Programmierhandbuch für Schrittmotor-steuerungen

Gültig ab Firmware 25.08.2011

NANOTEC ELECTRONIC GmbH & Co. KG
Kapellenstraße 6
D-85622 Feldkirchen bei München

Tel. +49 (0)89-900 686-0
Fax +49 (0)89-900 686-50
info@nanotec.de

Impressum

Nanotec® Electronic GmbH & Co. KG
Kapellenstraße 6
D-85622 Feldkirchen bei München

Tel.: +49 (0)89-900 686-0
Fax: +49 (0)89-900 686-50

Internet: www.nanotec.de

Originalbetriebsanleitung

Version/Änderungsübersicht

Version	Datum	Änderungen
V1.0	10.02.2009	Neuerstellung Befehlsreferenz (Firmware-Version 04.12.2008)
V2.0	11.12.2009	Befehlserweiterungen (Firmware-Version 10.10.2009), Ergänzung um Java-Programmierung und COM-Schnitt-stellen-Programmierung, deshalb Umbenennung in „Programmierhandbuch“
V2.1	28.01.2010	Befehlserweiterung
V2.2	11.02.2010	Befehlserweiterung Jerkfree-Rampe
V2.3	08.04.2010	Neuer Hinweis: Java-Programme und serielle Kommunikation gleichzeitig möglich
V2.4	03.11.2010	Befehlserweiterung und -korrekturen für serielle Kommunikation und Java-Programmierung; COM-Schnittstellen-Programmierung neu überarbeitet
V2.5	03.11.2011	Befehlserweiterung für COM-Schnittstellen-Programmierung und Korrekturen
V2.6	05.04.2012	Aktualisierung der Befehlsreferenz. Neu: Typ des Drehgebers einstellen

Inhalt

1 Zu diesem Handbuch

Zielgruppe

Dieses Dokument richtet sich an Programmierer, die eine eigene Steuerungssoftware für die Kommunikation mit den Steuerungen für folgende Nanotec Motoren programmieren wollen:

- SMCI12
- SMCI33 *
- SMCI35
- SMCI36
- SMCI47-S *
- SMCP33
- PD2-N
- PD4-N
- PD6-N

* bitte nachfolgenden Hinweis beachten!

Hinweis zu SMCI33 und SMCI47-S

Bei Steuerungen mit einer Firmware älter als 30.04.2009 kann das Update auf die neue Firmware, die in diesem Handbuch beschrieben wird, nicht vom Kunden durchgeführt werden.

Bitte schicken Sie uns diese Steuerungen ein, wir führen das Update schnell und natürlich kostenlos für Sie durch.

Inhalt des Handbuchs

Dieses Handbuch enthält eine Referenz aller Befehle zur Steuerung von Nanotec Motoren (Kapitel 2). Kapitel 3 beschreibt die Programmierung mit Java (NanoJEasy), Kapitel 4 beschreibt die Programmierung über die COM-Schnittstelle.

Bitte dringend beachten!

Vor der Verwendung der Befehlsreferenzen der Nanotec Firmware zur Erstellung eigener Steuerungsprogramme ist dieses Programmierhandbuch sorgfältig durchzulesen.

Nanotec behält sich im Interesse seiner Kunden das Recht vor, technische Änderungen und Weiterentwicklungen von Hard- und Software zur Verbesserung der Funktionalität dieses Produktes ohne besondere Ankündigung vorzunehmen.

Dieses Handbuch wurde mit der gebotenen Sorgfalt zusammengestellt. Es dient ausschließlich der technischen Beschreibung der Befehlsreferenzen der Nanotec Firmware und der Programmierung über JAVA, bzw der COM-Schnittstelle. Die Gewährleistung erstreckt sich gemäß unseren allgemeinen Geschäftsbedingungen ausschließlich auf Reparatur oder Umtausch defekter Geräte der Nanotec Schrittmotoren, eine Haftung für Schäden und Fehler durch fehlerhafte Verwendung der Befehlsreferenzen in der Programmierung für eigene Motorensteuerungen ist ausgeschlossen.

Für Kritik, Anregungen und Verbesserungsvorschläge wenden Sie sich bitte an die im Impressum (Seite 2) angegebene Adresse oder per Email an: info@nanotec.de

2 Befehlsreferenz der Nanotec Firmware

2.1 Allgemeine Informationen

2.1.1 Aufbau eines Befehls

Aufbau von Steuerungsbefehlen

Ein Befehl beginnt immer mit dem Startzeichen `'#'` und endet mit einem Carriage-Return `'\r'`. Alle dazwischenliegenden Zeichen sind ASCII-Zeichen (also keine Steuerzeichen).

Nach dem Startzeichen folgt zuerst die Adresse des Motors als ASCII-Dezimalzahl. Dieser Wert darf von 1 bis 254 betragen. Wird ein `'*'` anstatt der Zahl gesendet, werden alle am Bus hängenden Steuerungen angesprochen.

Darauf folgt der eigentliche Befehl, der im Allgemeinen aus einem ASCII-Zeichen und einer optionalen ASCII-Zahl besteht. Diese Zahl ist in der Dezimaldarstellung mit einem führenden Vorzeichen (`'+'` und `'-'`) zu schreiben.

Sendet der Nutzer eine Einstellung an die Firmware, ist bei positiven Zahlen das `'+'`-Zeichen nicht zwingend erforderlich.

Hinweis:
Manche Befehle bestehen aus mehreren Zeichen und andere wiederum benötigen keine Zahl als Parameter.

Reaktion der Steuerung

Hat eine Steuerung einen Befehl als für sich gültig erkannt, sendet sie als Bestätigung den Befehl als Echo, aber ohne das Startzeichen `'#'` zurück.

Hat die Steuerung einen unbekannten Befehl empfangen, antwortet diese mit einem dem Befehl nachgestellten Fragezeichen `'?'`.

Die Antwort der Steuerung wird wie der Befehl selbst mit einem Carriage-Return `'\r'` abgeschlossen.

Werden an die Steuerung ungültige Werte übergeben, werden diese ignoriert, aber trotzdem als Echo zurückgesendet.

Beispiel

An die Steuerung übergebener Wert:	`'#1G10000000\r'`
Antwort der Firmware:	`'1G10000000\r'` (Wert wird ignoriert)

Gibt die Steuerung auf einen beliebigen Befehl als Antwort `'?clock'`, bedeutet dies, dass gerade der Takt-Richtungs-Modus aktiv ist und die Taktfrequenz größer 65 kHz beträgt. Hierbei ist keine serielle Kommunikation mehr möglich. Um die Kommunikation wieder zu ermöglichen, ist eine Taktfrequenz kleiner 65 kHz einzustellen.

Beispiele

Setzen des Verfahrweges von Steuerung 1:	`'#1s1000\r'` → `'1s1000\r'`
Starten eines Satzes:	`'#1A\r'` → `'1A\r'`
Ungültiger Befehl:	`'#1°\r'` → `'1°?\r'`

CanOpen Schnittstellen-Spezifikation

Hinweise zur Programmierung mit CanOpen finden Sie im entsprechenden Handbuch zur Schnittstelle unter www.nanotec.de.

2.1.2 Langes Kommandoformat

Verwendung

Mit der Einführung der neuen Firmware wurden Befehle eingeführt, die aus mehr als nur einem Zeichen bestehen. Diese Kommandos dienen zum Lesen und Ändern von Maschinenparametern. Da diese in der Regel nur bei der Inbetriebnahme eingestellt werden müssen, hat die langsamere Übertragungsgeschwindigkeit aufgrund der Länge des Befehls keine Auswirkungen auf den Betrieb.

Aufbau langer Befehle

Ein langer Befehl beginnt mit dem bereits beschriebenen Adressierungsschema (`'#<ID>'`). Darauf folgt ein Doppelpunkt, der den Beginn eines langen Befehls ankündigt. Es folgt das Schlüsselwort und der Befehl, gefolgt von einem Carriage-Return-Zeichen (`'\r'`), das das Ende des Befehls anzeigt.

Ein langer Befehl kann sich aus den Zeichen `'A'` bis `'Z'` bzw. `'a'` bis `'z'`, sowie dem Unterstrich (`'_'`) zusammensetzen. Es wird zwischen Groß- und Kleinschreibung unterschieden. Ziffern sind nicht erlaubt.

Schlüsselwörter

Folgende Schlüsselwörter sind für die langen Kommandos definiert:

:CL für die Regler-Einstellungen und Motoreinstellungen (Closed-Loop)

:brake für die Motorsteuerung

:Capt für den Scope-Mode

Reaktion der Steuerung

Die Antwort der Firmware beginnt nicht mit einem `'#'` wie die Anfrage des Nutzers.

Nach dem Schlüsselwort folgt bei positiven Werten in der Antwort ein `'+'`-Zeichen. Bei negativen Werten folgt ein `'-'`-Zeichen.

Beide Zeichen (`'+'` und `'-'`) können als Trennzeichen verwendet werden.

Wird ein unbekanntes Schlüsselwort gesendet (unbekanntes Kommando), antwortet die Firmware mit einem Fragezeichen nach dem Doppelpunkt.

Beispiel

Unbekannter Befehl: `'#<ID>:CL_gibt_es_nicht\r'`

Antwort der Firmware: `'<ID>:?\r'`

Befehl zum Lesen eines Parameters

Lesebefehl

Zum Lesen eines Parameters wird nach dem Ende des Befehlsnamens mit einem Carriage-Return-Zeichen abgeschlossen.

Lesebefehl: `'#<ID>:Schlüsselwort_Kommando_abc\r'`

Antwort der Firmware

Die Firmware antwortet mit einem Echo des Befehls und dessen Wert.

Antwort: `'<ID>:Schlüsselwort_Kommando_abc+Wert\r'`

Befehl zum Ändern eines Parameters

Änderungsbefehl

Beim Ändern eines Parameters folgt nach dem Befehlsnamen ein `'='`-Zeichen gefolgt vom einzustellenden Wert. Bei positiven Werten ist ein `'+'`-Zeichen nicht zwingend erforderlich, aber auch nicht verboten. Der Befehl wird mit einem Carriage-Return-Zeichen abgeschlossen.

Änderungsbefehl: `'#<ID>:Schlüsselwort_Kommando_abc=Wert\r'`

Antwort der Firmware

Die Firmware antwortet mit einem Echo des Befehls als Bestätigung.

Antwort: `'<ID>:Schlüsselwort_Kommando_abc=Wert\r'`

Sehen Sie dazu auch das nachfolgende Beispiel.

Beispiel

Der Aufbau der langen Kommandobefehle ist an folgendem Beispiel gezeigt:

„Auslesen der Polpaare des Motors“

Parameter lesen	`'#1:CL_motor_pp\r'`
Antwort der Firmware	`'1:CL_motor_pp+50\r'`
Parameter ändern	`'#1:CL_motor_pp=100\r'`
Antwort der Firmware	`'1:CL_motor_pp=100\r'`

2.2 Befehlsübersicht

Nachfolgend finden Sie eine Übersicht über die seriellen Befehle der Nanotec Firmware (Zeichen und Parameter):

2.3 Lesebefehl

Funktion

Eine ganze Reihe von Einstellungen, die mit einem bestimmten Befehl gesetzt werden können, können mit einem entsprechenden Lesebefehl ausgelesen werden.

Befehl

Zeichen	Parameter
`'Z + Parameter'`	Der Lesebefehl setzt sich aus dem Zeichen `'Z'` und dem Befehl für den entsprechenden Parameter zusammen

Beispiel

Auslesen des Verfahrweges: `'#1Zs\r'` → `'1Zs1000\r'`

Antwort der Firmware

Liefert den jeweils gewünschten Parameter zurück.

Beschreibung

Dient zum Auslesen der aktuell gesetzten Werte einiger Befehle. Das Auslesen des Verfahrweges geschieht beispielsweise mit `'Zs'`, worauf die Firmware mit `'Zs1000'` antwortet.

Soll der Parameter eines bestimmen Satzes gelesen werden, ist dem jeweiligen Befehl die Nummer des Satzes voranzustellen.

Beispiel: `'#1Z5s'` → `'1Z5s2000'`

Eine Liste der Satzbefehle findet sich unter „*2.4 Sätze*“.

2.4 Sätze

Speichern von Verfahrwegen

Die Firmware unterstützt das Speichern von Verfahrwegen in Sätzen. Diese Daten werden in einem EEPROM abgelegt und gehen somit auch im ausgeschalteten Zustand nicht verloren.

Im EEPROM finden 32 Sätze mit den Satznummern 1 bis 32 Platz.

Gespeicherte Einstellungen pro Satz

Folgende Einstellungen werden in jedem Satz gespeichert:

Einstellung	Parameter	Siehe Abschnitt	Seite
Positionsmodus	`'p'`	*2.6.6 Positionierart setzen (neues Schema)*	47
Verfahrweg	`'s'`	*2.6.7 Verfahrweg einstellen*	49
Anfangsschrittfrequenz	`'u'`	*2.6.8 Minimalfrequenz einstellen*	49
Maximalschrittfrequenz	`'o'`	*2.6.9 Maximalfrequenz einstellen*	50
Zweite Maximalschritt-frequenz	`'n'`	*2.6.10 Maximalfrequenz 2 einstellen*	50
Beschleunigungs-rampe	`'b'`	*2.6.11 Beschleunigungsrampe einstellen*	51
Bremsrampe	`'B'`	*2.6.12 Bremsrampe einstellen*	51
Maximalruck für Beschleunigungs-rampe	`':b'`	*2.5.37 Maximalen Ruck für Beschleunigungsrampe setzen*	38
Maximalruck für Bremsrampe	`':B'`	*2.5.38 Maximalen Ruck für Bremsrampe setzen*	38
Drehrichtung	`'d'`	*2.6.14 Drehrichtung einstellen*	52
Drehrichtungsumkehr bei Wiederholungssätzen	`'t'`	*2.6.15 Richtungsumkehr einstellen*	53
Wiederholungen	`'W'`	*2.6.16 Wiederholungen einstellen*	53
Pause zwischen Wiederholungen und Folgesätzen	`'P'`	*2.6.17 Satzpause einstellen*	54
Satznummer des Folgesatzes	`'N'`	*2.6.18 Folgesatz einstellen*	54

2.5 Allgemeine Befehle

2.5.1 Motortyp einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_motor_type'`	0 bis 2	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Legt den Typ des angeschlossenen Motors fest:

- Wert 0: Schrittmotor
- Wert 1: BLDC-Motor mit Hallsensoren
- Wert 2: BLDC-Motor mit Hallsensoren und Drehgeber

Auslesen

Mit dem Befehl `':CL_motor_type'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.2 Phasenstrom einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'i'`	0 bis 150	ja	u8 (integer)	steuerungsabhängig

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Phasenstrom in Prozent. Werte über 100 sollten vermieden werden.

Auslesen

Mit dem Befehl `'Zi'` kann der aktuell gültige Wert ausgelesen werden.

2.5.3 Phasenstrom im Stillstand einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'r'`	0 bis 150	ja	u8 (integer)	steuerungsabhängig

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Strom der Stromreduzierung in Prozent. Dieser Wert ist wie der Phasenstrom relativ zum Endwert und nicht relativ zum Phasenstrom. Werte über 100 sollten vermieden werden.

Auslesen

Mit dem Befehl `'Zr'` kann der aktuell gültige Wert ausgelesen werden.

2.5.4 Spitzenstrom für BLDC einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':ipeak'`	0 bis 150	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Spitzenstrom für BLDC-Motoren in Prozent. Dieser Wert muß mindestens so groß sein wie der eingestellte Phasenstrom, sonst wird der Phasenstrom-Wert verwendet

Auslesen

Mit dem Befehl `':ipeak'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.5 Strom-Zeitkonstante für BLDC einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':itime'`	0 bis 65535	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Strom-Zeitkonstante für BLDC-Motoren in ms. Diese legt die Dauer fest, für die der eingestellte Spitzenstrom fließen darf.

Auslesen

Mit dem Befehl `':itime'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.6 Schrittmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'g'`	1, 2, 4, 5, 8, 10, 16, 32, 64, 254, 255	ja	u8 (integer)	2 = Halbschritt

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Schrittmodus. Die übergebene Zahl entspricht der Anzahl der Mikroschritte pro Vollschritt mit Ausnahme des Wertes 254, welcher den Vorschubkonstanten-modus auswählt, und mit Ausnahme des Wertes 255, welcher den adaptiven Schrittmodus auswählt.

Vorschubkonstantenmodus enthalten in Firmware neuer als 15.03.2010.

Auslesen

Mit dem Befehl `'Zg'` kann der aktuell gültige Wert ausgelesen werden.

2.5.7 Motoradresse einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'m'`	1 bis 254	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Motoradresse. Es ist darauf zu achten, dass nur eine Steuerung angeschlossen ist und die neu gesetzte Adresse nicht bereits von einem anderen Motor belegt ist, sonst ist keine Kommunikation mehr möglich.

Außerdem müssen eventuell vorhandene Adressdrehschalter an der Steuerung auf 0 stehen, da sonst die durch die Schalter eingestellte Adresse verwendet wird.

Adresse 0 und 255 sind für Fehlerfälle des EEPROMS reserviert.

2.5.8 Motor-ID einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':mt'`	0 bis 2147483647	ja	u32	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die in NanoPro eingestellte ID des Motors zurück oder setzt diese.

Diese Motor-ID gibt eindeutig Motortyp, Motorbezeichung und Anschlussart an (z.B. ST5918 parallel angeschlossen) und dient dazu, in der Steuerung zu hinterlegen, welcher Motor gerade wie angeschlossen ist (wird von NanoPro z.B. zur Ermittlung des maximal zulässigen Phasenstroms verwendet).

Auslesen

Mit dem Befehl `':mt'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.9 Endschalterverhalten einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'l'`	0 bis 4294967295	ja	u32 (integer)	17442

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt das Endschalterverhalten. Der Integer-Parameter wird als Bitmaske interpretiert. Die Bitmaske hat 16 Bit.

„Freifahrt“ bedeutet, dass die Steuerung bei Erreichen des Schalters mit der eingestellten unteren Geschwindigkeit vom Schalter herunterfährt.

„Stopp“ bedeutet, dass die Steuerung bei Erreichen des Schalters sofort anhält. Der Schalter bleibt dabei gedrückt.

Verhalten des internen Endschalters bei Referenzfahrt:

Bit0: Freifahrt vorwärts
Bit1: Freifahrt rückwärts (Default-Wert)
Es muss genau eines der beiden Bits gesetzt sein.

Verhalten bei Auslösen des internen Endschalters bei Normalfahrt:

Bit2: Freifahrt vorwärts
Bit3: Freifahrt rückwärts
Bit4: Stopp
Bit5: Ignorieren (Default-Wert)
Es muss genau eines der vier Bits gesetzt sein.
Diese Einstellung ist dann sinnvoll, wenn der Motor sich nicht mehr als eine Umdrehung drehen darf.

Verhalten des externen Endschalters bei Referenzfahrt:

Bit9: Frei vorwärts
Bit10: Frei rückwärts (Default-Wert)
Es muss genau eines der beiden Bits gesetzt sein.

Verhalten des externen Endschalters bei Normalfahrt:

Bit11: Freifahrt vorwärts
Bit12: Freifahrt rückwärts
Bit13: Stopp
Bit14: Ignorieren (Default-Wert)
Es muss genau eines der vier Bits gesetzt sein.
Mit dieser Einstellung kann der Verfahrweg des Motors durch einen Endschalter hart begrenzt werden.

Auslesen

Mit dem Befehl `'Zl'` kann der aktuell gültige Wert ausgelesen werden.

2.5.10 Fehlerkorrekturmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'U'`	0 bis 2	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Modus der Fehlerkorrektur:

- Wert 0: Aus
- Wert 1: Korrektur nach einer Fahrt
- Wert 2: Korrektur während einer Fahrt

Bei einem Motor ohne Drehgeber muss dieser Wert explizit auf 0 gesetzt werden, sonst versucht dieser ständig zu korrigieren, weil er von Schrittverlusten ausgeht.

Die Einstellung „Korrektur während einer Fahrt“ ist aus Kompatibilitätsgründen vorhanden und entspricht dem Verhalten „Korrektur nach einer Fahrt“. Für eine tatsächliche Korrektur während der Fahrt sollte der Closed-Loop-Modus benutzt werden.

Auslesen

Mit dem Befehl `'ZU'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.11 Satz für Autokorrektur einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'F'`	0 bis 32	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Aus dem gewählten Satz (Integer) werden Geschwindigkeit und Rampe für die Korrekturfahrt verwendet.

Ist 0 eingestellt, so wird keine Korrekturfahrt durchgeführt, sondern sofort ein Fehler ausgelöst, wenn die Fehlerkorrektur (Befehl `'U'`) aktiviert ist.

Siehe Befehl *2.5.10 Fehlerkorrekturmodus einstellen* `'U'`.

Auslesen

Mit dem Befehl `'ZF'` kann der aktuell gültige Wert ausgelesen werden.

2.5.12 Encoderrichtung einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'q'	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Wenn der Parameter auf `'1'` gesetzt ist, wird die Richtung des Drehencoders umgekehrt.

Auslesen

Mit dem Befehl `'Zq'` kann der aktuell gültige Wert ausgelesen werden.

2.5.13 Ausschwingzeit einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'o'	0 bis 250	ja	u8 (integer)	8

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Ausschwingzeit in 10ms Schritten zwischen Ende der Fahrt und der Überprüfung der Position durch den Drehgeber an.

Dieser Parameter ist nur gültig für die Positionsprüfung nach der Fahrt.
Siehe Befehl *2.5.10 Fehlerkorrekturmodus einstellen* `'U'`.

Zwischen Wiederholungs- oder Folgesätzen wird die Position nur geprüft, wenn die Pausezeit (siehe Befehl *2.6.17 Satzpause einstellen* `'P'`) länger als die Ausschwingzeit ist.

Nach einem Satz wird zuerst die Ausschwingzeit abgewartet, bevor der Motor sich wieder bereit meldet.

Auslesen

Mit dem Befehl `'Zo'` kann der aktuell gültige Wert ausgelesen werden.

2.5.14 Maximale Abweichung Drehgeber einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'X'`	0 bis 250	ja	u8 (integer)	2

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die maximale Abweichung in Schritten zwischen Soll-Position und Drehgeber-Position an.

Bei Schrittmodi größer als 1/10-Schritt bei 1,8° und 1/5 Schritt bei 0,9° Motoren muss dieser Wert größer 0 sein, da der Drehgeber selbst dann eine geringere Auflösung als die Mikroschritte des Motors hat.

Auslesen

Mit dem Befehl `'ZX'` kann der aktuell gültige Wert ausgelesen werden.

2.5.15 Zähler für Vorschubkonstante einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':feed_const_num'`	0 bis 2147483647	ja	u32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Zähler für die Vorschubkonstante. Diese legt die Anzahl der Schritte pro Umdrehung der Motorwelle für den Vorschubkonstantenschrittmodus fest. Die Vorschubkonstante wird nur benutzt, wenn sowohl Zähler als auch Nenner einen Wert ungleich 0 haben. Ansonsten wird die Drehgeberauflösung benutzt.

Auslesen

Mit dem Befehl `':feed_const_num'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.16 Nenner für Vorschubkonstante einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':feed_const_denum'`	0 bis 2147483647	ja	u32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Nenner für die Vorschubkonstante. Diese legt die Anzahl der Schritte pro Umdrehung der Motorwelle für den Vorschubkonstantenschrittmodus fest. Die Vorschubkonstante wird nur benutzt, wenn sowohl Zähler als auch Nenner einen Wert ungleich 0 haben. Ansonsten wird die Drehgeberauflösung benutzt.

Auslesen

Mit dem Befehl `':feed_const_denum'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.17 Positionsfehler zurücksetzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'D'`	-100000000 bis +100000000	ja	s32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt einen Fehler der Drehüberwachung zurück und setzt die aktuelle Position auf die, die der Drehgeber meldet (bei Eingabe ohne Parameter, C wird gleich I gesetzt, siehe Abschnitt *2.5.18* und *2.5.19*).

Bei Eingabe mit Parameter wird C und I auf Parameterwert gesetzt.
Bsp.: `'D100'` → C=100; I=100

2.5.18 Fehlerspeicher auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'E'	–	nein	–	–

Antwort der Firmware

Liefert den Index des Fehlerspeichers mit dem zuletzt aufgetretenen Fehler.

Beschreibung

Die Firmware beinhaltet 32 Fehlerspeicher.

Es werden die letzten 32 Fehler gespeichert. Ist Speicherposition 32 erreicht, wird der nächste Fehler wieder auf Speicherposition 1 gespeichert. In diesem Fall beinhaltet Speicherposition 2 also den ältesten noch auslesbaren Fehlercode.

Mit diesem Befehl wird der Index des Speicherplatzes mit dem zuletzt aufgetretenen Fehler, sowie der entsprechende Fehlercode, ausgelesen.

Auslesen

Mit dem Befehl `'Z'` + Indexnummer + `'E'` kann die Fehlernummer des jeweiligen Fehlerspeichers ausgelesen werden.
Bsp.: `'Z32E'` liefert die Fehlernummer von Index 32.

Fehlercodes

```
//! Error codes for error byte in EEPROM
#define ERROR_LOWVOLTAGE      0x01
#define ERROR_TEMP            0x02
#define ERROR_TMC             0x04
#define ERROR_EE              0x08
#define ERROR_QEI             0x10
#define ERROR_INTERNAL        0x20
```

Bedeutung

Fehler	Bedeutung
LOWVOLTAGE	Unterspannung
TMC	Treiberbaustein hat einen Fehler zurückgemeldet.
EE	Fehlerhafte Daten im Eprom, z.B. Schrittauflösung ist 25tel-Schritt.
QEI	Positionsfehler
INTERNAL	Interner Fehler (gleichzusetzen mit dem Windows-Bluescreen).

Status der Steuerung

Der Status der Steuerung kann mit dem Befehl *2.5.22 Status auslesen* `'$'` ausgelesen werden.

Tritt einer der oben aufgelisteten Fehler auf, so wechselt die Steuerung in den Zustand „Nicht bereit“ (Status-Bit 0 = 0, siehe *2.5.22 Status auslesen*) und der Ausgang 3 (Fehlerausgang) wird gesetzt. Wenn der Fehler reversibel und behoben ist, so kann er mittels des Befehls `'D'` (siehe *2.5.17 Positionsfehler zurücksetzen*) zurückgesetzt werden. Die Steuerung wechselt dann wieder in den Zustand „Bereit“ und der Fehlerausgang wird zurückgesetzt.

2.5.19 Drehgeberposition auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'I'	–	nein	–	–

Antwort der Firmware

Liefert die aktuelle Position des Motors laut Drehgeber zurück.

Beschreibung

Bei Motoren mit einem Drehgeber gibt dieser Befehl die aktuelle Position laut Drehgeber in Motorschritten zurück. Solange der Motor keine Schritte verloren hat, stimmen die Werte des Befehls *2.5.20 Position auslesen* 'C' und des Befehls *2.6.4 Aktuellen Satz auslesen* '|' (Pipe) überein.

Es ist dabei aber zu beachten, dass der Drehgeber für Schrittmodi höher als 1/10 bei 1,8° Motoren und höher als 1/5 bei 0,9° Motoren über eine zu geringe Auflösung verfügt und deswegen trotzdem Differenzen zwischen den beiden oben genannten Werten auftreten.

2.5.20 Position auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'C'	–	nein	–	–

Antwort der Firmware

Liefert die aktuelle Position zurück.

Beschreibung

Liefert die aktuelle Position des Motors in Schritten des jeweils eingestellten Schrittmodus. Diese Position ist relativ zu der Position der letzten Referenzfahrt.

Verfügt der Motor über einen Winkelgeber, sollte dieser Wert mit dem des Befehls 'I' bis auf eine kleine Toleranz übereinstimmen.

Die Toleranz ist abhängig von Schrittmodus und Motortyp (0,9° oder 1,8°), da der Winkelgeber eine geringere Auflösung als der Motor im Mikroschrittbetrieb hat.

Der Wertebereich ist der einer 32Bit signed Integer (Wertebereich ± 100000000).

2.5.21 „Motor ist referenziert“ abfragen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':is_referenced'`	0 und 1	nein	u8 (integer)	0

Antwort der Firmware

Wurde der Motor bereits referenziert, wird `'1'` zurückgemeldet, ansonsten `'0'`.

Beschreibung

Parameter ist `'1'` nach der Referenzfahrt.

Siehe auch *2.5.17 Positionsfehler zurücksetzen*.

2.5.22 Status auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'$'`	–	nein	–	–

Antwort der Firmware

Liefert den Status der Firmware als Bitmaske zurück.

Beschreibung

Die Bitmaske hat 8 Bit.

Bit 0: 1: Steuerung bereit

Bit 1: 1: Nullposition erreicht

Bit 2: 1: Positionsfehler

Bit 3: 1: Eingang 1 ist gesetzt während Steuerung wieder bereit ist. Tritt dann auf, wenn die Steuerung über Eingang 1 gestartet wurde und die Steuerung schneller wieder bereit ist, als der Eingang zurückgesetzt wurde.

Bit 4 und 6 sind immer auf 1, Bit 5 und 7 immer auf 0 gestellt.

2.5.23 Firmwareversion auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'v'	–	nein	–	–

Antwort der Firmware

Liefert den Versionsstring der Firmware zurück.

Beschreibung

Rückgabestring setzt sich aus mehreren Blöcken zusammen:

'v' Echo des Befehls

' ' Trennzeichen (Space)

Hardware: Möglich: SMCI47-S, PD6-N, PD4-N, PD2-N, SMCI33, SMCI35, SMCI36, SMCI12, SMCP33

'_' Trennzeichen

Kommunikation: 'USB' oder 'RS485'

'_' Trennzeichen

Releasedatum: tt-mm-jjjj z.B. 26-09-2007

'-' Trennzeichen

Revisionsnummer : revXXXX, z.B. rev1234

Beispiel einer kompletten Antwort

```
'001v SMCI47-S_RS485_17-05-2011-rev3711\r'
```

2.5.24 Betriebszeit seit Firmware-Update auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':optime'	–	nein	–	–

Antwort der Firmware

Liefert die Betriebszeit der Steuerung zurück.

Beschreibung

Liefert die Betriebszeit der Steuerung seit dem letzten Firmware-Update in Sekunden zurück. Wird ein Firmware-Update durchgeführt, so wird der Wert auf 0 zurückgesetzt und die Zählung beginnt von vorn.

2.5.25 Funktion der Digitaleingänge einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':port_in_a'` bis `':port_in_h'`	0 bis 13	ja	u8 (integer)	Unterschiedlich je nach Eingang

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Funktion, die der jeweilige Digitaleingang übernimmt. Jede Funktion wird durch eine eindeutige Nummer repräsentiert:

Eingangsfunktion	Nummer
Benutzerdefiniert	0
Starte Satz / Fehlerreset	1
Satzauswahl Bit 0	2
Satzauswahl Bit 1	3
Satzauswahl Bit 2	4
Satzauswahl Bit 3	5
Satzauswahl Bit 4	6
Externer Referenzschalter	7
Trigger	8
Richtung	9
Freigabe	10
Takt	11
Taktrichtungsmodus Moduswahl 1	12
Taktrichtungsmodus Moduswahl 2	13

Benutzerdefiniert (0) bedeutet, dass der Ein-/Ausgang von der Firmware nicht verwendet wird und dem Benutzer als General Purpose I/O zur Verfügung steht.

