegungssteuerung berechneten Soll-Position und der vom Geber des Moduls ermittelten Ist-Position.

Hinweis: Der zulässige maximale Schleppfehler des Moduls ist mit Hilfe der Funktion [PCube_setMaxDeltaPos](#) programmierbar.

Rückgabewert

<i>CLD_OK</i>	Aktueller Schleppfehler wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getActPos](#), [PCube_setMaxDeltaPos](#)

8.8.6.4 PCube_getCur

Aufgabe

Holt die aktuelle Stromaufnahme der Endstufe des Moduls

Prototyp

```
int PCube_getCur( int dev, int cube, float* pValue );
```

Beschreibung

Die aktuelle Stromaufnahme der Endstufe des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Hinweis: Die zulässige maximale Stromaufnahme der Endstufe ist mit Hilfe der Funktion [PCube_setMaxCur](#) programmierbar.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

<i>CLD_OK</i>	Aktuelle Stromaufnahme wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMaxCur](#)

8.8.7 Abfragen und Setzen von Begrenzungswerten

8.8.7.1 PCube_getMaxPos

Aufgabe

Holt die maximale Position des Moduls

Prototyp

```
int PCube_getMaxPos( int dev, int cube, float* pValue );
```

Beschreibung

Die maximale Position des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**

Rückgabewert

<i>CLD_OK</i>	Maximale Position wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Siehe auch [PCube_setMaxPos](#)

8.8.7.2 PCube_getMinPos

Aufgabe

Holt die minimale Position des Moduls

Prototyp

```
int PCube_getMinPos( int dev, int cube, float* pValue );
```

Beschreibung

Die minimale Position des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Minimale Position wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMinPos](#)

8.8.7.3 PCube_setMaxPos

Aufgabe

Setzt die maximale Position des Moduls

Prototyp

```
int PCube_setMaxPos( int dev, int cube, float pValue );
```

Beschreibung

Die maximale Position des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Maximale Position wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMaxPos](#)

8.8.7.4 PCube_setMinPos

Aufgabe

Setzt die minimale Position des Moduls

Prototyp

```
int PCube_setMinPos( int dev, int cube, float pValue );
```

Beschreibung

Die minimale Position des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Minimalwerte begrenzt.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Die minimale Position wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMinPos](#)

8.8.7.5 PCube_getHomeOffset

Aufgabe

Holt die benutzerdefinierte Nullposition des Moduls.

Prototyp

```
int PCube_getHomeOffset( int dev, int cube, float *pValue );
```

Beschreibung

Die benutzerdefinierte, zur Laufzeit änderbare Nullposition des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Nicht zu verwechseln mit `PCube_getDefHomeOffset`; holt die im Modul fest eingetragene benutzerdefinierte Nullposition.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Benutzerdefinierte Nullposition wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setHomeOffset](#)

8.8.7.6 PCube_setHomeOffset

Aufgabe

Setzt die benutzerdefinierte Nullposition des Moduls.

Prototyp

```
int PCube_setHomeOffset( int dev, int cube, float pValue );
```

Beschreibung

Die benutzerdefinierte Nullposition des Antriebsmoduls *cube* wird zur Laufzeit in *pValue* übergeben. Der neue Wert für das HomeOffset ist nur solange gültig, wie die Logik-Stromversorgung aufrecht erhalten wird.

Die Einheit der Position ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Benutzerdefinierte Nullposition wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Siehe auch [PCube_getHomeOffset](#)

8.8.7.7 PCube_getMaxVel

Aufgabe

Holt die maximale Geschwindigkeit des Moduls

Prototyp

```
int PCube_getMaxVel( int dev, int cube, float* pValue );
```

Beschreibung

Die maximale Geschwindigkeit des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**

Rückgabewert

<i>CLD_OK</i>	Maximale Geschwindigkeit wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMaxVel](#)

8.8.7.8 PCube_getMinVel

Aufgabe

Holt die minimale Geschwindigkeit des Moduls

Prototyp

```
int PCube_getMinVel( int dev, int cube, float* pValue );
```

Beschreibung

Die minimale Geschwindigkeit des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**.

Rückgabewert

<i>CLD_OK</i>	Minimale Geschwindigkeit wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMinPos](#)

8.8.7.9 PCube_setMaxVel

Aufgabe

Setzt die maximale Geschwindigkeit des Moduls

Prototyp

```
int PCube_setMaxVel ( int dev, int cube, float pValue );
```

Beschreibung

Die maximale Geschwindigkeit des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung! Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**.

Rückgabewert

<i>CLD_OK</i>	Maximale Geschwindigkeit wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMaxVel](#)

8.8.7.10 PCube_setMinVel

Aufgabe

Setzt die minimale Geschwindigkeit des Moduls

Prototyp

```
int PCube_setMinVel( int dev, int cube, float pValue );
```

Beschreibung

Die minimale Geschwindigkeit des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Minimalwerte begrenzt.

Achtung! Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Geschwindigkeit ist je nach Modultyp **m/s** oder **rad/s**.

Rückgabewert

<i>CLD_OK</i>	Minimale Geschwindigkeit wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMinVel](#)

8.8.7.11 PCube_getMaxAcc

Aufgabe

Holt die maximale Beschleunigung des Moduls

Prototyp

```
int PCube_getMaxAcc( int dev, int cube, float* pValue );
```

Beschreibung

Die maximale Beschleunigung des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Beschleunigung ist je nach Modultyp **m/s²** oder **rad/s²**

Rückgabewert

<i>CLD_OK</i>	Maximale Beschleunigung wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

Siehe auch [PCube_setMaxAcc](#)

8.8.7.12 PCube_setMaxAcc

Aufgabe

Setzt die maximale Beschleunigung des Moduls

Prototyp

```
int PCube_setMaxAcc( int dev, int cube, float pValue );
```

Beschreibung

Die maximale Beschleunigung des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung! Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Beschleunigung ist je nach Modultyp **m/s²** oder **rad/s²**.

Rückgabewert

<i>CLD_OK</i>	Maximale Beschleunigung wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMaxAcc](#)

8.8.7.13 PCube_getMinAcc

Aufgabe

Holt die minimale Beschleunigung des Moduls

Prototyp

```
int PCube_getMinAcc( int dev, int cube, float pValue );
```

Beschreibung

Die minimale Beschleunigung des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Beschleunigung ist je nach Modultyp **m/s²** oder **rad/s²**.

Rückgabewert

<i>CLD_OK</i>	Minimale Beschleunigung wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMinAcc](#)

8.8.7.14 PCube_setMinAcc

Aufgabe

Setzt die minimale Beschleunigung des Moduls

Prototyp

```
int PCube_setMinAcc( int dev, int cube, float pValue );
```

Beschreibung

Die minimale Beschleunigung des Antriebsmoduls *cube* wird in *pValue* übergeben. Alle weiteren Bewegungskommandos werden auf die angegebenen Maximalwerte begrenzt.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Beschleunigung ist je nach Modultyp **m/s²** oder **rad/s²**.

Rückgabewert

<i>CLD_OK</i>	Minimale Beschleunigung wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMinAcc](#)

8.8.7.15 PCube_getMaxCur

Aufgabe

Holt die maximal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
int PCube_getMaxCur( int dev, int cube, float* pValue );
```

Beschreibung

Die maximal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Stromaufnahme ist Ampere (A).

Rückgabewert

<i>CLD_OK</i>	Maximale Stromaufnahme wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMaxCur](#)

8.8.7.16 PCube_setMaxCur

Aufgabe

Setzt die maximal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
int PCube_setMaxCur( int dev, int cube, float* pValue );
```

Beschreibung

Die maximal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *cube* wird in *value* übergeben.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

<i>CLD_OK</i>	Maximale Stromaufnahme wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMaxCur](#)

8.8.7.17 PCube_getMinCur

Aufgabe

Holt die minimal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
int PCube_getMinCur( int dev, int cube, float* pValue );
```

Beschreibung

Die minimal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit der Stromaufnahme ist Ampere (A).

Rückgabewert

<i>CLD_OK</i>	Minimale Stromaufnahme wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMinCur](#)

8.8.7.18 PCube_setMinCur

Aufgabe

Setzt die minimal zulässige Stromaufnahme der Endstufe des Moduls

Prototyp

```
int PCube_setMinCur( int dev, int cube, float* pValue );
```

Beschreibung

Die minimal zulässige Stromaufnahme der Endstufe des Antriebsmoduls *cube* wird in *value* übergeben.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit der Stromaufnahme ist Ampere (**A**).

Rückgabewert

<i>CLD_OK</i>	Minimale Stromaufnahme wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMinCur](#)

8.8.7.19 PCube_getMaxDeltaPos

Aufgabe

Holt den maximal zulässigen Schleppfehler des Moduls

Prototyp

```
int PCube_getMaxDeltaPos( int dev, int cube, float* pValue );
```

Beschreibung

Der maximal zulässige Schleppfehler des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben.

Die Einheit des Schleppfehlers ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Maximaler Schleppfehler wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_setMaxDeltaPos](#)

8.8.7.20 PCube_setMaxDeltaPos

Aufgabe

Setzt den maximal zulässigen Schleppfehler des Moduls

Prototyp

```
int PCube_setMaxDeltaPos( int dev, int cube, float* pValue );
```

Beschreibung

Der maximal zulässige Schleppfehler des Antriebsmoduls *cube* wird in *value* übergeben.

Achtung!

Die angegebenen Werte dürfen die im Modul voreingestellten Werte nicht überschreiten. Höhere Werte werden auf die voreingestellten Werte begrenzt.

Die Einheit des Schleppfehlers ist je nach Modultyp **m** oder **rad**.

Rückgabewert

<i>CLD_OK</i>	Maximaler Schleppfehler wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Siehe auch [PCube_getMaxDeltaPos](#)

8.8.8 Abfragen und Setzen der Koeffizienten des Lageregelkreises

Diese Funktionen stehen erst ab der Version 2.5.1 zur Verfügung

8.8.8.1 PCube_getCO

Aufgabe

Holt den Koeffizienten CO des Lageregelkreises

Prototyp

```
int PCube_getCO( int dev, int cube, Int16* pValue );
```

Beschreibung

Der Koeffizient CO des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Dieser Wert stellt einen Verstärkungsfaktor dar. Je größer CO gesetzt ist, desto geringer ist die Verstärkung. Gültige Werte sind alle geraden Zahlen zwischen 12 und 48.

Rückgabewert

<i>CLD_OK</i>	Koeffizient CO wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.8.2 PCube_getDamp

Aufgabe

Holt den Dämpfungswert des Lageregelkreises

Prototyp

```
int PCube_getDamp( int dev, int cube, Int16* pValue );
```

Beschreibung

Die Dämpfung des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Dieser Wert stellt einen Dämpfungsfaktor dar. Je größer Damp gesetzt ist, desto größer ist die Dämpfung. Gültige Werte liegen zwischen 1 und 4.

Rückgabewert

<i>CLD_OK</i>	Dämpfungswert Damp wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

8.8.8.3 PCube_getA0

Aufgabe

Holt den Koeffizienten A0 des Lageregelkreises

Prototyp

```
int PCube_getA0( int dev, int cube, Int16* pValue );
```

Beschreibung

Der Koeffizient A0 des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Hier wird die Beschleunigungsfähigkeit des Antriebs angezeigt. Je größer A0 gesetzt ist, desto größere Beschleunigungsfähigkeit wird unterstellt. Gültige Werte liegen zwischen 1 und 12.

Rückgabewert

<i>CLD_OK</i>	Koeffizient A0 wurde geholt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

8.8.8.4 PCube_setC0

Aufgabe

Setzt den Koeffizienten C0 des Lageregelkreises

Prototyp

```
int PCube_setC0( int dev, int cube, Int16 pValue );
```

Beschreibung

Der Koeffizient C0 des Antriebsmoduls *cube* wird in *value* übergeben. Dieser Wert stellt einen Verstärkungsfaktor dar. Je größer C0 gesetzt wird, desto geringer wird die Verstärkung. Gültige Werte sind alle geraden Zahlen zwischen 12 und 48. Nach Setzen des oder der Koeffizienten muß [PCube_recalcPIDParams](#) aufgerufen werden, damit die Änderungen wirksam werden.

Rückgabewert

<i>CLD_OK</i>	Koeffizient C0 wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.8.5 PCube_setDamp

Aufgabe

Setzt den Dämpfungswert des Lageregelkreises

Prototyp

```
int PCube_setDamp( int dev, int cube, Int16 pValue );
```

Beschreibung

Die Dämpfung des Antriebsmoduls *cube* wird in *value* übergeben. Dieser Wert stellt einen Dämpfungsfaktor dar. Je größer Damp gesetzt wird, desto größer wird die Dämpfung. Gültige Werte liegen zwischen 1 und 4. Nach Setzen des oder der Koeffizienten muß `PCube_recalcPIDParams` aufgerufen werden, damit die Änderungen wirksam werden.

Rückgabewert

<i>CLD_OK</i>	Dämpfungswert Damp wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

8.8.8.6 PCube_setA0

Aufgabe

Setzt den Koeffizienten AO des Lageregelkreises

Prototyp

```
int PCube_setA0( int dev, int cube, Int16 pValue );
```

Beschreibung

Der Koeffizient AO des Antriebsmoduls *cube* wird über den Zeiger *pValue* übergeben. Hier wird die Beschleunigungsfähigkeit des Antriebs angesetzt. Je größer AO gesetzt wird, desto größere Beschleunigungsfähigkeit wird unterstellt. Gültige Werte liegen zwischen 1 und 12. Nach Setzen des oder der Koeffizienten muß [PCube_recalcPIDParams](#) aufgerufen werden, damit die Änderungen wirksam werden.

Rückgabewert

<i>CLD_OK</i>	Koeffizient AO wurde gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.8.7 PCube_recalcPIDParams

Aufgabe

Neuberechnen des Lageregelkreises

Prototyp

```
int PCube_recalcPIDParams( int dev, int cube );
```

Beschreibung

Die Neuberechnung des Lageregelkreises des Antriebsmoduls *cube* wird durchgeführt. Nach Setzen der Reglerparameter mittels [PCube_setCO](#), [PCube_setDamp](#) oder [PCube_setAO](#) muß diese Funktion aufgerufen werden, damit die Änderungen wirksam werden.

Rückgabewert

<i>CLD_OK</i>	Neuberechnung der Regelschleife wird durchgeführt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe 'Vorgehensweise bei Fehlern'.

8.8.9 Abfragen und Setzen der internen Digital-Ein- und Ausgänge

8.8.9.1 PCube_getDioData

Aufgabe

Holt die aktuell gesetzten Bits der internen Digital-Ein- und Ausgänge. Das Bit-Register wird über die Define-Konstante ACT_DIOSETUP angesprochen. Dies ist ein 32-Bit-Wort, das in der Variablen, auf die der Zeiger *pValue* zeigt, abgelegt wird. Nur die ersten 8 Bit in diesem Register haben eine bestimmte Bedeutung. Die Bits 1 - 4 repräsentieren die Inbits (Eingänge), die Bits 5 - 8 repräsentieren die Outbits (Ausgänge).

Prototyp

```
PCube_getDioData(int dev, int cube, UInt32 *bits)
```

Beschreibung

Die aktuell gesetzten Bits der Ein- und Ausgänge der internen DIOs des Moduls *cube* werden über den Zeiger *bits* übergeben.

Rückgabewert

<i>CLD_OK</i>	Bitregister der modulinternen DIOs wurde abgefordert.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

8.8.9.2 PCube_setDioData

Aufgabe

Setzt die Bits der internen Digital-Ein- und Ausgänge.

Prototyp

```
PCube_setDioData(int dev, int cube, UInt32 bits)
```

Beschreibung

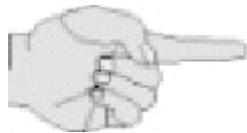
Die Bits der Ein- und Ausgänge der internen DIOs des Moduls *cube* werden in *bits* übergeben.

Rückgabewert

<i>CLD_OK</i>	Bitregister der modulinternen DIOs wurde neu gesetzt.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

Bei anderen Rückgabewerten siehe '*Vorgehensweise bei Fehlern*'.

Beispiel für das Lesen und Setzen der Bits interner DIOs:



```
int ival=0;
UInt32 ivalold;
char buffer[33];
retVal = PCube_getDioData( dev,idMap[module],&ivalold);
printf( "... Set Bit Nr (0-7): " );
scanf( "%d", &ival);
ivalold=ivalold|(UInt32)pow(2,ival);
retVal = PCube_setDioData( dev,idMap[module],ivalold);
retVal = PCube_getDioData( dev,idMap[module],&ivalold);
printf("\nNew Settings:%s\n",_itoa( ivalold, buffer, 2 ));
```

8.8.10 Abfrage von Vorgabewerten und sonstigen Funktionen

Die im vorhergehenden Abschnitt dargestellten Funktionen dienen zum Abfragen und Setzen von Vorgabewerten in den Antriebsmodulen, deren Überschreitung in der Regel einen Fehlerzustand bewirkt. In diesem Fall geht, wie beschrieben, das Modul in den Sicherheitszustand über. Die im folgenden genannten Funktionen PCube_getDefXXX dienen zum Abfragen der im Modul voreingestellten Maximalwerte, die nicht überschritten werden dürfen.

Wenn in den im vorhergehenden Abschnitt gezeigten Funktionen höhere als die voreingestellten Werte programmiert werden sollen, werden die neuen Werte in jedem Fall auf die voreingestellten begrenzt.

Auf eine ausführliche Beschreibung der Funktionen wird hier verzichtet, da die Parameter denen der im vorherigen Abschnitt beschriebenen Funktionen entsprechen.

Folgende Funktionen stehen zur Verfügung:

- `int PCube_getDefHomeOffset(int dev, int cube, float* pValue);`
- `int PCube_getDefGearRatio(int dev, int cube, float* pValue);`
- `int PCube_getDefLinearRatio(int dev, int cube, float* pValue);`
- `int PCube_getDefMinPos(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxPos(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxDeltaPos(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxDeltaVel(int dev, int cube, float* pValue);`
- `int PCube_getDefTorqueRatio(int dev, int cube, float* pValue);`
- `int PCube_getDefCurRatio(int dev, int cube, float* pValue);`
- `int PCube_getDefMinVel(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxVel(int dev, int cube, float* pValue);`
- `int PCube_getDefMinAcc(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxAcc(int dev, int cube, float* pValue);`
- `int PCube_getDefMinCur(int dev, int cube, float* pValue);`
- `int PCube_getDefMaxCur(int dev, int cube, float* pValue);`
- `int PCube_getDefHomeVel(int dev, int cube, float* pValue);`
- `int PCube_getDefHomeAcc(int dev, int cube, float* pValue);`
- `int PCube_getDefKpPosControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTnPosControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTvPosControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTlPosControl(int dev, int cube, float* pValue);`
- `int PCube_getDefKpVelControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTnVelControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTvVelControl(int dev, int cube, float* pValue);`
- `int PCube_getDefTlVelControl(int dev, int cube, float* pValue);`

- `int PCube_getDefDioSetup(int dev, int cube, UInt32* pValue);`
- `int PCube_getDefSerialNo(int dev, int cube, UInt32* pValue);`
- `int PCube_getDefConfig(int dev, int cube, UInt32* pValue);`
- `int PCube_getDefPulsesPerTurn(int dev, int cube, UInt32* pValue);`
- `int PCube_getDefCubeVersion(int dev, int cube, UInt16* pValue);`
- `int PCube_getDefServiceInterval(int dev, int cube, UInt16* pValue);`
- `int PCube_getDefBrakeTimeOut(int dev, int cube, UInt16* pValue);`
- `int PCube_getDefAddress(int dev, int cube, UInt8* pValue);`
- `int PCube_getDefMoveMode(int dev, int cube, UInt8* pValue);`
- `int PCube_getDefPrimaryBaudrate(int dev, int cube, UInt8* pValue);`
- `int PCube_getDefSecondaryBaudrate(int dev, int cube, UInt8* pValue);`
- `int PCube_getDefC0(int dev, int cube, Int16* pValue);`
- `int PCube_getDefDamp(int dev, int cube, Int16* pValue);`
- `int PCube_getDefA0(int dev, int cube, UInt16* pValue);`

8.8.11 Die I/O-Schnittstellen

8.8.11.1 EMS Thomas Wünsche I/O-Module

Die in der m5dll enthaltenen Funktionen zum Betreiben von I/O-Modulen der Firma EMS Thomas Wünsche werden im folgenden beschrieben. Es werden zur Zeit zwei Modultypen unterstützt: 0=CST-DO8-24V und 1=CST-DI8-24V.

8.8.11.1.1 TW_initModules

Aufgabe

Diese Funktion initialisiert ein Modul und kann vor Benutzung der weiteren Kommandos aufgerufen werden. Es ist zu beachten, daß diese Funktion nur verwendet werden darf, wenn sich lediglich ein TW-Modul am Bus befindet. Die Synchronisation mehrerer TW-Module wird durch Senden von 100 mit Zufallswerten initialisierten CAN-Telegrammen bei Aufruf der Funktion `PCube_openDevice` vorgenommen. Beim anschließenden Aufruf der Funktion `TW_initModules` geht diese Synchronisation wieder verloren.

Prototyp

```
int TW_initModules( int device );
```

Beschreibung

Rufen Sie diese Funktion nach Spannungsabfall in der Energieversorgung auf. Das Modul wird innerhalb dieser Funktion auf die eingestellte Baudrate vorbereitet.

Der Parameter *device* bezeichnet die Schnittstelle, über die das Modul angesteuert wird.

Rückgabewert

<i>CLD_OK</i>	Modul wurde erkannt.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.
<i>Anderer Wert</i>	Es wurde kein Modul am Bus gefunden.

8.8.11.1.2 TW_addModule

Aufgabe

Diese Funktion fügt ein Modul bestimmten Typs mit definierter Seriennummer der Laufzeitumgebung hinzu. Nach erfolgreichem Hinzufügen kann auf die I/O-Ports dieses Moduls zugegriffen werden.

Prototyp

```
int TW_addModule( int device, UInt8 typ, UInt32 serialnum-
  ber, int *index );
```

Beschreibung

Der Parameter *device* bezeichnet die Schnittstelle, über die die Module angesteuert werden. Die anzugebende Seriennummer ist diejenige, die auf dem Modul aufgedruckt ist.

Die Werte für den Parameter *typ* können Sie der m5.h-Datei entnehmen:

Typ	Konstante	Wert
CST-Digital-Ausgangsmodul mit 8 Ausgängen 24V	TW_DIG24V_OUT	0
CST-Digital-Eingangsmodul mit 8 Eingängen 24V	TW_DIG24V_IN	1
CST-Analog-Ausgangsmodul mit 2 Ausgängen 0-10V/12Bit	TW_ANA10V_OUT	2
CST-Analog-Eingangsmodul mit 4 Eingängen 0-10V/12Bit	TW_ANA10V_IN	3
CST-Analog-Ausgangsmodul mit 2 Ausgängen 0-25mA/12Bit	TW_ANA25MA_OUT	4
CST-Analog-Eingangsmodul mit 4 Eingängen 0-25mA/12Bit	TW_ANA25MA_IN	5

Rückgabewert

<i>CLD_OK</i>	Module wurden erkannt.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.
<i>Anderer Wert</i>	Modul wurde nicht korrekt hinzugefügt.

8.8.11.1.3 TW_getIO

Aufgabe

Diese Funktion fragt den Zustand eines I/O-Ports ab.

Prototyp

```
int TW_getIO( int device, int index, int port, Float *state
);
```

Beschreibung

Der Parameter *device* bezeichnet die Schnittstelle, über die die Module angesteuert werden. Die angegebene Seriennummer bestimmt das Modul, welches zuvor mit TW_addModule() hinzugefügt wurde.

Der Parameter *port* selektiert den Eingangs- oder Ausgangsport (0-7 bei digitalen I/O's, 0-1/4 bei analogen I/O's), dessen Zustand gelesen werden soll. Der Zustand des Moduls kann unabhängig vom Typ des Moduls gelesen werden.

In der Variablen *state* wird anschließend der Zustand des Ports geliefert, wobei 0 nicht aktiv und 1 aktiv ist (bei digitalen I/O's). Analoge Module liefern den abgetasteten Spannungswert (0-10V) bzw. den abgetasteten Stromwert (0-25mA).

Rückgabewert

<i>CLD_OK</i>	Zustand wurde gelesen.
<i>CLDERR_BADPARAM</i>	Zustand wurde nicht gesetzt, da falsche Parameter übergeben wurden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.11.1.4 TW_setIO

Aufgabe

Diese Funktion setzt den Zustand eines I/O-Ports.

Prototyp

```
int TW_setIO( int device, int index, int port, Float state );
```

Beschreibung

Der Parameter *device* bezeichnet die Schnittstelle, über die die Module angesteuert werden. Der angegebene Index bestimmt das Modul, welches zuvor mit `TW_addModule()` hinzugefügt wurde. Der Parameter *port* selektiert den Eingangs- oder Ausgangsport (0-7 bei digitalen I/O's, 0-1 bei analogen I/O's), dessen Zustand gesetzt werden soll. Der Zustand des Moduls kann unabhängig vom Typ des Moduls gesetzt werden, wobei allerdings nur bei Output-Modulen wirklich eine Änderung des Schaltvorgangs erfolgt.

In der Variablen *state* wird anschließend der Zustand des Ports gesetzt, wobei 0 nicht aktiv und 1 aktiv ist (bei digitalen I/O's). Analoge Module werden mit einem Spannungswert (0-10V) bzw. einen Stromwert (0-25mA) bedient.

Rückgabewert

<i>CLD_OK</i>	Zustand wurde gesetzt.
<i>CLDERR_BADPARAM</i>	Zustand wurde nicht gesetzt, da falsche Parameter übergeben wurden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.11.1.5 TW_getDigIOall

Aufgabe

Diese Funktion liest den Zustand aller Channels eines digitalen TW-Moduls.

Prototyp

```
int TW_getDigIOall( int device, int index, UInt8 *value );
```

Beschreibung

Der Parameter *device* bezeichnet die Schnittstelle, über die die Module angesteuert werden.

Die angegebene Seriennummer bestimmt das Modul, welches zuvor mit TW_addModule() hinzugefügt wurde.

In der Variablen *value* wird der Zustand der Channels geliefert, wobei 0 nicht aktiv und 1 aktiv ist.

Rückgabewert

<i>CLD_OK</i>	Zustand wurde gelesen.
<i>CLDERR_BADPARAM</i>	Zustand wurde nicht gesetzt, da falsche Parameter übergeben wurden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.11.1.6 TW_setDigIOall

Aufgabe

Diese Funktion setzt alle Channels eines digitalen TW-Moduls.

Prototyp

```
int TW_setDigIOall( int device, int index, UInt8 value );
```

Beschreibung

Der Parameter *device* bezeichnet die Schnittstelle, über die die Module angesteuert werden.

Die angegebene Seriennummer bestimmt das Modul, welches zuvor mit TW_addModule() hinzugefügt wurde.

Über die Variable *value* wird der Zustand aller Channels gesetzt, wobei 0 nicht aktiv und 1 aktiv ist.

Rückgabewert

<i>CLD_OK</i>	Zustand wurde gelesen.
<i>CLDERR_BADPARAM</i>	Zustand wurde nicht gesetzt, da falsche Parameter übergeben wurden.
<i>CLDERR_TRANSMISSIONERROR</i>	Übertragungsfehler bei der Kommunikation
<i>CLDERR_DRIVE_MODULETIMEOUT</i>	Antriebsmodul hat nicht reagiert.
<i>CLERR_WRONGHANDLE</i>	Verwendeter Zeiger auf Schnittstelle nicht initialisiert oder bereits gelöscht.

8.8.12 Besonderheiten bei speziellen Implementierungen

8.8.12.1 Die OR-PC5 CAN Schnittstelle

Diese Schnittstelle kann zusammen mit dem unixbasierten Echtzeitbetriebssystem QNX verwendet werden.

In der Implementierung des CAN-Treibers für die or Industrial Computers PC-5 Serie sind einige Besonderheiten zu beachten.

So besteht der Treiber aus zwei Teilen:

1. Den CAN-Server, der die Kommunikation mit dem CAN-Bus übernimmt.
2. Dem Softwaretreiber, der als lib vorliegt und zur Applikation hinzugelinkt werden muß. Dieser übernimmt die Kommunikation mit dem CAN-Server.

Der CAN-Serverprozeß *m5candr* muß gestartet sein, bevor ein Client gestartet werden soll. Dies ist sichergestellt, da der Clientprozeß anderenfalls eine Fehlermeldung produziert.

Des weiteren muß der Queuehandler von QNX gestartet sein. Dies kann in einem Startupscript automatisiert werden.

Der Handler lautet *Mqueue* und ist mittels *Mqueue&* zu starten.

Im Lieferumfang der Treiberdistribution befindet sich ein Beispielprogramm im Sourcecode und ein dazugehöriges Makefile, welches eine lauffähige Version des Programms *m5apitst* erzeugt.

Dieses ist mit folgender Kommandozeile zu starten:

```
./m5apitst ORCAN:1
```

Anschließend wird ein etwa 30 s andauernder Scan der angeschlossenen Module durchgeführt. Dies sollten PowerCube™-M5-Module sein, die möglichst nicht im M3-Modus betrieben werden.

Die Treiber sind mit root-Berechtigung zu starten.

8.8.13 Beispiel-Implementierungen in unterschiedliche Arbeitsumgebungen

8.8.13.1 MS Visual C++

Um die Funktionstüchtigkeit und Bewegungsmodi der PowerCube™-Module testen zu können, wurde das Programm *PCubeDemo* geschrieben. Das Testprogramm wurde in einer MS Visual C++ Umgebung, Version 6 als dialogbasierte MFC-Anwendung entworfen. Im Folgenden werden die Projektplanung und -erstellung ausführlich beschrieben: [Das Testprogramm PCubeDemo](#).

9. PowerCube™ Demonstrations- und Testprogramm (PCubeDemo.exe)

Diese Anleitung zeigt Ihnen an Hand eines konkreten Projektes, wie Sie unter Visual C++ 6.0 eine Anwendung entwickeln. Diese Anwendung greift auf die von der AMTEC GmbH herausgegebenen Bibliotheken zurück.

Was benötigen Sie um ein Projekt zu erstellen:

- Eine Entwicklungsumgebung, z.B. Visual C++ V6.0
- Die Bibliotheksdateien für die PowerCube™-Module: m5apiw32.dll, m5.h, m5apiw32.lib (*die beiden letzten Dateien werden mit Add Files to Project eingebunden*)
- Eine eingerichtete Schnittstellenkarte (z.B. IXXAT PC-CAN iPC-I320 V1.12, [siehe](#))
- Ein oder mehrere PowerCube™-Module (inklusive Netzteil und Kabel)

9.1 Typographie

Zur Hervorhebung und übersichtlicheren Darstellung werden die folgenden Schriftarten verwendet.

<code>reserved</code>	reservierte Wörter, Namen von Funktionen etc.
<code>listing</code>	Programmlistings
<code>parameter</code>	Parameter und Rückgabewerte

9.2 Anforderungen an das Anwendungsprogramm

Das zu realisierende Projekt soll ein Test- und Demonstrationprogrammes für die PowerCube™-Module sein. Das Programm soll nach dem Start den Hauptdialog öffnen und alle angeschlossenen PowerCube™-Module auflisten.

Der Hauptdialog soll über eine Menüleiste mit den folgenden Einträgen verfügen:

- **Exit:** Zum Verlassen des Programmes
- **Device:** Rescan, Initstring

Außerdem soll der Hauptdialog über eine Statusleiste verfügen, in der Fehler- Statusmeldungen des Programms zur Anzeige gebracht werden:

Im Hauptdialog selbst sollen folgende Punkte realisiert werden:

- Listenelement zur Anzeige der angeschlossenen Module mit ID, Position, Geschwindigkeit, Strom, Status
- Halt-Button

Ein rechter Mausklick auf ein Modul (im Listenelement) soll ein kontextsensitives Popup-Menü öffnen, um die folgenden Dinge mit **den zuvor ausgewählten Modulen** zu tun:

- Initialisieren (Homing)
- Reset
- Halt
- Stromwert auf 0 setzen

Ein Doppelklick auf ein Modul soll einen weiteren **nichtmodalen** Dialog öffnen, der die folgenden Punkte realisiert:

- Auswahlbox zur Auswahl der Bewegungsart (Ramp, Velocity, Current). Eine Änderung in dieser Auswahlbox soll die Anzeige aktualisieren.
- Einen RESET-/INIT-Button. (Zurücksetzen oder Referenzfahrt eines Moduls)
- Slider zum Einstellen der Bewegung. Die anzuzeigenden Bereichsgrenzen seien die Modulparameter maxpos/minpos, maxvel/minvel, maxcur/mincur.
- Parallel zum Slider soll es ein Eingabefeld geben, um die Daten über Tastatur eingeben zu können.
- Slider und Eingabefeld sind zu synchronisieren.
- Einen GO-/HALT-Button.
- Zwei Eingabefelder für Beschleunigung und Geschwindigkeit. Parallel dazu jeweils einen Schieberegler. Schieberegler und Eingabefeld sind zu synchronisieren.

Die Anwendung ist als dialogbasierte MFC-Applikation zu entwickeln. Sie soll die folgenden Anforderungen erfüllen:

- Hauptdialog bestehend aus:
- Listenelement zur Anzeige der folgenden Parameter
 - ID / Modul-Typ / Version / Seriennummer
 - Status

Es soll Threadprogrammierung zum Einsatz gebracht werden, um eine optimale Rechnerauslastung zu gewährleisten.

Alle Moduldaten (ROM-Defaultparameter und Modulparameter) sollen in einem globalen Array (bzw. in einer Klasse **CModData** gekapselt) abgelegt werden (Current, Velocity, Position, State usw.). Die Modulparameter werden von einem Thread geholt. Als Ende-Signal für den Thread soll eine Variable verwendet werden. Das Starten des Threads und die Darstellung der Daten im Listenelement erfolgt in einem WM-Timer-Event (ca. alle 100ms).

Es werden aber nicht nur Moduldaten in dem Array abgelegt, sondern auch Zustände des Anwendungsprogrammes. Zu diesen Informationen gehören:

```

Bool m_bModulSelected; ///true = Modul selected in Listcontrol
Bool m_bDialogOpen; ///true = Dialog for Modul is open
Bool m_bModulDeleted; ///True = Modul dleted (for example at Timeout)
UInt32 m_intCLD_DRIVE_errL; ///Errorcounter for CLD_DRIVE_XXX errors
UInt32 m_intCLD_DRIVE_errH; ///Errorcounter for CLD_DRIVE_XXX errors
UInt32 m_intCLDERR; ///Errorcounter for CLDERR_XXX errors
UInt32 m_intCLERR_L; ///Errorcounter for CLERR_XXX errors
UInt32 m_intCLERR_H; ///Errorcounter for CLERR_XXX errors

```

9.3 Konventionen

Bei Verwendung von Standarddatentypen sind die hier definierten Datentypen zu verwenden. Diese Datentypen stehen Ihnen automatisch zur Verfügung, wenn Sie die Datei m5.h inkludiert haben.