Beispiele

- Festlegen des Eingangs 3 als Triggereingang bei Steuerung 1:
  `'#1:port_in_c8\r'`
- Festlegen des Eingangs 6 als Takteingang bei Steuerung 2:
  `'#2:port_in_f11\r'`

Auslesen

Mit den Befehlen `':port_in_a'` bis `':port_in_h'` ohne Argument kann die aktuell für den jeweiligen Eingang eingestellte Funktion ausgelesen werden.

2.5.26 Funktion der Digitalausgänge einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':port_out_a'` bis `':port_out_h'`	0 bis 2	ja	u8 (integer)	Unterschiedlich je nach Ausgang

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Funktion, die der jeweilige Digitalausgang übernimmt. Jede Funktion wird durch eine eindeutige Nummer repräsentiert:

Eingangsfunktion	Nummer
Benutzerdefiniert	0
Bereit	1
Fahrend	2

Benutzerdefiniert (0) bedeutet, dass der Ein-/Ausgang von der Firmware nicht verwendet wird und dem Benutzer als General Purpose I/O zur Verfügung steht.

Beispiele

- Festlegen des Ausgangs 1 zur Fahrend-Anzeige bei Steuerung 1:
  `'#1:port_out_a2\r'`
- Festlegen des Ausgangs 2 zur Bereit-Anzeige bei Steuerung 2:
  `'#2:port_out_b1\r'`

Auslesen

Mit den Befehlen `':port_out_a'` bis `':port_out_h'` ohne Argument kann die aktuell für den jeweiligen Ausgang eingestellte Funktion ausgelesen werden.

2.5.27 Eingänge maskieren und demaskieren

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'L'`	0 bis 4294967295	ja	u32 (integer)	0x0003003f

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Ungültige Werte werden ignoriert, d.h. die gesamte Maske wird verworfen.

Beschreibung

Diese Bitmaske hat 32 Bit.

Setzt eine Bitmaske, die die Nutzung der Ein- und Ausgänge durch den Nutzer zulässt. Ist das Bit der entsprechenden I/Os auf '1' gesetzt, verwendet die Firmware diese I/Os. Ist es auf '0', sind die I/Os für den Nutzer verwendbar. Siehe auch Befehl *2.5.30 Ausgänge setzen* `'Y'`.

Nachfolgend die Belegung der Bits:	Bit auf 1:
Bit0: Eingang 1	1
Bit1: Eingang 2	2
Bit2: Eingang 3	4
Bit3: Eingang 4	8
Bit4: Eingang 5	16
Bit5: Eingang 6	32
Bit16: Ausgang 1	65536
Bit17: Ausgang 2	131072
Alle anderen Bits sind '0'	alle auf 1: 196671

Achtung:
Wird ein Bit beim Setzen der Maske nicht angesprochen, wird es automatisch auf '0' gesetzt, unabhängig vom Zustand! Es müssen alle Bits auf einmal gesetzt werden.

Werden ungültige Bitmasken gesetzt, werden diese verworfen, auch wenn die Firmware diese korrekt bestätigt.

Auslesen

Mit dem Befehl `'Y'` kann die aktuell eingestellte Maske ausgelesen werden.

Beispiele

Alle Bits sollen auf '0' gesetzt werden:

Send: #1L0\r

Read: 1L0\r

Bit3 und Bit5 sollen auf '1' gesetzt werden:

Send: #1L20\r

Read: 1L20\r

'20' deshalb, weil Bit3 mit dem Wert 4 und Bit5 mit dem Wert 16 angesprochen wird, also 4 + 16 = 20.

2.5.28 Polarität der Ein- und Ausgänge umkehren

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'h'	0 bis 4294967295	ja	u32 (integer)	0x0003003f

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Ungültige Werte werden ignoriert, d.h. die gesamte Maske wird verworfen.

Beschreibung

Setzt eine Bitmaske, mit der der Nutzer die Polarität der Ein- und Ausgänge umkehren kann. Ist das Bit des entsprechenden I/Os auf `'1'` gesetzt, findet keine Umkehrung statt. Ist es auf `'0'`, ist die Polarität des I/O invertiert.

Nachfolgend die Belegung der Bits:

Bit0: Eingang 1

Bit1: Eingang 2

Bit2: Eingang 3

Bit3: Eingang 4

Bit4: Eingang 5

Bit5: Eingang 6

Bit16: Ausgang 1

Bit17: Ausgang 2

Alle anderen Bits sind `'0'`.

Werden ungültige Bitmasken gesetzt, werden diese verworfen, auch wenn die Firmware diese korrekt bestätigt.

Auslesen

Mit dem Befehl `'Zh'` kann die aktuell eingestellte Maske ausgelesen werden.

2.5.29 Debounce-Zeit für Eingänge setzen (Entprellen)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'K'	0 bis 250	ja	u8 (integer)	20

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Zeit in ms, während der nach einer ersten Flanke an einem Eingang nicht auf darauf folgende Flanken reagiert wird. Erst nach Ablauf dieser Entprellzeit wird auf neue Flanken reagiert (Verriegelungslogik). Eine laufende Entprellzeit eines Eingangs hat keinen Einfluss auf die Erkennung von Flanken auf den anderen Eingängen.

Auslesen

Mit dem Befehl `'ZK'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.30 Ausgänge setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'Y'`	0 bis 4294967295	ja	u32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Diese Bitmaske hat 32 Bit.

Setzt die Ausgänge der Firmware, sofern diese für die freie Verwendung mittels des Befehls ***2.5.27 Eingänge maskieren und demaskieren*** `'L'` maskiert sind.

Ausgang 1 entspricht Bit 16 und Ausgang 2 Bit 17.

Auslesen

Mit dem Befehl `'ZY'` kann der aktuell eingestellte Wert ausgelesen werden.

Zusätzlich wird der Status der Eingänge angezeigt.

Bit0: Eingang 1

Bit1: Eingang 2

Bit2: Eingang 3

Bit3: Eingang 4

Bit4: Eingang 5

Bit5: Eingang 6

Bit6: `'0'` wenn Drehgeber gerade am Indexstrich, sonst `'1'`

Bit 16: Ausgang 1 (so wie er vom Nutzer eingestellt ist, auch wenn die Firmware diesen gerade bedient)

Bit 17: Ausgang 2 (so wie er vom Nutzer eingestellt ist, auch wenn die Firmware diesen gerade bedient)

Alle anderen Bits sind 0.

2.5.31 EEPROM Byte auslesen (Read EE Byte)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'(E'`	0 bis 65535	nein	u16	0

Antwort der Firmware

Liefert den Wert des Bytes im EEProm an der übergebenen Adresse.

Beschreibung

Liest ein Byte aus dem EEProm aus und gibt den Wert dieses Bytes zurück.

2.5.32 EEPROM Reset durchführen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'~'	–	nein	–	–

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Stellt die Werkseinstellungen wieder her. Die Steuerung benötigt eine Sekunde bis neue Befehle angenommen werden.

Während des Resets sollte kein Motor angeschlossen sein. Nach dem Reset sollte die Steuerung wenige Sekunden von der Stromversorgung getrennt werden.

2.5.33 Automatisches Senden des Status einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'J'	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Ist der Parameter auf '1' gesetzt, sendet die Firmware von sich aus nach Ende einer Fahrt den Status. Siehe Befehl *2.5.22 Status auslesen* '$', mit dem Unterschied, dass statt dem '$' ein kleines 'j' gesendet wird.

Auslesen

Mit dem Befehl 'ZJ' kann der aktuell gültige Wert ausgelesen werden.

2.5.34 Bootloader starten

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'@S'	–	nein	–	–

Antwort der Firmware

Keine Antwort, Bootloader antwortet mit '@OK'

Beschreibung

Dieser Befehl weist die Firmware an, den Bootloader zu starten. Die Firmware antwortet selbst nicht auf den Befehl. Der Bootloader antwortet mit '@OK'.

Der Bootloader selbst benötigt diesen Befehl ebenfalls, damit er sich nicht automatisch nach einer halben Sekunde wieder beendet. Deswegen muss dieser Befehl so oft gesendet werden, bis der Bootloader mit '@OK' antwortet. Der Bootloader verwendet das gleiche Adressierungsschema wie die Firmware selbst.

2.5.35 Umkehrspiel einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'z'`	0 bis 9999	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt das Umkehrspiel in Schritten an.

Die Einstellung dient dazu, das Spiel von nachgeschalteten Getrieben bei einem Drehrichtungswechsel auszugleichen.

Hierzu macht der Motor bei einem Drehrichtungswechsel die im Parameter eingestellte Anzahl von Schritten, bevor er beginnt, die Position zu inkrementieren.

Auslesen

Mit dem Befehl `'Zz'` kann der aktuell gültige Wert ausgelesen werden.

2.5.36 Rampe setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':ramp_mode'`	0, 1 und 2	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Rampe in allen Modi:

- `'0'` = die Trapez-Rampe ist ausgewählt
- `'1'` = die Sinus-Rampe ist ausgewählt
- `'2'` = die Jerkfree-Rampe ist ausgewählt

Dieser Parameter gilt für alle Modi außer Takt-Richtungs- und Drehmomentmodus (da diese Modi generell keine Rampe verwenden).

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.5.37 Maximalen Ruck für Beschleunigungsrampe setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':b'	1 bis 100000000	ja	u32 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den maximalen Ruck für die Beschleunigung.

Auslesen

Mit dem Befehl `'Z:b'` kann der aktuelle Wert ausgelesen werden.

Hinweis

Die tatsächliche Rampe ergibt sich aus den Werten für `'b'` und `':b'`.

- `'b'` = maximale Beschleunigung
- `':b'` = maximale Änderung der Beschleunigung (max. Ruck)

2.5.38 Maximalen Ruck für Bremsrampe setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':B'	1 bis 100000000	ja	u32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den maximalen Ruck für die Bremsrampe.

Wenn der Wert auf `'0'` gesetzt ist, wird zum Bremsen der gleiche Wert wie zum Beschleunigen (`':b'`) verwendet.

Auslesen

Mit dem Befehl `'Z:B'` kann der aktuelle Wert ausgelesen werden.

Hinweis

Die tatsächliche Rampe ergibt sich aus den Werten für `'B'` und `':B'`.

- `'B'` = maximale Beschleunigung
- `':B'` = maximale Änderung der Beschleunigung (max. Ruck)

2.5.39 Wartezeit für Abschalten der Bremsspannung setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':brake_ta'`	0 bis 65535	ja	u16 (integer)	0

Einheit

ms

Antwort der Firmware

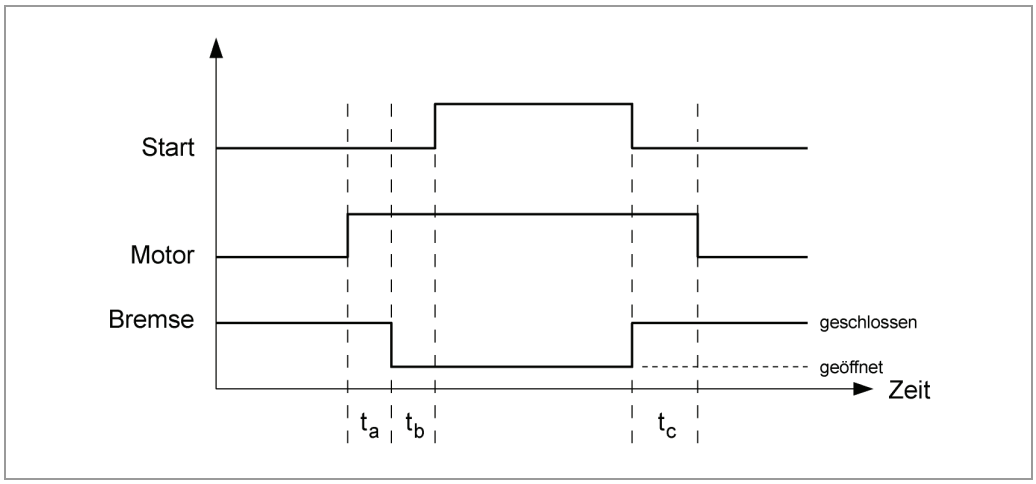
Bestätigt den Befehl durch Echo.

Beschreibung

Die externe Bremse kann über die folgenden Parameter eingestellt werden:

- Zeit ta:
  Wartezeit zwischen Einschalten des Motorstroms und Abschalten (Lösen) der Bremse in Millisekunden.
- Zeit tb:
  Wartezeit zwischen Abschalten (Lösen) der Bremse und Aktivieren der Bereitschaft in Millisekunden. Erst nach dieser Wartezeit werden Fahrbefehle ausgeführt.
- Zeit tc:
  Wartezeit zwischen Anschalten der Bremse und Abschalten des Motorstroms in Millisekunden. Der Motorstrom wird durch Rücksetzen des Freigabe-Eingangs abgeschaltet (siehe Abschnitt *2.5.25 „Funktion der Digitaleingänge einstellen“*).

Die Parameter geben jeweils Zeiten von 0 bis 65.536 Millisekunden an.
Defaultwerte der Steuerung nach einem Reset: 0 ms.



Beim Einschalten der Steuerung ist die Bremse zunächst aktiv und der Motor nicht bestromt. Zuerst wird der Motorstrom eingeschaltet und ta ms gewartet. Dann wird die Bremse gelöst und tb ms gewartet. Nach Ablauf von ta und tb werden Fahrbefehle ausgeführt.

Hinweis:
Während der Stromreduzierung wird die Bremse nicht aktiv geschaltet.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.5.40 Wartezeit für Motorbewegung setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':brake_tb'`	0 bis 65535	ja	u16 (integer)	0

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Wartezeit in Millisekunden zwischen Abschalten der Bremsspannung und dem Erlauben einer Motorbewegung.

Siehe auch Befehl *2.5.39 Wartezeit für Abschalten der Bremsspannung setzen* `'ta'` für weitere Informationen.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.5.41 Wartezeit für Abschalten Motorstrom setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':brake_tc'`	0 bis 65535	ja	u16 (integer)	0

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Wartezeit in Millisekunden zwischen Einschalten der Bremsspannung und dem Abschalten des Motorstroms.

Siehe auch Befehl *2.5.39 Wartezeit für Abschalten der Bremsspannung setzen* `'ta'` für weitere Informationen.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.5.42 Baudrate der Steuerung setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':baud'`	1 bis 12	ja	u8 (integer)	12

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Baudrate der Steuerung:

1 110
2 300
3 600
4 1200
5 2400
6 4800
7 9600
8 14400
9 19200
10 38400
11 57600
12 115200 (Defaultwert)

Hinweis:
Der neue Wert wird erst nach einem Neustart der Steuerung aktiviert (Strom aus/an).

Beispiel

Mit dem Befehl `'#1:baud=8'` wird die Baudrate der 1. Steuerung auf 14400 Baud gesetzt.

Auslesen

Mit dem Befehl `':baud'` kann der aktuell gültige Wert ausgelesen werden.

2.5.43 CRC-Prüfsumme einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':crc'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Schaltet die Überprüfung der seriellen Kommunikation mittels einer CRC-Prüfsumme (*cyclic redundancy check*) ein oder aus:

- Wert 0: CRC-Prüfung deaktiviert
- Wert 1: CRC-Prüfung aktiviert

Achtung:

Sobald die CRC-Prüfung aktiviert ist, muß zur Kommunikation mit der Steuerung mit jedem Befehl die korrekte CRC-Prüfsumme, vom Befehl durch einen Tabulator getrennt, mitgeschickt werden. Ist dies nicht der Fall, so führt die Steuerung den Befehl nicht aus und quittiert mit der Antwort `'<Befehl>?crc<Tab><Prüfsumme>'`.

Auslesen

Mit dem Befehl `':crc'` kann der aktuell eingestellte Wert ausgelesen werden.

Berechnung der CRC-Prüfsumme

Siehe Anhang.

2.5.44 Korrektur der Sinus-Kommutierung einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':cal_elangle_enable'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Schaltet die Korrektur der Sinus-Kommutierung ein oder aus:

- Wert 0: Korrektur deaktiviert
- Wert 1: Korrektur aktiviert

Hinweis:
Diese Funktion wirkt sich nur bei kalibrierten Motoren aus.

Auslesen

Mit dem Befehl `':cal_elangle_enable'` kann der aktuell eingestellte Wert ausgelesen werden.

2.5.45 Elektrischen Winkel setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':cal_elangle_data'`	0 bis 2147483647	nein	s32	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Wert des elektrischen Winkels.

Auslesen

Mit dem Befehl `'Z:cal_elangle_data'` kann der aktuelle Wert ausgelesen werden.

2.5.46 Hall-Konfiguration

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':hall_mode'`	0 bis 16777215	ja	u32	2371605

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Der Hall Mode gibt die Hall-Konfiguration eines angeschlossenen Brushless Motors als Integer-Wert an. Beispielsweise ist für die Motorentypen DB42S03, DB22M und DB87S01 der Wert 2371605 (0x243015 hexadezimal) zu verwenden, für die Motorentypen DB57 und DB22L dagegen der Wert 5309250 (0x510342 hexadezimal).

Für alle Nanotec Motoren kann der passende Wert bequem über NanoPro eingestellt werden.

Auslesen

Mit dem Befehl `':hall_mode'` kann der aktuell eingestellte Wert ausgelesen werden.

2.6 Satzbefehle

2.6.1 Motor starten

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'A'`	–	nein	–	–

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Startet die Fahrt mit den aktuell eingestellten Parametern.

2.6.2 Motor stoppen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'S'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Bricht die aktuelle Fahrt ab. Es werden die folgenden Rampen verwendet:

- Quickstop (H), wenn ohne Argument oder mit Argument `'0'`
- Bremsrampe (B), wenn mit Argument `'1'`

2.6.3 Satz aus EEPROM laden

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'y'`	1 bis 32	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Lädt die Satzdaten des im Parameter übergebenen Satzes aus dem EEPROM.

Siehe auch Befehl *2.6.5 Satz speichern* `'>'`.

2.6.4 Aktuellen Satz auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'|' (Pipe)	0 und 1	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo, wenn der Parameter auf `'1'` gesetzt wird. Sonst keine Antwort.

Beschreibung

Ist der Parameter auf `'0'`, antwortet die Firmware überhaupt nicht mehr auf Befehle, führt diese aber nach wie vor aus. Dies dient dazu, schnell Einstellungen an die Firmware zu schicken, ohne auf Bestätigungen zu warten.

Auslesen

Mit dem Befehl `'Z|'` schickt die Firmware alle Einstellungen des geladenen Satzes in einem Stück.

Mit `'Z5|'` werden die Daten des Satz 5 im EEPROM gesendet.

Das Format entspricht dem der jeweiligen Befehle.

Es ist zu beachten, dass bei der Antwort das `'|'`-Zeichen nicht gesendet wird. Siehe folgende Beispiele.

Beispiele

```
'#1Z|\r'
```

→ `'Zp+1s+1u+400o+860n+1000b+55800d+1t+0W+1P+0N+0\r'`

```
'#1Z5|\r'
```

→ `'Z5p+1s+400u+400o+1000n+1000b+2364d+0t+0W+1P+0N+0\r'`

2.6.5 Satz speichern

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'>'	1 bis 32	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dient zum Speichern der aktuell eingestellten Befehle (im RAM) in einem Satz im EEPROM. Der Parameter ist die Satznummer, in der die Daten gespeichert werden.

Während einer Fahrt sollte dieser Befehl nicht aufgerufen werden, da die aktuellen Werte sich durch Folgefahrten ändern.

Zu einem Satz gehören die folgenden Einstellungen bzw. Befehle:

Einstellung	Para-meter	Siehe Abschnitt	Seite
Positionsmodus	'p'	*2.6.6 Positionierart setzen (neues Schema)*	47
Verfahrweg	's'	*2.6.7 Verfahrweg einstellen*	49
Anfangsschrittfrequenz	'u'	*2.6.8 Minimalfrequenz einstellen*	49
Maximalschrittfrequenz	'o'	*2.6.9 Maximalfrequenz einstellen*	50
Zweite Maximalschritt-frequenz	'n'	*2.6.10 Maximalfrequenz 2 einstellen*	50
Beschleunigungsrampe	'b'	*2.6.11 Beschleunigungsrampe einstellen*	51
Bremsrampe	'B'	*2.6.12 Bremsrampe einstellen*	51
Maximalruck für Beschleunigungs-rampe	':b'	*2.5.37 Maximalen Ruck für Beschleunigungsrampe setzen*	38
Maximalruck für Brems-rampe	':B'	*2.5.38 Maximalen Ruck für Bremsrampe setzen*	38
Drehrichtung	'd'	*2.6.14 Drehrichtung einstellen*	52
Drehrichtungsumkehr bei Wiederholungssätzen	't'	*2.6.15 Richtungsumkehr einstellen*	53
Wiederholungen	'W'	*2.6.16 Wiederholungen einstellen*	53
Pause zwischen Wieder-holungen und Folge-sätzen	'P'	*2.6.17 Satzpause einstellen*	54
Satznummer des Folge-satzes	'N'	*2.6.18 Folgesatz einstellen*	54

2.6.6 Positionierart setzen (neues Schema)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'p'`	1 bis 19	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Wenn ungültige Werte eingestellt werden, wird die Positionierart `'p'` auf 1 gesetzt.

Beschreibung

Die Positionierarten `'p'` sind:

p	Modus	Funktionsweise
1	Relative Positionierung	Abhängig von den Satzparametern (siehe *2.4 Sätze*) wird relativ zur aktuellen Position der eingestellte Weg verfahren.
2	Absolute Positionierung	Abhängig von den Satzparametern (siehe *2.4 Sätze*) wird die vorgegebene Position als Absolut-Position angefahren. Die Drehrichtung ergibt sich aus der aktuellen und der vorgegebenen Position.
3	Interne Referenzfahrt	Abhängig von den Satzparametern (siehe *2.4 Sätze*) fährt der Motor so lange, bis der Indexstrich des Drehgebers erreicht wird. Danach fährt der Motor eine feste Anzahl von Schritten in die Gegenrichtung, so dass er den Indexstrich wieder verlässt. **Hinweis:** Dieser Modus ist nur für Motoren mit eingebautem und angeschlossenem Drehgeber zweckmäßig.
4	Externe Referenzfahrt	Abhängig von den Satzparametern (siehe *2.4 Sätze*) fährt der Motor so lange, bis der Endschalter erreicht wird. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe auch Befehl *2.5.9 Endschalterverhalten einstellen* `'l'`.
5	Drehzahlmodus	Wird der Motor gestartet, dreht der Motor bis zur Maximaldrehzahl mit der eingestellten Rampe hoch. Änderungen in der Geschwindigkeit oder Drehrichtung werden mit der eingestellten Rampe sofort angefahren, ohne dass der Motor zwischendurch gestoppt werden muss.
6	Flagpositioniermodus	Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.12 Trigger auslösen* `'T'` oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.7 Verfahrweg einstellen* `'s'`) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.10 Maximalfrequenz 2 einstellen* `'n'`).
7	Taktrichtungsmodus manuell links	Abhängig vom eingestellten Schrittmodus wird mit jedem Takt am Eingang 6 ein Schritt gefahren. Ist ein Eingangssignal als Richtung konfiguriert, wird entsprechend die Richtung gesetzt. Wird die Richtung nicht per Eingang gesetzt, wird entsprechend p=7 bzw. p=8 gefahren.
8	Taktrichtungsmodus manuell rechts	
9	Taktrichtungsmodus interne Referenzfahrt	Beim Starten des Modus wird eine interne Referenzfahrt durchgeführt (siehe p=3). Danach kann entsprechend des Takts am Eingang 6 schrittweise verfahren werden. Die Richtung wird durch einen Eingang mit der Einstellung „Richtung“ vorgegeben.
10	Taktrichtungsmodus	Beim Starten des Modus wird eine interne Referenzfahrt

p	Modus	Funktionsweise
	externe Referenzfahrt	durchgeführt (siehe p=4). Danach kann entsprechend des Takts am Eingang 6 schrittweise verfahren werden. Die Richtung wird durch einen Eingang mit der Einstellung „Richtung“ vorgegeben.
11	Analogmodus	Entsprechend des Pegels am Analogeingang (-10V bis +10V) und der Satzparameter wird die Drehzahl eingestellt.
12	Joystickmodus	Entsprechend des Pegels am Analogeingang (-10V bis +10V) und der Satzparameter wird die Drehzahl eingestellt. Zudem kann aber je nach Pegel in zwei Richtungen verfahren werden.
13	Analogpositioniermodus	Die Spannungshöhe am Analog-Eingang ist proportional zur gewünschten Position und ermöglicht dadurch ein Servo-Verhalten. Die Position wird entsprechend der Satzparameter angefahren.
14	HW-Referenzmodus	Dieser Modus dient zur definierten Initialisierung der Position im Closed-Loop-Betrieb (siehe auch interne Referenzfahrt, p=3)
15	Drehmomentmodus	Entsprechend des Pegels am Analogeingang wird ein festes Drehmoment eingestellt. Dieser Modus ist nur bei aktiviertem Closed-Loop-Modus anwendbar.
16	CL-Schnelltestmodus	Mit diesem Modus wird der Drehgeber-Indexversatz ermittelt.
17	CL-Testmodus	Mit diesem Modus werden Kalibrierwerte für den Closed-Loop Betrieb ermittelt. Vorausetzung ist ein angeschlossener und korrekt eingestellter Drehgeber, sowie ein aktivierter Closed-Loop-Betrieb.
18	CL-Autotune Modus	Mit diesem Modus werden die Reglerparameter für den Closed-Loop-Betrieb ermittelt. Voraussetzung ist ein angeschlossener und korrekt eingestellter Drehgeber, ein aktivierter Closed-Loop-Betrieb, sowie ein vorab erfolgreich durchgeführter CL-Testmodus.
19	CL-Schnelltestmodus 2	Mit diesem Modus wird ebenfalls der Drehgeber-Indexversatz ermittelt, wobei eine andere Methode verwendet wird.

Auslesen

Mit dem Befehl `'zp'` kann der aktuell gültige Wert ausgelesen werden.

Weitere Informationen

Weitere Informationen zu den Betriebsmodi können dem Benutzerhandbuch NanoPro entnommen werden.

2.6.7 Verfahrweg einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
's'	-100000000 bis +100000000	ja	s32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Befehl gibt den Verfahrweg in (Mikro-)Schritten an. Für die relative Positionierung sind nur positive Werte erlaubt. Die Richtung wird mit Befehl *2.6.14 Drehrichtung einstellen* `'d'` eingestellt.

Für die absolute Positionierung gibt dieser Befehl die Zielposition an. Negative Werte sind hier erlaubt. Die Drehrichtung aus Befehl *2.6.14 Drehrichtung einstellen* `'d'` wird ignoriert, da diese sich aus der aktuellen Position und der Zielposition ergibt.

Der Wertebereich ist der einer 32Bit signed Integer (Wertebereich ± 2^{31}).

Im Adaptiven Modus bezieht sich dieser Parameter auf Vollschritte.

Auslesen

Mit dem Befehl `'Zs'` kann der aktuell gültige Wert ausgelesen werden.

2.6.8 Minimalfrequenz einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'u'	1 bis 160000	ja	u32 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Minimalgeschwindigkeit in Hertz (Schritte pro Sekunde) an.

Bei einem Start eines Satzes beginnt der Motor, sich mit der Minimalgeschwindigkeit zu drehen. Er fährt dann mit der eingestellten Rampe (Befehl *2.6.11 Beschleunigungsrampe einstellen* `'b'`) bis zur Maximalgeschwindigkeit (Befehl *2.6.9 Maximalfrequenz einstellen* `'o'`) hoch.

Auslesen

Mit dem Befehl `'Zu'` kann der aktuell gültige Wert ausgelesen werden.

2.6.9 Maximalfrequenz einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'o'`	1 bis 1000000	ja	u32 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Maximalgeschwindigkeit in Hertz (Schritte pro Sekunde) an.

Die Maximalgeschwindigkeit wird erst nach Durchfahren der Beschleunigungsrampe erreicht.

Unterstützt höhere Frequenzen im Open-Loop-Betrieb:

- 1/2 Schritt: 32.000 Hz
- 1/4 Schritt: 64.000 Hz
- 1/8 Schritt: 128.000 Hz
- 1/16 Schritt: 256.000 Hz
- 1/32 Schritt: 512.000 Hz
- 1/64 Schritt: 1.000.000 Hz

Auslesen

Mit dem Befehl `'Zo'` kann der aktuell gültige Wert ausgelesen werden.

2.6.10 Maximalfrequenz 2 einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'n'`	1 bis 1000000	ja	u32 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Maximalgeschwindigkeit 2 in Hertz (Schritte pro Sekunde) an.

Die Maximalgeschwindigkeit 2 wird erst nach Durchfahren der Beschleunigungsrampe erreicht.

Unterstützt höhere Frequenzen im Open-Loop-Betrieb:

- 1/2 Schritt: 32.000 Hz
- 1/4 Schritt: 64.000 Hz
- 1/8 Schritt: 128.000 Hz
- 1/16 Schritt: 256.000 Hz
- 1/32 Schritt: 512.000 Hz
- 1/64 Schritt: 1.000.000 Hz

Dieser Wert findet ausschließlich im Flagpositioniermodus Anwendung. Siehe Befehl *2.6.6 Positionierart setzen (neues Schema)*.

Auslesen

Mit dem Befehl `'Zn'` kann der aktuell gültige Wert ausgelesen werden.

2.6.11 Beschleunigungsrampe einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'b'	1 bis 65535	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Beschleunigungsrampe an.

Zum Umrechnen der Parameters in die Beschleunigung in Hz/ms wird die folgende Formel verwendet:

Beschleunigung in Hz/ms = ((3000.0 / sqrt((float)<parameter>)) - 11.7).

Auslesen

Mit dem Befehl `'Zb'` kann der aktuell gültige Wert ausgelesen werden.

2.6.12 Bremsrampe einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'B'	0 bis 65535	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Bremsrampe an. Ist der gesetzte Wert 0, so bedeutet dies, dass für die Bremsrampe der für die Beschleunigungsrampe eingestellte Wert verwendet wird.

Auslesen

Mit dem Befehl `'ZB'` kann der aktuell gültige Wert ausgelesen werden.

2.6.13 Halterampe einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'H'`	0 bis 8000	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Halterampe an.
Bei einem Wert von 0 wird die Fahrt abrupt beendet.

Quickstop: Wird z.B. beim Überfahren des Endschalters verwendet.

Auslesen

Mit dem Befehl `'ZH'` kann der aktuell gültige Wert ausgelesen werden.

2.6.14 Drehrichtung einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'d'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Drehrichtung:

0: links

1: rechts

Auslesen

Mit dem Befehl `'Zd'` kann der aktuell gültige Wert ausgelesen werden.

2.6.15 Richtungsumkehr einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'t'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Bei Wiederholungssätzen wird die Drehrichtung des Motors bei jeder Wiederholung umgedreht falls dieser Parameter auf `'1'` gesetzt ist. Siehe Befehl *2.6.16 Wiederholungen einstellen* `'W'`.