```

typedef void Void;
typedef char Char;
typedef unsigned char Byte;
typedef unsigned char UInt8;
typedef signed short int Int16;
typedef unsigned short int UInt16;
typedef long int Int32;

```

```
typedef unsigned long UInt32;  
typedef float Float;  
typedef char Bool;  
typedef UInt16 CLDRet;  
typedef Int16 CLDID;  
typedef CLDID CLDDriveID;  
typedef CLDID CLDXtrnlID;  
typedef unsigned long Systime;
```

9.4 Anlegen eines Projektes

- Öffnen Sie Visual C++ V6.0
- Klicken Sie in der Menüleiste auf *File->new...* Es öffnet sich ein neues Fenster.
- Klicken Sie die Karteikarte *Projects*.
- Füllen Sie den Punkt *Project name* (z.B. PCubeDemo) und *Location* aus. Wenn Sie als Projektname PCubeDemo verwenden können Sie ganz einfach den mitgelieferten Source-Code in diesen Pfad kopieren und haben eine lauffähige Beispiel-Entwicklungsumgebung.
- Klicken Sie die in der Karteikarte Projects auf MFC AppWizard (exe).
- Klicken Sie auf *OK*.

Es öffnet sich ein neues Fenster.

MFC AppWizard Step 1

- Klicken Sie auf den Radio Button *Dialog based*
- Als Sprache wählen Sie z.B. English
- Klicken Sie auf den Button *Next*

Es öffnet sich ein neues Fenster.

MFC AppWizard Step 2 of 4

- Klicken Sie auf alle Checkboxes, außer *Automation* und *Windows sockets*.
- Geben Sie im unteren Eingabefeld einen Titel für den Hauptdialog ein.
- Klicken Sie auf den Button *Next*

Es öffnet sich ein neues Fenster.

MFC AppWizard Step 3 of 4

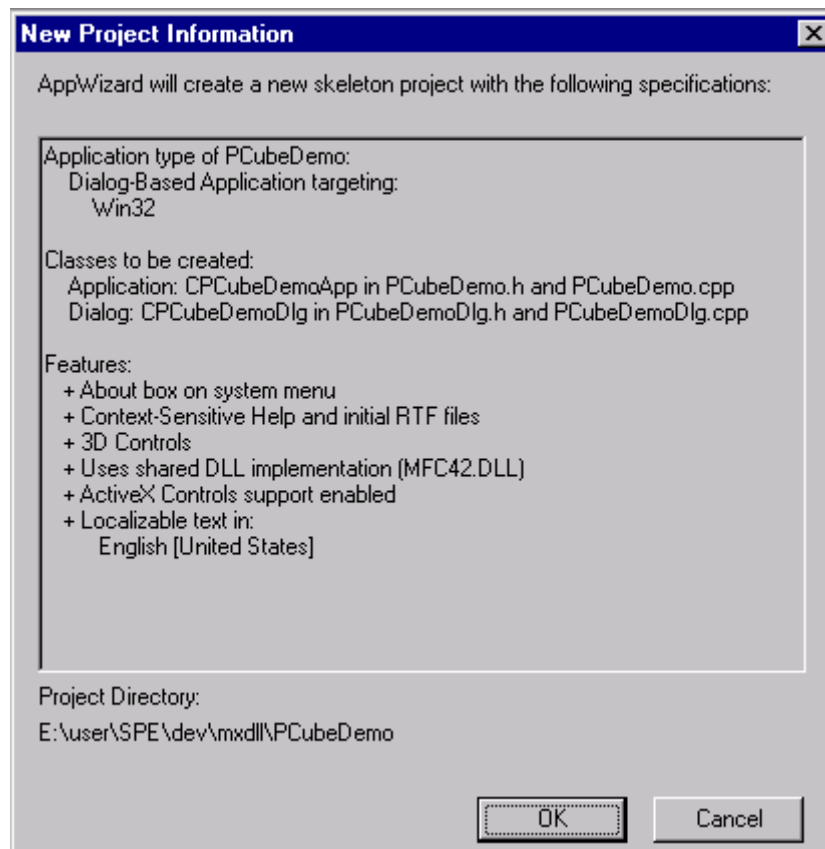
- Hier ist bereits der Radio Button *MFC Standard* ausgewählt und kann auch nicht geändert werden.
- Lassen Sie sich Kommentar generieren
- Bei der Frage nach der Art der Verwendung der MFC-Dll wählen Sie den Punkt *As shared DLL*
- Klicken Sie auf den Button *Next*

Es öffnet sich ein neues Fenster.

MFC AppWizard Step 4 of 4

In diesem Fenster behalten Sie alle Vorgaben bei und nehmen keine Änderungen vor.
Klicken Sie auf den Button *Finish*

Sie erhalten nun ein neues Fenster in dem Ihnen alle Einstellungen dargestellt werden.



Klicken Sie nach Überprüfung der Projektinformation den Button **OK** und Ihr Projekt wird angelegt.

Sie können nun bereits Ihr Projekt kompilieren. Allerdings haben Sie noch nicht viel getan. Sie sollten jetzt die Bibliotheksdateien in Ihrem Projekt aufnehmen. Kopieren Sie also die Dateien

- m5apiw32.lib
- m5.h
- m5apiw32.dll

in Ihren Projektpfad. Nun können Sie einfach den mitgelieferten Source-Code in Ihren neu erstellten Projektpfad kopieren und haben so eine lauffähige Beispiel-Entwicklungsumgebung. Sie sollten anschließend Ihr Projekt neu öffnen (durch Doppelklick auf die Datei PCubeDemo.dsw)

Anschließend

- klicken Sie in Visual C++ im Workspace-Fenster auf die Karteikarte **FileView**.
- Klicken Sie mit der rechten Maustaste auf **Header Files**.
- In dem sich öffnenden Kontextmenü wählen Sie den Punkt **Add Files to Folder**.
- Nun öffnet sich ein Filedialog indem Sie die Datei m5.h anklicken und mit **OK** bestätigen.
Wiederholen Sie diesen Vorgang für die Datei m5apiw32.lib

9.4.1 Hinzufügen der Menüleiste

- Klicken Sie im Projekt-Fenster auf die Ansicht **Resource View**
- Klicken Sie in der Menüleiste **Insert->Resource...**
- In dem sich öffnenden Fenster klicken Sie den Resource Typ: **Menu**
- Klicken Sie auf den Button **New**
- In Ihrem Projekt-Fenster gibt es nun ein neues Verzeichnis namens Menü und der darunterliegenden Resource **IDR_MENU1**. Das neu erstellte Menu wird auch sofort in einem Fenster im Arbeitsbereich geöffnet.
- Klicken Sie mit der rechten Taste in den Arbeitsbereich des Menü-Fensters.
- Es öffnet sich ein Kontextmenü in dem Sie auf **Properties** klicken.
- Nun öffnet sich das Eigenschaftsfenster. Klicken Sie auf den Nagel, so bleibt das Eigenschaftsfenster so lange geöffnet bis Sie es explizit schließen.
- Ändern Sie den Namen des Menüs von **IDR_MENU1** in **IDR_MAINMENU**. Achten Sie bitte unbedingt auf die Großschreibung.
- Bestätigen Sie Ihre Eingabe mit <Return> (optional, wird automatisch übernommen). Bevor Sie diesen Zustand speichern, müssen Sie mindestens einen Menüpunkt hinzufügen, da sonst das Menü vom Assistenten wieder aus dem Projekt gelöscht wird.

Füllen der Menüleiste

- Klicken Sie im Arbeitsbereich auf die zu erstellenden Menüpunkte.
- Füllen Sie *Caption* und *ID* nach dem folgenden Schema aus.
-

Hauptmenüpunkt	Caption Untermenü	ID Untermenü
Caption: &Exit	.	IDM_EXIT
Caption: &Device	&Rescan	IDM_RESCAN
	&Initstring	IDM_INITSTRING

- Wenn Sie für jeden Untermenüpunkt im Eigenschaftsfenster einen "Prompt" vergeben, so werden die Ressourcen automatisch in die String-Tabelle eingetragen. Wenn Sie anschließend den ClassWizard aufrufen, werden Sie aufgefordert die neue Ressource zuzuordnen. Der ClassWizard schlägt vor die Resource einer bestehenden Klasse zuzuordnen. Wählen Sie die Klasse Ihres Dialoges (hier: **PCubeDemoDlg**).

9.5 Hinzufügen eines Listenelementes

- Klicken Sie im Projektfenster doppelt auf den Dialog CPCubeDemoDlg.
- In der Werkzeugleiste klicken Sie auf das Listcontrol-Element.
- Positionieren die das Steuerelement in Ihrem Dialog.
- Öffnen Sie das Eigenschaftsfenster und tragen die ID: IDC_LISTMODULS ein. Unter *<Style->View* klicken Sie bitte auf Report.
- Im ClassWizard erzeugen Sie eine Controlvariable für dieses Steuerelement. Name: m_cListModuls.
- In der Funktion OnInitDialog() mit Hilfe der Funktion InsertColumn() das Steuerelement initialisieren.
- Dabei ist eine Struktur, die als Memberzeigervariable angelegt ist, sehr hilfreich (Spaltenüberschrift, Ausrichtung, Breite).
- Dieses Array wird im Konstrukt des Dialoges erzeugt, gefüllt in OnInitDialog().
- Die Anzahl der Spalten wird in einer Klassenkonstanten angelegt, die Initialisierung dieser Konstante erfolgt in der Memberinitialisierungsliste des Dialogkonstruktors.
- Das Array wird in OnInitDialog() gefüllt und einzelne Elemente davon an InsertColumn() übergeben.
- Nicht vergessen später das angelegte Zeigerarray wieder zu löschen.

Anlegen eines Kontextmenüs für das Listenelement

- Wie legen eine neue Menüleiste an. Jede Spalte dieser Menüleiste kann als Kontextmenü verwendet werden.
- Erzeugen Sie ein neues Menü und nennen es IDR_MENUKONTEXT.
- Füllen Sie Caption und ID nach dem folgenden Schema aus.

Hauptmenüpunkt	Caption Untermenü	ID Untermenü
Caption: ListModuls	&Init	IDM_INITMODULE
	&Reset	IDM_RESETPMODULE
	&Halt	IDM_HALTPMODULE
	O-&Current	IDM_OCURRENTPMODULE

Mit Hilfe des ClassWizard erzeugen Sie bitte aus der Windowsmessage **WM_CONTEXTMENU** die Memberfunktion **OnContextMenu** (für den Dialog CPCubeDemoDlg).

Diese Funktion hat die Aufgabe Einträge im Modul-Array vorzunehmen bzw. zu ändern. Die eigentliche Aktion findet erst bei einem Klick im Kontextmenü statt.

Innerhalb der erzeugten Funktion OnContextMenu() sollen die folgenden Dinge realisiert werden:

- Vor dem Klick mit der rechten Taste war mehr als ein Eintrag markiert und der Klick findet auf einer bereits markierten Zeile statt.
Der folgende Klick im Kontextmenü soll sich auf alle bereits markierten Module beziehen. Ein besonderer Eintrag oder Änderung im Modul-Array ist nicht erforderlich, da diese bereits beim Klickereignis des List-Controls erfolgt ist.
- Vor dem Klick mit der rechten Taste war mehr als ein Eintrag markiert und der Klick findet auf einer nicht markierten Zeile statt.
Hierbei wird von Microsoft die bestehenden Markierungen aufgehoben und die nichtmarkierte aber geklickte Zeile markiert, ohne das ein Klickereignis im List-Control ausgelöst wird. Somit müssen wir die Änderung (`m_bModulSelected = true`) im Modul-Array vornehmen.
- Vor dem Klick mit der rechten Taste war ein Eintrag (oder kein Eintrag) markiert.
Auch hier wird kein Klickereignis des List-Controls ausgelöst.

9.6 Anlegen eines Statusbars

- Klicken Sie im Projektfenster auf ResourceView und dort doppelt auf die String Table.
- Ein Statusbar läßt sich in mehrere Bereiche einteilen. Jeder Bereich erhält eine ID und mit in der String Table registriert werde.
Fügen Sie die folgenden Zeilen hinzu (Doppelklick auf leere Zeile)

- ID_INDICATOR_DATE
- ID_INDICATOR_TIME

- Klicken Sie in der Menüleiste von Visual C++ auf **View-> Resource Symbols**
- Fügen Sie einen neuen Statusleistenbezeichner ein (Klick auf **New**):
- Name: ID_MY_STATUS_BAR, Value übernehmen (nicht ändern) und mit **OK** bestätigen
- In der CPP-Datei Ihres Dialoges (PCubeDemoDlg.cpp) tragen Sie die folgenden Zeilen oberhalb der ersten Klassendefinition ein

```
static UINT indicators []=  
{  
    ID_SEPERATOR,  
    ID_SEPERATOR,  
    ID_INDICATOR_TIME,  
    ID_INDICATOR_DATE,  
}; // Achtung, das letzte Komma ist kein Tipfehler.
```

In der Header-Datei Ihres Dialoges sollte nun noch eine Membervariable für den Statusbar angelegt werden (privat). In der Cpp-Datei Ihres Dialoges (PCubeDemoDlg.cpp) fügen Sie in der Funktion CPCubeDemoDlg::OnInitDialog() den folgenden Code ein, um den Statusbar zu erzeugen.