Auslesen

Mit dem Befehl `'Zt'` kann der aktuell gültige Wert ausgelesen werden.

2.6.16 Wiederholungen einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'W'`	0 bis 254	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Anzahl der Durchgänge des aktuellen Satzes an.

Ein Wert von 0 bedeutet unendliche Wiederholungen.

Normalerweise ist ein Wert von 1 für einen Durchgang eingestellt.

Auslesen

Mit dem Befehl `'ZW'` kann der aktuell gültige Wert ausgelesen werden.

2.6.17 Satzpause einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'P'	0 bis 65535	ja	u16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Pause zwischen Wiederholungen von Sätzen oder zwischen Satz und Folgesatz in ms (Millisekunden) an.

Hat ein Satz keinen Folgesatz oder Wiederholung, wird die Pause nicht durchgeführt und der Motor ist sofort nach Ende der Fahrt wieder bereit.

Auslesen

Mit dem Befehl `'ZP'` kann der aktuell gültige Wert ausgelesen werden.

2.6.18 Folgesatz einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'N'	0 bis 32	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Nummer des Folgesatzes an. Ist der Parameter auf `'0'`, wird kein Folgesatz ausgeführt.

Auslesen

Mit dem Befehl `'ZN'` kann der aktuell gültige Wert ausgelesen werden.

2.7 Modusspezifische Befehle

2.7.1 Totbereich Joystickmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'='	0 bis 100	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Stellt den Totbereich im Joystickmodus ein.

Im Joystickmodus kann der Motor über eine Spannung am Analogeingang vorwärts und rückwärts verfahren werden.

Der Wertebereich in der Mitte zwischen Maximal- und Minimal-Spannung, bei dem der Motor sich nicht dreht, ist der Totbereich. Er wird in Prozent zur Größe des Bereichs angegeben.

Auslesen

Mit dem Befehl `'Z='` kann der aktuell eingestellte Totbereich ausgelesen werden.

2.7.2 Filter für Analog- und Joystickmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'f'	0 bis 255	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Im Analog- und Joystickmodus wird der Analogeingang verwendet, um die Drehzahl einzustellen. Mit dem Befehl `'f'` kann der Softwarefilter konfiguriert werden. Dabei gibt es zwei verschiedene Filterfunktionen je nach übergebenem Wert:

- 0 – 16: einfache Mittelwertbildung über die entsprechende Anzahl von Samples

Wert für Kommando „f“	Mittelwert über … Werte (1kHz Samplerate)
0	1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Wert für Kommando „f“	Mittelwert über … Werte (1kHz Samplerate)
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16

- 17 – 255: rekursiver Filter mit getrennt einstellbarer Zeitkonstante (Zeit, nach der sich der Filterausgang dem Filtereingang auf 50 % angenähert hat) und Hysterese (maximale Änderung des Werts am Filtereingang, gegenüber der der Filterausgang unempfindlich ist);
  f = (Bit 0-3: Zweierpotenz der Zeitkonstante in ms; Bit 4-7: Größe der Hysterese) + 16

		Hysterese in Bit (+-20mV)														
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
T in ms zum erreichen von 50% des Endwerts	**0**	32	48	64	80	96	112	128	144	160	176	192	208	224	240	0
	1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
	2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
	4	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
	8	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
	16	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
	32	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
	64	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
	128	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
	256	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
	512	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
	1024	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
	2048	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
	4096	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
	8192	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
	16384	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

Auslesen

Mit dem Befehl `'Zf'` kann der aktuell eingestellte Wert ausgelesen werden.

2.7.3 Minimalspannung für Analogmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'Q'	-100 bis +100	ja	s8 (integer)	-100

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt in 0,1V-Schritten den Bereichsanfang des Analogeingangs an.

Auslesen

Mit dem Befehl `'ZQ'` kann der aktuell gültige Wert ausgelesen werden.

2.7.4 Maximalspannung für Analogmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'R'	-100 bis +100	ja	s8 (integer)	100

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt in 0,1V-Schritten das Bereichsende des Analogeingangs an.

Auslesen

Mit dem Befehl `'ZR'` kann der aktuell gültige Wert ausgelesen werden.

2.7.5 Offset des Analogeingangs einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':aoa'	-32768 bis 32767	ja	s16	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Stellt den Offset des Analog-Eingangs ein.

Auslesen

Mit dem Befehl `':aoa'` kann der aktuell eingestellte Wert ausgelesen werden.

2.7.6 Verstärkung des Analogeingangs einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':aaa'`	0 bis 65535	ja	u16	32768

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Stellt die Verstärkung des Analog-Eingangs ein. Ein höherer Wert bewirkt eine höhere Steigung der Korrekturgeraden.

Auslesen

Mit dem Befehl `':aaa'` kann der aktuell eingestellte Wert ausgelesen werden.

2.7.7 Einschaltzähler zurücksetzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'%'`	1	ja	u32 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Der Einschaltzähler wird bei jedem Einschalten des Stroms um „1“ hochgezählt und gibt an, wie oft die Steuerung seit dem letzten Reset eingeschaltet wurde.
Wenn der Wert auf `'1'` gesetzt wird, wird der Einschaltzähler auf „0“ zurückgesetzt.

Auslesen

Mit dem Befehl `'Z%'` kann der aktuell gültige Wert ausgelesen werden.

2.7.8 Zeit bis zur Stromabsenkung einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`'G'`	0 bis 10000	ja	u16 (integer)	80

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Der Wert definiert die Wartezeit im Stillstand bis der Strom abgesenkt wird.

Auslesen

Mit dem Befehl `'ZG'` kann der aktuell gültige Wert ausgelesen werden.

2.7.9 Drehzahl erhöhen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'+'	–	nein	–	–

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Erhöht die Drehzahl im Drehzahlmodus um 100 Schritte/s.

2.7.10 Drehzahl verringern

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'-'	–	nein	–	–

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Verringert die Drehzahl im Drehzahlmodus um 100 Schritte/s.

2.7.11 Drehzahl auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':v'	– 2147483648 bis 2147483647	nein	s32	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die aktuelle Drehzahl des Motors zurück (nur im Drehzahlmodus).

Auslesen

Mit dem Befehl `':v'` kann der aktuelle Wert ausgelesen werden, wenn Closed-Loop aktiv ist.

2.7.12 Trigger auslösen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'T'	–	nein	–	–

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Auslöser für den Flagpositionsmodus.

Vor Auslösen des Trigger fährt der Motor mit konstanter Drehzahl.

Nach Auslösen des Triggers fährt der Motor noch die eingestellte Strecke ab der Position, bei der der Trigger ausgelöst wurde und stoppt dann.

2.7.13 Interpolationszeitraum für Taktrichtungsmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':clock_interp'`	0 bis 16383	ja	u16 (integer)	320

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Interpolationszeitraum für den Taktrichtungsmodus in 33 Mikrosekunden-Schritten.

Beispiel

Gesetzter Wert: 320 – ein Takt am Takteingang ist nach 320 * 33 µs =~ 10 ms abgearbeitet worden.

Auslesen

Mit dem Befehl `':clock_interp'` kann der aktuell eingestellte Wert ausgelesen werden.

2.8 Befehle für JAVA-Programm

2.8.1 Java-Programm an Steuerung übertragen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(J'	0 bis 268500991	ja	s32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Wird von NanoPro bzw. NanoJEasy selbstständig ausgeführt.

2.8.2 Geladenes Java-Programm starten

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JA'	0	nein	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl mit '(JA+', wenn das Programm erfolgreich gestartet wurde, bzw. mit '(JA-', wenn das Programm nicht gestartet werden konnte (kein gültiges oder gar kein Programm in der Steuerung geladen).

Beschreibung

Der Befehl startet das in der Steuerung geladene Java-Programm.

2.8.3 Laufendes Java-Programm stoppen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JS'	0	nein	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl mit '(JS+', wenn das Programm erfolgreich gestoppt wurde, bzw. mit '(JS-', wenn das Programm bereits beendet war.

Beschreibung

Der Befehl stoppt das gerade laufende Java-Programm.

2.8.4 Java-Programm beim Einschalten der Steuerung automatisch starten

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JB'	0 bis 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl mit `'(JB=1'`, wenn das Programm automatisch startet, bzw. mit `'(JB=0'`, wenn das Programm nicht automatisch startet.

Beschreibung

Mit diesem Befehl wird festgelegt, ob das Programm automatisch gestartet werden soll:

- `'0'` = Programm nicht automatisch starten
- `'1'` = Programm automatisch starten

Die Funktion sollte nur dann gewählt werden, wenn

- ein Java-Programm auf der Steuerung vorhanden ist
- das Programm bereits getestet wurde und in Ordnung ist
- keine Endlosschleifen mit Sende-Befehlen in dem Programm vorkommen

Sonst verursacht dieser Befehl beim Neustart der Steuerung einen Overflow an der Schnittstelle und das Programm kann nicht mehr angehalten werden.

2.8.5 Fehler des Java-Programms auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JE'	0 bis 255	nein	u8 (integer)	0

Antwort der Firmware

Liefert den Index des Fehlerspeichers mit dem zuletzt aufgetretenen Fehler. Siehe Abschnitt *3.8 Mögliche Java-Fehlermeldungen*.

Beschreibung

Dieser Befehl liest den letzten Fehler aus.

2.8.6 Warnung des Java-Programms auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JW'	0 bis 255	nein	u8 (integer)	0

Antwort der Firmware

Gibt die letzte aufgetretene Warnung zurück. Derzeit nur:

- '0' = keine Warnung
- 'WARNING_FUNCTION_NOT_SUPPORTED'

Beschreibung

Dieser Befehl liest die letzte Warnung aus.

2.9 Regelkreis-Einstellungen

2.9.1 Closed-Loop-Modus aktivieren

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_enable'`	0 bis 3	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Wird der Wert auf '`1`', '`2`' oder '`3`' gesetzt, wird die Firmware angewiesen, den Regelkreis zu aktivieren. Dieser wird aber erst dann aktiviert, wenn gewisse Voraussetzungen erfüllt sind:

Wert	Beschreibung
0	Der Regelkreis wird sofort deaktiviert.
1	Closed Loop wird aktiviert, sobald der Index erkannt wurde und die Steuerung wieder im Status „Bereit“ ist („Auto-Enable nach der Fahrt").
2	Closed Loop wird aktiviert, sobald der Index erkannt wurde („Auto-Enable während der Fahrt").
3	Closed Loop wird aktiviert, sobald ein kurzer CL-Testlauf durchgeführt wurde (Modus 19: ‚`p19`‘). Diese Einstellung ist ab Firmware Version 24-10-2011 verfügbar.

Wichtige Bedingungen

Folgende Bedingungen sind beim Aktivieren des Regelkreises unbedingt einzuhalten:

- Die Einstellungen von `':CL_Motor_pp'`, `':CL_rotenc_inc'` und `':CL_rotenc_rev'` müssen mit den technischen Daten des angeschlossenen Schrittmotors übereinstimmen.
  Siehe dazu Befehle *2.9.10 Polpaare des Motors einstellen*, *2.9.12 Anzahl der Inkremente einstellen* und *2.9.13 Anzahl der Wellenumdrehungen einstellen*.
- Jedes Mal, wenn ein neuer Motor angeschlossen wird (auch wenn es der gleiche Typ ist), muss eine Kalibrierfahrt durchgeführt werden (Modus 17: 'p17“).

ACHTUNG:
Wird eine der beiden Bedingungen nicht erfüllt, kommt es möglicherweise zu einem Hochdrehen des Motors bis über seine maximale mechanische Belastbarkeit!

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.2 Status Closed-Loop-Modus auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_is_enabled'`	0 und 1	nein	u8 (integer)	0

Antwort der Firmware

Meldet den Status zurück:

- `'0'` = nicht enabled
- `'1'` = enabled

Beschreibung

Liest den Status des Closed-Loop-Modus aus.

2.9.3 Regelungstyp für Drehzahlmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':speedmode_control'`	0 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt den Regelungstyp für den Drehzahlmodus an:

- `'0'` = Geschwindigkeitsregelkreis
- `'1'` = Positionsregelkreis

Mit diesem Parameter wird der Typ des Regelkreises festgelegt, welcher im Drehzahlmodus zur Regelung verwendet wird, wenn Closed Loop aktiviert ist.

Auslesen

Mit dem Befehl `':speedmode_control'` kann der aktuell eingestellte Wert ausgelesen werden.

2.9.4 Toleranzfenster für Endposition einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_position_window'`	0 bis 2147483647	ja	u32 (integer)	0

Einheit

Inkremente

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Ist der Regelkreis aktiv, ist dies ein Kriterium, wann die Firmware die Endposition als erreicht betrachtet. Der Parameter gibt hierzu ein Toleranzfenster in Inkrementen des Drehgebers an.

Ist die tatsächlich gemessene Position innerhalb der gewünschten Endposition + – der in diesem Parameter einstellbaren Toleranz und wird diese Bedingung für eine bestimmte Zeit eingehalten, gilt die Endposition als erreicht.

Die Zeit für dieses Zeitfenster wird im Parameter ':CL_position_window_time“ eingestellt. Siehe dazu Befehl *2.9.5 Zeit für Toleranzfenster der Endposition einstellen*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.5 Zeit für Toleranzfenster der Endposition einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_position_window_time'`	0 bis 65535	ja	u16 (integer)	0

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt die Zeit in Millisekunden für den Parameter ':CL_position_window“ an.
Siehe dazu Befehl *2.9.4. Toleranzfenster für Endposition einstellen*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.6 Maximal erlaubten Schleppfehler einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_following_error_window'`	0 bis 2147483647	ja	u32 (integer)	100

Einheit

Inkremente

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Ist der Regelkreis aktiv, gibt dieser Parameter den maximal erlaubten Schleppfehler in Inkrementen des Drehgebers an.

Weicht die Ist-Position zu einem beliebigen Zeitpunkt mehr als dieser Parameter von der Soll-Position ab, wird ein Positionsfehler ausgelöst und der Regelkreis wird abgeschaltet.

Zusätzlich kann mit dem Parameter ':CL_following_error_timeout“ eine Zeit angegeben werden, wie lange der Schleppfehler größer als die Toleranz sein darf, ohne einen Positionsfehler auszulösen. Siehe dazu Befehl *2.9.7 Zeit für maximalen Schleppfehler einstellen*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.7 Zeit für maximalen Schleppfehler einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_following_error_timeout'`	0 bis 65535	ja	u16 (integer)	100

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann eine Zeit in Millisekunden angegeben werden, wie lange der Schleppfehler größer als die Toleranz sein darf, ohne einen Positionsfehler auszulösen. Siehe dazu Befehl *2.9.6 Maximal erlaubten Schleppfehler einstellen*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.8 Maximal erlaubte Drehzahlabweichung

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_speed_error_window'`	0 bis 2147483647	ja	u32 (integer)	150

Einheit

Inkremente

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Ist der Regelkreis aktiv, gibt dieser Parameter die maximal erlaubte Drehzahlabweichung an.

Zusätzlich kann mit dem Parameter `':CL_speed_error_timeout'` eine Zeit angegeben werden, wie lange die Drehzahlabweichung größer als die Toleranz sein darf. Siehe dazu Befehl *2.9.9 Zeit für maximal erlaubte Drehzahlabweichung*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.9 Zeit für maximal erlaubte Drehzahlabweichung

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_speed_error_timeout'`	0 bis 65535	ja	u16 (integer)	250

Einheit

ms

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann eine Zeit in Millisekunden angegeben werden, wie lange die Drehzahlabweichung größer als die Toleranz sein darf. Siehe dazu Befehl *2.9.8 Maximal erlaubte Drehzahlabweichung*.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.10 Polpaare des Motors einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_motor_pp'`	1 bis 65535	ja	u16 (integer)	50

Einheit

Anzahl der Polpaare

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Der Parameter stellt die Anzahl der Polpaare des angeschlossenen Motors ein.

Hinweis:
Nach einem Ändern dieses Parameters **muss** die Firmware neu gestartet werden (Strom abstecken).

Die Anzahl der Polpaare entspricht ¼ der Anzahl der Vollschritte pro Umdrehung bei Schrittmotoren und 1/6 der Anzahl der Vollschritte pro Umdrehung bei BLDC-Motoren. Die üblichen Werte betragen derzeit 50 und 100 bei Schrittmotoren und 2 und 4 bei BLDC-Motoren. Falsche Werte haben zur Folge, dass der Regelkreis nicht funktioniert.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.11 Typ des Drehgebers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_rotenc_type'`	0 bis 3	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Typ des Drehgebers, der angeschlossen ist. Jeder Typ wird durch einen eindeutigen Wert repräsentiert:

Wert	Drehgeber-Typ
0	Kein Drehgeber
1	Inkrementeller Drehgeber mit Index
2	Inkrementeller Drehgeber ohne Index
3	Absoluter Drehgeber, singleturn

Dieser Befehl ist ab Firmware Version 24-10-2011 verfügbar.

2.9.12 Anzahl der Inkremente einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_rotenc_inc'`	1 bis 65535	ja	u16 (integer)	2000

Einheit

Inkremente

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt die Anzahl der Inkremente des Drehgebers pro einer bestimmten Anzahl von Wellenumdrehungen an. Die Anzahl der Umdrehungen kann mit dem Parameter `':CL_rotenc_rev'` eingestellt werden. Siehe dazu Befehl *2.9.13 Anzahl der Wellenumdrehungen einstellen*.

Hinweis:
Nach einem Ändern dieses Parameters **muss** die Firmware neu gestartet werden (Strom abstecken).

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.13 Anzahl der Wellenumdrehungen einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_rotenc_rev'`	1	ja	u16 (integer)	1

Einheit

Umdrehungen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt die Anzahl der Wellenumdrehungen für den Parameter `':CL_rotenc_inc'` an. Siehe Befehl *2.9.11 Typ des Drehgebers einstellen*.

Diese Einstellung exisitiert aus Kompatibilitätsgründen. Er sollte immer auf „1“ gesetzt werden. Andere Werte haben zur Folge, dass der Regelkreis nicht funktioniert. Die Umrechnung für die Fehlerkorrektur ohne Regelkreis funktioniert aber auch dann.

Hinweis:
Nach einem Ändern dieses Parameters **muss** die Firmware neu gestartet werden (Strom abstecken).

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.14 Zähler des P-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_v_Z'`	0 bis 65535	ja	u16 (integer)	1

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Proportionalteils des Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.15 Nenner des P-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_v_N'`	0 bis 15	ja	u8 (integer)	3

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Proportionalteils des Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.16 Zähler des I-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_v_Z'`	0 bis 65535	ja	u16 (integer)	1

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Integralteils des Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.17 Nenner des I-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_v_N'`	0 bis 15	ja	u8 (integer)	4

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Integralteils des Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.18 Zähler des D-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_v_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Differentialteils des Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.19 Nenner des D-Anteils des Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_v_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Differentialteils des Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.20 Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_csv_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Proportionalteils des kaskadierenden Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.21 Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_csv_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Proportionalteils des kaskadierenden Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.22 Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_csv_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Integralteils des kaskadierenden Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.23 Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_csv_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Integralteils des kaskadierenden Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.24 Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_csv_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Differentialteils des kaskadierenden Geschwindigkeitsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.25 Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_csv_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Differentialteils des kaskadierenden Geschwindigkeitsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.26 Zähler des P-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_s_Z'`	0 bis 65535	ja	u16 (integer)	100

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Proportionalteils des Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.27 Nenner des P-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_s_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Proportionalteils des Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.28 Zähler des I-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_s_Z'`	0 bis 65535	ja	u16 (integer)	1

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Integralteils des Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.29 Nenner des I-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_s_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Integralteils des Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.30 Zähler des D-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_s_Z'`	0 bis 65535	ja	u16 (integer)	200

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Differentialteils des Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.31 Nenner des D-Anteils des Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_s_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Differentialteils des Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.32 Zähler des P-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_css_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Proportionalteils des kaskadierenden Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'`gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.33 Nenner des P-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KP_css_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Proportionalteils des kaskadierenden Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.34 Zähler des I-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_css_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Integralteils des kaskadierenden Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.35 Nenner des I-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KI_css_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Integralteils des kaskadierenden Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.36 Zähler des D-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_css_Z'`	0 bis 65535	ja	u16 (integer)	0

Einheit

Zähler

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Zähler des Differentialteils des kaskadierenden Positionsreglers an.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.37 Nenner des D-Anteils des kaskadierenden Positionsreglers einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_KD_css_N'`	0 bis 15	ja	u8 (integer)	0

Einheit

Nenner in 2er Potenzen

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Dieser Parameter gibt den Nenner des Differentialteils des kaskadierenden Positionsreglers in 2er Potenzen an.

0 = 1

1 = 2

2 = 4

3 = 8

usw.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.9.38 Stützstellenabstand für Lastwinkelkurve einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_la_node_distance'`	1 bis 65535	ja	u16 (integer)	4096

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den Stützstellenabstand für die Lastwinkelkurve.

Auslesen

Mit dem Befehl `':CL_la_node_distance'` kann der aktuell eingestellte Wert ausgelesen werden.

2.9.39 Untergrenze für Kaskadenregler einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':ca'`	0 bis 2147483647	ja	u32	327680

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Befehl wird die Untergrenze eingestellt, ab welcher der Kaskadenregler zugeschaltet werden soll. Zusammen mit dem Befehl `':cs'` kann somit eine Hysterese eingestellt werden.Auslesen

Mit dem Befehl `':ca'` kann der aktuell eingestellte Wert ausgelesen werden.

2.9.40 Obergrenze für Kaskadenregler einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':cs'`	0 bis 2147483647	ja	u32	512

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Befehl wird die Obergrenze eingestellt, bis zu welcher der Kaskadenregler zugeschaltet ist. Zusammen mit dem Befehl `':ca'` kann somit eine Hysterese eingestellt werden.

Auslesen

Mit dem Befehl `':cs'` kann der aktuell eingestellte Wert ausgelesen werden.

2.9.41 Status des Kaskadenreglers auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':ce'`	0 und 1	nein	u8	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt an, ob der Kaskadenregler gerade aktiv ist.

Auslesen

Mit dem Befehl `':ce'` kann der aktuelle Wert ausgelesen werden.

2.10 Durch Testlauf ermittelte motorabhängige Lastwinkelwerte für den Closed-Loop-Mode

Allgemeines

Beim ersten Einsatz einer Steuerung mit dem dazugehörigen Motor muss ein Testlauf gestartet werden. Dabei werden motorabhängige Lastwinkelwerte für den Closed-Loop-Mode von der Steuerung ermittelt und fest gespeichert.

Diese Lastwinkelwerte sind mit NanoPro les- und speicherbar, um sie bei einem Steuerungswechsel wieder zurückschreiben zu können..

2.10.1 Offset Encoder/Motor auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_poscnt_offset'`	-32768 bis +32767	ja	s16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Der beim Testlauf ermittelte Offset zwischen Encoder und Motor wird ausgelesen.

2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_la_a'` bis `':CL_la_j'`	-32768 bis +32767	ja	s16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Die beim Testlauf ermittelten geschwindigkeitsabhängigen Lastwinkelmesswerte des Motors (Closed Loop load angle) werden mit folgenden Befehlen ausgelesen und können mit diesen auch wieder gesetzt werden:

- `':CL_la_a'`
- `':CL_la_b'`
- `':CL_la_c'`
- `':CL_la_d'`
- `':CL_la_e'`
- `':CL_la_f'`
- `':CL_la_g'`
- `':CL_la_h'`
- `':CL_la_i'`
- `':CL_la_j'`

Auslesen

Mit dem Befehl `':CL_la_a'` bis `':CL_la_j'` kann der aktuell eingestellte Wert ausgelesen werden.

2.10.3 Geschwindigkeitsmesswerte des Testlaufs auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_ola_v_a'` bis **`':CL_ola_v_g'`**	-32768 bis +32767	ja	s16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Die beim Testlauf ermittelten Geschwindigkeitsmesswerte (Closed Loop load angle velocity) werden ausgelesen:

- `':CL_ola_v_a'`
- `':CL_ola_v_b'`
- `':CL_ola_v_c'`
- `':CL_ola_v_d'`
- `':CL_ola_v_e'`
- `':CL_ola_v_f'`
- `':CL_ola_v_g'`

Diese Werte sind nur nach dem Testlauf auslesbar. Sie geben die Geschwindigkeiten an, bei denen der entsprechende Lastwinkel gemessen wurde. Sie werden nicht im EEPROM gespeichert und verschwinden folglich nach einem Neustart der Steuerung.

2.10.4 Strommesswerte des Testlaufs auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_ola_i_a'` bis `':CL_ola_i_g'`	-32768 bis +32767	ja	s16 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Die beim Testlauf ermittelten Strommesswerte (Closed Loop load angle current) werden ausgelesen:

- `':CL_ola_i_a'`
- `':CL_ola_i_b'`
- `':CL_ola_i_c'`
- `':CL_ola_i_d'`
- `':CL_ola_i_e'`
- `':CL_ola_i_f'`
- `':CL_ola_i_g'`

Diese Werte sind nur nach dem Testlauf auslesbar. Sie geben die Ströme an, bei denen der entsprechende Lastwinkel gemessen wurde. Sie werden nicht im EEPROM gespeichert und verschwinden folglich nach einem Neustart der Steuerung.

2.10.5 Lastwinkelmesswerte des Testlaufs auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':CL_ola_l_a'` bis `':CL_ola_l_g'`	-2147483648 bis +2147483647	ja	s32 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Die beim Testlauf ermittelten Lastwinkelmesswerte (Closed Loop load angle position) werden ausgelesen:

- `':CL_ola_l_a'`
- `':CL_ola_l_b'`
- `':CL_ola_l_c'`
- `':CL_ola_l_d'`
- `':CL_ola_l_e'`
- `':CL_ola_l_f'`
- `':CL_ola_l_g'`

Diese Werte sind nur nach dem Testlauf auslesbar. Sie geben die gemessenen Lastwinkel an und sind eine Kopie der CL_la_*-Werte. Sie werden nicht im EEPROM gespeichert und verschwinden folglich nach einem Neustart der Steuerung.

2.11 Scope-Mode

2.11.1 Integration eines Scopes

Beschreibung

Im Scope-Mode werden die zu messenden Größen ausgewählt und an den Motor übertragen. Der Motor führt anschließend eine Messung durch und übermittelt das Ergebnis in Echtzeit an die Steuerungssoftware NanoPro zurück.

- Die Daten werden Binär übertragen.
- Die Daten werden nach Priorität sortiert nacheinander übertragen.
- Jedes Datenpaket enthält als letztes Datenbyte eine CRC8 Checksumme.

Beispiele

Jede Datenquelle kann separat gewählt werden:

`':Capt_Time=10'` → sendet alle 10 ms die gewählten Daten

`':Capt_Time=0'` → beendet den Scope-Modus

`':Capt_sPos=1'` → die Sollposition wird gewählt

`':Capt_sPos=0'` → die Sollposition wird abgewählt

Defaultmäßig ist keine Datenquelle gewählt.

Datenwort wenn `':Capt_sCurr=1'` und `':Capt_iln=1'`

`':Capt_sCurr_BYTE'`

`':Capt_iln_BYTE_HI'`

`':Capt_iln_BYTE_LO CRC'`

2.11.2 Samplerate einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':Capt_Time'	0 bis 65535	ja	u16 (integer)	0

Priorität

–

Einheit

ms (Millisekunden)

Beschreibung

Der Parameter definiert das Zeitintervall in ms, in dem die gewählten Daten gesendet werden.

`'0'` deaktiviert die Scopefunktion.

Beispiel

`':Capt_Time=10'` → Sende alle 10 ms die gewählten Daten.

`':Capt_Time=0'` → beendet den Scope-Modus

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.11.3 Sollposition des Rampengenerators auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_sPos'`	0 und 1	ja	u8 (integer)	0

Priorität

1

Einheit

Schritte

Beschreibung

Liefert die Sollposition, die vom Rampengenerator erzeugt wird.

Beispiel

`'1'` = die Sollposition wird gewählt

`'0'` = die Sollposition wird abgewählt

2.11.4 Istposition des Drehgebers auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_iPos'`	0 und 1	ja	u8 (integer)	0

Priorität

2

Einheit

Schritte

Beschreibung

Liefert die aktuelle Drehgeberposition.

Beispiel

`'1'` = die Istposition wird gewählt

`'0'` = die Istposition wird abgewählt

2.11.5 Sollstrom der Motoransteuerung auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_sCurr'`	0 und 1	ja	u8 (integer)	0

Priorität

3

Einheit

keine

32767 entspricht 150% des Maximalstroms (Wert kann auch negativ werden).

Beschreibung

Liefert den Sollstrom, der für die Ansteuerung des Motors verwendet wird.

Beispiel

`':Capt_sCurr=1'` → der Sollstrom wird gewählt

`':Capt_sCurr=0'` → der Sollstrom wird abgewählt

2.11.6 Ist-Spannung der Steuerung auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_iVolt'`	0 und 1	ja	u8 (integer)	0

Priorität

4

Einheit

Wertebereich 0 – 1023 (10Bit)

1023 entspricht 66,33 V

0 entspricht 0 V

Beschreibung

Liefert die Spannung, die an der Steuerung anliegt.

Beispiel

`':Capt_iVolt=1'` → die anliegende Spannung wird gewählt

`':Capt_iVolt=0'` → die anliegende Spannung wird abgewählt

2.11.7 Digitaleingänge auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_iln'`	0 und 1	ja	u8 (integer)	0

Priorität

5

Einheit

keine

Beschreibung

Liefert die Bitmaske der Eingänge.

Beispiel

`':Capt_iln=1'` → die Bitmaske der Eingänge wird gewählt

`':Capt_iln=0'` → die Bitmaske der Eingänge wird abgewählt

2.11.8 Spannung am Analogeingang auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_iAnalog'`	0 und 1	ja	u8 (integer)	0

Priorität

6

Einheit

0 entspricht –10 V

1023 entspricht +10 V

Beschreibung

Liefert die Spannung des Analogeingangs.

Beispiel

`':Capt_iAnalog=1'` → die Spannung des Analogeingangs wird gewählt

`':Capt_iAnalog=0'` → die Spannung des Analogeingangs wird abgewählt

2.11.9 CAN-Buslast auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_iBus'`	0 und 1	ja	u8 (integer)	0

Priorität

7

Einheit

%

Ungültige Werte werden ignoriert.

Beschreibung

Liefert die ungefähre Auslastung des CAN-Busses in %.

Beispiel

`':Capt_iBus=1'` → die Auslastung des CAN-Busses wird gewählt

`':Capt_iBus=0'` → die Auslastung des CAN-Busses wird abgewählt

2.11.10 Temperatur der Steuerung auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_lTemp'`	0 und 1	ja	u8 (integer)	0

Priorität

8

Einheit

Wertebereich 0 – 1023

Beschreibung

Liefert die in der Steuerung gemessene Temperatur.