```

if (!m_wndStatusBar.Create(WS_CHILD | WS_VISIBLE | CCS_BOTTOM |
SBARS_SIZEGRIP,

CRect(0,0,0,0), this, ID_MY_STATUSBAR))
{
    TRACE0( "Failed to create status bar\n" );

    return -1; // Fehler beim Erzeugen
}

// Divide Statusbar into areas (define always right corner of each area

const int nParts = 4;

CRect rect;

m_wndStatusBar.GetClientRect(&rect);

UINT16 widths[nParts] =

{
    rect.right-(int)(0.4*(rect.right-rect.left)),

    rect.right-(int)(0.2*(rect.right-rect.left)),

    rect.right-(int)(0.1*(rect.right-rect.left)),

    -1
};

```

9.7 Anlegen eines nichtmodalen Unter-Dialoges (CModulDlg)

- Um zum Hauptdialog einen neuen Dialog hinzuzufügen, geht man folgendermaßen vor:
 - Wählen Sie im *Workspace-Fenster* der Visual C++ - Programmierumgebung (linkes Fenster) das Registerblatt **ResourceView** aus.
 - Öffnen Sie mit einem Klick auf das +-Symbol am Eintrag "*Projektname*" *resources* und klicken mit der rechten Maustaste auf den Eintrag *Dialog*.
 - Wählen Sie im Kontextmenü **insert...** aus und in dem anschließend geöffneten Dialogfenster die Option **new**. Ein neues Dialogfenster wird erzeugt.
- Neue Kontrollelemente werden dem Dialogfenster wie folgt hinzugefügt:
 - Klicken Sie mit der linken Maustaste im *Control*-Fenster auf das gewünschte Kontrollelement, dann ziehen Sie mit gedrückter linker Maustaste ein Rechteck auf, in dem das neue Kontrollelement erstellt wird. Das Kontrollelement kann durch Selektieren mit der linken Maustaste (Cursor wird zum Fadenkreuz) und Ziehen bei gedrückter linker Maustaste positioniert werden. Das *Control*-Fenster öffnet sich zusammen mit dem neuen Dialogfenster auf der rechten Seite. Sollte es versehentlich geschlossen werden, kann es mit einem rechten Mausklick auf die Toolbar des Visual C++ - Hauptfensters und Wählen des Kontextmenüpunkts *Controls* wiederhergestellt werden. Folgende Kontrollelemente wurden verwendet:
 - *Static Text* -Feld: Hier wird der Text vorgegeben und kann vom Benutzer nicht durch Editieren verändert werden, wohl aber zur Laufzeit vom Programm

selbst. Dient meist zur Beschreibung von Eingabefeldern (*Edit Box*).

- *Group Box* (Gruppenfeld): Dient zur graphischen Abgrenzung gleichartiger Kontrollelemente.
 - *Edit Box*-Feld: Hier kann der Benutzer zur Laufzeit des Programms Eingaben machen. Das Eingabefeld kann z.B. mit der Membervariablen einer Klasse verknüpft sein, so daß der Benutzer den Variableninhalt über die Eingabe in die *Edit Box* verändern kann.
 - *Slider*-Element (Schieberegler): Spezialform der Bildlaufleiste. Ist der Schieberegler mit einer Membervariablen verknüpft, kann der Variableninhalt graphisch, d.h. durch Ziehen des Reglers verändert werden.
 - *Radio Button* (Optionsfeld): Hier können alternative Auswahlen getroffen werden, d.h. es kann immer nur ein Optionsfeld, das zu einer Gruppe gehört, selektiert werden.
 - *Check Button* (Kontrollkästchen): Hier können keines oder mehrere Kontrollkästchen gleichzeitig markiert (gecheckt) sein.
 - *Push Button* (Befehlsschaltfläche): Löst eine Aktion oder eine Abfolge von Aktionen aus, wenn er gedrückt (mit der linken Maustaste angeklickt) wird.
- Um eine Gruppe von Radio-Buttons zu erhalten, die nur eine Auswahl zulassen, muß zunächst ein Radio Button auf dem Dialogfenster erzeugt werden. Dieser wird durch rechten Mausklick selektiert und im Kontextmenü wird der Menüpunkt *Properties* ausgewählt und im Properties-Fenster eine eindeutige und sprechende ID vergeben. Nach Schließen des Properties-Fenster wird erneut das Kontextmenü aufgerufen und der Menüpunkt **copy** ausgewählt. Anschließend werden mit einem rechten Mausklick auf eine freie Stelle des Dialogfensters und Auswahl des Kontextmenüpunktes **paste** die gewünschte Anzahl von Optionsfeldern erzeugt. Dann öffnet man im ersten Radio Button den Kontextmenüeintrag *Properties/Registerblatt General* und wählt (nur dort) die Eigenschaft **Group** aus. Schließlich erhalten alle anderen Radio Buttons ebenfalls eindeutige ID's.
 - Alle auf dem Dialogfenster erzeugten Kontrollelemente müssen eindeutige und sprechende (zur besseren Identifikation) ID's erhalten.
 - Nachdem alle Kontrollelemente erzeugt wurden und eindeutige ID's erhalten haben, wird der Klassenassistent, zu erreichen unter *Hauptmenü Visual C++/View/Class Wizard*, aufgerufen. es erscheint ein Dialogfenster, indem Sie dazu aufgefordert werden, für das von ihnen neu angelegte Dialogfenster eine neue Klasse zu erstellen. Sie sollten *create a new class* auswählen und mit *OK* bestätigen.
 - Im Klassenassistenten (*Class Wizard/Registerblatt Member Variables*) werden allen Kontrollelementen, die im Programm angesprochen bzw. deren Eigenschaften verändert werden sollen, Membervariable zugeordnet (s. Tabelle 1).
 - Im Registerblatt *Message Maps* des Klassenassistenten können den Kontrollelementen (ID's beachten) sowie der Dialogklasse selbst Message-Handler (Funktionen, die auf best. Ereignisse reagieren) hinzugefügt werden.
 - Zusätzlich benötigte Membervariable können mittels Rechtsklick im *Workspace-Fenster/ Registerblatt Classview* auf den Eintrag der neu erzeugten Dialogklasse *CModulDlg* über den Kontextmenüpunkt *Add Member Variable...* hinzugefügt werden (s. Tabelle 2).

Nachfolgend werden tabellarisch die im Klassenassistenten erzeugten Membervariablen und Message-Handler des Modul-Dialogs aufgeführt:

Control-ID's und Membervariable sowie Messages

Objekt-ID	Datentyp	Memervariable	Messages
IDC_BUTTON_ENLARGE	CButton	m_ctrlButtonEnlarge	BN_CLICKED
IDC_BUTTON_GO			BN_CLICKED
IDC_BUTTON_HALT			BN_CLICKED
IDC_BUTTON_INIT			BN_CLICKED
IDC_BUTTON_RESET			BN_CLICKED
IDC_BUTTON_FLAGS	CButton	m_ctrlButtonFlags	BN_CLICKED
IDC_EDIT_ACC	float	m_flEditAcc	EN_SETFOCUS EN_CHANGE EN_KILLFOCUS
IDC_EDIT_POS	float	m_flEditPos	EN_SETFOCUS EN_CHANGE EN_KILLFOCUS
IDC_EDIT_VEL	float	m_flEditVel	EN_SETFOCUS EN_CHANGE EN_KILLFOCUS
IDC_STATIC_MAXPOS	CString CStatic	m_strMaxPos m_ctrlMaxPos	
IDC_STATIC_MINPOS	CString CStatic	m_strMinPos m_ctrlMinPos	
IDC_STATIC_MAXACC	CString CStatic	m_strMaxAcc m_ctrlMaxAcc	
IDC_STATIC_MINACC	CString CStatic	m_strMinAcc m_ctrlMinAcc	
IDC_STATIC_MAXVEL	CString CStatic	m_strMaxVel m_ctrlMaxVel	
IDC_STATIC_MINVEL	CString CStatic	m_strMinVel m_ctrlMinVel	
IDC_STATIC_ACTCUR	CString	m_ctrlActCur	
IDC_STATIC_ACTPOS	CString	m_ctrlActPos	
IDC_STATIC_ACTVEL	CString	m_ctrlActVel	
IDC_STATIC_DELTAPOS	CString	m_ctrlDeltaPos	
IDC_RADIO_POS	int	m_intRadio	BN_CLICKED
	CButton	m_ctrlRadioPos	
IDC_RADIO_CUR	erscheint nicht bei Member Variables		BN_CLICKED
IDC_RADIO_VEL	erscheint nicht bei Member Variables		BN_CLICKED
IDC_SLIDER_ACC	CSliderCtrl	m_ctrlSliderAcc	
IDC_SLIDER_POS	CSliderCtrl	m_ctrlSliderPos	
IDC_SLIDER_VEL	CSliderCtrl	m_ctrlSliderVEL	
IDC_STATIC_FRAME	CButton	m_ctrlStaticFrame	
IDC_STATIC_FLAGS	CStatic	m_ctrlStaticFlags	
IDC_STATIC_POS	CStatic	m_ctrlStaticPos	
IDC_CHECK_ACC	bool	m_bCheckAcc	
	CButton	m_ctrlCheckAcc	
IDC_CHECK_BRA	bool	m_bCheckBra	
	CButton	m_ctrlCheckBra	
IDC_CHECK_COM	bool	m_bCheckCom	
	CButton	m_ctrlCheckCom	
IDC_CHECK_CPU	bool	m_bCheckCPU	
	CButton	m_ctrlCheckCPU	

IDC_CHECK_DEC	bool	m_bCheckDec	
	CButton	m_ctrlCheckDec	
IDC_CHECK_END	bool	m_bCheckEnd	
	CButton	m_ctrlCheckEnd	
IDC_CHECK_ERR	bool	m_bCheckErr	
	CButton	m_ctrlCheckErr	
IDC_CHECK_FET	bool	m_bCheckFet	
	CButton	m_ctrlCheckFet	
IDC_CHECK_FULL	bool	m_bCheckFull	
	CButton	m_ctrlCheckFull	
IDC_CHECK_HALL	bool	m_bCheckHard	
	CButton	m_ctrlCheckHard	
IDC_CHECK_HALT	bool	m_bCheckHalt	
	CButton	m_ctrlCheckHalt	
IDC_CHECK_HARD	bool	m_bCheckHard	
	CButton	m_ctrlCheckHard	
IDC_CHECK_HOK	bool	m_bCheckHOK	
	CButton	m_ctrlCheckHOK	
IDC_CHECK_INPRO	bool	m_bCheckInpro	
	CButton	m_ctrlCheckInpro	
IDC_CHECK_INTEGRAL	bool	m_bCheckIntegral	
	CButton	m_ctrlCheckIntegral	
IDC_CHECK_LIM	bool	m_bCheckLim	
	CButton	m_ctrlCheckLim	
IDC_CHECK_MOV	bool	m_bCheckMov	
	CButton	m_ctrlCheckMov	
IDC_CHECK_POW	bool	m_bCheckPow	
	CButton	m_ctrlCheckPow	
IDC_CHECK_S1	bool	m_bCheckS1	
	CButton	m_ctrlCheckS1	
IDC_CHECK_S2	bool	m_bCheckS2	
	CButton	m_ctrlCheckS2	
IDC_CHECK_SHORT	bool	m_bCheckShort	
	CButton	m_ctrlCheckShort	
IDC_CHECK_SOFT	bool	m_bCheckSoft	
	CButton	m_ctrlCheckSoft	
IDC_CHECK_SR	bool	m_bCheckSR	
	CButton	m_ctrlCheckSR	
IDC_CHECK_STEADY	bool	m_bCheckSteady	
	CButton	m_ctrlCheckSteady	
IDC_CHECK_TOW	bool	m_bCheckTow	
	CButton	m_ctrlCheckTow	
IDC_CHECK_VOLT	bool	m_bCheckVolt	
	CButton	m_ctrlCheckVolt	
IDC_CHECK_WDG	bool	m_bCheckWDG	
		CButton	
CModulDlg			WM_HSCROLL

Beim Erzeugen der Funktionen werden die vom Klassenassistenten vorgeschlagenen Namen übernommen.

Nachfolgend werden die zusätzlichen Membervariablen, die keinem über das Kontextmenü im Workspace-Fenster/Class View erzeugt wurden, aufgelistet:

Zusätzliche Membervariablen

Membervariable	Datentyp
m_bFlag	bool
m_bSize	bool
m_bEdit	bool
m_bScroll	bool
m_intX	int
m_intY	int
m_intCx	int
m_intCy	int
m_intIndex	int
m_pApp	CPCubeDemoApp*
m_pDialog	CDialog*

Der so erzeugte Dialog muß nun von Hand in die Umgebung integriert werden, da der Klassenassistent keine nichtmodalen Dialoge unterstützt.

9.7.1 Integration des nichtmodalen Dialoges

Die Integration eines nichtmodalen Dialoges geschieht in Anlehnung an Kapitel 7 des Buches "Inside Visual C++ 6.0" von David Kruglinski.

- Fügen Sie in der Header-Datei des Unterdialoges (ModulDlg.h) die folgenden privaten Member-Variablen ein.

```
CDialog* m_pDialog; (Dieser Zeiger soll später auf den aufrufenden Dialog zeigen.)
UInt16 m_intIndex; (Diese Variable enthält den Index des ausgewählten Moduls.)
```

- Fügen Sie die folgende Funktionsdeklaration ein:

```
CModulDlg( CDialog* pDlg, UInt16 index); Konstruktor für den nichtmodalen Dialog.
```

```
Bool Create();
```

- Außerhalb der Klassendeklaration muß nun eine ID für eine eigene Message vereinbart werden. Fügen Sie bitte folgende Define-Anweisung ein:

```
#define WM_CLOSEDLG WM_USER + 5
Die ersten fünf ID's werden vom Anwendungsgerüst verwendet.
```

- In der C-Datei des Unterdialoges (ModulDlg.cpp) muß der eben deklarierte Konstruktor definiert werden.

```
CModulDlg::CModulDlg( CDialog* pDlg, UInt16 index )
{
    m_pDialog = pDlg;
    m_intIndex = index;
}
```

- In den Standardkonstruktor (für das modale Öffnen dieses Dialoges) müssen die Variablen natürlich auch initialisiert werden.

```
m_pDialog = NULL;
m_intIndex = 0;
```

- Bearbeiten Sie die deklarierte Funktion Create() wie folgt;

```
Bool CModulDlg::Create()
{
    return CDialog::Create(CModulDlg::IDD)
}
```

Bei nichtmodalen Dialogen dürfen nicht die Basisklassenfunktionen von OnOK() und OnCancel() aufgerufen werden. Das würde aber passieren, sobald Sie die <Return> oder <ESC> drücken. Aus diesem Grund müssen Sie unbedingt die Funktionen überladen.

```

void CModulDlg::OnCancel()
{
    if( m_pDialog != NULL )
        m_pDialog->PostMessage(WM_CLOSEDLG, m_intIndex); // for
        modeless Dialog
    else
        CDialog::OnCancel(); // for modal Dialog
}

void CModulDlg::OnOK()
{
    // Bewahrt das Dialogfenster davor, durch Drücken der <Return>-Taste
    geschlossen zu werden

    // CDialog::OnOK();
}

```

Erläuterung der Funktion: `BOOL PostMessage(UINT message, WPARAM wParam = 0, LPARAM lParam = 0);`

`wParam` und `lParam` werden zur Übergabe von Nachrichtendaten verwendet. Man kann diesen Datentypen nach Belieben Werte zuweisen. Wir werden `wParam` dazu verwenden, den Index des Moduls zurückzugeben, für das dieser Dialog geöffnet worden ist.

- Includieren Sie die nötiger Header-Dateien

```

#include "PCubeDemoDlg.h"
#include "ModulDlg.h"

```

- Fügen Sie in der Header-Datei des Oberdialoges (PCubeDemoDlg.h) die folgende private Member-variable ein.

```
CModulDlg* m_pDlg[31];
```

(Diese Zeiger sollen später auf die geöffneten Unterdialoge zeigen.)

- Damit das Anlegen des Dialogarrays zu keinen Fehlermeldungen führt müssen Sie die Header-Datei (ModulDlg.h) includieren, oder Sie verwenden außerhalb der Klassendeklaration die Vorwärtsdeklaration :

```
class CModulDlg;
```

Guter Stil ist nun die Zeiger zu initialisieren. Verwenden Sie dazu eine Schleife im Konstruktor von `CPCubeDemoDlg` ;

```

for ( int i = 0 ; i < 31 ; i++ )
{
    m_pDlg[i] = NULL;
}

```

- Modifizieren Sie den Konstruktor, die Funktion `OnInitDialog()` oder eine beliebige andere Funktion, in der Sie auf einen konkreten Dialog zeigen wollen. Wir werden später genauer auf diese Problematik zu sprechen kommen!

```
m_pDlg[i] = new CModulDlg( this, i ); //this zeigt auf Oberdialog, i
ist der Index des Moduls.
```

Vergessen Sie nicht an entsprechender Stelle den Speicher mit `delete` wieder freizugeben.

- Modifizieren Sie eine Funktion Ihrer Wahl um den Dialog zu erzeugen und sichtbar zu machen. Wir tun dies z.B. bei einem Doppelklick auf das List-Control.

```
m_pDlg[i]->Create();
```

```
m_pDlg[i]->ShowWindow( SW_SHOW );
```

Möchten Sie das der sich öffnende Dialog an einer bestimmten Stelle erscheint so können Sie die Funktion ShowWindow (.....) auch durch die Funktion SetWindowPos(.....) ersetzen. Diese Funktion beinhaltet implizit ein ShowWindow(. . .).

Vergessen Sie nicht an entsprechender Stelle mit der Funktion m_pDlg[i]->DestroyWindow() die Fenster wieder zu löschen.

Wie bereits erwähnt unterstützt der Klassenassistent keine benutzerdefinierten Nachrichten. Sie müssen deshalb die folgenden Erweiterungen vornehmen:

- In der Header-Datei (PCubeDemoDlg.h) müssen Sie vor DECLARE_MESSAGE_MAP() und außerhalb von afx_msg die Nachrichtenfunktion deklarieren.

```
afx_msg LRESULT OnCloseDlg(WPARAM wParam, LPARAM lParam);
```

- In der Datei PCubeDemoDlg.cpp müssen Sie nach BEGIN_MESSAGE_MAP und vor AFX_MSG_MAP die folgende Nachricht definieren:

```
ON_MESSAGE( WM_CLOSEDLG, OnCloseDlg)
```

- Sie müssen auch die Nachrichtenfunktion selbst definieren

```
LRESULT CPCubeDemoDlg::OnCloseDlg( WPARAM wParam, LPARAM lParam)
{
    if ( m_pdlg[ wParam ] ) // wParam ist der Index des geöffneten
        Dialoges (Modul)
        m_pDlg[wParam]->DestroyWindow();
    return 0L;
}
```

Das Abfragen ob der Zeiger auf den Dialog überhaupt existiert ist nicht zwingend, da DestroyWindow() auch auf nicht existierende Fenster angewendet werden darf.

9.8 Grundprinzip des Programmaufbaus

Mit unserem Programm können (pro Schnittstelle) bis zu 31 Module angeschlossen werden. Alle Parameter der Module müssen im Programm gespeichert werden und natürlich jedem Unterdialog (Moduldialog) zur Verfügung stehen. Aus diesem Grund werden wir die Moduldaten in einer eigenen Klasse kapseln. Diese Klasse soll den Namen *CModData* tragen. In diese Klasse fügen Sie bitte alle benötigten Modulparameter ein (zunächst global). Wir benötigen zwei weitere Variable, die uns ohne ein Objekt dieser Klasse zu haben zur Verfügung stehen müssen. Diese Variablen betreffen die Anzahl der Module und die Geräteadresse. Wir werden diese Variablen statisch anlegen. In unserem Hauptdialog legen wir dann ein Array von Pointern auf diese Klasse an. Immer wenn wir nach Modulen scannen, wird für jedes Modul ein Dialog und ein Modularray angelegt. Da der Dialog den Zeiger auf den Oberdialog und einen entsprechenden Index mit übergeben bekommt und das Modularray global angelegt ist, kann jeder Dialog auf sein zugehöriges Array zugreifen.

Vorgehensweise

- Klicken Sie in der Entwicklungsumgebung in der Menüleiste **Insert->New class...**
- Den sich öffnenden Dialog füllen Sie wie folgt aus:
Class type: Generic Class (kann ausgewählt werden)
Name: CModData
- In Ihrem Workspace-Fenster wählen Sie nun das Registerblatt **ClassView** aus und klicken doppelt auf die Klasse *CModData*
- Fügen Sie nach belieben globale Modulparameter ein. Vergessen Sie nicht die beiden statischen Variablen

```
static UInt16 m_intCount;
static Int16 m_intDevice;
```

- Nun sollten Sie im Konstruktor dieser Klasse alle Variablen initialisieren.
- Die beiden statischen Variablen werden außerhalb der Klasse *CModData* wie folgt definiert:

```
UInt16 CModData::m_intCount = 0;
Int16 CModData::m_intDevice = -1;
```

Nun müssen Sie in der Header-Datei des Hauptdialoges (PCubeDemoDlg.h) zwei Zeigerarrays anlegen. Ein globale Array für die Zeiger auf *CModData* und ein privates Zeigerarray auf die zu öffnenden Dialoge. Letzteres haben wir bereits angelegt und initialisiert:

```
public:
    •CModData* m_ptrCubeData[31];
```

- Damit das Anlegen des Modularrays zu keinen Fehlermeldungen führt, müssen Sie die Header-Datei (ModData.h) includieren, oder Sie verwenden, außerhalb der Klassendeklaration, die Vorwärtsdeklaration :

```
class CModData;
```

- Guter Stil ist es nun die Zeiger zu initialisieren. Verwenden Sie dazu die Schleife im Konstruktor von *CPCubeDemoDlg* und erweitern diese um eine Zeile;

```
for ( int i = 0 ; i < 31 ; i++ )
{
    m_ptrCubeData[i] = NULL;
    m_pDlg[i] = NULL;
}
```

9.9 Der Initialisierungs- und Scanthread

Nachdem die Anwendung gestartet wird muß zunächst nach angeschlossenen Modulen gesucht werden. Das soll als Thread in der Funktion OnRescan() (Klick in die Menüleiste *Devices->Rescan*) geschehen. Nachdem Module gefunden wurden, soll der Thread für jedes angeschlossene Modul die Defaultparameter lesen. Anschließend soll der Thread in einer Endlosschleife in bestimmten Zeitabständen aktuelle Parameter lesen und das Modul-Array aktualisieren.

- Bevor wir das tun müssen wir noch einige Membervariablen in unserer Klasse aufnehmen.

```
Bool m_bThread; //soll laufenden Thread signalisieren
HANDLE m_hThread; //Handler auf gestarteten Thread
```

- Starten eines Threads.

```
//
*****
// Starting a Thread to open a Device
//
*****
m_bThread = true;
UInt threadID;
m_hThread = CreateThread( NULL,

                                                                    •0,
                                                                    (LPTHREAD_START_ROUTINE)scanning,
                                                                    (LPVOID)this,
                                                                    0,
                                                                    (LPDWORD)&threadID);

if ( !m_hThread )
{
    m_bThread = false;
    TRACE("Thread -find Modules- not started");
}
else
{
    TRACE("Scanning for PowerCube Moduls ...");
    m_wndStatusBar.SetText("Scanning for PowerCube Moduls
    ...",0,0);
}
```

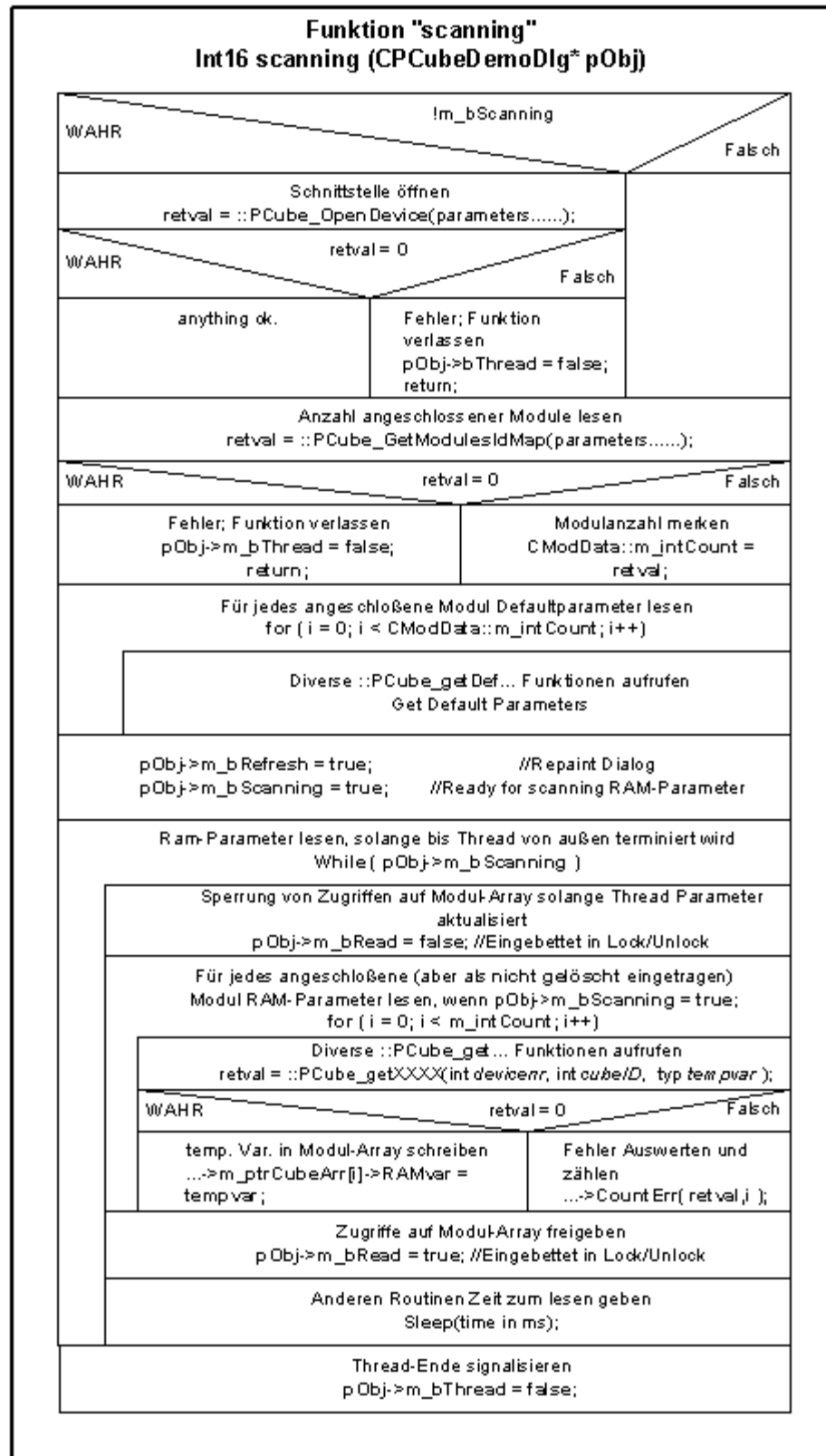
Der Thread bekommt einen Zeiger auf eine Funktion als Parameter übergeben. Diese Funktion wird dann im Hintergrund ausgeführt. Die Funktion muß zuvor natürlich deklariert und definiert werden. Bitte deklarieren und definieren Sie die Funktion außerhalb der Klasse.