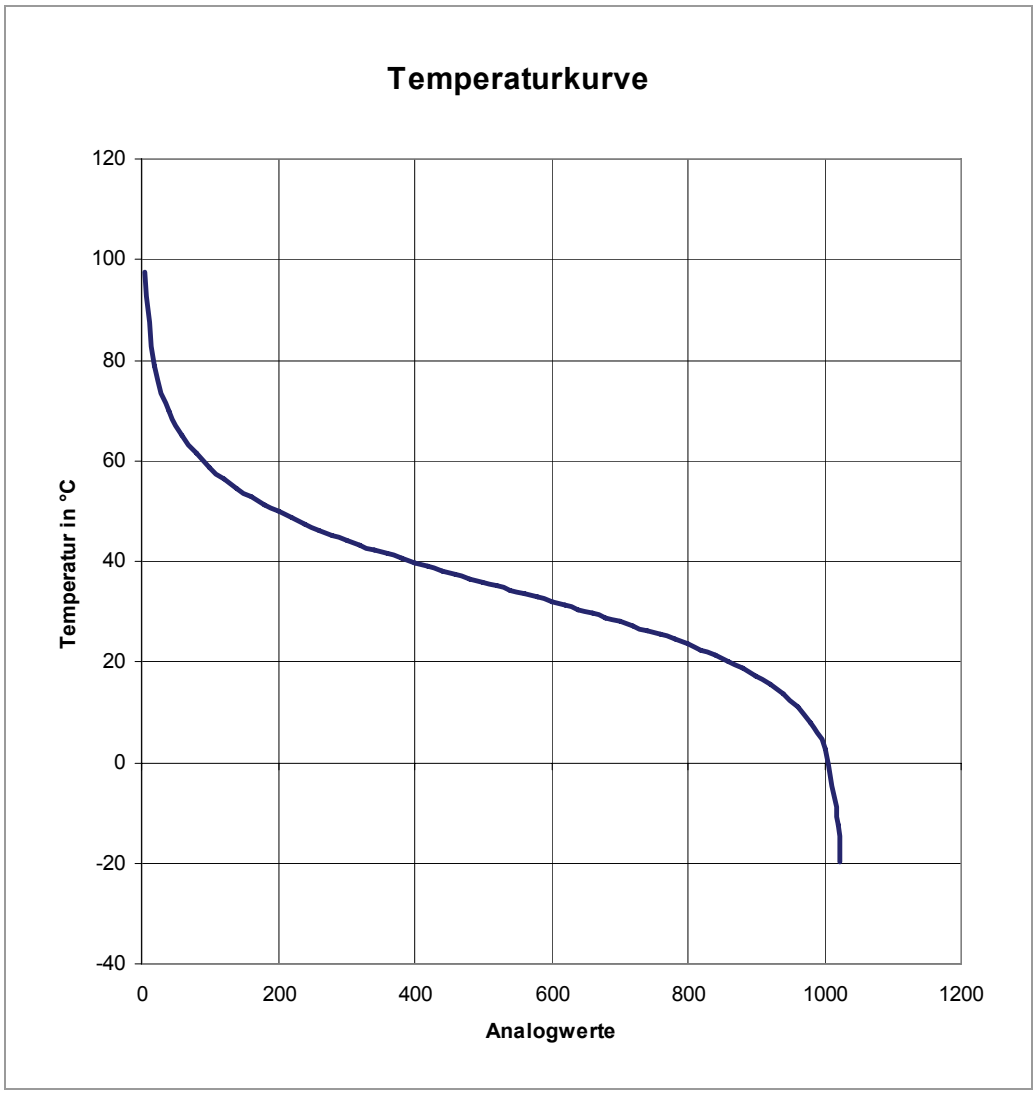
Beispiel

`':Capt_ITemp=1'` → die Temperatur der Steuerung wird gewählt

`':Capt_ITemp=0'` → die Temperatur der Steuerung wird abgewählt

Temperaturkurve

Die Steuerungen liefern den raw-Messwert des A/D Wandlers. Um aus diesem Wert die Temperatur der Steuerung zu berechnen, muss die Temperaturkurve des Messsensors eingerechnet werden.



Umrechnung

Die Umrechnung des raw-Messwerts x in die Temperatur T (°C) erfolgt nach folgender Formel:

T = **[**1266500 / (4250 + log((x/1023) * 0,33 / (1-(x/1023))) * 298)**]** – 273

Wertetabelle

Messwert x	Temperatur T (°C)	Messwert x	Temperatur T (°C)
5	97,48	520	35,09
20	78,82	540	34,33
40	70,03	560	33,57
60	64,98	580	32,82
80	61,41	600	32,05
100	58,64	620	31,28
120	56,36	640	30,5
140	54,42	660	29,71

Messwert x	Temperatur T (°C)	Messwert x	Temperatur T (°C)
160	52,71	680	28,9
180	51,19	700	28,07
200	49,8	720	27,22
220	48,53	740	26,34
240	47,35	760	25,43
260	46,24	780	24,48
280	45,2	800	23,48
300	44,21	820	22,41
320	43,26	840	21,28
340	42,34	860	20,05
360	41,46	880	18,71
380	40,61	900	17,21
400	39,78	920	15,5
420	38,97	940	13,5
440	38,17	960	11,03
460	37,39	980	7,75
480	36,62	1000	2,64
500	35,85	1020	-12,45
		1022	-19,87

Programmbeispiel (C#)

```
double computeTemperature(UInt16 value) {
double adc_max = 1023;
double R0 = 33000;
double TnK = 298;
double BK = 4250;
double Rn = 100000;
double bruch = value / adc_max;
double Rt = bruch * R0 / (1 - bruch);
double log = Math.Log(Rt / Rn);
double T = 0;
if ((vaue > 1) && (value < 1023)) {
T = (BK * TnK) / (BK + Math.Log(Rt / Rn) * TnK)- 273;
}
return T;
}
```

2.11.11 Schleppfehler auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':Capt_lFollow'`	0 und 1	ja	u8 (integer)	0

Priorität

9

Einheit

Schritte

Beschreibung

Liefert die Differenz zwischen Soll- und Ist-Position.

Beispiel

`':Capt_IFollow=1'` → die Differenz zwischen Soll- und Ist-Position wird gewählt

`':Capt_IFollow=0'` → die Differenz zwischen Soll- und Ist-Position wird abgewählt

2.12 Konfiguration des Stromreglers für Steuerungen mit dsp-Drive

2.12.1 P-Anteil des Stromreglers im Stillstand einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KP_low'`	0 bis 1000	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der P-Anteil des Stromreglers für Steuerungen mit dsp-Drive im Stillstand eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.12.2 P-Anteil des Stromreglers während der Fahrt einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KP_hig'`	0 bis 1000	ja	u16 (integer)	10

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der P-Anteil des Stromreglers für Steuerungen mit dsp-Drive während der Fahrt eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.12.3 Skalierungsfaktor zur drehzahlabh. Anpassung des P-Anteils des Reglers während der Fahrt einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KP_scale'`	0 bis 1000	ja	u16 (integer)	58

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der Skalierungsfaktor zur drehzahlabhängigen Anpassung des P-Anteils des Stromreglers für Steuerungen mit dsp-Drive während der Fahrt eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.12.4 I-Anteil des Stromreglers im Stillstand einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KI_low'`	0 bis 1000	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der I-Anteil des Stromreglers für Steuerungen mit dsp-Drive im Stillstand eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.12.5 I-Anteil des Stromreglers während der Fahrt einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KI_hig'`	0 bis 1000	ja	u16 (integer)	10

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der I-Anteil des Stromreglers für Steuerungen mit dsp-Drive während der Fahrt eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

2.12.6 Skalierungsfaktor zur drehzahlabh. Anpassung des I-Anteils des Reglers während der Fahrt einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
`':dspdrive_KI_scale'`	0 bis 1000	ja	u16 (integer)	200

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der Skalierungsfaktor zur drehzahlabhängigen Anpassung des I-Anteils des Stromreglers fürSteuerungen mit dsp-Drive während der Fahrt eingestellt werden.

Normalerweise keine Änderung nötig.

Auslesen

Wird das Schlüsselwort ohne `'= + Wert'` gesendet, kann der aktuell eingestellte Wert ausgelesen werden.

3 Programmierung mit Java (NanoJEasy)

3.1 Übersicht

Zu diesem Kapitel

Dieses Kapitel enthält eine kurze Übersicht über die Programmiersprache der Nanotec Schrittmotorsteuerungen. Die Steuerungen enthalten eine Java Virtual Machine (VM), die um einige herstellerspezifische Funktionen erweitert wurde.

Einschränkungen

Aufgrund der zugrunde liegenden Hardware weist die aktuelle VM folgende Einschränkungen auf:

- Der verfügbare Programmspeicher auf der Steuerung hängt von der Firmware-Version ab.
- Der Stack sowie der Heap sind auf 50 Einträge begrenzt → rekursive Funktionsaufrufe sind nur begrenzt möglich.
- Es werden keine Threads unterstützt.

Verwendete Abkürzungen

VM	Virtuelle Maschine
Java SE	Java Standard Edition
JDK	Java Developent Kit
JRE	Java Runtime Environment

Voraussetzungen

Um ein Programm für die Steuerung zu entwickeln müssen folgende Voraussetzungen erfüllt sein:

- Programmierumgebung NanoJEasy installiert
- SMCI47-S
- SMCP33
- SMCI33
- SMCI35
- SMCI36
- SMCI12
- PD6-N
- PD4-N
- PD2-N

Gleichzeitige Kommunikation über serielle Schnittstelle

NanoJ läuft als virtuelle Maschine unabhängig von der eigentlichen Firmware und kommuniziert mit dieser über die gleichen Funktionen, die auch von der seriellen Schnittstelle aufgerufen werden.

Während die Steuerung serielle Befehle empfängt und abarbeitet, kann deshalb gleichzeitig ein Java-Programm laufen.

Hinweis:
Es sollte allgemein nur dann mit seriellen Befehlen gearbeitet werden, wenn das Java-Programm zu dem Zeitpunkt nicht aktiv auf die Steuerung einwirkt.

3.2 Befehlsübersicht

Nachfolgend finden Sie eine Auflistung der Befehle für die Programmierung mit Java (NanoJEasy):

capture-Befehle

cl-Befehle

util-Befehle

3.3 NanoJEasy installieren

Allgemeines

Bei NanoJEasy handelt es sich um eine Programmierumgebung zur Entwicklung von Java-Programmen, welche auf Nanotec Schrittmotorsteuerungen ablauffähig sind und eine erweiterte Programmierung der Steuerungen ermöglichen.

NanoJEasy enthält den frei verfügbaren Gnu-Java-Compiler (gcj) zur Übersetzung der Java-Programme.

Vorgehensweise

Führen Sie die Installation wie folgt durch:

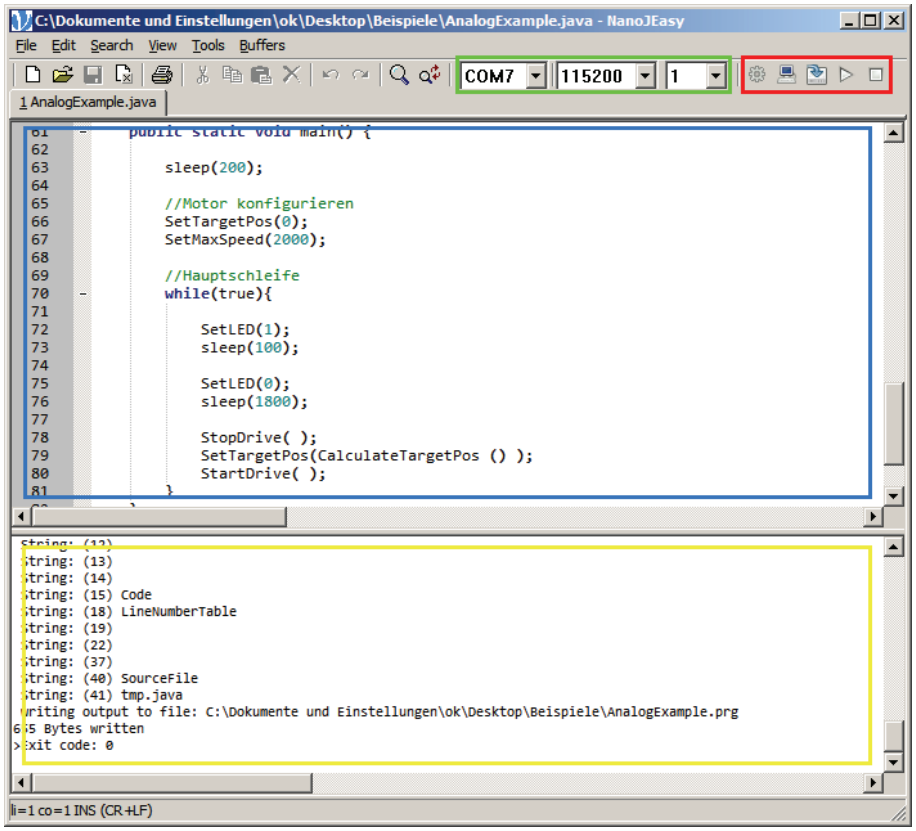
Schritt	Durchführung
1	Doppelklicken Sie auf die Datei setup.exe.
2	Wählen Sie die gewünschte Sprache aus.
3	Bestätigen Sie, dass Sie die Lizenzbestimmungen anerkennen.
4	Wählen Sie den Ordner aus, in dem NanoJEasy installiert werden soll.
5	Bestätigen oder Ändern Sie den vorgeschlagenen Startmenüeintrag für NanoJEasy.
6	Starten Sie die Installation.

3.4 Arbeiten mit NanoJEasy

3.4.1 Das Hauptfenster von NanoJEasy

Screenshot

Im folgenden Screenshot sind alle wichtigen Elemente des NanoJEasy-Hauptfensters markiert:



Erläuterung der Bereiche

- Mit den grün markierten Bedienelementen lassen sich folgende Kommunikationsparameter einstellen:
  - Auswahl eines der vorhandenen COM-Ports
  - Auswahl einer Baudrate
  - Auswahl einer Motornummer
- Mit den rot markierten Buttons können folgende Aktionen durchgeführt werden:
  - Übersetzen und Linken des aktuellen Programms
  - Simulation des aktuellen Programms
  - Übertragen des aktuellen Programms in die Steuerung
  - Ausführen des in der Steuerung befindlichen Programms
  - Stoppen des in der Steuerung laufenden Programms
- Im blau markierten Textbereich wird der Programmquelltext bearbeitet.
- Im gelb markierten Ausgabebereich erscheinen Meldungen zur Übersetzung, Simulation, Übertragung und Ausführung des entwickelten Programms.

3.4.2 Ablauf der Entwicklung mit NanoJEasy

Entwicklungsablauf

Der Entwicklungsablauf mit NanoJEasy folgt normalerweise folgendem Schema:

Stufe	Beschreibung
1	Programm im Textbereich erstellen.
2	Programm übersetzen und linken.
3	Optional: Programm simulieren.
4	Einstellungen der Kommunikationsparameter überprüfen.
5	Programm auf die Steuerung übertragen.
6	Programm auf der Steuerung ausführen.

Wichtige Hinweise zur Programmierung

Bei der Programmierung sollten unbedingt folgende Hinweise beachtet werden:

- Quelltextdateien müssen mit der Zeichenkodierung UTF-8 erstellt werden. NanoJEasy verwendet standardmäßig diese Zeichenkodierung.
- Der Klassenname in der Quelltextdatei muss mit dem Namen der Quelltextdatei übereinstimmen. Beispiel: die Datei „Testprogramm.java“ muss die Klasse „class Testprogramm“ enthalten.
- Die Java-Befehle zur Kommunikation mit der Steuerung stoßen die jeweilige Aktion der Steuerung nur an, warten aber nicht, bis die Steuerung die Aktion ausgeführt hat. Wenn das Java-Programm auf die Ausführung der Aktion warten soll, so muss nach dem Befehl zur Ausführung eine Wartezeit eingefügt werden, z.B. `'Sleep(2000);'`. Siehe hierzu auch die Beispielprogramme.

Befehl vervollständigen beim Eingeben

Geben Sie einen Befehl wie folgt ein:

Schritt	Durchführung
1	Geben Sie die ersten Buchstaben eines Befehls ein, z.B. `'Set'` von `'SetCurrent'`.
2	Drücken Sie die Tasten <Strg> + <Leertaste>. Eine Auswahlliste von Befehlen, die mit `'Set'` beginnen, erscheint.
3	Markieren Sie in der Auswahlliste mit den Pfeiltasten „Auf“ und „Ab“ einen Befehl.
4	Drücken Sie die Taste „Enter“, um den Befehl auszuwählen.

Simulation starten und beenden

Gehen Sie zum Starten und Beenden der Simulation wie folgt vor:

Schritt	Durchführung
1	Klicken Sie auf den „Simulation starten“-Button (s.o.). Es erscheinen fortlaufend die Ausgaben des Emulators im Ausgabebereich.
2	Drücken Sie zum Beenden der Simulation auf die Tasten <Strg> + <Pause>.

3.4.3 Integrierte Befehle

Klassen und Funktionen

Die VM enthält integrierte Funktionen, die im Programm verwendet werden können. Die Funktionen sind in insgesamt sechs verschiedenen Klassen zusammengefasst, welche im Quellcode eingebunden werden können.

Die nachfolgenden Abschnitte geben Aufschluss über die einzelnen Klassen und ihre enthaltenen Funktionen.

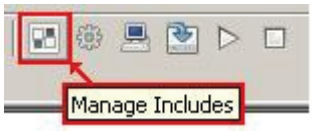
Einbinden einer Klasse

Die sechs verschiedenen Klassen sind im Package nanotec enthalten und müssen durch folgende Eingabe am Programmanfang eingebunden werden:

```
import nanotec.*;
```

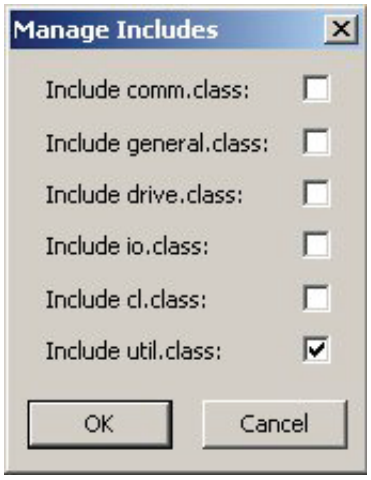
Welche der Klassen wirklich beim Übertragen auf die Steuerung eingebunden wird, muss zusätzlich in NanoJEasy eingestellt werden.

Die „Manage Includes“ - Schaltfläche im oberen rechten Bereich der Anwendung



öffnet den Einbindungs-Manager.

Die benötigten Klassen lassen sich dann einfach per aktivierter CheckBox einbinden:



Aufrufen von Funktionen

Die einzelnen Funktionen einer Klasse werden im Quelltext wie folgt aufgerufen:

```
[Name der Klasse].[Name der Funktion]();
```

Beispiel:

```
drive.StartDrive();
```

3.5 Klassen und Funktionen

3.5.1 Klasse „capture“

Anwendung

Die Klasse capture dient zum Konfigurieren des Scope Modes. Mit den folgenden Funktionen kann die Steuerung so konfiguriert werden, dass sie Steuerungsgrößen ermittelt und über die serielle Schnittstelle verschickt. Siehe auch Abschnitt 2.11.

capture.SetCaptTime

Deklaration:

```
static native void SetCaptTime(int time);
```

Diese Funktion setzt die Samplerate.

Die Funktion entspricht dem seriellen Befehl `':Capt_Time<time>'`, siehe Befehl *2.11.2 Samplerate einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptTime

Deklaration:

```
static native void GetCaptTime(int time);
```

Diese Funktion liest die Samplerate.

Die Funktion entspricht dem seriellen Befehl `':Capt_Time'`, siehe Befehl *2.11.2 Samplerate einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptsPos

Deklaration:

```
static native void SetCaptsPos(int value);
```

Diese Funktion wählt die Sollposition an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_sPos<value>'`, siehe Befehl *2.11.3 Sollposition des Rampengenerators auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptsPos

Deklaration:

```
static native int GetCaptsPos();
```

Diese Funktion liest aus, ob die Sollposition gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_sPos'`, siehe Befehl *2.11.3 Sollposition des Rampengenerators auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptiPos

Deklaration:

```
static native void SetCaptiPos(int value);
```

Diese Funktion wählt die Istposition an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_iPos<value>'`, siehe Befehl *2.11.4 Istposition des Drehgebers auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptiPos

Deklaration:

```
static native int GetCaptiPos();
```

Diese Funktion liest aus, ob die Istposition gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_iPos'`, siehe Befehl *2.11.4 Istposition des Drehgebers auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptsCurr

Deklaration:

```
static native void SetCaptsCurr(int value);
```

Diese Funktion wählt den Sollstrom an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_sCurr<value>'`, siehe Befehl *2.11.5 Sollstrom der Motoransteuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptsCurr

Deklaration:

```
static native int GetCaptsCurr();
```

Diese Funktion liest aus, ob den Sollstrom gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_sCurr'`, siehe Befehl *2.11.5 Sollstrom der Motoransteuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptiVolt

Deklaration:

```
static native void SetCaptiVolt(int value);
```

Diese Funktion wählt die Istspannung an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_iVolt<value>'`, siehe Befehl *2.11.6 Ist-Spannung der Steuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptiVolt

Deklaration:

```
static native int GetCaptiVolt();
```

Diese Funktion liest aus, ob die Istspannung gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_iVolt'`, siehe Befehl *2.11.6 Ist-Spannung der Steuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptiIn

Deklaration:

```
static native void SetCaptiIn(int value);
```

Diese Funktion wählt die Bitmaske der Eingänge an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_iln<value>'`, siehe Befehl *2.11.7 Digitaleingänge auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptiIn

Deklaration:

```
static native int GetCaptiIn();
```

Diese Funktion liest aus, ob die Bitmaske der Eingänge gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_iln'`, siehe Befehl *2.11.7 Digitaleingänge auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptiAnalog

Deklaration:

```
static native void SetCaptiAnalog(int value);
```

Diese Funktion wählt die Spannung am Analogeingang an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_iAnalog<value>'`, siehe Befehl *2.11.8 Spannung am Analogeingang auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptiAnalog

Deklaration:

```
static native int GetCaptiAnalog();
```

Diese Funktion liest aus, ob die Spannung am Analogeingang gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_iAnalog'`, siehe Befehl *2.11.8 Spannung am Analogeingang auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptiBus

Deklaration:

```
static native void SetCaptiBus(int value);
```

Diese Funktion wählt die Auslastung des CAN-Busses an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_iBus<value>'`, siehe Befehl *2.11.9 CAN-Buslast auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptiBus

Deklaration:

```
static native int GetCaptiBus();
```

Diese Funktion liest aus, ob die Auslastung des CAN-Busses gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_iBus'`, siehe Befehl *2.11.9 CAN-Buslast auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptlTemp

Deklaration:

```
static native void SetCaptlTemp(int value);
```

Diese Funktion wählt die Temperatur der Steuerung an/ab.

Die Funktion entspricht dem seriellen Befehl `':Capt_lTemp<value>'`, siehe Befehl *2.11.10 Temperatur der Steuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptlTemp

Deklaration:

```
static native int GetCaptlTemp();
```

Diese Funktion liest aus, ob die Temperatur der Steuerung gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_lTemp'`, siehe Befehl *2.11.10 Temperatur der Steuerung auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.SetCaptlFollow

Deklaration:

```
static native void SetCaptlFollow(int value);
```

Diese Funktion wählt den Schleppfehler an/aus.

Die Funktion entspricht dem seriellen Befehl `':Capt_lFollow<offset>'`, siehe Befehl *2.11.11 Schleppfehler auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

capture.GetCaptIFollow

Deklaration:

```
static native int GetCaptIFollow();
```

Diese Funktion liest aus, ob der Schleppfehler der Steuerung gewählt ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':Capt_IFollow'`, siehe Befehl *2.11.11 Schleppfehler auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.2 Klasse „cl“

Anwendung

Die Klasse cl dient der Konfiguration des closed loop. Es können die PID Parameter eingestellt sowie der Status des closed loop manipuliert werden.

cl.SetClosedLoop

Deklaration:

```
static native void SetClosedLoop(int value);
```

Diese Funktion aktiviert/deaktiviert den Regelkreis. Der Modus wird erst aktiviert, wenn eine interne Referenzfahrt durchgeführt wurde bzw. wenn bei aktiviertem auto enable mehr als eine Umdrehung gefahren wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_enable<value>'`, siehe Befehl *2.9.1 Closed-Loop-Modus aktivieren*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetClosedLoop

Deklaration:

```
static native int GetClosedLoop();
```

Diese Funktion liest aus, ob der Regelkreis aktiviert/deaktiviert ist.

Die Funktion entspricht dem seriellen Befehl `':CL_enable'`, siehe Befehl *2.9.1 Closed-Loop-Modus aktivieren*.

Enthalten in Firmware neuer als 15.03.2010.

cl.IsClosedLoopEnabled

Deklaration:

```
static native int IsClosedLoopEnabled();
```

Diese Funktion liest aus, ob der Regelkreis aktiviert/deaktiviert ist.

- Wert 0: Regelkreis nicht aktiv
- Wert 1: Regelkreis aktiv (nur wenn spezielle Referenzfahrt durchgeführt wurde)

Die Funktion entspricht dem seriellen Befehl `':CL_is_enabled'`, siehe Befehl *2.9.2 Status Closed-Loop-Modus auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPvZ

Deklaration:

```
static native void SetKPvZ(int value);
```

Diese Funktion setzt den Zähler des P-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_Z<value>'`, siehe Befehl *2.9.14 Zähler des P-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPvZ

Deklaration:

```
static native int GetKPvZ();
```

Diese Funktion liest den Zähler des P-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_Z'`, siehe Befehl *2.9.14 Zähler des P-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPvN

Deklaration:

```
static native void SetKPvN(int value);
```

Diese Funktion setzt den Nenner des P-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_N<value>'`, siehe Befehl *2.9.15 Nenner des P-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPvN

Deklaration:

```
static native int GetKPvN();
```

Diese Funktion liest den Nenner des P-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_N'`, siehe Befehl *2.9.15 Nenner des P-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIvZ

Deklaration:

```
static native void SetKIvZ(int value);
```

Diese Funktion setzt den Zähler des I-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_Z<value>'`, siehe Befehl *2.9.16 Zähler des I-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIvZ

Deklaration:

```
static native int GetKIvZ();
```

Diese Funktion liest den Zähler des I-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_Z'`, siehe Befehl *2.9.16 Zähler des I-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIvN

Deklaration:

```
static native void SetKIvN(int value);
```

Diese Funktion setzt den Nenner des I-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_N<value>'`, siehe Befehl *2.9.17 Nenner des I-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIvN

Deklaration:

```
static native int GetKIvN();
```

Diese Funktion liest den Nenner des I-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_N'`, siehe Befehl *2.9.17 Nenner des I-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDvZ

Deklaration:

```
static native void SetKDvZ(int value);
```

Diese Funktion setzt den Zähler des D-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_Z<value>'`, siehe Befehl *2.9.18 Zähler des D-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDvZ

Deklaration:

```
static native int GetKDvZ();
```

Diese Funktion liest den Zähler des D-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_Z'`, siehe Befehl *2.9.18 Zähler des D-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDvN

Deklaration:

```
static native void SetKDvN(int value);
```

Diese Funktion setzt den Nenner des D-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl ':CL_KD_v_N<value>', siehe Befehl *2.9.19 Nenner des D-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDvN

Deklaration:

```
static native int GetKDvN();
```

Diese Funktion liest den Nenner des D-Anteils des Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl ':CL_KD_v_N', siehe Befehl *2.9.19 Nenner des D-Anteils des Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPsZ

Deklaration:

```
static native void SetKPsZ(int value);
```

Diese Funktion setzt den Zähler des P-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl ':CL_KP_s_Z<value>', siehe Befehl *2.9.26 Zähler des P-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPsZ

Deklaration:

```
static native int GetKPsZ();
```

Diese Funktion liest den Zähler des P-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl ':CL_KP_s_Z', siehe Befehl *2.9.26 Zähler des P-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPsN

Deklaration:

```
static native void SetKPsN(int value);
```

Diese Funktion setzt den Nenner des P-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl ':CL_KP_s_N<value>', siehe Befehl *2.9.27 Nenner des P-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPsN

Deklaration:

```
static native int GetKPsN();
```

Diese Funktion liest den Nenner des P-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_s_N'`, siehe Befehl *2.9.27 Nenner des P-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIsZ

Deklaration:

```
static native void SetKIsZ(int value);
```

Diese Funktion setzt den Zähler des I-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_Z<value>'`, siehe Befehl *2.9.28 Zähler des I-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIsZ

Deklaration:

```
static native int GetKIsZ();
```

Diese Funktion liest den Zähler des I-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_Z'`, siehe Befehl *2.9.28 Zähler des I-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIsN

Deklaration:

```
static native void SetKIsN(int value);
```

Diese Funktion setzt den Nenner des I-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_N<value>'`, siehe Befehl *2.9.29 Nenner des I-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIsN

Deklaration:

```
static native int GetKIsN();
```

Diese Funktion liest den Nenner des I-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_N'`, siehe Befehl *2.9.29 Nenner des I-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDsZ

Deklaration:

```
static native void SetKDsZ(int value);
```

Diese Funktion setzt den Zähler des D-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_Z<value>'`, siehe Befehl *2.9.30 Zähler des D-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDsZ

Deklaration:

```
static native int GetKDsZ();
```

Diese Funktion liest den Zähler des D-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_Z'`, siehe Befehl *2.9.30 Zähler des D-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDsN

Deklaration:

```
static native void SetKDsN(int value);
```

Diese Funktion setzt den Nenner des D-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_N<value>'`, siehe Befehl *2.9.31 Nenner des D-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDsN

Deklaration:

```
static native int GetKDsN();
```

Diese Funktion liest den Nenner des D-Anteils des Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_N'`, siehe Befehl *2.9.31 Nenner des D-Anteils des Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPcsvZ

Deklaration:

```
static native void SetKPcsvZ(int value);
```

Diese Funktion setzt den Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_Z<value>'`, siehe Befehl *2.9.20 Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPcsvZ

Deklaration:

```
static native int GetKPcsvZ();
```

Diese Funktion liest den Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_Z'`, siehe Befehl *2.9.20 Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPcsvN

Deklaration:

```
static native void SetKPcsvN(int value);
```

Diese Funktion setzt den Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_N<value>'`, siehe Befehl *2.9.21 Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPcsvN

Deklaration:

```
static native int GetKPcsvN();
```

Diese Funktion liest den Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_N'`, siehe Befehl *2.9.21 Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIcsvZ

Deklaration:

```
static native void SetKIcsvZ(int value);
```

Diese Funktion setzt den Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_Z<value>'`, siehe Befehl *2.9.22 Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIcsvZ

Deklaration:

```
static native int GetKIcsvZ();
```

Diese Funktion liest den Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_Z'`, siehe Befehl *2.9.22 Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIcsvN

Deklaration:

```
static native void SetKIcsvN(int value);
```

Diese Funktion setzt den Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_N<value>'`, siehe Befehl *2.9.23 Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIcsvN

Deklaration:

```
static native int GetKIcsvN();
```

Diese Funktion liest den Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_N'`, siehe Befehl *2.9.23 Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDcsvZ

Deklaration:

```
static native void SetKDcsvZ(int value);
```

Diese Funktion setzt den Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_Z<value>'`, siehe Befehl *2.9.24 Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDcsvZ

Deklaration:

```
static native int GetKDcsvZ();
```

Diese Funktion liest den Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_Z'`, siehe Befehl *2.9.24 Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDcsvN

Deklaration:

```
static native void SetKDcsvN(int value);
```

Diese Funktion setzt den Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_N<value>'`, siehe Befehl *2.9.25 Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDcsvN

Deklaration:

```
static native int GetKDcsvN();
```

Diese Funktion liest den Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_N'`, siehe Befehl *2.9.25 Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPcssZ

Deklaration:

```
static native void SetKPcssZ(int value);
```

Diese Funktion setzt den Zähler des P-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_Z<value>'`, siehe Befehl *2.9.32 Zähler des P-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPcssZ

Deklaration:

```
static native int GetKPcssZ();
```

Diese Funktion liest den Zähler des P-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_Z'`, siehe Befehl *2.9.32 Zähler des P-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKPcssN

Deklaration:

```
static native void SetKPcssN(int value);
```

Diese Funktion setzt den Nenner des P-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_N<value>'`, siehe Befehl *2.9.33 Nenner des P-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKPcssN

Deklaration:

```
static native int GetKPcssN();
```

Diese Funktion liest den Nenner des P-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_N'`, siehe Befehl *2.9.33 Nenner des P-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIcssZ

Deklaration:

```
static native void SetKIcssZ(int value);
```

Diese Funktion setzt den Zähler des I-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_Z<value>'`, siehe Befehl *2.9.34 Zähler des I-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIcssZ

Deklaration:

```
static native int GetKIcssZ();
```

Diese Funktion liest den Zähler des I-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_Z'`, siehe Befehl *2.9.34 Zähler des I-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKIcssN

Deklaration:

```
static native void SetKIcssN(int value);
```

Diese Funktion setzt den Nenner des I-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_N<value>'`, siehe Befehl *2.9.35 Nenner des I-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKIcssN

Deklaration:

```
static native int GetKIcssN();
```

Diese Funktion liest den Nenner des I-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_N'`, siehe Befehl *2.9.35 Nenner des I-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDcssZ

Deklaration:

```
static native void SetKDcssZ(int value);
```

Diese Funktion setzt den Zähler des D-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_Z<value>'`, siehe Befehl *2.9.36 Zähler des D-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDcssZ

Deklaration:

```
static native int GetKDcssZ();
```

Diese Funktion liest den Zähler des D-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_Z'`, siehe Befehl *2.9.36 Zähler des D-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetKDcssN

Deklaration:

```
static native void SetKDcssN(int value);
```

Diese Funktion setzt den Nenner des D-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_N<value>'`, siehe Befehl *2.9.37 Nenner des D-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetKDcssN

Deklaration:

```
static native int GetKDcssN();
```

Diese Funktion liest den Nenner des D-Anteils des kaskadierenden Positionsreglers.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_N'`, siehe Befehl *2.9.37 Nenner des D-Anteils des kaskadierenden Positionsreglers einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetPositionWindow

Deklaration:

```
static native void SetPositionWindow(int value);
```

Diese Funktion setzt das Toleranzfenster für die Endposition im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window<value>'`, siehe Befehl *2.9.4 Toleranzfenster für Endposition einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetPositionWindow

Deklaration:

```
static native int GetPositionWindow();
```

Diese Funktion liest das Toleranzfenster für die Endposition im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window'`, siehe Befehl *2.9.4 Toleranzfenster für Endposition einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetPositionWindowTime

Deklaration:

```
static native void SetPositionWindowTime(int time);
```

Diese Funktion setzt die Zeit für das Toleranzfenster der Endposition im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window_time<time>'`, siehe Befehl *2.9.5 Zeit für Toleranzfenster der Endposition einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetPositionWindowTime

Deklaration:

```
static native int GetPositionWindowTime();
```

Diese Funktion liest die Zeit für das Toleranzfenster der Endposition im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window_time'`, siehe Befehl *2.9.5 Zeit für Toleranzfenster der Endposition einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetFollowingErrorWindow

Deklaration:

```
static native void SetFollowingErrorWindow(int value);
```

Diese Funktion setzt den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_window<value>'`, siehe Befehl *2.9.6 Maximal erlaubten Schleppfehler einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetFollowingErrorWindow

Deklaration:

```
static native int GetFollowingErrorWindow();
```

Diese Funktion liest den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_window'`, siehe Befehl *2.9.6 Maximal erlaubten Schleppfehler einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetFollowingErrorTimeout

Deklaration:

```
static native void SetFollowingErrorTimeout(int time);
```

Diese Funktion setzt die Zeit für den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_timeout<time>'`, siehe Befehl *2.9.7 Zeit für maximalen Schleppfehler einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetFollowingErrorTimeout

Deklaration:

```
static native int GetFollowingErrorTimeout();
```

Diese Funktion liest die Zeit für den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_timeout'`, siehe Befehl *2.9.7 Zeit für maximalen Schleppfehler einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetSpeedErrorWindow

Deklaration:

```
static native void SetSpeedErrorWindow(int value);
```

Diese Funktion setzt die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_window<value>'`, siehe Befehl *2.9.8 Maximal erlaubte Drehzahlabweichung*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetSpeedErrorWindow

Deklaration:

```
static native int GetSpeedErrorWindow();
```

Diese Funktion liest die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_window'`, siehe Befehl *2.9.8 Maximal erlaubte Drehzahlabweichung*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetSpeedErrorTimeout

Deklaration:

```
static native void SetSpeedErrorTimeout(int time);
```

Diese Funktion setzt die Zeit für die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_timeout<time>'`, siehe Befehl *2.9.9 Zeit für maximal erlaubte Drehzahlabweichung*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetSpeedErrorTimeout

Deklaration:

```
static native int GetSpeedErrorTimeout();
```

Diese Funktion liest die Zeit für die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_timeout'`, siehe Befehl *2.9.9 Zeit für maximal erlaubte Drehzahlabweichung*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle1

Deklaration:

```
static native void SetCLLoadAngle1(int value);
```

Diese Funktion setzt den Lastwinkel 1 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_a<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle1

Deklaration:

```
static native int GetCLLoadAngle1();
```

Diese Funktion liest den Lastwinkel 1 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_a'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle2

Deklaration:

```
static native void SetCLLoadAngle2(int value);
```

Diese Funktion setzt den Lastwinkel 2 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_b<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle2

Deklaration:

```
static native int GetCLLoadAngle2();
```

Diese Funktion liest den Lastwinkel 2 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_b'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle3

Deklaration:

```
static native void SetCLLoadAngle3(int value);
```

Diese Funktion setzt den Lastwinkel 3 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_c<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle3

Deklaration:

```
static native int GetCLLoadAngle3();
```

Diese Funktion liest den Lastwinkel 3 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_c'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle4

Deklaration:

```
static native void SetCLLoadAngle4(int value);
```

Diese Funktion setzt den Lastwinkel 4 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_d<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle4

Deklaration:

```
static native int GetCLLoadAngle4();
```

Diese Funktion liest den Lastwinkel 4 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_d'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle5

Deklaration:

```
static native void SetCLLoadAngle5(int value);
```

Diese Funktion setzt den Lastwinkel 5 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_e<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle5

Deklaration:

```
static native int GetCLLoadAngle5();
```

Diese Funktion liest den Lastwinkel 5 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_e'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle6

Deklaration:

```
static native void SetCLLoadAngle6(int value);
```

Diese Funktion setzt den Lastwinkel 6 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_f<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle6

Deklaration:

```
static native int GetCLLoadAngle6();
```

Diese Funktion liest den Lastwinkel 6 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_f'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLLoadAngle7

Deklaration:

```
static native void SetCLLoadAngle7(int value);
```

Diese Funktion setzt den Lastwinkel 7 des Motors aus dem Closed-Loop-Testlauf.

Die Funktion entspricht dem seriellen Befehl `':CL_la_g<value>'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLLoadAngle7

Deklaration:

```
static native int GetCLLoadAngle7();
```

Diese Funktion liest den Lastwinkel 7 des Motors aus dem Closed-Loop-Testlauf aus.

Die Funktion entspricht dem seriellen Befehl `':CL_la_g'`, siehe Befehl *2.10.2 Lastwinkelmesswerte des Motors setzen/auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLNodeDistance

Deklaration:

```
static native void SetCLNodeDistance(int value);
```

Diese Funktion setzt den Stützstellenabstand für die Lastwinkelkurve.

Die Funktion entspricht dem seriellen Befehl `':CL_la_node_distance<value>'`, siehe Befehl *2.9.38 Stützstellenabstand für Lastwinkelkurve einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLNodeDistance

Deklaration:

```
static native int GetCLNodeDistance();
```

Diese Funktion liest den Stützstellenabstand für die Lastwinkelkurve.

Die Funktion entspricht dem seriellen Befehl `':CL_la_node_distance'`, siehe Befehl *2.9.38 Stützstellenabstand für Lastwinkelkurve einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.SetCLPoscntOffset

Deklaration:

```
static native void SetCLPoscntOffset(int offset);
```

Diese Funktion setzt den Offset zwischen Encoder und Motor.

Die Funktion entspricht dem seriellen Befehl `':CL_poscnt_offset<offset>'`, siehe Befehl *2.10.1 Offset Encoder/Motor auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetCLPoscntOffset

Deklaration:

```
static native int GetCLPoscntOffset();
```

Diese Funktion liest den beim Testlauf ermittelten Offset zwischen Encoder und Motor aus.

Die Funktion entspricht dem seriellen Befehl `':CL_poscnt_offset'`, siehe Befehl *2.10.1 Offset Encoder/Motor auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

cl.GetVelocityActualValue

Deklaration:

```
static native int GetVelocityActualValue();
```

Diese Funktion liest die aktuelle Drehzahl aus (nur im Closed-Loop-Betrieb).

Die Funktion entspricht dem seriellen Befehl `':v'`, siehe Befehl *2.7.11 Drehzahl auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.3 Klasse „comm“

Anwendung

Die Klasse comm dient der Konfiguration der seriellen Kommunikation sowie zum Senden von Daten.

comm.SendInt

Deklaration:

```
static native void SendInt(int in);
```

Sendet den angegebenen Integer-Wert über die serielle Schnittstelle.

comm.SendLong

Deklaration:

```
static native void SendLong(long in);
```

Sendet den angegebenen Long-Wert über die serielle Schnittstelle.

comm.SetBaudrate

Deklaration:

```
static native void SetBaudrate(int value);
```

Diese Funktion setzt die Baudrate der Steuerung.

Die Funktion entspricht dem seriellen Befehl `':baud<value>'`, siehe Befehl *2.5.42 Baudrate der Steuerung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

comm.GetBaudrate

Deklaration:

```
static native int GetBaudrate();
```

Diese Funktion liest die Baudrate der Steuerung.

Die Funktion entspricht dem seriellen Befehl `':baud'`, siehe Befehl *2.5.42 Baudrate der Steuerung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

comm.SetCRC

Deklaration:

```
static native void SetCRC(int value);
```

Schaltet die Überprüfung der seriellen Kommunikation mittels einer CRC-Prüfsumme (cyclic redundancy check) ein oder aus:

- Wert 0: CRC-Prüfung deaktiviert
- Wert 1: CRC-Prüfung aktiviert

Die Funktion entspricht dem seriellen Befehl `':crc<value>'`, siehe Befehl *2.5.43 CRC-Prüfsumme einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

comm.GetCRC

Deklaration:

```
static native int GetCRC();
```

Diese Funktion liest, ob die Überprüfung der seriellen Kommunikation mittels einer CRC Prüfsumme ein oder aus ist.

Die Funktion entspricht dem seriellen Befehl `':crc'`, siehe Befehl *2.5.43 CRC-Prüfsumme einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

comm.SetSupressResponse

Deklaration:

```
static native void SetSupressResponse(int value);
```

Diese Funktion aktiviert oder deaktiviert die Antwortunterdrückung beim Senden.

- value = 0: Antwortunterdrückung ein
- value = 1: Antwortunterdrückung aus

Die Funktion entspricht dem seriellen Befehl `'|<value>'`, siehe Befehl *2.6.4 Aktuellen Satz auslesen*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.4 Klasse „config“

Anwendung

Die Klasse config dient der Konfiguration von allgemeinen Steuerungseinstellungen.

config.SetSendStatusWhenCompleted

Deklaration:

```
static native void SetSendStatusWhenCompleted(int flag);
```

Diese Funktion schaltet das selbständige Senden eines Status am Ende einer Fahrt ein/aus.

- sendStatus = 0: automatisches Senden aus
- sendStatus = 1: automatisches Senden ein

Die Funktion entspricht dem seriellen Befehl `'J<flag>'`, siehe Befehl *2.5.33 Automatisches Senden des Status einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetSendStatusWhenCompleted

Deklaration:

```
static native int GetSendStatusWhenCompleted();
```

Diese Funktion liest, ob das selbständige Senden eines Status am Ende einer Fahrt eingeschaltet ist.

- sendStatus = 0: automatisches Senden aus
- sendStatus = 1: automatisches Senden ein

Die Funktion entspricht dem seriellen Befehl `'ZJ'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetRecordForAutoCorrect

Deklaration:

```
static native void SetRecordForAutoCorrect(int record);
```

Diese Funktion konfiguriert die automatische Fehlerkorrektur des Motors.

Die Funktion entspricht dem seriellen Befehl `'F<record>'`, siehe Befehl *2.5.11 Satz für Autokorrektur einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetRecordForAutoCorrect

Deklaration:

```
static native int GetRecordForAutoCorrect();
```

Diese Funktion liest aus, welcher Satz für die automatische Fehlerkorrektur gesetzt ist.

Die Funktion entspricht dem seriellen Befehl `'ZF'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetEncoderDirection

Deklaration:

```
static native void SetEncoderDirection(int value);
```

Diese Funktion setzt die Encoderdrehrichtung. Ist der Parameter value 1, so wird die Richtung des Drehencoders umgekehrt.

Die Funktion entspricht dem seriellen Befehl `'q<value>'`, siehe Befehl *2.5.12 Encoderrichtung einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetEncoderDirection

Deklaration:

```
static native int GetEncoderDirection();
```

Diese Funktion liest aus, ob die Encoderdrehrichtung umgekehrt wird.

Die Funktion entspricht dem seriellen Befehl `'Zq'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetSwingOutTime

Deklaration:

```
static native void SetSwingOutTime(int time);
```

Diese Funktion setzt die Ausschwingzeit.

Die Funktion entspricht dem seriellen Befehl `'O<time>'`, siehe Befehl *2.5.13 Ausschwingzeit einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetSwingOutTime

Deklaration:

```
static native int GetSwingOutTime();
```

Diese Funktion liest die Ausschwingzeit aus.

Die Funktion entspricht dem seriellen Befehl `'ZO'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetAngleDeviationMax

Deklaration:

```
static native void SetAngleDeviationMax(int value);
```

Diese Funktion setzt die maximale Winkelabweichung zwischen Sollposition und Drehgeberwert.

Die Funktion entspricht dem seriellen Befehl `'X<value>'`, siehe Befehl *2.5.14 Maximale Abweichung Drehgeber einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetAngleDeviationMax

Deklaration:

```
static native int GetAngleDeviationMax();
```

Diese Funktion liest die maximale Winkelabweichung zwischen Sollposition und Drehgeberwert aus.

Die Funktion entspricht dem seriellen Befehl `'ZX'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetCurrentReductionTime

Deklaration:

```
static native void SetCurrentReductionTime(int value);
```

Diese Funktion setzt die Wartezeit im Stillstand bis der Strom abgesenkt wird.

Die Funktion entspricht dem seriellen Befehl `'G<value>'`, siehe Befehl *2.7.8 Zeit bis zur Stromabsenkung einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetCurrentReductionTime

Deklaration:

```
static native int GetCurrentReductionTime();
```

Diese Funktion liest die Wartezeit im Stillstand bis der Strom abgesenkt wird.

Die Funktion entspricht dem seriellen Befehl `'ZG'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetReverseClearance

Deklaration:

```
static native void SetReverseClearance(int value);
```

Diese Funktion setzt das Umkehrspiel in Schritten.

Die Funktion entspricht dem seriellen Befehl `'z<value>'`, siehe Befehl *2.5.35 Umkehrspiel einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetReverseClearance

Deklaration:

```
static native int GetReverseClearance();
```

Diese Funktion liest das Umkehrspiel in Schritten.

Die Funktion entspricht dem seriellen Befehl `'Zz'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.ResetEEProm

Deklaration:

```
static native void ResetEEProm();
```

Diese Funktion setzt alle Einstellungen der Steuerung auf die Defaultwerte (Werkseinstellungen) zurück.

Die Funktion entspricht dem seriellen Befehl `'~'`, siehe Befehl *2.5.32 EEPROM Reset durchführen*.

Enthalten in Firmware neuer als 15.03.2010.

ACHTUNG: diese Funktion löscht auch das Java Programm! Das Programm läuft noch bis zum Ende (da im Speicher), lässt sich danach aber nicht mehr starten

config.SetMotorPP

Deklaration:

```
static native void SetMotorPP(int value);
```

Diese Funktion setzt die Motor Polpaare.

Die Funktion entspricht dem seriellen Befehl `':CL_motor_pp<value>'`, siehe Befehl *2.9.10 Polpaare des Motors einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetMotorPP

Deklaration:

```
static native int GetMotorPP();
```

Diese Funktion liest die Motor Polpaare.

Die Funktion entspricht dem seriellen Befehl `':CL_motor_pp'`, siehe Befehl *2.9.10 Polpaare des Motors einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetRotencInc

Deklaration:

```
static native void SetRotencInc(int value);
```

Diese Funktion setzt die Anzahl der Inkremente des Drehgebers.

Die Funktion entspricht dem seriellen Befehl `':CL_rotenc_inc<value>'`, siehe Befehl *2.9.12 Anzahl der Inkremente einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetRotencInc

Deklaration:

```
static native int GetRotencInc();
```

Diese Funktion liest die Anzahl der Inkremente des Drehgebers.

Die Funktion entspricht dem seriellen Befehl `':CL_rotenc_inc'`, siehe Befehl *2.9.12 Anzahl der Inkremente einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetBrakeTA

Deklaration:

```
static native void SetBrakeTA(int time);
```

Diese Funktion setzt die Wartezeit für das Abschalten der Bremsspannung.

Die Funktion entspricht dem seriellen Befehl `':brake_ta<time>'`, siehe Befehl *2.5.39 Wartezeit für Abschalten der Bremsspannung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetBrakeTA

Deklaration:

```
static native int GetBrakeTA();
```

Diese Funktion liest die Wartezeit für das Abschalten der Bremsspannung.

Die Funktion entspricht dem seriellen Befehl `':brake_ta'`, siehe Befehl *2.5.39 Wartezeit für Abschalten der Bremsspannung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetBrakeTB

Deklaration:

```
static native void SetBrakeTB(int time);
```

Diese Funktion setzt die Zeit zwischen dem Abschalten der Bremsspannung und dem Erlauben einer Motorbewegung.

Die Funktion entspricht dem seriellen Befehl `':brake_tb<time>'`, siehe Befehl *2.5.40 Wartezeit für Motorbewegung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetBrakeTB

Deklaration:

```
static native int GetBrakeTB();
```

Diese Funktion liest die Zeit zwischen dem Abschalten der Bremsspannung und dem Erlauben einer Motorbewegung.

Die Funktion entspricht dem seriellen Befehl `':brake_tb'`, siehe Befehl *2.5.40 Wartezeit für Motorbewegung setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetBrakeTC

Deklaration:

```
static native void SetBrakeTC(int time);
```

Diese Funktion setzt die Wartezeit für das Abschalten des Motorstroms.

Der Motorstrom wird durch Rücksetzen des Freigabe-Eingangs abgeschaltet (siehe Abschnitt *2.5.25 „Funktion der Digitaleingänge einstellen“*).

Die Funktion entspricht dem seriellen Befehl `':brake_tc<time>'`, siehe Befehl *2.5.41 Wartezeit für Abschalten Motorstrom setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetBrakeTC

Deklaration:

```
static native int GetBrakeTC();
```

Diese Funktion liest die Wartezeit für das Abschalten des Motorstroms.

Der Motorstrom wird durch Rücksetzen des Freigabe-Eingangs abgeschaltet (siehe Abschnitt *2.5.25 „Funktion der Digitaleingänge einstellen“*).

Die Funktion entspricht dem seriellen Befehl `':brake_tc'`, siehe Befehl *2.5.41 Wartezeit für Abschalten Motorstrom setzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetErrorCorrection

Deklaration:

```
static native void SetErrorCorrection(int value);
```

Diese Funktion setzt den Drehgeberüberwachungsmodus.

Die Funktion entspricht dem seriellen Befehl `'U<value>'`, siehe Befehl *2.5.10 Fehlerkorrekturmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetErrorCorrection

Deklaration:

```
static native int GetErrorCorrection();
```

Diese Funktion liest den Drehgeberüberwachungsmodus aus.

Die Funktion entspricht dem seriellen Befehl `'ZU'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetSpeedmodeControl

Deklaration:

```
static native void SetSpeedmodeControl(int value);
```

Diese Funktion setzt den Regelungstyp für den Drehzahlmodus.

Die Funktion entspricht dem seriellen Befehl `':speedmode_control<value>'`, siehe Befehl *2.9.3 Regelungstyp für Drehzahlmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetSpeedmodeControl

Deklaration:

```
static native int GetSpeedmodeControl();
```

Diese Funktion liest den Regelungstyp für den Drehzahlmodus.

Die Funktion entspricht dem seriellen Befehl `':speedmode_control'`, siehe Befehl *2.9.3 Regelungstyp für Drehzahlmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetCLMotorType

Deklaration:

```
static native void SetCLMotorType(int value);
```

Diese Funktion legt den Typ des angeschlossenen Motors fest.

Die Funktion entspricht dem seriellen Befehl `':CL_motor_type<value>'`, siehe Befehl *2.5.1 Motortyp einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetCLMotorType

Deklaration:

```
static native int GetCLMotorType();
```

Diese Funktion liest den Typ des angeschlossenen Motors.

Die Funktion entspricht dem seriellen Befehl `':CL_motor_type'`, siehe Befehl *2.5.1 Motortyp einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetFeedConstNum

Deklaration:

```
static native void SetFeedConstNum(int value);
```

Diese Funktion setzt den Zähler der Vorschubkonstanten.

Die Funktion entspricht dem seriellen Befehl `':feed_const_num<value>'`, siehe Befehl *2.5.15 Zähler für Vorschubkonstante einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetFeedConstNum

Deklaration:

```
static native int GetFeedConstNum();
```

Diese Funktion liest den Zähler der Vorschubkonstanten.

Die Funktion entspricht dem seriellen Befehl `':feed_const_num'`, siehe Befehl *2.5.15 Zähler für Vorschubkonstante einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetFeedConstDenum

Deklaration:

```
static native void SetFeedConstDenum(int value);
```

Diese Funktion setzt den Nenner der Vorschubkonstanten.

Die Funktion entspricht dem seriellen Befehl `':feed_const_denum<value>'`, siehe Befehl *2.5.16 Nenner für Vorschubkonstante einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetFeedConstDenum

Deklaration:

```
static native int GetFeedConstDenum();
```

Diese Funktion liest den Nenner der Vorschubkonstanten.

Die Funktion entspricht dem seriellen Befehl `':feed_const_denum'`, siehe Befehl *2.5.16 Nenner für Vorschubkonstante einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetCurrentTime

Deklaration:

```
static native void SetCurrentTime(int time);
```

Diese Funktion setzt die Strom-Zeitkonstante für BLDC.

Die Funktion entspricht dem seriellen Befehl `':itime<time>'`, siehe Befehl *2.5.5 Strom-Zeitkonstante für BLDC einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetCurrentTime

Deklaration:

```
static native int GetCurrentTime();
```

Diese Funktion liest die Strom-Zeitkonstante für BLDC.

Die Funktion entspricht dem seriellen Befehl `':itime'`, siehe Befehl *2.5.5 Strom-Zeitkonstante für BLDC einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetCurrentPeak

Deklaration:

```
static native void SetCurrentPeak(int value);
```

Diese Funktion setzt den Strom-Spitzenwert für BLDC.

Die Funktion entspricht dem seriellen Befehl `':ipeak <value>'`, siehe Befehl *2.5.4 Spitzenstrom für BLDC einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetCurrentPeak

Deklaration:

```
static native int GetCurrentPeak();
```

Diese Funktion liest den Strom-Spitzenwert für BLDC.

Die Funktion entspricht dem seriellen Befehl `':ipeak'`, siehe Befehl *2.5.4 Spitzenstrom für BLDC einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.ResetStartCount

Deklaration:

```
static native void ResetStartCount(int value);
```

Diese Funktion setzt den Einschaltzähler zurück.

value kann nur den Wert 1 annehmen.

Die Funktion entspricht dem seriellen Befehl `'%<value>'`, siehe Befehl *2.7.7 Einschaltzähler zurücksetzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetStartCount

Deklaration:

```
static native int GetStartCount();
```

Diese Funktion liest den Einschaltzähler.

Die Funktion entspricht dem seriellen Befehl `'Z%'`, siehe Befehl *2.7.7 Einschaltzähler zurücksetzen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetLimitSwitchBehavior

Deklaration:

```
static native void SetLimitSwitchBehavior(int value);
```

Diese Funktion setzt das Endschalterverhalten.

value kann nur den Wert 1 annehmen.

Die Funktion entspricht dem seriellen Befehl `'l<value>'`, siehe Befehl *2.5.9 Endschalterverhalten einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetLimitSwitchBehavior

Deklaration:

```
static native int GetLimitSwitchBehavior();
```

Diese Funktion liest das Endschalterverhalten aus.

Die Funktion entspricht dem seriellen Befehl `'Zl'`, siehe Befehl *2.5.9 Endschalterverhalten einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.SetMotorAddress

Deklaration:

```
static native void SetMotorAddress(int value);
```

Diese Funktion setzt die Motoradresse.

Die Funktion entspricht dem seriellen Befehl `'m<value>'`, siehe Befehl *2.5.7 Motoradresse einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

config.GetMotorAddress

Deklaration:

```
static native int GetMotorAddress();
```

Diese Funktion liest die Motoradresse.

Die Funktion entspricht dem seriellen Befehl `'Zm'`, siehe Befehl *2.5.7 Motoradresse einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.5 Klasse „drive“

drive.StartDrive

Deklaration:

```
static native void StartDrive();
```

Diese Funktion startet den Motor. Es werden dabei die aktuell eingestellten Satzdaten (Modus, Geschwindigkeit, Rampen, etc.) verwendet.

Die Funktion entspricht dem seriellen Befehl `'A'`, siehe Befehl *2.6.1 Motor starten*.

drive.StopDrive

Deklaration:

```
static native void StopDrive(int type);
```

Bricht die aktuelle Fahrt ab; type legt fest, wie gestoppt wird:

type = 0: Es wird ein Quickstop ausgeführt (Bremsung mit sehr steiler Rampe)

type = 1: Es wird mit der normalen Bremsrampe gebremst

Im Drehzahl-, Analog- und Joystickmodus die einzige Möglichkeit, den Motor in den Bereit-Zustand zu bringen.

Es werden keine Rampen gefahren, sondern der Motor sofort zum Stillstand gebracht. Dadurch können bei hohen Geschwindigkeiten Schrittverluste entstehen.

In den 3 oben genannten Modi sollte deswegen vor dem Stopp-Befehl die Geschwindigkeit heruntergefahren werden.

Die Funktion entspricht dem seriellen Befehl `'S'`, siehe Befehl *2.6.2 Motor stoppen*.

drive.SetMaxSpeed

Deklaration:

```
static native void SetMaxSpeed(int value);
```

Gibt die Maximalfrequenz in Hertz (Schritte pro Sekunde) an.

Die Maximalfrequenz wird erst nach Durchfahren der Beschleunigungsrampe erreicht.

Die Funktion entspricht dem seriellen Befehl `'o<value>'`, siehe Befehl *2.6.9 Maximalfrequenz einstellen*.

drive.GetMaxSpeed

Deklaration:

```
static native int GetMaxSpeed();
```

Liest den aktuell gültigen Wert der Maximalfrequenz in Hertz (Schritte pro Sekunde) aus.

Die Funktion entspricht dem seriellen Befehl `'Zo'`, siehe *2.3 Lesebefehl*.

drive.SetMaxSpeed2

Deklaration:

```
static native void SetMaxSpeed2(int speed);
```

Funktion setzt die obere Maximalfrequenz.

Die Funktion entspricht dem seriellen Befehl `'n<value>'`, siehe Befehl *2.6.10 Maximalfrequenz 2 einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetMaxSpeed2

Deklaration:

```
static native int GetMaxSpeed2();
```

Funktion liest die obere Maximalfrequenz.

Die Funktion entspricht dem seriellen Befehl `'Zn'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetMinSpeed

Deklaration:

```
static native void SetMinSpeed (int value);
```

Gibt die Minimalgeschwindigkeit in Hertz (Schritte pro Sekunde) an und ist nur im Open-Loop-Betrieb verwendbar.

Beim Start eines Satzes beginnt der Motor, sich mit der Minimalgeschwindigkeit zu drehen. Er beschleunigt dann mit der eingestellten Rampe bis zur Maximalgeschwindigkeit.

Die Funktion entspricht dem seriellen Befehl `'u<value>'`, siehe Befehl *2.6.8 Minimalfrequenz einstellen*.

drive.GetMinSpeed

Deklaration:

```
static native int GetMinSpeed();
```

Liest den aktuell gültigen Wert der Minimalgeschwindigkeit in Hertz (Schritte pro Sekunde) aus.

Die Funktion entspricht dem seriellen Befehl `'Zu'`, siehe *2.3 Lesebefehl*.

drive.SetAcceleration

Deklaration:

```
static native void SetAcceleration(int value);
```

Gibt die Beschleunigungsrampe an.

Zum Umrechnen der Parameter in die Beschleunigung in Hz/ms wird die folgende Formel verwendet:

Beschleunigung in Hz/ms = ((3000.0 / sqrt((float)<value>)) - 11.7).

Die Funktion entspricht dem seriellen Befehl `'b<value>'`, siehe Befehl *2.6.11 Beschleunigungsrampe einstellen*.

drive.GetAcceleration

Deklaration:

```
static native int GetAcceleration();
```

Liest den aktuell gültigen Wert der Beschleunigungsrampe aus.

Die Funktion entspricht dem seriellen Befehl `'Zb'`, siehe *2.3 Lesebefehl*.

drive.SetDeceleration

Deklaration:

```
static native void SetDeceleration(int value);
```

Diese Funktion setzt die Bremsrampe.

Die Funktion entspricht dem seriellen Befehl `'B<value>'`, siehe Befehl *2.6.12 Bremsrampe einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetDeceleration

Deklaration:

```
static native int GetDeceleration();
```

Diese Funktion liest die Bremsrampe aus.

Die Funktion entspricht dem seriellen Befehl `'ZB'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetDecelerationHalt

Deklaration:

```
static native void SetDecelerationHalt(int value);
```

Diese Funktion setzt die Quickstoprampe.

Die Funktion entspricht dem seriellen Befehl `'H<value>'`, siehe Befehl *2.6.13 Halterampe einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetDecelerationHalt

Deklaration:

```
static native int GetDecelerationHalt();
```

Diese Funktion liest die Quickstoprampe aus.