```
int16 scanning (CPCubeDemoDlg* pObj);
```

Die Funktion bekommt als Parameter einen Zeiger auf den Oberdialog. Somit kann im Thread auf globale Membervariablen und -funktionen des Dialoges zurückgegriffen werden.

Die Funktion scanning

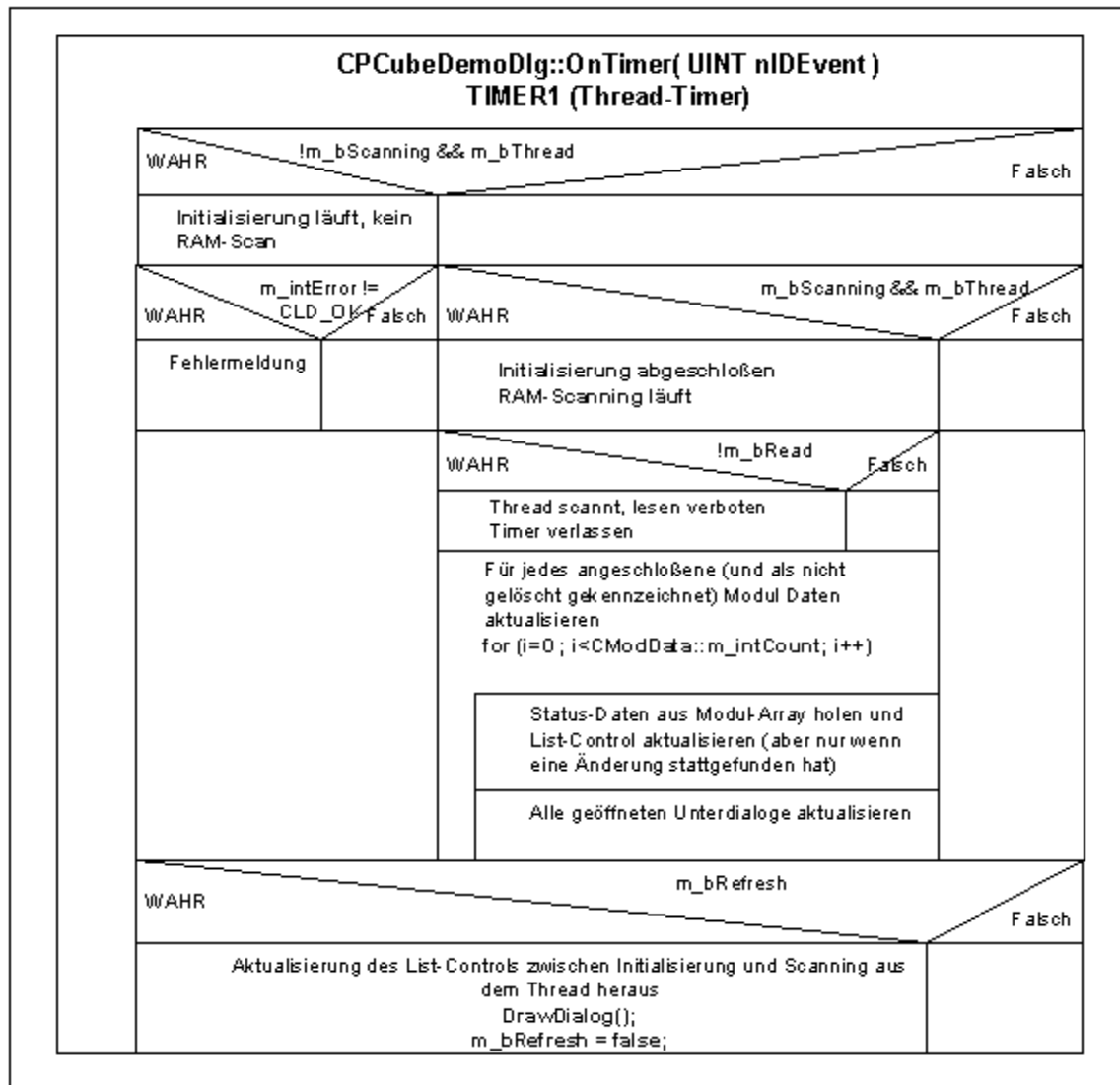
Die Funktion scanning ist in der Datei PCubeDemoDlg.h global deklariert und in der Datei PCubeDemoDlg.cpp global definiert.

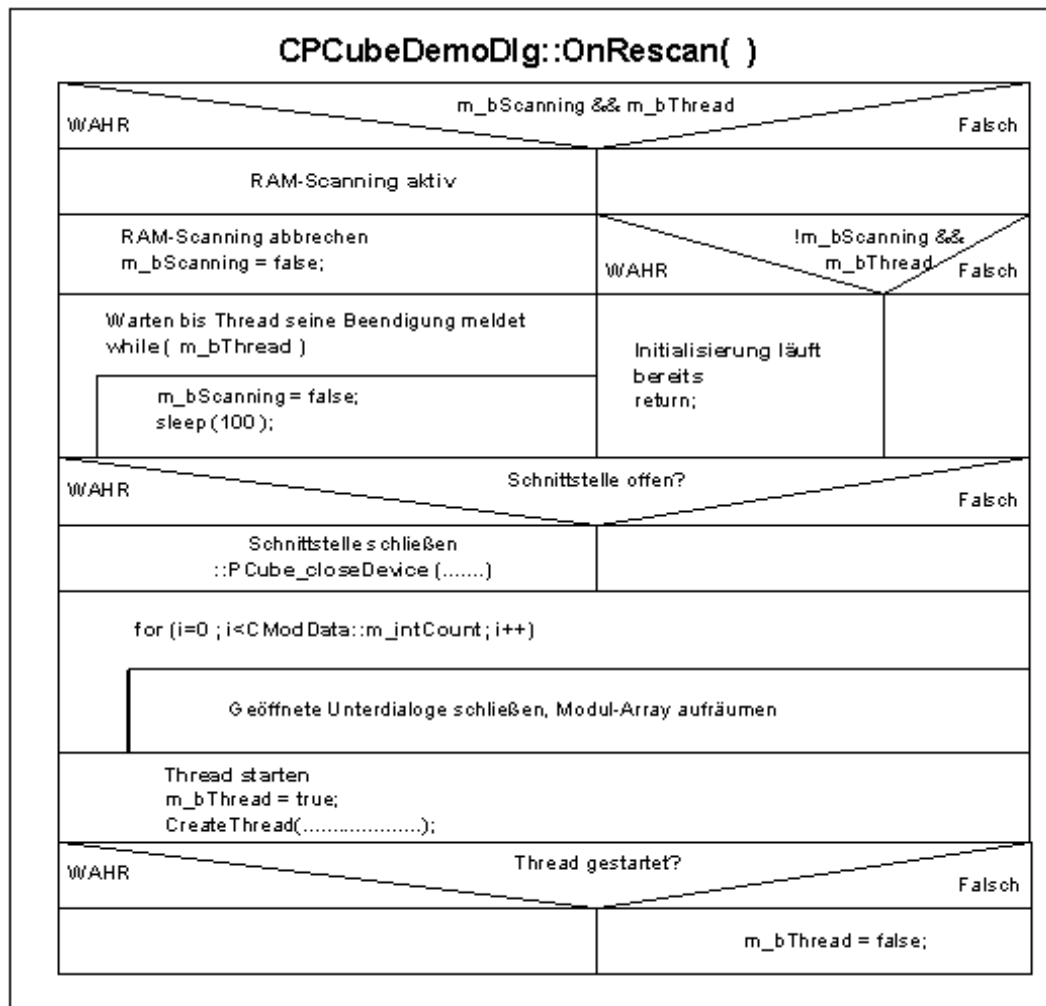


9.10 Funktionaler Ablauf des Programmes

Nachdem das Programm gestartet wurde, wird der Hauptdialog aufgebaut und ein Timer gestartet. Dieser Timer ist dafür zuständig alle 200 ms die vom Thread in das Modul-Array geschriebenen Daten zu lesen und die Bildschirmanzeige zu aktualisieren.

Der Thread wird erst gestartet, wenn der entsprechende Menüpunkt (Rescan()) geklickt wird. Was der Timer und die Funktion Rescan im einzelnen abarbeiten ist den beiden folgenden Flußdiagrammen zu entnehmen. Der Thread läuft vollkommen eigenständig im Hintergrund und aktualisiert alle 200ms die Daten des Modul-Arrays. Parallel dazu werden im Timer alle 200 ms diese Daten aus dem Modul-Array gelesen. Diese beiden Vorgänge sind zu synchronisieren um zu vermeiden das der Timer Moduldaten ließt obwohl der Thread erst ein halben Datensatz in dieses Array geschrieben hat. Zu diesem Zweck wird im Thread eine boolsche Variable (m_bRead) solange auf "false" gesetzt, wie der Thread braucht um die Daten zu schreiben. Der Timer kann diese Daten nur lesen , wenn die Variable (m_bRead) "true" ist. Um eine Zugriffsverletzung während des Schreibens und Lesens der boolschen Variable zu verhindern, wird das Lesen und Schreiben der Variable in einer "Critical Section" gesichert.





9.11 Fehlerabarbeitung

Jede PowerCube™-Funktion liefert einen Fehlerstatus zurück, der im Idealfall "0" (CLD_OK) ist. Daher ist der Rückgabewert einer jeden Funktion zu prüfen. Beim Lesen der RAM-Parameter darf ein Schreiben ins Modul-Array nur bei fehlerfreiem Funktionsablauf durchgeführt werden. Nun macht es keinen Sinn bei jedem kleinen Fehler das gesamte Programm abzubrechen. Vielmehr wird hier die folgende Strategie durchgeführt. Alle auftretenden Fehler werden aufsummiert und nach ca. 2 Sekunden wieder gelöscht, wenn nicht mehr als 10 Fehler aufgetreten sind. Sind mehr als zehn Fehler aufgetreten wird das entsprechende Modul abgeschaltet bzw. aus dem List-Contol gelöscht und steht von nun an nicht mehr zur Verfügung.

Es wurde nicht für jeden möglichen Fehler eine eigene Variable vergeben, sondern die Fehlererfassung und -auswertung erfolgt auf Bitebene. In eine Variable vom Typ Long int (32 Bit passen 8 Halbwörter a 4 Bit. Somit kann ein Fehler maximal bis zur Zahl 15 auflaufen (2 hoch 4 - 1). Um nun an die einzelnen Bitfelder heranzukommen wird das 32-Bit Wort mit einer 4-Bit-Maske UND-verknüpft. Dies gewährleistet, das nur das entsprechende Bitfeld ausgewertet wird und alle übrigen Bits null sind. Anschließend wird das Bitfeld solange nach rechts geschoben bis die auszuwertende Bitgruppe an den Bitpositionen 0-3 liegt. Anschließend kann über einen einfachen Integervergleich die Fehlerzahl ermittelt werden.

Mit der Aufsummierung verhält es sich ähnlich, nur das hier eine Addition mit einem Bitfeld "0001" stattfindet.

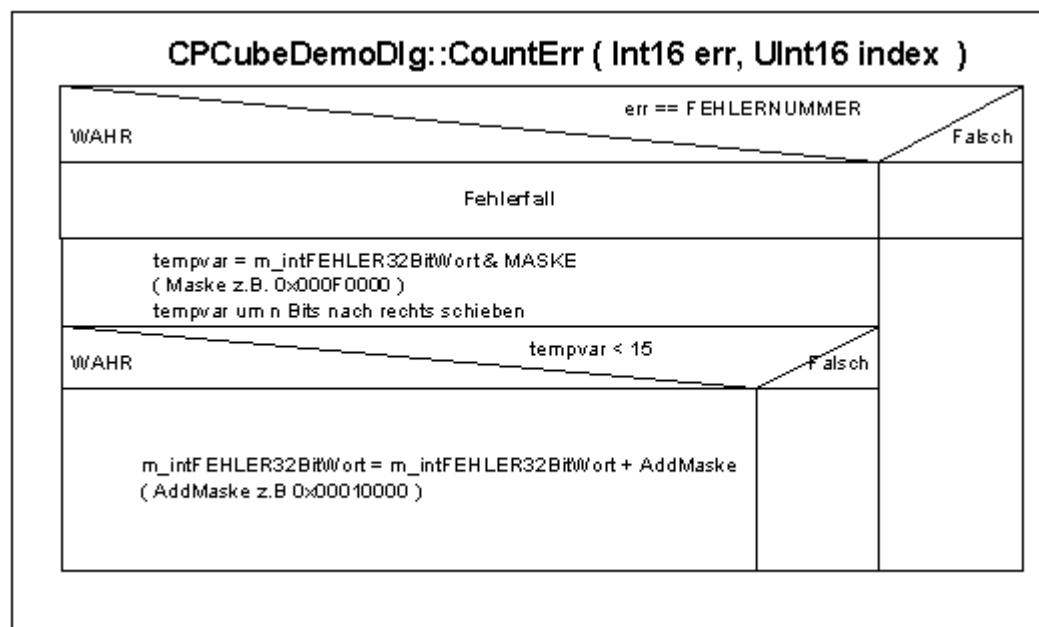
Die Fehlersummierung findet in der Funktion CPCubeDemoDlg::CountErr (.....) statt. Diese Funktion wird vom Thread aufgerufen. Die Fehlerauswertung wird im Timer (Timer2, Intervall 2Sek.) aufgerufen und findet in der Funktion CPCubeDemoDlg::CheckSumErr (.....) statt.

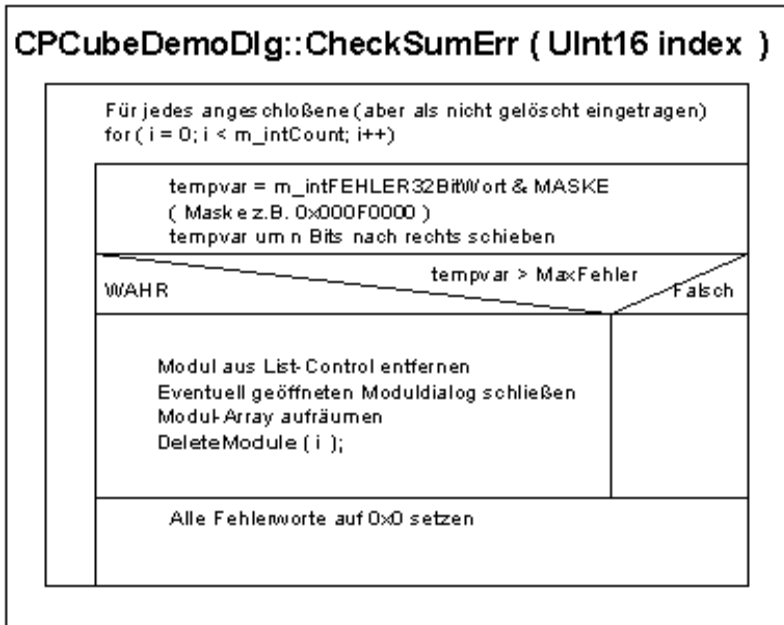
Beispiel (Summenbildung) :

Fehlerwort:	0010 0010 0000 0110 1111 0011 0001 (Fehlerzahl Bitgruppe 2 = 3)
Fehler der zweiten Bitgruppe +1	0000 0000 0000 0000 0000 0001 0000
Ergebnis der UND-Verknüpfung:	0010 0010 0000 0110 1111 0100 0001 (Fehlerzahl Bitgruppe 2 = 4)

Beispiel (Auswertung) :

Fehlerwort:	0010 0010 0000 0110 1111 0011 0001
Maskierung	0000 0000 0000 0000 0000 1111 0000
Ergebnis der UND-Verknüpfung:	0000 0000 0000 0000 0000 0011 0000
Schieben des Ergebnisses um 4 Bits nach rechts :	0000 0000 0000 0000 0000 0000 0011 (= Fehlerzahl 3)





9.12 Die Funktionsweise des Moduldialogs

- In die Funktion CPCubeDemoDlg::OnDblickListmoduls() des Hauptdialogs werden drei Objekte vom Typ RECT erzeugt:

```

RECT* lpRect0;
RECT* lpRect1;
RECT* lpRect2;
lpRect0 = new RECT;
lpRect1 = new RECT;
lpRect2 = new RECT;

```

- Diese Rechteck-Objekte erhalten nun die Dimensionen des Hauptdialogs, des vollständigen Moduldialogs und des Teilbereichs vom Moduldialog, der dargestellt werden soll:

```

this->GetWindowRect ( lpRect0 );
m_pDlg[nItem]->GetWindowRect ( lpRect1 );
m_pDlg[nItem]->m_ctrlStaticFrame.GetWindowRect ( lpRect2 );

```

- Dann werden die Koordinaten des Hauptdialogs in eigenen Membervariablen gespeichert. Schließlich wird noch die Position des Frames, ab dem das Modulfenster unsichtbar bleiben soll, ermittelt und der Moduldialog wird mit Hilfe dieser Werte angezeigt. Die Addition des Terms (20*nItem) sorgt für eine kaskadierte Darstellung der einzelnen Modulfenster:

```

m_intX = lpRect0->left;
m_intY = lpRect0->top;
m_intCy = lpRect0->bottom - lpRect0->top;

```

```

Int32 top_dlg;
Int32 top_dlg_frm;
top_dlg_frm = lpRect2->top;
top_dlg = lpRect1->top;
m_pDlg[nItem]->SetWindowPos ( &wndTopMost,
m_intX + (20*nItem),
m_intY + m_intCy + (20*nItem),
m_pDlg[nItem]->m_intCx,
top_dlg_frm-top_dlg,
SWP_SHOWWINDOW );

```

Das im Kapitel **Inbetriebnahme**, Abschnitt [Das Testprogramm PCubeDemo](#) beschriebene Verhalten des Moduldialogs kann z.B. durch folgende Programmierschritte erreicht werden:

9.12.1 Moduldialog initialisieren

- Die Funktion `CModulDlg::OnInitDialog()` wird, falls nicht beim Erstellen der Moduldialog-Klasse bereits erzeugt, im Klassenassistenten durch das Hinzufügen der Nachricht `WM_INITDIALOG` zur Klasse `CModulDlg` sowie anschließendem Betätigen des **Add Function**-Buttons bereitgestellt.
- Zwecks Erleichterung der Schreibearbeit werden nun lokale Variable des Typs `float` deklariert, denen die jeweiligen minimalen bzw. maximalen Begrenzungswerte für Position, Geschwindigkeit und Beschleunigung (im Betriebssystem der Module eingebrennt) zugewiesen werden:

```
float DefMaxPos;
float DefMinPos;
float DefMaxAcc;
float DefMinAcc;
float DefMaxVel;
float DefMinVel;
DefMinPos = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinPos;
DefMaxPos = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxPos;
DefMinAcc = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinAcc;
DefMaxAcc = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxAcc;
DefMinVel = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinVel;
DefMaxVel = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxVel;
```

- Der Bereich der Slider wird zwischen 0 und 1000 festgelegt:

```
m_ctrlSliderAcc.SetRangeMin(0);
m_ctrlSliderPos.SetRangeMin(0);
m_ctrlSliderVel.SetRangeMin(0);
m_ctrlSliderAcc.SetRangeMax(1000);
m_ctrlSliderPos.SetRangeMax(1000);
m_ctrlSliderVel.SetRangeMax(1000);
```

- Im Klassenassistenten werden den *Static Text*-Felder zur Beschriftung der Slider-Bereichsgrenzen Membervariablen vom Typ `CString` zugeordnet (Auswahl der Karteikarte *Member Variables*, Selektieren des gewünschten Controls und Drücken des Buttons **Add Variable**). Diesen Membervariablen werden mittels der *Format*-Funktion die Begrenzungswerte zugeordnet, beispielsweise

```
if ( m_pDialog->m_ptrCubeArr[m_intIndex]->m_nModuleType ==
    LINEAR_DRIVE )
{
    m_strMinPos.Format("%.3f mm\n", DefMinPos * 1000);
    m_strMaxPos.Format("%.3f mm\n", DefMaxPos * 1000);
    m_strMinAcc.Format("%.f mm/s^2\n", DefMinAcc * 1000);
    m_strMaxAcc.Format("%.3f mm/s^2\n", DefMaxAcc * 1000);
    m_strMinVel.Format("%.f mm/s\n", DefMinVel * 1000);
    m_strMaxVel.Format("%.3f mm/s\n", DefMaxVel * 1000);
}
```


- Die im Frame *actual module parameter* angezeigten Werte werden analog aus der Container-Klasse *CModData* geholt und mit der Format-Funktion dargestellt.
- Da Slider und Modul unterschiedliche Bewegungsbereiche haben, müssen Funktionen hinzugefügt werden, die die Sliderposition in Modulwerte umrechnen und umgekehrt (Funktionen einfügen über Kontextmenüpunkt *Add Member Function* bei rechtem Mausklick auf die Klasse *CModulDlg* im *Class View*), z.B.:

```
float CModulDlg::GetPosFromCtrl(int pos)
{
    return (((float)pos *
        ( m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxPos -
        m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinPos ) /
        (float)1000.0) +
        m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinPos );
}
```

liefert die aktuelle Position des Sliders
und

```
int CModulDlg::GetPosForSlider()
{
    float term;
    term = (m_pDialog->m_ptrCubeArr[m_intIndex]->ActPos -
        m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinPos ) * 1000 /
        ( m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxPos -
        m_pDialog->m_ptrCubeArr[m_intIndex]->DefMinPos );
    return (int)term;
}
```

übergibt dem Slider die aktuelle Position des Moduls. Dabei ist zu beachten, daß die Funktion *m_ctrlSliderGetPos()*, die die momentane Position des Sliders liefert, nur Integer-Werte zurückgibt.

- Dann wird die Position des Sliders der aktuellen Position des Moduls angepaßt und der Inhalt des Editfeldes auf Null gesetzt, da hier nur vom Benutzer mittels Eingabe oder Sliderbewegung vorgegebene Positionswerte angezeigt werden:

```
m_ctrlSliderPos.SetPos(GetPosForSlider());
m_flEditPos = 0.0f;
```

9.12.2 Scroll-Ereignis bearbeiten

- Die Funktion *CModulDlg::OnHScroll()* (reagiert auf Ereignisse an horizontalen Scrollcontrols wie auch Slidern) wird im Klassenassistenten durch das Hinzufügen der Nachricht *WM_HSCROLL* zur Klasse *CModulDlg* sowie anschließendem Betätigen des **Add Function**-Buttons bereitgestellt. Diese Funktion liefert den Wert *nSBCode* (spezifiziert das Scroll-Ereignis), den Wert *nPos* (Position des Sliders) sowie den Wert *pScrollBar* (Zeiger auf die betätigte Scrollbar bzw. Slider) zurück.
- Durch Abfragen des Zeigers und des Scroll-Ereignisses reagiert man spezifisch auf den oberen Slider mit der Membervariablen *m_ctrlSliderPos* und auf den Zustand *SB_ENDSCROLL*, der eintritt, wenn der Slider durch Loslassen der linken Maustaste freigegeben wurde:

```

if (pScrollBar == (CScrollBar*)&m_ctrlSliderPos)
{
    m_bScroll = TRUE;
    if ( nSBCode == SB_ENDSCROLL )
        m_bScroll = FALSE;
}

```

Beim Betätigen des obersten Sliders mit der Membervariable *m_ctrlSliderPos* wird eine boolsche Variable *m_bScroll* auf *true* gesetzt (wird benötigt, um die unterschiedlichen Verhaltensweisen beim Scrollen und nach Freigabe des Sliders zu realisieren, s.u.); beim Ereignis SB_ENDSCROLL wird diese Variable wieder auf *false* gesetzt.

- Solange der Slider *m_ctrlSliderPos* betätigt wird (*m_bScroll == true*), werden je nach Bewegungsmodus die entsprechenden Bewegungsbefehle zum Modul geschickt. Im nachfolgenden Beispiel für Linearmodule erscheint ein konstanter Umrechnungsfaktor von 1000, der durch die Umrechnung von m (Einheit im Betriebssystem des Moduls) in mm (Anzeige im Moduldialog) erforderlich wird:

```

if ( m_pDialog->m_ptrCubeArr[m_intIndex]->m_nModuleType ==
LINEAR_DRIVE )
{
    switch ( m_intRadio )
    {
        case RAMPMODE:
            m_flEditPos = GetPosFromCtrl(m_ctrlSliderPos.Get-
Pos()) * 1000;
            ::PCube_moveRamp( CModData::m_intDevice,
m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID ,
GetPosFromCtrl(m_ctrlSliderPos.GetPos()),
m_flEditVel / 1000,
m_flEditAcc / 1000);
            break;

        case VELOCITYMODE:
            ::PCube_moveVel( CModData::m_intDevice,
m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID ,
2 * GetVelFromCtrl(m_ctrlSliderPos.GetPos()) );
            m_flEditPos = GetVelFromCtrl(m_ctrlSliderPos.Get-
Pos()) * 2000;
            break;

        case CURRENTMODE:
            ::PCube_moveCurrent( CModData::m_intDevice,
m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID ,
GetCurFromCtrl(m_ctrlSliderPos.GetPos()) * 2);
            m_flEditPos = GetCurFromCtrl(m_ctrlSliderPos.Get-
Pos()) * 2);
            break;
    }
}

```

Zur Umrechnung von Rad [rad] in Grad [°] bei Verwendung von Drehmodulen müssen Funktionen zur Konvertierung eingeführt werden:

```
float CModulDlg::GetDegreeFromRad(float rad)
{
    return ( rad * 360.0f / (2.0f * PI) );
}

```

und

```
float CModulDlg::GetRadFromDegree(float deg)
{
    return ( deg * 2.0f * PI / 360.0f );
}

```

PI = Definekonstante :

```
define PI 3.1415926535897932384626433832795f

```

Diese werden dann bei der Berechnung der Parameter für die Bewegungsfunktionen eingesetzt, z.B.:

case RAMPMODE:

```
    m_flEditPos = GetDegreeFromRad(GetPosFromCtrl
    (m_ctrlSliderPos.GetPos()));
    ::PCube_moveRamp( CModData::m_intDevice,
    m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID ,
    GetPosFromCtrl(m_ctrlSliderPos.GetPos()),
    GetRadFromDegree(m_flEditVel),
    GetRadFromDegree(m_flEditAcc) );

```

- Beim Loslassen des Sliders *m_ctrlSliderPos* sollen je nach Bewegungsmodus unterschiedliche Verhaltensweisen des Sliders realisiert werden. Befindet sich das Modul im Geschwindigkeits- bzw. Strommodus, wird beim Ereignis *SB_ENDSCROLL* jeweils ein Bewegungskommando gesendet, das die Geschwindigkeit bzw. Stromaufnahme auf Null zurücksetzt:

```
if ( nSBCode == SB_ENDSCROLL )
{
    m_bScroll = FALSE;
    switch ( m_intRadio )
    {
        case VELOCITYMODE:
            m_ctrlSliderPos.SetPos(0);
            m_flEditPos = GetVelFromCtrl(m_ctrlSliderPos.Get-
            Pos());
            ::PCube_moveVel( CModData::m_intDevice,
            m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID ,
            0 );
            break;

        case CURRENTMODE:
            m_ctrlSliderPos.SetPos(0);
            m_flEditPos = GetCurFromCtrl
            (m_ctrlSliderPos.GetPos());
            ::PCube_moveCurrent( CMod-Data::m_intDevice,
            m_pDialog->m_ptrCubeArr[m_intIndex]
            ->m_intCubeID ,
            0 );
            break;
    }
}

```

- Im Positionsmodus wird der Slider nach Freigabe des Controls auf die aktuelle Position des Moduls zurückgesetzt und folgt ihr bis zum Erreichen der Endposition. Dies wird in der Funktion `CModulDlg::DrawDialog()` durchgeführt.

9.12.3 Die Funktion `CModulDlg::DrawDialog()`

Im Timer des Hauptdialogs, der die aktuellen Moduldaten alle 200 ms ausliest, wird auch die Funktion `CModulDlg::DrawDialog()` aufgerufen. Diese Funktion wird, wie oben bereits beschrieben, als Memberfunktion der Klasse `CModulDlg` eingerichtet. Darin werden folgende Programmfunktionalitäten realisiert:

- Wie oben beschrieben, wird der Slider `m_ctrlSliderPos` im Rampenmodus (Positionsmodus), nachdem er freigegeben wurde, mit der aktuellen Position des Moduls versorgt und folgt so der Modulbewegung bis zum Erreichen der Endposition:

```
if ( m_intRadio == RAMPMODE && m_bScroll == FALSE )
{
    m_ctrlSliderPos.SetPos(GetPosForSlider());
}
```

- Die Auflösung des Sliders `m_ctrlSliderPos` muß hochgesetzt werden. Geschieht dies gar nicht oder nur beim Initialisieren des Dialogs in der Funktion `CModulDlg::OnInitDialog()`, so wird der Slider nicht aktualisiert, wenn er sich gerade am linken Ende (Position 0) des Sliderbereichs befindet:

```
m_ctrlSliderPos.SetTicFreq(1);
```

- Die im Moduldialog eingefügten Kontrollkästchen werden je nach aktuellem Modulstatus aktiviert und mit Häkchen versehen oder deaktiviert. Dies geschieht durch Abfrage des Statusworts `CubeState` und binäre Addition mit dem jeweils interessierenden Statusbit (siehe dazu in der Header-Datei `m5.h` unter "Drive States M5"). Als Beispiel soll die Abfrage dienen, ob sich das Modul gerade in der Beschleunigungsphase einer Rampenbewegung befindet:

```
if ( m_pDialog->m_ptrCubeArr[m_intIndex]->CubeState &
STATE_RAMP_ACC )
{
    m_bCheckAcc = TRUE;
    m_ctrlCheckAcc.SetCheck(1);
    m_ctrlCheckAcc.EnableWindow(TRUE);
}
else
{
    m_bCheckAcc = FALSE;
    m_ctrlCheckAcc.SetCheck(0);
    m_ctrlCheckAcc.EnableWindow(FALSE);
}
```

- Die aktuellen Modulwerte für Position, Geschwindigkeit, Stromaufnahme und Schleppfehler, die im Frame *actual module parameter* befinden, werden neu geholt und angezeigt:

```

m_strActCur.Format("act cur: %.2f A\n", m_pDialog-
>m_ptrCubeArr[m_intIndex]->Cur);

switch ( m_pDialog->m_ptrCubeArr[m_intIndex]->m_nModuleType )
{
    case LINEAR_DRIVE:
        m_strActPos.Format("act pos: %.3f mm\n",
            m_pDialog->m_ptrCubeArr[m_intIndex]->ActPos * 1000);
        m_strActVel.Format("act vel: %.3f mm/s\n",
            m_pDialog->m_ptrCubeArr[m_intIndex]->ActVel * 1000);
        m_strDeltaPos.Format("act deltapos: %.3f mm\n",
            m_pDialog->m_ptrCubeArr[m_intIndex]->DeltaPos * 1000);
        break;

    case ROTARY_DRIVE:
        m_strActPos.Format("act pos: %.2f °\n",
            GetDegreeFromRad(m_pDialog->m_ptrCubeArr[m_intIndex]
            ->ActPos));
        m_strActVel.Format("act vel: %.2f °/s\n",
            GetDegreeFromRad(m_pDialog->m_ptrCubeArr[m_intIndex]
            ->ActVel));
        m_strDeltaPos.Format("act deltapos: %.3f °\n",
            GetDegreeFromRad(m_pDialog->m_ptrCubeArr[m_intIndex]
            ->DeltaPos));
        break;
}

```

- Beim Editieren eines Eingabefeldes wird eine boolsche Variable *m_bEdit* auf *true* gesetzt. Erst beim Verlassen des Editfelds (bei der Nachricht WM_KILLFOCUS des entsprechenden Controls) wird die Variable auf *false* zurückgesetzt. In der Funktion CModulDlg::DrawDialog() wird daraufhin ein UpdateData(false) ausgelöst (bewirkt, daß der Moduldialog aktualisiert und neu gezeichnet wird). Das geschieht, damit der Wert in der Editbox bei einer Eingabe nicht ständig vom alten Wert überschrieben wird. Dies bedeutet aber auch, daß während der Eingabe in ein Editfeld keine aktuellen Modulparameter angezeigt werden können. Betroffen sind sowohl Statusanzeigen in den Kontrollkästchen als auch die Werte im Frame *actual module parameter*.

```

if( m_bEdit == FALSE )
{
    UpdateData(false);
}

```

9.12.4 Die Befehlsschaltflächen (Buttons)

- Der **Go**-Button: Beim Drücken des **Go**-Buttons wird in Abhängigkeit vom eingestellten Bewegungsmodus ein Bewegungskommando abgeschickt, welches die in den Eingabefeldern eingestellten Bewegungsparameter (Rampenmodus: `m_flEditPos` = Position, `m_flEditAcc` = Beschleunigung, `m_flEditVel` = Geschwindigkeit; Geschwindigkeitsmodus: `m_flEditPos` = Geschwindigkeit; Strommodus: `m_flEditPos` = Stromaufnahme) übernimmt. Beispiel für den Rampenmodus (Positionsmodus):

```

if ( m_intRadio == RAMPMODE )
{
    switch ( m_pDialog->m_ptrCubeArr[m_intIndex]
            ->m_nModuleType )
    {
        case LINEAR_DRIVE:
            m_ctrlSliderAcc.SetPos(GetAcc-
                ForSlider(m_flEditAcc / 1000));
            m_ctrlSliderVel.SetPos(GetVel-
                ForSlider(m_flEditVel / 1000));
            ::PCube_moveRamp( CModData::m_intDevice,
                m_pDialog->m_ptrCubeArr[m_intIndex]
                ->m_intCubeID,
                m_flEditPos / 1000,
                m_flEditVel / 1000,
                m_flEditAcc / 1000 );
            break;

        case ROTARY_DRIVE:
            m_ctrlSliderAcc.SetPos(GetAccForSlider(GetRad-
                FromDegree(m_flEditAcc)));
            m_ctrlSliderVel.SetPos(GetVelForSlider(GetRad-
                FromDegree(m_flEditVel)));
            ::PCube_moveRamp( CModData::m_intDevice,
                m_pDialog->m_ptrCubeArr[m_intIndex]
                ->m_intCubeID,
                GetRadFromDegree(m_flEditPos),
                GetRadFromDegree(m_flEditVel),
                GetRadFromDegree(m_flEditAcc) );
            break;
    }
}

```

Im Geschwindigkeits- bzw. Strommodus dauert die Bewegung so lange an, bis ein HALT-Kommando die Fahrt beendet oder die Endlagen erreicht sind.

- Der **Halt**-Button: Beim Betätigen des **Halt**-Buttons wird das Kommando `::PCube_haltModule()` an das dem jeweiligen Moduldialog zugeordnete Modul geschickt. Das Kommando löst ein sofortiges Beenden der Modulbewegung und das Aktivieren der Bremse aus. Der nun gesetzte *Halt*-Flag zeigt einen Sicherheitsstatus an, der nur durch Drücken des **Reset**-Knopfes wieder aufgehoben werden kann.

```

int retval = ::PCube_haltModule( CModData::m_intDevice,
    m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID );

```

- Der **Reset**-Button: Das Auslösen des **Reset**-Knopfes bewirkt das Senden des `::PCube_resetModule()`-Befehls an das betroffene Modul. Dadurch werden z.B. das HALT-Flag sowie einige Fehler-Flags zurückgesetzt. Der Befehl muß auch nach einer erfolgreichen Referenzfahrt gegeben werden.

```
int retval = ::PCube_resetModule( CModData::m_intDevice,
m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID );
```

Des Weiteren werden die Werte für Geschwindigkeit und Beschleunigung im Rampenmodus wieder auf die Grundeinstellungen zurückgesetzt, die bei der Modulinitialisierung vorgegeben wurden.

- Der **Init**-Button: Durch Betätigen der **Init**-Befehlsschaltfläche wird der Befehl `::PCube_syncModule()` an das Modul gesendet, wodurch die Homing Procedure (Referenzfahrt) ausgelöst wird. Nach erfolgreicher Referenzfahrt wird das Statusbit *HOK* gesetzt. Erst danach kann eine Fahrt im Rampenmodus ausgeführt werden.

```
retval = ::PCube_syncModule( CModData::m_intDevice,
m_pDialog->m_ptrCubeArr[m_intIndex]->m_intCubeID );
```

- Die <<<-Buttons: Die <<<-Buttons erlauben das Anzeigen oder Wegblenden von Fensterinhalten. Die Programmierung der Schaltflächen erfolgt äquivalent zu den am Anfang des Kapitels "Die Funktionsweise des Moduldialogs" beschriebenen Schritten. Der Status der Fenstergröße wird mit den beiden boolschen Variablen *m_bSize* (*true* = Bewegungsmodus sichtbar) und *m_bFlag* (*true* = Statusflags sichtbar) beschrieben.

9.12.5 Die Optionsfelder (Radio Buttons)

- Beim Betätigen der Optionsfelder wird die Variable *m_intRadio* gesetzt und damit der Bewegungsmodus vorgegeben. Gleichzeitig wird der Slider *m_ctrSliderPos* neu initialisiert (Rampenmodus: Slider wird auf aktuelle Modulposition gesetzt; Geschwindigkeits- und Strommodus: Slider wird in die Mitte gesetzt => neuer Nullpunkt). Im Geschwindigkeits- und Strommodus werden die Sliderbereichsgrenzen von *m_ctrSliderPos* neu gesetzt (± 500), um eine Bewegung in beide Richtungen (positive und negative Geschwindigkeits- bzw. Stromwerte) zu erlauben. Je nach Bewegungsmodus werden die neuen Bereichsgrenzen für die entsprechende Bewegungsart den Variablen *m_strMinPos* und *m_strMaxPos* mitgegeben, die sie links und rechts des Sliders *m_ctrSliderPos* anzeigen. Der Inhalt des Editfeldes *m_fEditPos* wird auf Null gesetzt, die Bezeichnung des Editfeldes dem Bewegungsmodus angepaßt. Abhängig vom Bewegungsmodus werden die Slider- und Editbox-Controls für Geschwindigkeit und Beschleunigung (unterhalb Frame *Move Category*) aktiviert oder deaktiviert (Rampenmodus: aktiv; Geschwindigkeits- und Strommodus: inaktiv). Dies soll beispielhaft für das Aktivieren des mit "Current" bezeichneten Optionsfeldes dargestellt werden:

```
float DefMinCur;
float DefMaxCur;

// neue Bereichsgrenzen für Bewegung im Strommodus setzen
DefMaxCur = m_pDialog->m_ptrCubeArr[m_intIndex]->DefMaxCur;
DefMinCur = -DefMaxCur;

// neue Bereichsgrenzen für Slider im Strommodus setzen
m_ctrlSliderPos.SetRangeMin(-500);
m_ctrlSliderPos.SetRangeMax(500);

// neue Bezeichnung für Eingabefeld setzen
m_ctrlStaticPos.SetWindowText("Cur:");

// Bewegungsmodus setzen (sind über Define-Konstanten am Anfang der
Implementierungsdatei definiert)
m_intrRadio = CURRENTMODE;

// Anzeige der neuen Bereichsgrenzen der Bewegung
m_strMinPos.Format("%.3f A\n", DefMinCur);
m_strMaxPos.Format("%.3f A\n", DefMaxCur);

m_flEditPos = 0;

// Setzt die Sliderposition auf den neuen Nullpunkt in der Mitte
m_ctrlSliderPos.SetPos( (m_ctrlSliderPos.GetRangeMax() -
m_ctrlSliderPos.GetRangeMin()) / 10 );
m_ctrlSliderPos.SetPos(0);

// Deaktiviert die Kontrollelemente für die Eingabe von Geschwind-
igkeits- und Beschleunigungsvorgaben im Rampenmodus
m_ctrlMinAcc.EnableWindow(FALSE);
m_ctrlMaxAcc.EnableWindow(FALSE);
m_ctrlEditAcc.EnableWindow(FALSE);
m_ctrlSliderAcc.EnableWindow(FALSE);
m_ctrlMinVel.EnableWindow(FALSE);
m_ctrlMaxVel.EnableWindow(FALSE);
m_ctrlEditVel.EnableWindow(FALSE);
m_ctrlSliderVel.EnableWindow(FALSE);

UpdateData(FALSE);
```


9.12.6 Tooltips (QuickInfos)

- In der Funktion `CModulDlg::OnInitDialog()` wird der Moduldialog für das Anzeigen von Tooltips vorbereitet:

```
this->EnableToolTips(TRUE);
```

- Bei Steuerelementen, die sich in einem nicht von `CFrameWnd` abgeleiteten Fenster befinden, was beispielsweise bei Steuerelementen innerhalb eines Dialogfeldes oder einer Formularansicht der Fall ist, muß dem übergeordneten Fenster eine Behandlungsfunktion für die Nachricht `TTN_NEEDTEXT` zur Verfügung gestellt werden, um QuickInfos für untergeordnete Steuerelemente darstellen zu können. Daher wird in der Nachrichtenzuordnung von `CModulDlg` (`BEGIN_MESSAGE_MAP`) folgender Eintrag eingefügt:

```
ON_NOTIFY_EX(TTN_NEEDTEXT, 0, OnToolTipNotify)
```

Darin sind:

`TTN_NEEDTEXT`: Nachricht, daß Text für die QuickInfos zur Verfügung gestellt werden muß,

`0`: QuickInfo-ID (immer Null),

`OnToolTipNotify`: Die aufzurufende Member-Funktion, wenn Text für diese Schaltfläche benötigt wird.

- Die Member-Funktion `BOOL CModulDlg::OnToolTipNotify(UINT id, NMHDR *pNMHDR, LRESULT *pResult)` wird mittels des Klassenassistenten neu angelegt. Der im folgenden Beispiel angegebene Funktionsaufbau (Quelle: MSDN) kann für die meisten Tooltip-Implementierungen genutzt werden:

```
TOOLTIPTEXT *pTTT = (TOOLTIPTEXT *)pNMHDR;
UINT nID = pNMHDR->idFrom;
if (pTTT->uFlags & TTF_IDISHWND)
{
    // idFrom ist der HWND des Werkzeugs
    nID = ::GetDlgCtrlID((HWND)nID);
    if(nID)
    {
        pTTT->lpszText = MAKEINTRESOURCE(nID);
        pTTT->hinst = AfxGetResourceHandle();
        return(TRUE);
    }
}
return(FALSE);
```

Abschließend werden Zeichenfolgenressourcen für die einzelnen Checkbox-Kontrollelemente in der String Table angelegt. Dazu öffnet man in der *Resource-View* Karteikarte im Workspacefenster der Visual C++ - Entwicklungsumgebung mit einem linken Mausklick auf das + die *PCubeDemo resources*. Dann wird der Ordner *String Table* geöffnet, so daß die eigentliche Ressource sichtbar wird. Ein Doppelklick öffnet die *String Table* Ressource. Am Ende der Tabelle befindet sich ein leeres umrahmtes Feld. Ein Doppelklick darauf öffnet ein Dialogfeld *String Properties*. Darin befindet sich ein Feld *ID*, in dem mittels einer Listbox die gewünschte Kontrollelement-ID ausgewählt werden kann. Dieser ID kann man nun die gewünschte Zeichenkette (Text des Tooltips) zuordnen.

10. Anhang

10.1 Weltweite Vertriebs- und Servicestellen

Weltweit

AMTEC GmbH

Pankstraße 8-10

13127 Berlin

Tel.: (+49 30) 47 49 90 - 0

Fax.: (+49 30) 47 49 90 - 99

E-Mail: info@amtec-robotics.com

Homepage: <http://www.amtec-robotics.com> oder <http://www.powercube.de>

Großbritannien und Irland

Camtec Automation UK Ltd.

PO Box 2563

Bath BA1 3YR - GB

Tel.: +44 1225 316 400

E-Mail: auto@camtech.ndirect.co.uk

Homepage: <http://www.camtech.co.uk>

Schweiz

Angst + Pfister AG Zürich

Thurgauerstraße 66

CH - 8052 Zürich

Tel.: (+41 1) 3 06 61 11

Fax.: (+41 1) 3 06 65 08

E-Mail: schweiz@angst-pfister.com

Homepage: <http://www.angst-pfister.com>

Singapur

Servo Dynamics Pte Ltd

No 10 Kaki Bukit Road 1

#01-30 Kaki Bukit Industrial Park Singapore 416175

Tel.: (+65) 844 02 88

Fax.: (+65) 844 00 70

E-Mail: servodynamics@servo.com.sg

Homepage: <http://www.servo.com.sg>

Korea (Süd)

Seoul Techno Company

Room#602 Mijin Bd.

#116-2, Samsung-Dong, Kangnam-Ku Seoul 135-090, Korea

Tel.: (+82) 2 539 - 8330 / 1

Fax: (+82) 2 539 - 8332

10.2 EG-Konformitätserklärung

Serien-Nr.: 1998 - 07 - 01

Der Unterzeichnende, welcher den Hersteller

AMTEC GmbH
Pankstraße 8 - 10
13127 Berlin

vertritt,
 erklärt hiermit die Übereinstimmung des Produktes

PowerCube™

mit den Bestimmungen der nachfolgend genannten EG-Richtlinie
 (einschließlich aller zutreffenden Änderungen)

Referenz-Nr.	Titel
89 / 336 / E W G	EMV - Richtlinie

und daß die nachfolgend benannten Normen zur Anwendung gelangt sind.

Nr.	Ausgabe	Titel	Teil
E N 500 81	1.92	Störaussendung	1
E N 500 82	3.95	Störfestigkeit	2

Berlin, 07. Juni 1998

AMTEC GmbH

Dr. Sawa Tschakarow
 Geschäftsführer

10.3 Frequently Asked Questions

Was muß ich tun, wenn sich das Modul außerhalb seiner Endlagen befindet?

Es gibt zwei Endlagen, die überschritten werden können.

Softwareendlagen

Die Softwareendlagen können bei Überlastbetrieb überfahren werden, weil die Bewegung nicht abrupt beendet werden kann (Massenträgheit) ohne den Stromregler zu überlasten. Sollten Sie Ihr Modul extrem überlasten, kann es sogar über die Hardwareendlagen fahren.

Außerdem ist es im reinen Strombetrieb (PCube_moveCurrent) möglich die Softwareentlagen zu überschreiten.

Hardwareendlagen

Die Hardwareendlagen können,

- mit dem Kommando PCube_moveCurrent(),
- von Hand, im stromlosen Zustand (PCube_moveCurrent(0)),

überfahren werden.

Es gibt zwei Möglichkeiten ein Modul wieder zwischen die Endlagen zu bringen.

mit Motorkraft:

geben Sie dem Modul das Kommando PCube_moveCurrent(positiver Stromwert). Der Stromwert sollte bei ca 10% des Nominalstromes liegen. Bewegt sich Ihr Modul gar nicht oder in die falsche Richtung geben Sie dem Modul das Kommando PCube_moveCurrent(negativer Stromwert). Wenn sich Ihr Modul wieder im Arbeitsbereich befindet, senden Sie PCube_haltModule() und anschließend PCube_resetModule() an Ihr Modul.

manuell:

geben Sie dem Modul das Kommando PCube_moveCurrent(0). Ihr Modul ist nun stromlos (achtung, auch die Bremse wird gelöst) und kann manuell positioniert werden.

Anschließend sollten Sie das Kommando PCube_resetModule() an das Modul senden.

Was muß ich tun, wenn ein Hindernis die Modulbewegung behindert?

Da sich dieser Fall nie ganz ausschließen läßt, gibt es zwei Schutzmechanismen. Diese Schutzmechanismen können eine Kollision zwar auch nicht verhindert, sorgen jedoch für eine Abschaltung des entsprechenden Moduls.

Schleppfehlermechanismus

Bei der Ausführung von Bewegungen gibt es immer eine momentane Soll- und Ist-Position. Die Differenz der beiden Positionen wird als Schleppfehler bezeichnet. Überschreitet der Schleppfehler einen im Modul gespeicherten Grenzwert, so schaltet das Modul mit einer Fehlermeldung ab.

Selbstüberwachung des Stromreglers

Die Stromregelung in den PowerCube™-Modulen besitzt einen eigenen Prozessor (Risc-CPU). Dieser Prozessor überwacht die Temperatur des Motors und des Brückenverstärkers, sowie die Stromaufnahme des Brückenverstärkers. Übersteigt die Temperatur einen bestimmten Wert, oder wird für mehr als 2 Sekunden der dreifachen Nennstrom verbraucht, schaltet das PowerCube™-Modul mit einer Fehlermeldung ab.

In beiden Fällen erhält man nach Statusabfrage eine Fehlermeldung und das Modul schaltet ab. Um das Modul dann wieder für Kommandos zugänglich zu machen, müssen Sie das Kommando `PCube_resetModule()` an das Modul senden.

Welche Bedeutung hat die Fehlermeldung "STATE_POW_VOLT_ERR" ?

Dieser Fehler signalisiert Ihnen das die Zwischenkreisspannung des Motors unter den zulässigen Minimalwert gesunken ist. Zu diesem Fehler kann es kommen, wenn Sie Ihre Module überlasten, bzw. Ihr Netzteil unterdimensioniert ist.

Wieviele Module kann ich an ein Interface anschließen?

Das ist unterschiedlich!

CAN-Interface: bis zu 31 Module

RS232 (ohne zusätzliche Einrichtungen) : bis zu 8 Module

RS 485 : bis zu 31 Module

Profibus: bis zu 31 Module

Ich bekomme häufiger die Fehlermeldung "STATE_COMM_ERROR". Was muß ich tun?

Überprüfen Sie zunächst ob der Bus mit dem richtigen Abschlußwiderstand versehen ist. Wenn dies nicht die Fehlerquelle war untersuchen Sie bitte die Verbindungskabel und Steckerverschraubung auf Defekte.

Kann ich Module mit RS232-Schnittstelle und Module mit CAN-Interface von einem gemeinsamen Steuergerät aus nutzen ?

Ja, allerdings müssen Ihre Module dafür konfiguriert sein. Sie sollten sich jedoch über den Sinn Gedanken machen. Alle Module, die in einem System zusammenarbeiten, sollten auch über den gleichen Bus angesteuert werden. Ein praktisches Beispiel für Ihre Anwendung wäre hingegen die Steuerung zweier Teilsysteme mit unterschiedlichen Ansteuerungen vornehmen zu wollen.

Was geschieht, wenn ich einem Modul den Befehl gebe, über eine seiner Endlagen zu fahren?

Sie können nur mit dem Kommando `PCube_moveCurrent()` über die Endlagen fahren.

Verwenden Sie z.B das Kommando `PCube_moveRamp(...)`, dann wird das Modul dieses Kommando nur bis zu seiner Endlage durchführen. Es ist nicht möglich über die im Modul eingebrannten Parameter hinaus zu fahren!

Ein Beispiel:

Sie haben ein Modul, mit dem Sie nur in X-Richtung fahren können. Sie stehen bei $X=0$ und X_{max} sei 1000.

Nun wollen Sie die folgende Bewegung ausführen: Bewegung um 1200 und anschließend Bewegung um - 400.

Sie erwarten also von Ihrem Modul, daß es nach den beiden Bewegungen auf der Position 800 steht. Das tut es aber nicht. Es steht auf der Position 600, weil die erste Bewegung auf $X=1000$ umgerechnet wurde. Auch hier wieder der Hinweis, fragen Sie in regelmäßigen Abständen den Modulstatus ab, damit Sie erfahren ob Ihr Modul eine Fehlermeldung generiert.

Wie erfolgt die Einbindung der Module an die Busleitung?

Sie können unter RS232 kein Modul entfernen, weil dann das ganze System nicht mehr läuft. Bei Benutzung eines CAN-Interfaces wird die Systemkommunikation an der Stelle unterbrochen, an der ein Modul aus der Busleitung entfernt wird.

Arbeiten Module mit unterschiedlichen Betriebssystemversionen zusammen?

Ja. Sie müssen nur darauf achten, daß Ihre Anwendungssoftware einen reduzierten Befehlssatz verwendet, nämlich den der ältesten Softwareversion die in Ihrem System vorhanden ist (Abwärtskompatibilität). Wir versorgen Sie aber auch gern mit einer neuen Version.

Kann ich die Module an Netzspannung anschließen?

Nein das ist nicht möglich. Sie sollten unbedingt ein ausreichend dimensioniertes Netzteil verwenden, d.h. je nach Baugröße und Modulanzahl Gleichspannung 24 V oder 48 V mit 5 A, 10 A, 20 A oder 40 A.

Was muß ich beachten, wenn ich an einem Modul die Endlagen verändert habe?

Sie müssen unbedingt die Parameter des Moduls anpassen (MinPos und MaxPos). Das tun Sie am besten mit EasyConfig. Anschließend sollten Sie eine Referenzfahrt durchführen.

Wenn ich einen Parameter geändert habe, bleibt diese Änderung dann für alle Zeiten erhalten?

Wenn Sie Parameter mit Ihrem Anwendungsprogramm ändern, so bleiben die Änderungen nur solange erhalten bis Sie das Modul abschalten. Die von Ihnen geänderten Parameter werden nicht dauerhaft im Modul abgelegt. Das können Sie nur mit EasyConfig tun.

Was heißt CAN high/low?

Die Übertragung im CAN-Protokoll erfolgt als Differenzsignal-Übertragung (Zweidraht). Die Information befindet sich in der Differenz der beiden Signale. Die High-Leitung kann Signalwerte zwischen 2,75 V und 4,5 V annehmen und die Low-Leitung kann Signalwerte zwischen 0,5 und 2,25 V annehmen.

Sind die Leitungen 2 und 5 im Kabel ebenfalls verdreht ?

Die Datenleitungen 1 und 3 sowie 2 und 5 sind gedreht angeordnet und jeweils geschirmt. Beide Abschirmungen sind mit Pin 4 (Masse) verbunden.

Wie oft können die Stecker gesteckt werden?

Die Pins der Stecker und Buchsen sind vergoldet, um optimale Leitungseigenschaften zu gewährleisten. Leider ist diese Vergoldung nicht dauerhaft, wenn die Verbindung oft gesteckt wird. Sie sollten die Verbindung nicht öfter als 150 mal trennen und wieder verbinden. Sie sollten die Verbindung auch unbedingt verschrauben.

Gibt es eine Tabelle für die möglichen Einstellungen aller Module in EasyConfig?

Nein.

Woher erhalte ich die neuesten Versionen von Hard- und Software?

Entweder von Ihrem Vertragshändler oder direkt bei AMTEC. Hier stehen Ihnen die neusten Versionen zum Download bereit. Die Internetadressen lauten:

<http://www.amtec-robotics.com> und <http://www.powercube.de>

Ist es möglich, neue Software auf alter Hardware zu betreiben?

Sie können auf alten Modulen (V1.4.xx, V2.3.22 bis V2.4.10) nicht das allerneueste Betriebssystem installieren. Das wird von dem Programm easyConfig geprüft und verhindert. EasyConfig kann erst ab Modulversionen 2.5.00 eingesetzt werden. Wir unterscheiden zwei Generationen von Modulen.

MORSE-Module (2.3.22 bis 2.4.10)

PowerCube™-Module (ab Version 2.5.00)

Kann es zur Überlastung der Module kommen?

Wenn Sie einige Grundregeln beachten kann es zu keiner Überlastung der Module kommen. Dem Datenblatt Ihres Moduls können Sie den nominalen und maximalen Strombedarf entnehmen. Die Module können dauerhaft mit dem Nominalstrom betrieben werden. Kurzzeitig (2 Sekunden) dürfen Sie den dreifachen Nominalstrom ziehen. Anschließend muß unbedingt der Strombedarf für mindestens 20 Sekunden im Nominalbereich liegen.

Bei den Modulen handelt es sich um mechatronische Elemente, deren Bauteile fettgeschmiert sind. Die Maximaldaten der Module sollten also erst nach einer Warmlaufphase abgefordert werden.

Nach einem Halt-Kommando reagiert mein Module nicht mehr, was habe ich falsch gemacht?

Wenn Sie ein Halt-Kommando an ein Modul senden wird das Modul angehalten und in einen Sicherheitszustand überführt. Das Modul kann erst wieder in Betrieb genommen werden, nachdem das PCube_resetModule0-Kommando gesendet wurde. Sie sollten dann mit dem letzten Kommando vor dem Halt fortfahren.

Wie sensibel ist die Stromregelung, kann man die Stromwerte für Kraftmessungen heranziehen?

Sie können die Stromwerte für Kraftmessungen heranziehen. Dafür müssen Sie sich aber selbst eine Strom/Krafttabelle erzeugen, indem Sie mit einem Kraftmesser die Kräfte bei bestimmten Stromwerten aufnehmen. Beachten Sie aber, daß nur bei Motoren ohne Getriebe (System Motor) und vergleichbaren Betriebstemperaturen die Abweichung von dieser Nennkurve nicht größer als $\pm 5\%$ wird. Bei Getrieben (Drive Unit) und variierenden Betriebstemperaturen können bereits nach 30 Minuten Abweichungen um $\pm 80\%$ auftreten. Daher sollten Kraftmessungen mittels Stromwerten nur nach Rücksprache mit AMTEC vorgenommen werden.

Muß die erworbene Software registriert werden?

Die M5DII muß nicht registriert werden, da sie nur sinnvoll in Zusammenarbeit mit PowerCube™-Modulen funktioniert. Alle Anwendungsprogramme sollten Sie registrieren lassen