Die Funktion entspricht dem seriellen Befehl `'ZH'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetRampType

Deklaration:

```
static native void SetRampType(int ramp);
```

Diese Funktion setzt den Rampentyp.

Die Funktion entspricht dem seriellen Befehl `':ramp_mode<ramp>'`, siehe Befehl *2.5.36 Rampe setzen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetRampType

Deklaration:

```
static native int GetRampType();
```

Diese Funktion liest den Rampentyp aus.

Die Funktion entspricht dem seriellen Befehl `':ramp_mode'`, siehe *2.5.36 Rampe setzen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetJerk

Deklaration:

```
static native void SetJerk(int value);
```

Diese Funktion setzt den maximalen Ruck für die Beschleunigung in 100/s³.

Die Funktion entspricht dem seriellen Befehl `':b<value>'`, siehe Befehl *2.5.37 Maximalen Ruck für Beschleunigungsrampe setzen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetJerk

Deklaration:

```
static native int GetJerk();
```

Diese Funktion gibt den maximalen Ruck für die Beschleunigung in 100/s³ aus.

Die Funktion entspricht dem seriellen Befehl `'Z:b'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetBrakeJerk

Deklaration:

```
static native void SetBrakeJerk(int value);
```

Diese Funktion setzt den Bremsruck in 100/s³.

Die Funktion entspricht dem seriellen Befehl `':B<value>'`, siehe Befehl *2.5.38 Maximalen Ruck für Bremsrampe setzen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetBrakeJerk

Deklaration:

```
static native int GetBrakeJerk();
```

Diese Funktion liest den Bremsruck in 100/s³ aus.

Die Funktion entspricht dem seriellen Befehl `'Z:B'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.IsReferenced

Deklaration:

```
static native int IsReferenced();
```

Die Funktion liest aus, ob der Motor referenziert ist oder nicht.

Die Funktion entspricht dem seriellen Befehl `':is_referenced'`, siehe *2.5.21 „Motor ist referenziert“ abfragen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.IncreaseFrequency

Deklaration:

```
static native void IncreaseFrequency();
```

Die Funktion errhöht die Drehzahl im Drehzahlmodus um 100 Schritte/s.

Die Funktion entspricht dem seriellen Befehl `'+'`, siehe *2.7.9 Drehzahl erhöhen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.DecreaseFrequency

Deklaration:

```
static native void DecreaseFrequency();
```

Die Funktion verringert die Drehzahl im Drehzahlmodus um 100 Schritte/s.

Die Funktion entspricht dem seriellen Befehl `'+'`, siehe *2.7.10 Drehzahl verringern*.

Enthalten in Firmware neuer als 15.03.2010.

drive.TriggerOn

Deklaration:

```
static native void TriggerOn();
```

Auslöser für den Flagpositionsmodus.

Die Funktion entspricht dem seriellen Befehl `'T'`, siehe *2.7.12 Trigger auslösen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetTargetPos

Deklaration:

```
static native void SetTargetPos(int value);
```

Gibt den Verfahrweg in (Mikro-)Schritten an. Für die relative Positionierung sind nur positive Werte erlaubt. Die Richtung wird mit SetDirection eingestellt.

Für die absolute Positionierung gibt dieser Befehl die Zielposition an. Negative Werte sind hier erlaubt. Die mit SetDirection eingestellte Drehrichtung wird ignoriert, da diese sich aus der aktuellen Position und der Zielposition ergibt.

Der Wertebereich ist von -100.000.000 bis +100.000.000.

Im adaptiven Modus bezieht sich dieser Parameter auf Halbschritte.

Die Funktion entspricht dem seriellen Befehl `'s<value>'`, siehe Befehl *2.6.7 Verfahrweg einstellen*.

drive.GetTargetPos

Deklaration:

```
static native int GetTargetPos();
```

Liest den aktuell gültigen Wert des Verfahrwegs in (Mikro-)Schritten aus.

Die Funktion entspricht dem seriellen Befehl `'Zs'`, siehe *2.3 Lesebefehl*.

drive.SetMode

Deklaration:

```
static native void SetMode(int value);
```

Die Positionierarten `'p'` sind:

Positioniermodus	
p=1	Relative Positionierung; Der Befehl *2.6.7 Verfahrweg einstellen* `'s'` gibt den Verfahrweg relativ zur aktuellen Position an. Der Befehl *2.6.14 Drehrichtung einstellen* `'d'` gibt die Richtung an. Der Parameter *2.6.7 Verfahrweg einstellen* `'s'` muss positiv sein.
p=2	Absolute Positionierung; Der Befehl *2.6.7 Verfahrweg einstellen* `'s'` gibt die Zielposition relativ zur Referenzposition an. Der Befehl *2.6.14 Drehrichtung einstellen* `'d'` wird ignoriert.
p=3	Interne Referenzfahrt; Der Motor läuft mit der unteren Geschwindigkeit in die Richtung, die in Befehl *2.6.14 Drehrichtung einstellen* `'d'` eingestellt ist, bis er den Indexstrich des Drehgeber erreicht. Danach läuft der Motor eine feste Anzahl von Schritten, so dass er den Indexstrich wieder verlässt. Für die Richtung des Freifahrens siehe Befehl *2.5.9 Endschalterverhalten einstellen* `'l'`. Dieser Modus macht nur bei Motoren mit eingebautem und angeschlossenem Drehgeber Sinn.
p=4	Externe Referenzfahrt; Der Motor läuft mit der oberen Geschwindigkeit in die Richtung, die in Befehl *2.6.14 Drehrichtung einstellen* `'d'` eingestellt ist, bis er den Endschalter erreicht hat. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe Befehl *2.5.9 Endschalterverhalten einstellen* `'l'`.
Drehzahlmodus	
p=5	Drehzahlmodus; Wird der Motor gestartet, dreht der Motor bis zur Maximaldrehzahl mit der eingestellten Rampe hoch. Änderungen in der Geschwindigkeit oder Drehrichtung werden mit der eingestellten Rampe sofort angefahren, ohne dass der Motor zwischendurch gestoppt werden muss.
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus

Flagpositioniermodus	
p=6	Flagpositioniermodus; Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.12 Trigger auslösen* `'T'` oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.7 Verfahrweg einstellen* `'s'`) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.10 Maximalfrequenz 2 einstellen* `'n'`).
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Taktrichtungsmodus	
p=7	Manuell links.
p=8	Manuell rechts.
p=9	Interne Referenzfahrt; siehe Positioniermodus
p=10	Externe Referenzfahrt; siehe Positioniermodus
Analogmodus	
p=11	Analogmodus
Joystickmodus	
p=12	Joystickmodus
Analogpositioniermodus	
p=13	Analogpositioniermodus
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
HW-Referenzmodus	
p=14	HW-Referenzmodus
Drehmomentmodus	
p=15	Drehmomentmodus
CL-Schnelltestmodus	
p=16	CL-Schnelltestmodus
CL-Testmodus	
p=17	CL-Testmodus

drive.GetMode

Deklaration:

```
static native int GetMode();
```

Liest die aktuelle Positionierart aus.

Die Funktion entspricht dem seriellen Befehl `'Zp'`, siehe *2.3 Lesebefehl*.

drive.SetCurrent

Deklaration:

```
static native void SetCurrent(int value);
```

Setzt den Phasenstrom in Prozent. Werte über 100 sollten vermieden werden.

Die Funktion entspricht dem seriellen Befehl `'i<value>'`, siehe Befehl *2.5.2 Phasenstrom einstellen*.

drive.GetCurrent

Deklaration:

```
static native int GetCurrent();
```

Liest den aktuell eingestellten Phasenstrom in Prozent aus.

Die Funktion entspricht dem seriellen Befehl `'Zi'`, siehe *2.3 Lesebefehl*.

drive.SetCurrentReduction

Deklaration:

```
static native void SetCurrentReduction(int value);
```

Setzt den Strom der Stromreduzierung bei Stillstand in Prozent. Dieser Wert ist wie der Phasenstrom relativ zum Endwert. Werte über 100 sollten vermieden werden.

Die Funktion entspricht dem seriellen Befehl `'r<value>'`, siehe Befehl *2.5.3 Phasenstrom im Stillstand einstellen*.

drive.GetCurrentReduction

Deklaration:

```
static native int GetCurrentReduction();
```

Liest den aktuell eingestellten Phasenstrom im Stillstand in Prozent aus.

Die Funktion entspricht dem seriellen Befehl `'Zr'`, siehe *2.3 Lesebefehl*.

drive.GetStatus

Deklaration:

```
static native int GetStatus();
```

Gibt den aktuellen Status der Steuerung als Bitmaske zurück.

Bit 0 ready

Bit 1 reference

Bit 2 posError

Bit 3 endStartActive

Bit 4-7 mode

Die Funktion entspricht dem seriellen Befehl `'$'`, siehe Befehl *2.5.22 Status auslesen*.

drive.SetDirection

Deklaration:

```
static native void SetDirection(int value);
```

Setzt die Drehrichtung:

value=0 Drehrichtung links

value=1 Drehrichtung rechts

Die Funktion entspricht dem seriellen Befehl `'d<value>'`, siehe Befehl *2.6.14 Drehrichtung einstellen*.

drive.GetDirection

Deklaration:

```
static native int GetDirection();
```

Liest die aktuell eingestellte Drehrichtung aus.

Die Funktion entspricht dem seriellen Befehl 'Zd, siehe *2.3 Lesebefehl*.

drive.SetDirectionReversing

Deklaration:

```
static native void SetDirectionReversing (int value);
```

Diese Funktion setzt die Drehrichtungsumkehr.

Die Funktion entspricht dem seriellen Befehl `'t<value>'`, siehe Befehl *2.6.15 Richtungsumkehr einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetDirectionReversing

Deklaration:

```
static native int GetDirectionReversing ();
```

Diese Funktion liest den Wert der Drehrichtungsumkehr.

Die Funktion entspricht dem seriellen Befehl `'Zt'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetRepeat

Deklaration:

```
static native void SetRepeat (int repeat);
```

Diese Funktion setzt die Anzahl der Wiederholungen.

Die Funktion entspricht dem seriellen Befehl `'W<repeat>'`, siehe Befehl *2.6.16 Wiederholungen einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetRepeat

Deklaration:

```
static native int GetRepeat ();
```

Diese Funktion liest die Anzahl der Wiederholungen.

Die Funktion entspricht dem seriellen Befehl `'ZW'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetPause

Deklaration:

```
static native void SetPause (int pause);
```

Gibt die Pause zwischen Wiederholungen von Sätzen oder zwischen Satz und Folgesatz in ms (Millisekunden) an.

Die Funktion entspricht dem seriellen Befehl `'P<pause>'`, siehe Befehl *2.6.17 Satzpause einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetPause

Deklaration:

```
static native int GetPause ();
```

Diese Funktion liest die Pausenzeit in Millisekunden.

Die Funktion entspricht dem seriellen Befehl `'ZP'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.SetNextRecord

Deklaration:

```
static native void SetNextRecord (int record);
```

Diese Funktion setzt den Folgesatz.

Die Funktion entspricht dem seriellen Befehl `'N< record>'`, siehe Befehl *2.6.18 Folgesatz einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetNextRecord

Deklaration:

```
static native int GetNextRecord ();
```

Diese Funktion liest die Nummer des Folgesatzes.

Die Funktion entspricht dem seriellen Befehl `'ZN'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

drive.GetEncoderPosition

Deklaration:

```
static native int GetEncoderPosition();
```

Liest die aktuelle Position des Drehgebers aus.

Die Funktion entspricht dem seriellen Befehl `'I'`, siehe Befehl *2.5.19 Drehgeberposition auslesen*.

drive.GetDemandPosition

Deklaration:

```
static native int GetDemandPosition();
```

Liest die aktuelle Position des Motors aus.

Die Funktion entspricht dem seriellen Befehl `'C'`, siehe Befehl *2.5.20 Position auslesen*.

drive.SetPosition

Deklaration:

```
static native void SetPosition(int value);
```

Setzt einen Fehler der Drehgeberüberwachung zurück und setzt die Ist- und Soll-Position auf den übergebenen Wert.

Die Funktion entspricht dem seriellen Befehl `'D<value>'`, siehe Befehl *2.5.17 Positionsfehler zurücksetzen.*

Funktion enthalten in Firmware neuer als 15.03.2010.

drive.LoadDataSet

Deklaration:

```
public static native void LoadDataSet (int whichone);
```

Parameter: int whichone 1-32

Rückgabe: keine

Lädt den gewählten Datensatz in die Steuerung. Die Datensätze können mittels NanoPro konfiguriert werden.

Die Funktion entspricht dem seriellen Befehl `'y'`, siehe Befehl 2.6.3 Satz aus EEPROM laden.

drive.SaveDataSet

Deklaration:

```
static native void SaveDataSet(int whichone);
```

Parameter: int whichone 1-32

Rückgabe: keine

Schreibt die Werte im Speicher der Steuerung in den gewählten Datensatz.

Die Funktion entspricht dem seriellen Befehl `'>'`, siehe Befehl *2.6.5. Satz speichern.*

Funktion enthalten in Firmware neuer als 15.03.2010.

3.5.6 Klasse „dspdrive“

Anwendung

Die Klasse dspdrive dient zur Konfiguration des Stromreglers bei Steuerungen, welche mit DSPdrive ausgestattet sind.

dspdrive.SetDSPDrivePLow

Deklaration:

```
static native void SetDSPDrivePLow(int value);
```

Diese Funktion setzt den P-Anteil des Stromreglers im Stillstand.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_low<value>'`, siehe Befehl *2.12.1 P-Anteil des Stromreglers im Stillstand einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDrivePLow

Deklaration:

```
static native int GetDSPDrivePLow();
```

Diese Funktion liest den P-Anteil des Stromreglers im Stillstand.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_low'`, siehe Befehl *2.12.1 P-Anteil des Stromreglers im Stillstand einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.SetDSPDrivePHigh

Deklaration:

```
static native void SetDSPDrivePHigh(int value);
```

Diese Funktion setzt den P-Anteil des Stromreglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_hig<value>'`, siehe Befehl *2.12.2 P-Anteil des Stromreglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDrivePHigh

Deklaration:

```
static native int GetDSPDrivePHigh();
```

Diese Funktion liest den P-Anteil des Stromreglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_hig'`, siehe Befehl *2.12.2 P-Anteil des Stromreglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.SetDSPDrivePScale

Deklaration:

```
static native void SetDSPDrivePScale(int value);
```

Diese Funktion setzt den Skalierungsfaktor zur drehzahlabhängigen. Anpassung des P-Anteils des Reglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_scale<value>'`, siehe Befehl *2.12.3 Skalierungsfaktor zur drehzahlabh. Anpassung des P-Anteils des Reglers während der Fahrt einstellen.*

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDrivePScale

Deklaration:

```
static native int GetDSPDrivePScale();
```

Diese Funktion liest den Skalierungsfaktor zur drehzahlabhängigen Anpassung des P-Anteils des Reglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_scale'`, siehe Befehl *2.12.3 Skalierungsfaktor zur drehzahlabh. Anpassung des P-Anteils des Reglers während der Fahrt einstellen.*

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.SetDSPDriveILow

Deklaration:

```
static native void SetDSPDriveILow(int value);
```

Diese Funktion setzt den I-Anteil des Stromreglers im Stillstand.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KI_low<value>'`, siehe Befehl *2.12.4 I-Anteil des Stromreglers im Stillstand einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDriveILow

Deklaration:

```
static native int GetDSPDriveILow();
```

Diese Funktion liest den I-Anteil des Stromreglers im Stillstand.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KP_low'`, siehe Befehl *2.12.4 I-Anteil des Stromreglers im Stillstand einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.SetDSPDriveIHigh

Deklaration:

```
static native void SetDSPDriveIHigh(int value);
```

Diese Funktion setzt den I-Anteil des Stromreglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KI_hig<value>'`, siehe Befehl *2.12.5 I-Anteil des Stromreglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDriveIHigh

Deklaration:

```
static native int GetDSPDriveIHigh();
```

Diese Funktion liest den I-Anteil des Stromreglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KI_hig'`, siehe Befehl *2.12.5 I-Anteil des Stromreglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.SetDSPDriveIScale

Deklaration:

```
static native void SetDSPDriveIScale(int value);
```

Diese Funktion setzt den Skalierungsfaktor zur drehzahlabhängigen Anpassung des I-Anteils des Reglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KI_scale<value>'`, siehe Befehl *2.12.6 Skalierungsfaktor zur drehzahlabh. Anpassung des I-Anteils des Reglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

dspdrive.GetDSPDriveIScale

Deklaration:

```
static native int GetDSPDriveIScale();
```

Diese Funktion liest den Skalierungsfaktor zur drehzahlabhängigen Anpassung des I-Anteils des Reglers während der Fahrt.

Die Funktion entspricht dem seriellen Befehl `':dspdrive_KI_scale'`, siehe Befehl *2.12.6 Skalierungsfaktor zur drehzahlabh. Anpassung des I-Anteils des Reglers während der Fahrt einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.7 Klasse „io“

Anwendung

Die Klasse io dient zur Verwaltung der digitalen und analogen Ein- und Ausgänge.

io.SetLED

Deklaration:

```
static native void SetLED(int in);
```

Setzt die Fehler-LED.

1: LED ein

2: LED aus

io.SetDigitalOutput

Deklaration:

```
static native void SetDigitalOutput(int value);
```

Setzt die digitalen Ausgänge der Steuerung bit-codiert.

io.GetDigitalOutput

Deklaration:

```
static native int GetDigitalOutput();
```

Liest die aktuell eingestellte Bitmaske für die digitalen Ausgänge aus.

io.GetDigitalInput

Deklaration:

```
static native int GetDigitalInput();
```

Liest die aktuell anliegenden digitalen Eingänge aus.

io.GetAnalogInput

Deklaration:

```
static native int GetAnalogInput(int Port);
```

Liest die aktuellen Werte der analogen Eingänge aus. Port gibt dabei den zu lesenden Port an: 1 für den ersten Analogport, 2 für den zweiten Port (wenn vorhanden).

io.SetAnalogDead

Deklaration:

```
static native void SetAnalogDead(int analogDead);
```

Diese Funktion setzt den Totbereich des Analogeingangs.

Die Funktion entspricht dem seriellen Befehl `'=<value>'`, siehe Befehl *2.7.1 Totbereich Joystickmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetAnalogDead

Deklaration:

```
static native int GetAnalogDead();
```

Diese Funktion liest den Totbereich des Analogeingangs.

Die Funktion entspricht dem seriellen Befehl `'Z='`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetAnalogFilter

Deklaration:

```
static native void SetAnalogFilter(int filter);
```

Diese Funktion setzt den Wert für den Filter des Analogeingangs.

Die Funktion entspricht dem seriellen Befehl `'f<filter>'`, siehe Befehl *2.7.2 Filter für Analog- und Joystickmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetAnalogFilter

Deklaration:

```
static native int GetAnalogFilter();
```

Diese Funktion liest den Wert für den Filter des Analogeingangs aus.

Die Funktion entspricht dem seriellen Befehl `'Zf'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInputMaskEdge

Deklaration:

```
static native void SetInputMaskEdge(int mask);
```

Diese Funktion setzt die Polarität der Ein- und Ausgänge.

Die Funktion entspricht dem seriellen Befehl `'h<mask>'`, siehe Befehl *2.5.27 Eingänge maskieren und demaskieren*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInputMaskEdge

Deklaration:

```
static native int GetInputMaskEdge();
```

Diese Funktion liest die aktuelle Polarität der Ein- und Ausgänge aus.

Die Funktion entspricht dem seriellen Befehl `'Zh'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetDebounceTime

Deklaration:

```
static native void SetDebounceTime(int time);
```

Diese Funktion setzt die Entprellzeit für die Eingänge in Millisekunden.

Die Funktion entspricht dem seriellen Befehl `'K<time>'`, siehe Befehl *2.5.29 Debounce-Zeit für Eingänge setzen (Entprellen)*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetDebounceTime

Deklaration:

```
static native int GetDebounceTime();
```

Diese Funktion liest die Entprellzeit für die Eingänge in Millisekunden aus.

Die Funktion entspricht dem seriellen Befehl `'ZK'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput1Selection

Deklaration:

```
static native void SetInput1Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 1.

Die Funktion entspricht dem seriellen Befehl `':port_in_a<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput1Selection

Deklaration:

```
static native int GetInput1Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 1 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_a'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput2Selection

Deklaration:

```
static native void SetInput2Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 2.

Die Funktion entspricht dem seriellen Befehl `':port_in_b<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput2Selection

Deklaration:

```
static native int GetInput2Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 2 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_b'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput3Selection

Deklaration:

```
static native void SetInput3Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 3.

Die Funktion entspricht dem seriellen Befehl `':port_in_c<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput3Selection

Deklaration:

```
static native int GetInput3Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 3 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_c'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput4Selection

Deklaration:

```
static native void SetInput4Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 4.

Die Funktion entspricht dem seriellen Befehl `':port_in_d<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput4Selection

Deklaration:

```
static native int GetInput4Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 4 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_d'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput5Selection

Deklaration:

```
static native void SetInput5Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 5.

Die Funktion entspricht dem seriellen Befehl `':port_in_e<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput5Selection

Deklaration:

```
static native int GetInput5Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 5 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_e'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput6Selection

Deklaration:

```
static native void SetInput6Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 6.

Die Funktion entspricht dem seriellen Befehl `':port_in_f<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput6Selection

Deklaration:

```
static native int GetInput6Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 6 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_f'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput7Selection

Deklaration:

```
static native void SetInput7Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 7.

Die Funktion entspricht dem seriellen Befehl `':port_in_g<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput7Selection

Deklaration:

```
static native int GetInput7Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 7 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_g'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetInput8Selection

Deklaration:

```
static native void SetInput8Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitaleingang 8.

Die Funktion entspricht dem seriellen Befehl `':port_in_h<function>'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetInput8Selection

Deklaration:

```
static native int GetInput8Selection();
```

Diese Funktion liest die Funktion für den Digitaleingang 8 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_h'`, siehe Befehl *2.5.25 Funktion der Digitaleingänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput1Selection

Deklaration:

```
static native void SetOutput1Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 1.

Die Funktion entspricht dem seriellen Befehl `':port_out_a<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput1Selection

Deklaration:

```
static native int GetOutput1Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 1 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_a'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput2Selection

Deklaration:

```
static native void SetOutput2Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 2.

Die Funktion entspricht dem seriellen Befehl `':port_out_b<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput2Selection

Deklaration:

```
static native int GetOutput2Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 2 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_b'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput3Selection

Deklaration:

```
static native void SetOutput3Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 3.

Die Funktion entspricht dem seriellen Befehl `':port_out_c<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput3Selection

Deklaration:

```
static native int GetOutput3Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 3 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_c'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput4Selection

Deklaration:

```
static native void SetOutput4Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 4.

Die Funktion entspricht dem seriellen Befehl `':port_out_d<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput4Selection

Deklaration:

```
static native int GetOutput4Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 4 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_d'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput5Selection

Deklaration:

```
static native void SetOutput5Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 5.

Die Funktion entspricht dem seriellen Befehl `':port_out_e<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput5Selection

Deklaration:

```
static native int GetOutput5Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 5 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_e'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput6Selection

Deklaration:

```
static native void SetOutput6Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 6.

Die Funktion entspricht dem seriellen Befehl `':port_out_f<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput6Selection

Deklaration:

```
static native int GetOutput6Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 6 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_f'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput7Selection

Deklaration:

```
static native void SetOutput7Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 7.

Die Funktion entspricht dem seriellen Befehl `':port_out_g<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput7Selection

Deklaration:

```
static native int GetOutput7Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 7 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_g'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetOutput8Selection

Deklaration:

```
static native void SetOutput8Selection(int function);
```

Diese Funktion setzt die Funktion für den Digitalausgang 8.

Die Funktion entspricht dem seriellen Befehl `':port_out_h<function>'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetOutput8Selection

Deklaration:

```
static native int GetOutput8Selection();
```

Diese Funktion liest die Funktion für den Digitalausgang 8 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_h'`, siehe Befehl *2.5.26 Funktion der Digitalausgänge einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetAnalogMin

Deklaration:

```
static native void SetAnalogMin(int value);
```

Diese Funktion setzt die minimale Spannung für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `'Q<value>'`, siehe Befehl *2.7.3 Minimalspannung für Analogmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetAnalogMin

Deklaration:

```
static native int GetAnalogMin();
```

Diese Funktion liest die minimale Spannung für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `'ZQ'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

io.SetAnalogMax

Deklaration:

```
static native void SetAnalogMax(int value);
```

Diese Funktion setzt die maximale Spannung für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `'R<value>'`, siehe Befehl *2.7.4 Maximalspannung für Analogmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

io.GetAnalogMax

Deklaration:

```
static native int GetAnalogMax();
```

Diese Funktion liest die maximale Spannung für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `'ZR'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

3.5.8 Klasse „util“

util.GetMillis

Deklaration:

```
static native int GetMillis();
```

Liest die Zeit seit dem Einschalten der Steuerung in Millisekunden aus.

util.Sleep

Deklaration:

```
static void Sleep(int ms);
```

Wartet für ms Millisekunden.

util.TestBit

Deklaration:

```
static boolean TestBit(int value, int whichone);
```

Prüft, ob ein Bit gesetzt ist.

value	=	Wert, der das zu prüfende Bit enthält
whichone	=	Gibt an, welches Bit getestet werden soll 0 entspricht dem niederwertigsten Bit
Rückgabe	=	true, wenn das Bit gesetzt ist, false sonst

util.SetBit

Deklaration:

```
static int SetBit(int value, int whichone);
```

Setzt ein Bit in einem Integer.

Value	=	Wert, in dem das Bit gesetzt werden soll
whichone	=	Gibt an, welches Bit gesetzt werden soll 0 entspricht dem niederwertigsten Bit
Rückgabe	=	Der veränderte Wert

util.ClearBit

Deklaration:

```
static int ClearBit(int value, int whichone);
```

Löscht ein Bit in einem Integer.

Value	=	Wert, in dem das Bit gelöscht werden soll
whichone	=	Gibt an, welches Bit gelöscht werden soll 0 entspricht dem niederwertigsten Bit
Rückgabe	=	Der veränderte Wert

util.SetStepMode

Deklaration:

```
static native void SetStepMode(int value);
```

Diese Funktion setzt den Schrittmodus.

Die Funktion entspricht dem seriellen Befehl `'g<value>'`, siehe Befehl *2.5.6 Schrittmodus einstellen*.

Enthalten in Firmware neuer als 15.03.2010.

util.GetStepMode

Deklaration:

```
static native int GetStepMode();
```

Diese Funktion liest den Schrittmodus.

Die Funktion entspricht dem seriellen Befehl `'Zg'`, siehe *2.3 Lesebefehl*.

Enthalten in Firmware neuer als 15.03.2010.

3.6 Java Programmbeispiele

Es folgen einige kurze Beispielprogramme. Die Programme liegen sowohl als Quellcode, als auch in bereits compilierter Form im Verzeichnis „Beispiele“.

3.6.1 AnalogExample.java

```
/** liest alle 2 Sekunden den Analogwert und fährt eine daraus
 *  berechnete Position an
 *
 * */
import nanotec.*;
class AnalogExample {
        /** liest den Analogwert und berechnet daraus
         *  eine Zielposition
         *
         * */
        static int CalculateTargetPos( ){
                int pos = io.GetAnalogInput( 1 );
                pos = (pos * 2) - 1000;
                return pos;
        }

        public static void main() {
                //Motor konfigurieren
                util.SetStepMode(4);            //1/4 Schritt
                drive.SetTargetPos(0);  //Zielposition:0
                drive.SetMaxSpeed(2000);        //Geschwindigkeit
                drive.SetMode(2);               //absolute Positionierung
                //Hauptschleife
                while(true){
                        io.SetLED(1);
                        util.Sleep(100);

                        io.SetLED(0);
                        util.Sleep(1800);

                        drive.StopDrive( 0 );
                        drive.SetTargetPos( CalculateTargetPos () );
                        drive.StartDrive( );
                }
```

```
        }
    }
```

3.6.2 DigitalExample.java

```
/** wenn Eingang 1 aktiv ist, wird die led eingeschaltet
 *  wenn Eingang 2 aktiv ist, wird der Wert des Analogeingangs über die
 *  serielle Schnittstelle gesendet
 *
 *
 * */

import nanotec.*;

class DigitalExample {

    public static void main() {

        int input = 0;
        int cnt = 0;

        //Hauptschleife
        while(true){

            input =  io.GetDigitalInput( );

            //Bit 0 entspricht Eingang 1
            if( util.TestBit(input,0) ){
                io.SetLED(1);
            } else {
                io.SetLED(0);
            }

            cnt ++;
            //Analogwert nicht permanent senden, da sonst nur //schlecht
            lesbar
            if( util.TestBit(input,1) && ((cnt % 50) == 0)){
                comm.SendInt( io.GetAnalogInput(1) );
            }
        }
```

```
                }
            }
```

3.6.3 TimerExample.java

```
/** Beispiel für einen mit GetMillis() realisierten Timer
 *
 *  Das Programm lässt die Rote LED blinken
 * */

import nanotec.*;

class TimerExample {

        public static void main() {

                //Hauptschleife
                while(true){
                        io.SetLED(1);
                        util.Sleep(200);

                        io.SetLED(0);
                        util.Sleep(1800);
                }
        }
}
```

3.6.4 ConfigDriveExample.java

```
/** Konfiguriert den motor auf absolut positionierung
 *  und fährt zwischen 2 positionen mit verschiedenen geschwindigkeiten
 *  hin und her
 * */

import nanotec.*;

class ConfigDriveExample {

        public static void main() {

                //Motor konfigurieren
                drive.SetMode(2);                       //Absolut Positionierung
                drive.SetMinSpeed(100);
                drive.SetAcceleration(2000);            //Rampe
                drive.SetCurrent(10);                   //Strom
                drive.SetCurrentReduction(1);   //Strom für Reduzierung
                util.SetStepMode(2);                    //1/2 Schritt modus


                //Hauptschleife
                while(true){

                        drive.SetMaxSpeed(1000);        //Geschwindigkeit
                        drive.SetTargetPos(1000);       //Ziel

                        drive.StartDrive( );
                        util.Sleep(4000);               //4 Sekunde warten

                        drive.SetMaxSpeed(2000);        //Geschwindigkeit
                        drive.SetTargetPos(10); //Ziel
                        drive.StartDrive( );
                        util.Sleep(2000);               //2 Sekunden warten
                }
        }
}
```

3.6.5 DigitalOutput.java

```
/**setzt die Ausgänge und sendet den aktuellen Status
 * über die serielle Schnittstelle
 *
 * */

import nanotec.*;

class DigitalOutput {

        public static void main() {

                util.Sleep(200);

                while(1 == 1){
                        io.SetDigitalOutput(1);
                        comm.SendInt( io.GetDigitalOutput( ) );
                        util.Sleep(1000);

                        io.SetDigitalOutput(2);
                        comm.SendInt( io.GetDigitalOutput( ) );
                        util.Sleep(1000);

                        io.SetDigitalOutput(4);
                        comm.SendInt( io.GetDigitalOutput( ) );
                        util.Sleep(1000);

                        io.SetDigitalOutput(7);
                        comm.SendInt( io.GetDigitalOutput( ) );
                        util.Sleep(1000);

                        io.SetDigitalOutput(0);
                        comm.SendInt( io.GetDigitalOutput( ) );
                        util.Sleep(1000);
                }
        }
}
```

3.6.6 ExportAnalogIn.java

```
/** Liest den Analogwert und Skaliert Ihn. Das Ergebnis
 * wird in die Einstellung "Totbereich Joystikmodus" geschrieben.
 * Somit kann der jeweils aktuelle Wert mit dem Befehl 'Z='
 * ausgelesen werden (z.B. #1Z= für Motor ID 1)
 * Bitte Beachten: da die Einstellung für den Totbereich verändert wird,
 * kann dieses Programm nicht zusammen mit einem Analogmodus betrieben
 * werden.
*/

import nanotec.*;

class ExportAnalogIn {

        public static void main() {
                while(true){
                        util.Sleep(1000);
                        io.SetAnalogDead((io.GetAnalogInput(1) - 500) / 10);
                }
        }

}
```

3.7 Manuelles Übersetzen und Übertragen eines Programms ohne NanoJEasy

3.7.1 Erforderliche Tools

Einleitung

Alternativ zum Übersetzen und Übertragen von Programmen aus der Programmierumgebung heraus können Programme auch manuell übersetzt und übertragen werden. Es wird jedoch empfohlen, NanoJEasy zu verwenden, da dies komfortabler und weniger fehleranfällig ist.

Java SE

NanoJEasy enthält den freien Java-Compiler gcj des GNU-Projekts zum Übersetzen der Java-Dateien. Dieser befindet sich innerhalb des NanoJEasy-Installationsverzeichnisses im Unterverzeichnis java/bin.

Alternativ kann auch die Standard Java Implementierung Java SE der Firma Oracle verwendet werden. Dazu kann das JDK (Java Development Kit) kostenlos von der Webseite oracle.com heruntergeladen werden.

ejvm_linker

Der ejvm_linker ist ein Kommandozeilen-Programm, welches Java.class-Dateien so konvertiert, dass Sie von der Steuerung verarbeitet werden können.

Das Programm muss nicht unbedingt installiert werden. Es ist jedoch hilfreich, wenn Sie es in die PATH-Variable eintragen. Damit können Sie beim Starten des Programms das Eingeben des kompletten Pfads vermeiden.

Gehen Sie zum Eintragen des Programms in die PATH-Variable wie folgt vor:

Schritt	Durchführung
1	Wählen Sie unter Start -> Einstellungen -> Systemsteuerung -> System die Registerkarte „Erweitert“.
2	Klicken Sie auf die Schaltfläche <Umgebungsvariablen>.
3	Markieren Sie im Fenster „Systemvariablen“ die Variable.
4	Klicken Sie unter dem Fenster „Systemvariablen“ auf <Bearbeiten>.
5	Geben Sie unter „Wert der Variablen“ den Installationspfad von NanoJEasy ein.
6	Klicken Sie auf <OK>.

Firmware-Utility

Das Firmware-Utility (Version 1.2 oder höher benötigt) dient zum Übertragen von Firmware bzw. Programmdateien auf eine Steuerung. Das Programm muss nicht installiert werden, das Ausführen der firmware_util.exe genügt.

ejvm_emulator

Der ejvm_emulator dient zum Funktionstest des Programms auf dem PC. Mit dem Emulator können Probleme wie ein Stacküberlauf der VM simuliert werden.

3.7.2 Programm übersetzen

Das Programm muss mit dem GNU Java Compiler übersetzt werden:

`gcj.exe –C Meinprogramm.java`

Alternativ kann das Programm mit dem normalen Java SE Compiler übersetzt werden:

`javac.exe Meinprogramm.java`

Das Ergebnis ist ein .class File, welches das fertige Programm in binärer Form enthält:

`Meinprogramm.class`

„*Meinprogramm*“ ist der Platzhalter für den Namen Ihres Programms.

3.7.3 Programm linken und konvertieren

Überblick

Bevor das Programm auf die Steuerung übertragen werden kann, muss es gelinkt und konvertiert werden. Dies erfolgt mithilfe der ejvm_linker.exe. Bei der Konvertierung werden auch einige Überprüfungen durchgeführt, insbesondere die Programmgröße.

ejvm_linker.exe starten

Geben Sie ein:

`ejvm_linker.exe Meinprogramm.class Meinprogramm.prg`

„*Meinprogramm*“ ist der Platzhalter für den Namen Ihres Programms.

Im Regelfall ist zusätzlich die Angabe der Nanotec-Klassen notwendig, die hinzugelinkt werden sollen:

`ejvm_linker.exe Meinprogramm.class` **`nanotec\comm.class nanotec\config.class nanotec\drive.class nanotec\io.class nanotec\cl.class nanotec\util.class nanotec\dspdrive.class nanotec\capture.class`** `Meinprogramm.prg`

Ergebnis

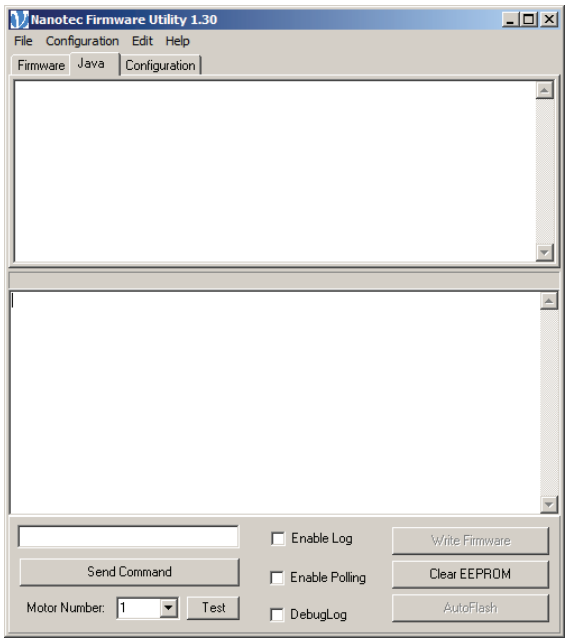
Das Ergebnis des Linkens und Konvertierens ist eine .prg-Datei, welche in die Steuerung geladen werden kann:

`Meinprogramm.prg`

3.7.4 Programm auf die Steuerung übertragen

Dialogfenster Firmware-Utility

Die Übertragung auf die Steuerung erfolgt mit Firmware-Utility:



Vorgehensweise

Gehen Sie zum Eintragen des Programms in die PATH-Variable wie folgt vor:

Schritt	Durchführung
1	Öffnen Sie den Menüpunkt „Configuration“ und tragen Sie den korrekten Com-Port und eine Baudrate von 115.200 ein.
2	Überprüfen Sie, ob die im Eingabefeld „Motor Number“ stehende Nummer mit der Stellung des Hex-Schalters der Steuerung übereinstimmt (siehe hierzu das Handbuch der Steuerung).
3	Öffnen Sie den Menüpunkt File -> Open und wählen Sie die .prg-Datei Ihres Programms aus. Das obere Textfeld von Firmware-Utility wird ausgefüllt.
4	Klicken Sie zum Übertragen des Programms zur Steuerung auf die Schaltfläche <Transfer Program>.

3.7.5 Programm ausführen

PD4 Utility

Mit Firmware-Utility können auch serielle Kommandos an die Steuerung übertragen werden. Hierfür geben Sie das gewünschte Kommando in das Textfeld über der Schaltfläche <Send Command> ein.

Es gibt die in den folgenden Absätzen genannten Befehle:

(JA ... Geladenes Java-Programm starten

Dieser Befehl startet das Programm. Als Antwort erhält man (JA+ wenn das Programm erfolgreich gestartet wurde bzw. (JA- wenn das Programm nicht gestartet werden konnte (kein gültiges oder gar kein Programm auf der Steuerung installiert). Siehe auch Abschnitt *2.8.2 Geladenes Java-Programm starten*.

(JS ... Laufendes Java-Programm stoppen

Dieser Befehl stoppt das Programm.

Als Antwort erhält man (JS+ wenn das Programm erfolgreich gestoppt wurde bzw. (JS- wenn das Programm bereits beendet war. Siehe auch Abschnitt *2.8.3 Laufendes Java-Programm stoppen*.

(JB ... Java-Programm beim Einschalten der Steuerung automatisch starten

Mit diesem Befehl kann festgelegt werden, ob das Programm beim Einschalten der Steuerung automatisch gestartet wird:

- (JB=1 das Programm wird automatisch gestartet.
- (JB=0 das Programm wird nicht automatisch gestartet.

Siehe auch Abschnitt *2.8.4 Java-Programm beim Einschalten der Steuerung automatisch starten*.

(JE ... Fehler des Java-Programms auslesen

Dieser Befehl liest den letzten Fehler aus:

- ERROR_NOT_NATIVE 1
- ERROR_FUNCTION_PARAMETER_TYPE 2
- ERROR_FUNCTION_NOT_FOUND 3
- ERROR_NOT_LONG 4
- ERROR_UNKNOWN_OPCODE 5
- ERROR_TOO_MANY_PARAMS 6
- ERROR_NO_MAIN_METHOD 7
- ERROR_CP_OUT_OF_RANGE 8
- ERROR_LOCAL_VAR_OUT_OF_RANGE 9
- ERROR_NOT_AN_VAR_IDX A
- ERROR_VAR_IS_NO_INT B
- ERROR_STACK_OVERFLOW C
- ERROR_STACK_UNDERFLOW D
- ERROR_HEAP_OVERFLOW E
- ERROR_HEAP_UNDERFLOW F
- ERROR_FRAME_OVERLOW 10
- ERROR_UNKNOWN_DATATYPE 11
- ERROR_LOCAL_VAR_OVERFLOW 12

Siehe auch Abschnitt *2.8.5 Fehler des Java-Programms auslesen* und *3.8 Mögliche Java-Fehlermeldungen*.

(JW ... Warnung auslesen

Dieser Befehl liest die letzte Warning aus:

WARNING_FUNCTION_NOT_SUPPORTED 1

Um Ausgaben des Programms angezeigt zu bekommen, muss der Haken „Debug Log“ gesetzt sein (siehe Programmbeispiel „DigitalOutput.java“). Siehe auch Abschnitt *2.8.6 Warnung des Java-Programms auslesen*.

3.8 Mögliche Java-Fehlermeldungen

Bedeutung der Fehlermeldungen

Die mit dem Befehl '(JE“ ausgelesenen Fehlermeldungen haben folgende Bedeutung:

Index	Fehlermeldung	Bedeutung
1	ERROR_NOT_NATIVE	Dieser Befehl wird von der Steuerung nicht unterstützt.
2	ERROR_FUNCTION_PARAMETER_TYPE	Der Übergabeparameter einer Funktion hat den falschen Typ (z.B. „float“ anstatt „int“).
3	ERROR_FUNCTION_NOT_FOUND	Es wurde eine unbekannte Funktion aufgerufen. Überprüfen, ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
4	ERROR_NOT_LONG	Es wird ein falscher Datentyp verwendet (sollte „long“ sein).
5	ERROR_UNKNOWN_OPCODE	Es wird eine nicht unterstützte Java-Funktion aufgerufen (z.B. „new“).
6	ERROR_TOO_MANY_PARAMS	Die Anzahl der Parameter bei einem Funktionsaufruf stimmt nicht.
7	ERROR_NO_MAIN_METHOD	Die Funktion „public static void main()“ fehlt.
8	ERROR_CP_OUT_OF_RANGE	Speicherfehler: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
9	ERROR_LOCAL_VAR_OUT_OF_RANGE	Speicherfehler: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
A	ERROR_NOT_AN_VAR_IDX	Speicherfehler: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
B	ERROR_VAR_IS_NO_INT	Es wird ein falscher Datentyp verwendet (sollte „int“ sein).
C	ERROR_STACK_OVERFLOW	Stack-Überlauf: es wurden zu viele Funktionsaufrufe ineinander geschachtelt (möglicherweise zu tiefe Rekursion).

Index	Fehlermeldung	Bedeutung
D	ERROR_STACK_UNDERFLOW	Stack-Unterlauf: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
E	ERROR_HEAP_OVERFLOW	Heap-Überlauf: es wurden zu viele Funktionsaufrufe ineinander geschachtelt (möglicherweise zu tiefe Rekursion).
F	ERROR_HEAP_UNDERFLOW	Heap-Unterlauf: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).
10	ERROR_FRAME_OVERLOW	Frame Überlauf: es wurden zu viele Klassenaufrufe verwendet.
11	ERROR_UNKNOWN_DATATYPE	Es wird ein unbekannter Datentyp verwendet.
12	ERROR_LOCAL_VAR_OVERFLOW	Speicherfehler: überprüfen ob alle Dateien eingebunden sind. Siehe auch Abschnitt *3.4.3 Integrierte Befehle* (Einbindungs-Manager).

Siehe auch Abschnitt *2.8.5 Fehler des Java-Programms auslesen* und Abschnitt *3.7.5 Programm ausführen*.

4 Programmierung über die COM-Schnittstelle

4.1 Übersicht

Zu diesem Kapitel

Dieses Kapitel enthält eine Übersicht über die COM-Schnittstelle für das Programmieren der Nanotec Schrittmotorsteuerungen.

Betriebssysteme und NanoPro-Versionen

Die benötigten Funktionen für eine serielle Kommunikation mit den Schrittmotorsteuerungen sind im Moment ausschließlich für das Betriebssystem Windows und deren Derivate (x64) geschrieben.

Diese Dokumentation ist ab der NanoPro-Version 1.60.0.0 und SDK-Version 1.60.0.0 gültig.

Voraussetzungen

Um ein Programm zur Ansteuerung für die Schrittmotorsteuerungen zu entwickeln, sollten folgende Voraussetzungen erfüllt sein:

- Es sollten Programmierkenntnisse vorhanden sein.
- Das SDK (Software Development Kit) für „NanoPro“ sollte installiert sein. Durch dessen Installation wird die CommandsPD4I.dll registriert.
- Das .net-Framework 2.0 muss installiert sein.

Programmierumgebungen

Als Programmierumgebung kann Microsoft Visual Studio oder jede andere geeignete hochsprachige IDE verwendet werden. Die mit NanoPro mitgelieferten Beispielprojekte wurden mit Microsoft Visual Studio erstellt.

Programmbeispiele

Einige Beispiele für die Benutzung der CommandsPD4I sind im NanoPro-Installationsverzeichnis im Unterverzeichnis SDK\example zu finden. Alle Beispiele sind als Projekte für Microsoft Visual Studio realisiert.

4.2 Befehlsübersicht

Nachfolgend finden Sie eine Auflistung der Befehle für die Programmierung über die COM-Schnittstelle:

4.3 Beschreibung der Funktionen

4.3.1 Allgemein

Methoden

Es gibt zwei Kategorien von Methoden:

- Set-Methoden, welche Informationen an die Steuerung übergeben. Mit dem Rückgabewert bei den Set-Methoden kann geprüft werden, ob die Information auch zur Steuerung gesendet worden ist.
- Get-Methoden, die Informationen von der Steuerung holen.

Abruf des Status der Objekte

Explizit können nach jedem Methodenaufruf mit folgenden Funktionen Informationen über den Status des Objekts abgerufen werden:

- Errorflag — diese Funktion liefert den Fehlerstatus zurück
- ErrorNumber — diese Funktion gibt die Fehlernummer zurück
- ErrorMessageString — diese Funktion liefert eine Beschreibung des Fehlers zurück

4.3.2 Auflistung der Funktionen

ErrorFlag

Definition:

```
bool ErrorFlag
```

Hat diese Variable den Wert true, so ist ein Fehler aufgetreten.

ErrorNumber

Definition:

```
int ErrorNumber
```

In dieser Variable wird die Nummer eines eventuell aufgetretenen Fehlers gespeichert.

ErrorMessageString

Definition:

```
string ErrorMessageString
```

In dieser Variable wird die Beschreibung eines eventuell aufgetretenen Fehlers gespeichert.

SerialPorts

Definition:

```
string[] SerialPorts
```

Dieses Feld enthält eine Liste der vorhandenen seriellen Schnittstellen des Computersystems.

SelectedPort

Definition:

```
string SelectedPort
```

Mit Hilfe dieser Variable wird die zu verwendende serielle Schnittstelle festgelegt (z.B. „COM1“).

Baudrate

Definition:

```
int Baudrate
```

Mit Hilfe dieser Variable wird die zu verwendende Übertragungsrate festgelegt.

Supportlog

Definition:

```
bool Supportlog
```

Mit dieser Variable kann festgelegt werden, ob ein Supportlog geschrieben werden soll.

GetAvailableMotorAddresses

Definition:

```
IList<int> GetAvailableMotorAddresses
```

Dieses Feld enthält eine Liste der möglichen Motoradressen.

MotorAddresse

Definition:

```
int MotorAddresse
```

Mit Hilfe dieser Variable wird die Motoradresse festgelegt, mit der kommuniziert werden soll.

GetStatusByte

Definition:

```
byte GetStatusByte()
```

Mit dieser Funktion kann das Statusbyte der Steuerung abgefragt werden.

Die Funktion entspricht dem seriellen Befehl `'$'`.

IsMotorReady

Definition:

```
bool IsMotorReady()
```

Diese Funktion liefert true zurück, wenn das Bit 0 im Statusbyte gesetzt ist (Steuerung ist bereit).

IsAtReferencePosition

Definition:

```
bool IsAtReferencePosition()
```

Diese Funktion liefert true zurück, wenn das Bit 1 im Statusbyte gesetzt ist (Nullposition erreicht).

HasPositionError

Definition:

```
bool HasPositionError()
```

Diese Funktion liefert true zurück, wenn das Bit 2 im Statusbyte gesetzt ist (Positionsfehler).

HasEndedTravelProfileAndStartInputStillActive

Definition:

```
bool HasEndedTravelProfileAndStartInputStillActive()
```

Diese Funktion liefert true zurück, wenn das Bit 3 im Statusbyte gesetzt ist (Eingang 1 ist gesetzt, während Steuerung wieder bereit ist).

IsPositionModeActive

Definition:

```
bool IsPositionModeActive()
```

Diese Funktion liefert true zurück, wenn der Positionsmodus aktiv ist.

IsSpeedModeActive

Definition:

```
bool IsSpeedModeActive()
```

Diese Funktion liefert true zurück, wenn der Drehzahlmodus aktiv ist.

IsFlagPositionModeActive

Definition:

```
bool IsFlagPositionModeActive()
```

Diese Funktion liefert true zurück, wenn der Flagpositionsmodus aktiv ist.

IsClockDirectionModeActive

Definition:

```
bool IsClockDirectionModeActive()
```

Diese Funktion liefert true zurück, wenn der Takt-Richtungsmodus aktiv ist.

IsJoyStickModeActive

Definition:

```
bool IsJoyStickModeActive()
```

Diese Funktion liefert true zurück, wenn der Joystickmodus aktiv ist.

IsAnalogModeActive

Definition:

```
bool IsAnalogModeActive()
```

Diese Funktion liefert true zurück, wenn der Analogmodus aktiv ist.

IsTorqueModeActive

Definition:

```
bool IsTorqueModeActive()
```

Diese Funktion liefert true zurück, wenn der Drehmomentmodus aktiv ist.

IsMasterModeActive

Definition:

```
bool IsMasterModeActive()
```

Diese Funktion liefert true zurück, wenn der Mastermodus („!10“) aktiv ist.

StartTravelProfile

Definition:

```
bool StartTravelProfile()
```

Mit dieser Funktion kann das Fahrprofil gestartet werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'A'`.

StopTravelProfile

Definition:

```
bool StopTravelProfile()
```

Mit dieser Funktion kann das Fahrprofil gestoppt werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'S1'`.

QuickStopTravelProfile

Definition:

```
bool  QuickStopTravelProfile()
```

Mit dieser Funktion kann das Fahrprofil schnell gestoppt werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'S'`.

IncreaseFrequency

Definition:

```
bool IncreaseFrequency()
```

Diese Funktion erhöht die Frequenz des Motors.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'+'`.

DecreaseFrequency

Definition:

```
bool DecreaseFrequency()
```

Diese Funktion erniedrigt die Frequenz des Motors.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'-'`.

TriggerOn

Definition:

```
bool TriggerOn()
```

Diese Funktion sendet den Trigger-Befehl an den Motor.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'T'`.

SetRamp

Definition:

```
bool SetRamp(int ramp)
```

Diese Funktion setzt die Beschleunigungsrampe.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'b'`.

SetBreak

Definition:

```
bool SetBreak(double breakTime)
```

Diese Funktion setzt die Pausenzeit in Millisekunden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'P'` .

ChooseRecord

Definition:

```
bool ChooseRecord(int recordNumber)
```

Diese Funktion lädt einen bestimmten Satz (Fahrprofil).

Der Parameter recordNumber ist dabei die Satznummer (Fahrprofil), die geladen werden soll.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'y'`.

GetRamp

Definition:

```
int GetRamp(int operationNumber)
```

Diese Funktion liest die Beschleunigungsrampe aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zb'`.

GetBreak

Definition:

```
int GetBreak(int operationNumber)
```

Diese Funktion liest die Pausenzeit in Millisekunden.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'ZP'`.

SetDirection

Definition:

```
bool SetDirection(int direction)
```

Diese Funktion setzt die Drehrichtung des Motors.

- direction = 0 entspricht links
- direction = 1 entspricht rechts

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'d'`.

SetMaxFrequency

Definition:

```
bool SetMaxFrequency(int maxFrequency)
```

Diese Funktion setzt die Zielfrequenz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'o'`.

GetMaxFrequency

Definition:

```
int GetMaxFrequency(int operationNumber)
```

Diese Funktion liest die Zielfrequenz.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zo'`.

SetRotationMode

Definition:

```
bool SetRotationMode(int rotationMode)
```

Diese Funktion setzt den Drehgeberüberwachungsmodus.

- rotationMode = 0 entspricht ausgeschaltet
- rotationMode = 1 entspricht Prüfen am Ende
- rotationMode = 2 entspricht Prüfen während der Fahrt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Einstellung „Prüfen während der Fahrt“ ist aus Kompatibilitätsgründen vorhanden und entspricht dem Verhalten „Prüfen am Ende“. Für eine tatsächliche Korrektur während der Fahrt sollte der Closed-Loop-Modus benutzt werden.

Die Funktion entspricht dem seriellen Befehl `'U'`.

ResetPositionError

Definition:

```
bool ResetPositionError(bool useEncoderValue, int position)
```

Mit dieser Funktion kann ein Positionsfehler zurückgesetzt und der Wert des Positionszählers gesetzt werden.

- useEncoderValue = true: setze Positionszähler auf Wert, den der Drehgeber anzeigt
- useEncoderValue = false: setze Positionszähler auf Wert der Variable Position

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'D'`.

ResetAllSettings

Definition:

```
bool ResetAllSettings()
```

Diese Funktion setzt alle Einstellungen der Steuerung auf die Defaultwerte (Werkseinstellungen) zurück.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'~'`.

GetVersion

Definition:

```
string GetVersion()
```

Diese Funktion gibt den Versionsstring der Steuerung zurück.

Die Funktion entspricht dem seriellen Befehl `'v'`.

SetSendStatusWhenCompleted

Definition:

```
bool SetSendStatusWhenCompleted(bool sendStatus)
```

Diese Funktion schaltet das selbständige Senden eines Status am Ende einer Fahrt.

- sendStatus = 0: automatisches Senden aus
- sendStatus = 1: automatisches Senden ein

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'J'`.

GetSendStatusWhenCompleted

Definition:

```
bool GetSendStatusWhenCompleted()
```

Diese Funktion liest, ob das selbständige Senden eines Status am Ende einer Fahrt eingeschaltet ist.

- sendStatus = 0: automatisches Senden aus
- sendStatus = 1: automatisches Senden ein

Die Funktion entspricht dem seriellen Befehl `'ZJ'`.

GetPosition

Definition:

```
int GetPosition()
```

Diese Funktion gibt den Wert des Positionszählers aus.

Die Funktion entspricht dem seriellen Befehl `'C'`.

GetIO

Definition:

```
int GetIO()
```

Diese Funktion gibt den Status der Eingänge als Integer-Wert zurück.

Die Funktion entspricht dem seriellen Befehl `'ZY'`.

SetIO

Definition:

```
bool SetIO(int io)
```

Diese Funktion setzt den Status der Ausgänge über einen Integer-Wert.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'Y'`

SetInputMaskEdge

Definition:

```
bool SetInputMaskEdge(int ioMask)
```

Diese Funktion setzt die Polarität der Ein- und Ausgänge.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Für eine genaue Beschreibung der Verwendung siehe den seriellen Befehl `'h'`.

GetInputMaskEdge

Definition:

```
int GetInputMaskEdge()
```

Diese Funktion gibt die aktuelle Polarität der Ein- und Ausgänge zurück.

Die Funktion entspricht dem seriellen Befehl `'Zh'`.

SetRecord

Definition:

```
bool SetRecord(int recordNumber)
```

Diese Funktion speichert die zuvor gesetzten Satzparameter in dem Satz mit der übergebenen Nummer.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'>'`.

SetPlay

Definition:

```
bool SetPlay(int play)
```

Diese Funktion setzt den Totbereich des Analogeingangs.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'='`.

GetPlay

Definition:

```
int GetPlay()
```

Diese Funktion gibt den Wert für den Totbereich des Analogeingangs zurück.

Die Funktion entspricht dem seriellen Befehl `'Z='`.

SetDebounceTime

Definition:

```
bool SetDebounceTime(int debounceTime)
```

Diese Funktion setzt die Entprellzeit für die Eingänge in Millisekunden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'K'`.

GetDebounceTime

Definition:

```
int GetDebounceTime()
```

Diese Funktion gibt die Entprellzeit für die Eingänge in Millisekunden zurück.

Die Funktion entspricht dem seriellen Befehl `'ZK'`.

SetSoftwareFilter

Definition:

```
bool SetSoftwareFilter(int softwareFilter)
```

Diese Funktion setzt den Wert für den Filter des Analogeingangs.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'f'`.

GetSoftwareFilter

Definition:

```
int GetSoftwareFilter()
```

Diese Funktion liest den Wert für den Filter des Analogeingangs aus.

Die Funktion entspricht dem seriellen Befehl `'Zf'`.

SetStepMode

Definition:

```
bool SetStepMode(int stepMode)
```

Diese Funktion setzt den Schrittmodus.

- stepMode = 1 entspricht Vollschritt
- stepMode = 2 entspricht Halbschritt
- stepMode = 4 entspricht Viertelschritt
- stepMode = 5 entspricht Füntelschritt
- stepMode = 8 entspricht Achtelschritt
- stepMode = 10 entspricht Zehntelschritt
- stepMode = 16 entspricht 16tel Schritt
- stepMode = 32 entspricht 32stel Schritt
- stepMode = 64 entspricht 64stel Schritt
- stepMode = 254 entspricht Vorschubkonstante
- stepMode = 255 entspricht Adaptiver Mikroschritt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'g'`.

GetStepMode

Definition:

```
int GetStepMode()
```

Diese Funktion liest den aktuellen Schrittmodus aus.

- Rückgabe = 1 entspricht Vollschritt
- Rückgabe = 2 entspricht Halbschritt
- Rückgabe = 4 entspricht Viertelschritt
- Rückgabe = 5 entspricht Füntelschritt
- Rückgabe = 8 entspricht Achtelschritt
- Rückgabe = 10 entspricht Zehntelschritt
- Rückgabe = 16 entspricht 16tel Schritt
- Rückgabe = 32 entspricht 32stel Schritt
- Rückgabe = 64 entspricht 64stel Schritt
- Rückgabe = 254 entspricht Vorschubkonstante
- Rückgabe = 255 entspricht Adaptiver Mikroschritt

Die Funktion entspricht dem seriellen Befehl `'Zg'`.

SetMotorAddress

Definition:

```
bool SetMotorAddress(int newMotorAddress)
```

Diese Funktion setzt die Motoradresse.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'m'`.

GetMotorAddress

Definition:

```
int GetMotorAddress(int selectedMotor)
```

Diese Funktion liest die Motoradresse aus. Der Wert des übergebenen Parameters selectedMotor ist egal, da der Befehl an alle Busteilnehmer gesendet wird.

Achtung:
Bei Verwendung dieses Befehls sollte nur eine Steuerung an den RS485-Bus angeschlossen sein.

GetErrorAddress

Definition:

```
int GetErrorAddress()
```

Diese Funktion liest die Fehleradresse aus, an der sich der letzte Fehlercode befindet.

Die Funktion entspricht dem seriellen Befehl `'E'`.

GetError

Definition:

```
int GetError(int errorAddress)
```

Diese Funktion liest den Fehler (Status) an der übergebenen Adresse.

Die Funktion entspricht dem seriellen Befehl `'ZE'`.

SetEnableAutoCorrect

Definition:

```
bool SetEnableAutoCorrect(string recordNumber, bool
autoCorrect)
```

Diese Funktion konfiguriert die automatische Fehlerkorrektur des Motors.

Der Wert von autoCorrect gibt an, ob eine Korrektur stattfinden soll.

Der Parameter recordNumber ist dabei die Satznummer (Fahrprofil), mit der ein eventueller Fehler korrigiert werden soll.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'F'`.

GetEnableAutoCorrect

Definition:

```
int GetEnableAutoCorrect(int errorAddress)
```

Diese Funktion liest aus, welcher Satz für die automatische Fehlerkorrektur gesetzt ist.

Die Funktion entspricht dem seriellen Befehl `'ZF'`.

SetSwingOutTime

Definition:

```
bool SetSwingOutTime(int swingOutTime)
```

Diese Funktion setzt die Ausschwingzeit.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'O'`.

GetSwingOutTime

Definition:

```
int GetSwingOutTime()
```

Diese Funktion liest die Ausschwingzeit aus.

Die Funktion entspricht dem seriellen Befehl `'ZO'`.

SetNextOperation

Definition:

```
bool SetNextOperation(int operationNumber)
```

Diese Funktion setzt den Folgesatz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'N'`.

GetNextOperation

Definition:

```
int GetNextOperation(int operationNumber)
```

Diese Funktion liest die Nummer des Folgesatzes.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'ZN'`.

SetPhaseCurrent

Definition:

```
bool SetPhaseCurrent(int phaseCurrent)
```

Diese Funktion setzt den Phasenstrom in Prozent. Werte über 100 sollten vermieden werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'i'`.

GetPhaseCurrent

Definition:

```
int GetPhaseCurrent()
```

Diese Funktion gibt den Phasenstrom in Prozent zurück.

Die Funktion entspricht dem seriellen Befehl `'Zi'`.

SetCurrentReduction

Definition:

```
bool SetCurrentReduction(int currentReduction)
```

Diese Funktion setzt den Phasenstrom bei Stillstand in Prozent. Werte über 100 sollten vermieden werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'r'`.

GetCurrentReduction

Definition:

```
int GetCurrentReduction()
```

Diese Funktion gibt den Phasenstrom bei Stillstand in Prozent zurück.

Die Funktion entspricht dem seriellen Befehl `'Zr'`.

SetLimitSwitchBehavior

Definition:

```
bool SetLimitSwitchBehavior(int refBehaviorsInternal, int
norBehaviorsInternal, int refBehaviorsExternal, int
norBehaviorsExternal)
```

Diese Funktion setzt das Endschalterverhalten.

Dabei bedeuten die einzelnen Parameter:

- refBehaviorsInternal = Verhalten des internen Endschalters bei Referenzfahrt
- norBehaviorsInternal = Verhalten des internen Endschalters bei Normalfahrt
- refBehaviorsExternal = Verhalten des externen Endschalters bei Referenzfahrt
- norBehaviorsExternal = Verhalten des externen Endschalters bei Normalfahrt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Für eine genaue Beschreibung der Verwendung siehe den seriellen Befehl `'l'`.

GetLimitSwitchBehavior

Definition:

```
bool GetLimitSwitchBehavior(out int refBehaviorsInternal,
out int norBehaviorsInternal, out int
refBehaviorsExternal, out int norBehaviorsExternal)
```

Diese Funktion liest das Endschalterverhalten aus.

Dabei bedeuten die einzelnen Rückgabeparameter:

- refBehaviorsInternal = Verhalten des internen Endschalters bei Referenzfahrt
- norBehaviorsInternal = Verhalten des internen Endschalters bei Normalfahrt
- refBehaviorsExternal = Verhalten des externen Endschalters bei Referenzfahrt
- norBehaviorsExternal = Verhalten des externen Endschalters bei Normalfahrt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Für eine genaue Beschreibung der Verwendung siehe den seriellen Befehl `'l'`.

SetReverseClearance

Definition:

```
bool SetReverseClearance(int reverseClearance)
```

Diese Funktion setzt das Umkehrspiel in Schritten.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'z'`.

GetReverseClearance

Definition:

```
int GetReverseClearance()
```

Diese Funktion gibt das Umkehrspiel in Schritten aus.

Die Funktion entspricht dem seriellen Befehl `'Zz'`.

SetAnalogueMin

Definition:

```
bool SetAnalogueMin(double analogueMin)
```

Diese Funktion setzt die minimale Spannung für den Analogeingang.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'Q'`.

GetAnalogueMin

Definition:

```
double GetAnalogueMin()
```

Diese Funktion gibt die minimale Spannung für den Analogeingang aus.

Die Funktion entspricht dem seriellen Befehl `'ZQ'`.

SetAngelDeviationMax

Definition:

```
bool SetAngelDeviationMax(int deviation)
```

Diese Funktion setzt die maximale Winkelabweichung zwischen Sollposition und Drehgeberwert.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'X'`.

GetAngelDeviationMax

Definition:

```
int GetAngelDeviationMax()
```

Diese Funktion gibt die maximale Winkelabweichung zwischen Sollposition und Drehgeberwert aus.

Die Funktion entspricht dem seriellen Befehl `'ZX'`.

SetAnalogueMax

Definition:

```
bool SetAnalogueMax(double analogueMax)
```

Diese Funktion setzt die maximale Spannung für den Analogeingang.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'R'`.

GetAnalogueMax

Definition:

```
double GetAnalogueMax()
```

Diese Funktion gibt die maximale Spannung für den Analogeingang aus.

Die Funktion entspricht dem seriellen Befehl `'ZR'`.

SetPositionType

Definition:

```
bool SetPositionType(int positionType)
```

Diese Funktion setzt die Positionierart.

- positionType = 1 entspricht relativ; abhängig vom Operationsmodus
- positionType = 2 entspricht absolut; abhängig vom Operationsmodus
- positionType = 3 entspricht interner Referenzfahrt;
- positionType = 4 entspricht externer Referenzfahrt;

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Für eine genaue Beschreibung der Verwendung siehe den seriellen Befehl `'p'`.

GetPositionType

Definition:

```
int GetPositionType(int operationNumber)
```

Diese Funktion liest die Positionierart aus.

- 1 entspricht relativ; abhängig vom Operationsmodus
- 2 entspricht absolut; abhängig vom Operationsmodus
- 3 entspricht interner Referenzfahrt;
- 4 entspricht externer Referenzfahrt

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der der Positionstyp gelesen werden soll.

Für eine genaue Beschreibung der Verwendung siehe den seriellen Befehl `'p'`.

SetSteps

Definition:

```
bool SetSteps(int steps)
```

Diese Funktion setzt die Anzahl der Schritte.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'s'`.

GetSteps

Definition:

```
int GetSteps(int operationNumber)
```

Diese Funktion liest die Anzahl der Schritte aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zs'`.

SetStartFrequency

Definition:

```
bool SetStartFrequency(int startFrequency)
```

Diese Funktion setzt die Startfrequenz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'u'`.

GetStartFrequency

Definition:

```
int GetStartFrequency(int operationNumber)
```

Diese Funktion gibt die Startfrequenz aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zu'`.

SetMaxFrequency2

Definition:

```
bool SetMaxFrequency2(int maxFrequency)
```

Diese Funktion setzt die obere Maximalfrequenz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'n'`.

GetMaxFrequency2

Definition:

```
int GetMaxFrequency2(int operationNumber)
```

Diese Funktion gibt die obere Maximalfrequenz aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zn'`.

SetSuppressResponse

Definition:

```
bool SetSuppressResponse(int suppress)
```

Diese Funktion aktiviert oder deaktiviert die Antwortunterdrückung beim Senden.

- suppress = 0: Antwortunterdrückung ein
- suppress = 1: Antwortunterdrückung aus

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'|'`.

GetRotationMode

Definition:

```
int GetRotationMode()
```

Diese Funktion liest den Drehgeberüberwachungsmodus.

- 0 bedeutet keine Überwachung
- 1 bedeutet eine Prüfung am Ende
- 2 bedeutet eine Prüfung während der Fahrt

Die Einstellung „Prüfen während der Fahrt“ ist aus Kompatibilitätsgründen vorhanden und entspricht dem Verhalten „Prüfen am Ende“. Für eine tatsächliche Korrektur während der Fahrt sollte der Closed-Loop-Modus benutzt werden.

Die Funktion entspricht dem seriellen Befehl `'ZU'`.

GetDirection

Definition:

```
int GetDirection(int operationNumber)
```

Diese Funktion liest die Drehrichtung des Motors aus.

- 0 entspricht links
- 1 entspricht rechts

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zd'`.

SetDirectionReverse

Definition:

```
bool SetDirectionReverse(bool directionReverse)
```

Diese Funktion setzt die Drehrichtungsumkehr.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'t'`.

GetDirectionReverse

Definition:

```
bool GetDirectionReverse(int operationNumber)
```

Diese Funktion liest die Drehrichtungsumkehr aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Zt'`.

SetEncoderDirection

Definition:

```
bool SetEncoderDirection(bool encoderDirection)
```

Diese Funktion setzt die Encoderdrehrichtung. Ist der Parameter encoderDirection true, so wird die Richtung des Drehencoders umgekehrt.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'q'`.

GetEncoderDirection

Definition:

```
bool GetEncoderDirection()
```

Diese Funktion gibt aus, ob die Encoderdrehrichtung umgekehrt wird.

Die Funktion entspricht dem seriellen Befehl `'Zq'`.

GetEncoderRotary

Definition:

```
int GetEncoderRotary()
```

Diese Funktion liest die Encoderposition aus.

Die Funktion entspricht dem seriellen Befehl `'I'`.

SetRampType

Definition:

```
bool SetRampType(int rampType)
```

Diese Funktion setzt den Rampentyp.

- rampType = 0: Trapezrampe
- rampType = 1: Sinusrampe
- rampType = 2: Jerkfreerampe

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':ramp_mode'`.

GetRampType

Definition:

```
int GetRampType()
```

Diese Funktion gibt den Rampentyp aus.

- rampType = 0: Trapezrampe
- rampType = 1: Sinusrampe
- rampType = 2: Jerkfreerampe

Die Funktion entspricht dem seriellen Befehl `':ramp_mode'`.

SetJerk

Definition:

```
bool SetJerk(int jerk)
```

Diese Funktion setzt den Ruck in 100/s³.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':b'`.

GetJerk

Definition:

```
int GetJerk(int operationNumber)
```

Diese Funktion gibt den Ruck in 100/s³ aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Z:b'`.

SetBrakeRamp

Definition:

```
bool SetBrakeRamp(int rampBrake)
```

Diese Funktion setzt die Bremsrampe.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'B'`.

GetBrakeRamp

Definition:

```
int GetBrakeRamp(int operationNumber)
```

Diese Funktion liest die Bremsrampe aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'ZB'`.

SetBrakeJerk

Definition:

```
bool SetBrakeJerk(int jerk)
```

Diese Funktion setzt den Bremsruck in 100/s³.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':B'`.

GetBrakeJerk

Definition:

```
int GetBrakeJerk(int operationNumber)
```

Diese Funktion gibt den Bremsruck in 100/s³ aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'Z:B'`.

SetQuickStoppRamp

Definition:

```
bool SetQuickStoppRamp(int rampQuickStopp)
```

Diese Funktion setzt die Quickstoprampe.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'H'`.

GetQuickStoppRamp

Definition:

```
int GetQuickStoppRamp(int operationNumber)
```

Diese Funktion liest die Quickstoprampe aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'ZH'`.

SetRepeat

Definition:

```
bool SetRepeat(int repeats)
```

Diese Funktion setzt die Anzahl der Wiederholungen.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `'W'`.

GetRepeat

Definition:

```
int GetRepeat(int operationNumber)
```

Diese Funktion liest die Anzahl der Wiederholungen aus.

Der Parameter operationNumber ist dabei die Satznummer (Fahrprofil), aus der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `'ZW'`.

SetModus8

Definition:

```
bool SetModus8()
```

Diese Funktion setzt den Operationsmodus 14, welcher einer internen Referenzfahrt entspricht. Bei älteren Firmwares war eine Fahrt in diesem Operationsmodus nötig, um den Closed-Loop-Modus zu aktivieren.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

SetKalibrierModus

Definition:

```
bool SetKalibrierModus()
```

Diese Funktion setzt den Operationsmodus 17, welcher den Kalibrierlauf des CL-Assistenten durchführt.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

SetClosedLoop

Definition:

```
bool SetClosedLoop(int value)
```

Diese Funktion aktiviert oder deaktiviert den Closed Loop Modus.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_enable'`.

GetClosedLoop

Definition:

```
int GetClosedLoop()
```

Diese Funktion gibt aus, ob der Closed Loop Modus aktiviert ist.

Die Funktion entspricht dem seriellen Befehl `':CL_enable'`.

GetCLLoadAngle

Definition:

```
int GetCLLoadAngle(int tripelNumber)
```

Diese Funktion liest einen Lastwinkel des Motors aus dem Closed-Loop-Testlauf aus.

Der Parameter tripelNumber ist dabei die Nummer (0-9) des Werts, der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `':CL_la_a'` bis `':CL_la_j'`.

GetClosedLoopOlaCurrent

Definition:

```
int GetClosedLoopOlaCurrent(int tripelNumber)
```

Diese Funktion liest einen Korrekturwert des Stromreglers aus dem Closed-Loop-Testlauf aus.

Der Parameter tripelNumber ist dabei die Nummer (0-6) des Werts, der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `':CL_ola_i_a'` bis `':CL_ola_i_g'`.

GetClosedLoopOlaVelocity

Definition:

```
int GetClosedLoopOlaVelocity(int tripelNumber)
```

Diese Funktion liest einen Korrekturwert des Geschwindigkeitsreglers aus dem Closed-Loop-Testlauf aus.

Der Parameter tripelNumber ist dabei die Nummer (0-6) des Werts, der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `':CL_ola_v_a'` bis `':CL_ola_v_g'`.

GetClosedLoopOlaLoadAngle

Definition:

```
int GetClosedLoopOlaLoadAngle(int tripelNumber)
```

Diese Funktion liest einen Korrekturwert des Positionsreglers aus dem Closed-Loop-Testlauf aus.

Der Parameter tripelNumber ist dabei die Nummer (0-6) des Werts, der gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl `':CL_ola_l_a'` bis `':CL_ola_l_g'`.

SetPositionWindow

Definition:

```
bool SetPositionWindow(int positionWindow)
```

Diese Funktion setzt das Toleranzfenster für die Endposition im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window'`.

GetPositionWindow

Definition:

```
int GetPositionWindow()
```

Diese Funktion gibt den Wert für das Toleranzfenster für die Endposition im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window'`.

SetPositionWindowTime

Definition:

```
bool SetPositionWindowTime(int time)
```

Diese Funktion setzt die Zeit für das Toleranzfenster der Endposition im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window_time'`.

GetPositionWindowTime

Definition:

```
int GetPositionWindowTime()
```

Diese Funktion gibt den Wert für die Zeit für das Toleranzfenster der Endposition im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_position_window_time'`.

SetFollowingErrorWindow

Definition:

```
bool SetFollowingErrorWindow(int followingErrorWindow)
```

Diese Funktion setzt den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_window'`.

GetFollowingErrorWindow

Definition:

```
int GetFollowingErrorWindow()
```

Diese Funktion gibt den Wert für den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_window'`.

SetSpeedErrorWindow

Definition:

```
bool SetSpeedErrorWindow(int speedErrorWindow)
```

Diese Funktion setzt die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_window'`.

GetSpeedErrorWindow

Definition:

```
int GetSpeedErrorWindow()
```

Diese Funktion gibt den Wert für die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_window'`.

SetFollowingErrorTimeout

Definition:

```
bool SetFollowingErrorTimeout(int timeout)
```

Diese Funktion setzt die Zeit für den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_timeout'`.

GetFollowingErrorTimeout

Definition:

```
int GetFollowingErrorTimeout()
```

Diese Funktion gibt den Wert für die Zeit für den maximal erlaubten Schleppfehler im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_following_error_timeout'`.

SetSpeedErrorTimeout

Definition:

```
bool SetSpeedErrorTimeout(int timeout)
```

Diese Funktion setzt die Zeit für die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_timeout'`.

GetSpeedErrorTimeout

Definition:

```
int GetSpeedErrorTimeout()
```

Diese Funktion gibt den Wert für die Zeit für die maximal erlaubte Drehzahlabweichung im Closed-Loop-Betrieb aus.

Die Funktion entspricht dem seriellen Befehl `':CL_speed_error_timeout'`.

SetRotencInc

Definition:

```
bool SetRotencInc(int rotencInc)
```

Diese Funktion setzt die Anzahl der Inkremente des Drehgebers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_rotenc_inc'`.

GetRotencInc

Definition:

```
int GetRotencInc()
```

Diese Funktion gibt die Anzahl der Inkremente des Drehgebers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_rotenc_inc'`.

SetBrakeTA

Definition:

```
bool SetBrakeTA(UInt32 brake)
```

Diese Funktion setzt die Wartezeit für das Abschalten der Bremsspannung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':brake_ta'`.

GetBrakeTA

Definition:

```
int GetBrakeTA()
```

Diese Funktion gibt die Wartezeit für das Abschalten der Bremsspannung aus.

Die Funktion entspricht dem seriellen Befehl `':brake_ta'`.

SetBrakeTB

Definition:

```
bool SetBrakeTB(UInt32 brake)
```

Diese Funktion setzt die Zeit zwischen dem Abschalten der Bremsspannung und dem Erlauben einer Motorbewegung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':brake_tb'`.

GetBrakeTB

Definition:

```
int GetBrakeTB()
```

Diese Funktion gibt die Zeit zwischen dem Abschalten der Bremsspannung und dem Erlauben einer Motorbewegung aus.

Die Funktion entspricht dem seriellen Befehl `':brake_tb'`.

SetBrakeTC

Definition:

```
bool SetBrakeTC(UInt32 brake)
```

Diese Funktion setzt die Wartezeit für das Abschalten des Motorstroms.

Der Motorstrom wird durch Rücksetzen des Freigabe-Eingangs abgeschaltet (siehe Abschnitt *2.5.25 „Funktion der Digitaleingänge einstellen“*).

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':brake_tc'`.

GetBrakeTC

Definition:

```
int GetBrakeTC()
```

Diese Funktion gibt die Wartezeit für das Abschalten des Motorstroms aus.

Der Motorstrom wird durch Rücksetzen des Freigabe-Eingangs abgeschaltet (siehe Abschnitt *2.5.25 „Funktion der Digitaleingänge einstellen“*).

Die Funktion entspricht dem seriellen Befehl `':brake_tc'`.

SetKPsZ

Definition:

```
bool SetKPsZ(int value)
```

Diese Funktion setzt den Zähler des P-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_s_Z'`.

GetKPsZ

Definition:

```
int GetKPsZ()
```

Diese Funktion gibt den Zähler des P-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_s_Z'`.

SetKPsN

Definition:

```
bool SetKPsN(int value)
```

Diese Funktion setzt den Nenner des P-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_s_N'`.

GetKPsN

Definition:

```
int GetKPsN()
```

Diese Funktion gibt den Nenner des P-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_s_N'`.

SetKIsZ

Definition:

```
bool SetKIsZ(int value)
```

Diese Funktion setzt den Zähler des I-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_Z'`.

GetKIsZ

Definition:

```
int GetKIsZ()
```

Diese Funktion gibt den Zähler des I-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_Z'`.

SetKIsN

Definition:

```
bool SetKIsN(int value)
```

Diese Funktion setzt den Nenner des I-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_N'`.

GetKIsN

Definition:

```
int GetKIsN()
```

Diese Funktion gibt den Nenner des I-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_s_N'`.

SetKDsZ

Definition:

```
bool SetKDsZ(int value)
```

Diese Funktion setzt den Zähler des D-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_Z'`.

GetKDsZ

Definition:

```
int GetKDsZ()
```

Diese Funktion gibt den Zähler des D-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_Z'`.

SetKDsN

Definition:

```
bool SetKDsN(int value)
```

Diese Funktion setzt den Nenner des D-Anteils des Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_N'`.

GetKDsN

Definition:

```
int GetKDsN()
```

Diese Funktion gibt den Nenner des D-Anteils des Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_s_N'`.

SetKPvZ

Definition:

```
bool SetKPvZ(int value)
```

Diese Funktion setzt den Zähler des P-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_Z'`.

GetKPvZ

Definition:

```
int GetKPvZ()
```

Diese Funktion gibt den Zähler des P-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_Z'`.

SetKPvN

Definition:

```
bool SetKPvN(int value)
```

Diese Funktion setzt den Nenner des P-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_N'`.

GetKPvN

Definition:

```
int GetKPvN()
```

Diese Funktion gibt den Nenner des P-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_v_N'`.

SetKIvZ

Definition:

```
bool SetKIvZ(int value)
```

Diese Funktion setzt den Zähler des I-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_Z'`.

GetKIvZ

Definition:

```
int GetKIvZ()
```

Diese Funktion gibt den Zähler des I-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_Z'`.

SetKIvN

Definition:

```
bool SetKIvN(int value)
```

Diese Funktion setzt den Nenner des I-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_N'`.

GetKIvN

Definition:

```
int GetKIvN()
```

Diese Funktion gibt den Nenner des I-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_v_N'`.

SetKDvZ

Definition:

```
bool SetKDvZ(int value)
```

Diese Funktion setzt den Zähler des D-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_Z'`.

GetKDvZ

Definition:

```
int GetKDvZ()
```

Diese Funktion gibt den Zähler des D-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_Z'`.

SetKDvN

Definition:

```
bool SetKDvN(int value)
```

Diese Funktion setzt den Nenner des D-Anteils des Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_N'`.

GetKDvN

Definition:

```
int GetKDvN()
```

Diese Funktion gibt den Nenner des D-Anteils des Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_v_N'`.

SetKPcssZ

Definition:

```
bool SetKPcssZ(int value)
```

Diese Funktion setzt den Zähler des P-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_Z'`.

GetKPcssZ

Definition:

```
int GetKPcssZ()
```

Diese Funktion gibt den Zähler des P-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_Z'`.

SetKPcssN

Definition:

```
bool SetKPcssN(int value)
```

Diese Funktion setzt den Nenner des P-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_N'`.

GetKPcssN

Definition:

```
int GetKPcssN()
```

Diese Funktion gibt den Nenner des P-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_css_N'`.

SetKIcssZ

Definition:

```
bool SetKIcssZ(int value)
```

Diese Funktion setzt den Zähler des I-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_Z'`.

GetKIcssZ

Definition:

```
int GetKIcssZ()
```

Diese Funktion gibt den Zähler des I-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_Z'`.

SetKIcssN

Definition:

```
bool SetKIcssN(int value)
```

Diese Funktion setzt den Nenner des I-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_N'`.

GetKIcssN

Definition:

```
int GetKIcssN()
```

Diese Funktion gibt den Nenner des I-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_css_N'`.

SetKDcssZ

Definition:

```
bool SetKDcssZ(int value)
```

Diese Funktion setzt den Zähler des D-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_Z'`.

GetKDcssZ

Definition:

```
int GetKDcssZ()
```

Diese Funktion gibt den Zähler des D-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_Z'`.

SetKDcssN

Definition:

```
bool SetKDcssN(int value)
```

Diese Funktion setzt den Nenner des D-Anteils des kaskadierenden Positionsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_N'`.

GetKDcssN

Definition:

```
int GetKDcssN()
```

Diese Funktion gibt den Nenner des D-Anteils des kaskadierenden Positionsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_css_N'`.

SetKPcsvZ

Definition:

```
bool SetKPcsvZ(int value)
```

Diese Funktion setzt den Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_Z'`.

GetKPcsvZ

Definition:

```
int GetKPcsvZ()
```

Diese Funktion gibt den Zähler des P-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_Z'`.

SetKPcsvN

Definition:

```
bool SetKPcsvN(int value)
```

Diese Funktion setzt den Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_N'`.

GetKPcsvN

Definition:

```
int GetKPcsvN()
```

Diese Funktion gibt den Nenner des P-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KP_csv_N'`.

SetKIcsvZ

Definition:

```
bool SetKIcsvZ(int value)
```

Diese Funktion setzt den Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_Z'`.

GetKIcsvZ

Definition:

```
int GetKIcsvZ()
```

Diese Funktion gibt den Zähler des I-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_Z'`.

SetKIcsvN

Definition:

```
bool SetKIcsvN(int value)
```

Diese Funktion setzt den Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_N'`.

GetKIcsvN

Definition:

```
int GetKIcsvN()
```

Diese Funktion gibt den Nenner des I-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KI_csv_N'`.

SetKDcsvZ

Definition:

```
bool SetKDcsvZ(int value)
```

Diese Funktion setzt den Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_Z'`.

GetKDcsvZ

Definition:

```
int GetKDcsvZ()
```

Diese Funktion gibt den Zähler des D-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_Z'`.

SetKDcsvN

Definition:

```
bool SetKDcsvN(int value)
```

Diese Funktion setzt den Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_N'`.

GetKDcsvN

Definition:

```
int GetKDcsvN()
```

Diese Funktion gibt den Nenner des D-Anteils des kaskadierenden Geschwindigkeitsreglers aus.

Die Funktion entspricht dem seriellen Befehl `':CL_KD_csv_N'`.

SetInput1Selection

Definition:

```
bool SetInput1Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 1.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_a'`.

GetInput1Selection

Definition:

```
InputSelection GetInput1Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 1 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_a'`.

SetInput2Selection

Definition:

```
bool SetInput2Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 2.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_b'`.

GetInput2Selection

Definition:

```
InputSelection GetInput2Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 2 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_b'`.

SetInput3Selection

Definition:

```
bool SetInput3Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 3.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_c'`.

GetInput3Selection

Definition:

```
InputSelection GetInput3Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 3 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_c'`.

SetInput4Selection

Definition:

```
bool SetInput4Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 4.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_d'`.

GetInput4Selection

Definition:

```
InputSelection GetInput4Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 4 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_d'`.

SetInput5Selection

Definition:

```
bool SetInput5Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 5.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_e'`.

GetInput5Selection

Definition:

```
InputSelection GetInput5Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 5 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_e'`.

SetInput6Selection

Definition:

```
bool SetInput6Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 6.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_f'`.

GetInput6Selection

Definition:

```
InputSelection GetInput6Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 6 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_f'`.

SetInput7Selection

Definition:

```
bool SetInput7Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 7.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_g'`.

GetInput7Selection

Definition:

```
InputSelection GetInput7Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 7 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_g'`.

SetInput8Selection

Definition:

```
bool SetInput8Selection(InputSelection inputSelection)
```

Diese Funktion setzt die Funktion für den Digitaleingang 8.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_in_h'`.

GetInput8Selection

Definition:

```
InputSelection GetInput8Selection()
```

Diese Funktion gibt die Funktion für den Digitaleingang 8 aus.

Die Funktion entspricht dem seriellen Befehl `':port_in_h'`.

SetOutput1Selection

Definition:

```
bool SetOutput1Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 1.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_a'`.

GetOutput1Selection

Definition:

```
OutputSelection GetOutput1Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 1 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_a'`.

SetOutput2Selection

Definition:

```
bool SetOutput2Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 2.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_b'`.

GetOutput2Selection

Definition:

```
OutputSelection GetOutput2Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 2 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_b'`.

SetOutput3Selection

Definition:

```
bool SetOutput3Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 3.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_c'`.

GetOutput3Selection

Definition:

```
OutputSelection GetOutput3Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 3 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_c'`.

SetOutput4Selection

Definition:

```
bool SetOutput4Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 4.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_d'`.

GetOutput4Selection

Definition:

```
OutputSelection GetOutput4Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 4 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_d'`.

SetOutput5Selection

Definition:

```
bool SetOutput5Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 5.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_e'`.

GetOutput5Selection

Definition:

```
OutputSelection GetOutput5Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 5 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_e'`.

SetOutput6Selection

Definition:

```
bool SetOutput6Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 6.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_f'`.

GetOutput6Selection

Definition:

```
OutputSelection GetOutput6Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 6 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_f'`.

SetOutput7Selection

Definition:

```
bool SetOutput7Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 7.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_g'`.

GetOutput7Selection

Definition:

```
OutputSelection GetOutput7Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 7 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_g'`.

SetOutput8Selection

Definition:

```
bool SetOutput8Selection(OutputSelection outputSelection)
```

Diese Funktion setzt die Funktion für den Digitalausgang 8.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':port_out_h'`.

GetOutput8Selection

Definition:

```
OutputSelection GetOutput8Selection()
```

Diese Funktion gibt die Funktion für den Digitalausgang 8 aus.

Die Funktion entspricht dem seriellen Befehl `':port_out_h'`.

SetFeedConstNum

Definition:

```
bool SetFeedConstNum(int feedConstNum)
```

Diese Funktion setzt den Zähler der Vorschubkonstanten.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':feed_const_num'`.

GetFeedConstNum

Definition:

```
int GetFeedConstNum()
```

Diese Funktion gibt den Zähler der Vorschubkonstanten aus.

Die Funktion entspricht dem seriellen Befehl `':feed_const_num'`.

SetFeedConstDenum

Definition:

```
bool SetFeedConstDenum(int feedConstDenum)
```

Diese Funktion setzt den Nenner der Vorschubkonstanten.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':feed_const_denum'`.

GetFeedConstDenum

Definition:

```
int GetFeedConstDenum()
```

Diese Funktion gibt den Nenner der Vorschubkonstanten aus.

Die Funktion entspricht dem seriellen Befehl `':feed_const_denum'`.

SetCurrentPeak

Definition:

```
bool SetCurrentPeak(int currentPeak)
```

Diese Funktion setzt den Strom-Spitzenwert für BLDC.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':ipeak'`.

GetCurrentPeak

Definition:

```
int GetCurrentPeak()
```

Diese Funktion gibt den Strom-Spitzenwert für BLDC aus.

Die Funktion entspricht dem seriellen Befehl `':ipeak'`.

SetCurrentTime

Definition:

```
bool SetCurrentTime(int currentTime)
```

Diese Funktion setzt die Strom-Zeitkonstante für BLDC.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl `':itime'`.

GetCurrentTime

Definition:

```
int GetCurrentTime()
```

Diese Funktion gibt die Strom-Zeitkonstante für BLDC aus.

Die Funktion entspricht dem seriellen Befehl `':itime'`.

GetAnalogAmplitude

Definition:

```
int GetAnalogAmplitude()
```

Liest die Amplitude für den Analogeingang aus.

Die Funktion entspricht dem seriellen Befehl `'Z:aaa'`.

SetAnalogAmplitude

Definition:

```
bool SetAnalogAmplitude(int analogAmplitude)
```

Setzt die Amplitude für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `':aaa'`.

GetAnalogOffset

Definition:

```
int GetAnalogOffset()
```

Liest den Offset für den Analogeingang aus.

Die Funktion entspricht dem seriellen Befehl `'Z:aoa'`.

SetAnalogOffset

Definition:

```
bool SetAnalogOffset(int analogOffset)
```

Setzt den Offset für den Analogeingang.

Die Funktion entspricht dem seriellen Befehl `':aoa'`.

GetCascIsEnabled

Definition:

```
bool GetCascIsEnabled()
```

Gibt zurück, ob der Kaskadenregler aktiv ist `'Z:ce'`.

GetCascStart

Definition:

```
int GetCascStart()
```

Liest die Startfrequenz für den Kaskadenregler aus.

Die Funktion entspricht dem seriellen Befehl `'Z:ca'`.

SetCascStart

Definition:

```
bool SetCascStart(int frequency)
```

Setzt die Startfrequenz für den Kaskadenregler.

Die Funktion entspricht dem seriellen Befehl `':ca'`.

GetCascStop

Definition:

```
int GetCascStop()
```

Liest die Endfrequenz für den Kaskadenregler aus.

Die Funktion entspricht dem seriellen Befehl `'Z:cs'`.

SetCascStop

Definition:

```
bool SetCascStop(int frequency)
```

Setzt die Endfrequenz für den Kaskadenregler.

Die Funktion entspricht dem seriellen Befehl `':cs'`.

GetCLNodeDistance

Definition:

```
int GetCLNodeDistance()
```

Liest den CL-Stützstellenabstand aus.

Die Funktion entspricht dem seriellen Befehl `'Z:CL_la_node_distance'`.

SetCLNodeDistance

Definition:

```
bool SetCLNodeDistance(int nodeDistance)
```

Setzt den CL-Stützstellenabstand.

Die Funktion entspricht dem seriellen Befehl `':CL_la_node_distance'`.

GetClockInterpolated

Definition:

```
int GetClockInterpolated()
```

Liest den Clockmode Steilheits-Faktor aus.

Die Funktion entspricht dem seriellen Befehl `'Z:clock_interp '`.

SetClockInterpolated

Definition:

```
bool SetClockInterpolated(int gradient)
```

Setzt den Clockmode Steilheits-Faktor.

Die Funktion entspricht dem seriellen Befehl `':clock_interp '`.

GetCLPosCNTOffset

Definition:

```
int GetCLPosCNTOffset()
```

Liest den Drehgeberindex-Versatz aus.

Die Funktion entspricht dem seriellen Befehl `'Z:CL_poscnt_offset '`.

SetCLPosCNTOffset

Definition:

```
bool SetCLPosCNTOffset(int posCNTOffset)
```

Setzt den Drehgeberindex-Versatz.

Die Funktion entspricht dem seriellen Befehl `':CL_poscnt_offset '`.

SendCommandString

Definition:

```
bool SendCommandString(String commandString)
```

Sendet den übergebenen String an die Steuerung.

4.4 Programmbeispiele

Einleitung

Einige Beispiele für die Benutzung der CommandsPD4I Funktionsbibliothek sind im NanoPro-Installationsverzeichnis im Unterverzeichnis SDK\example zu finden. Alle Beispiele sind als Projekte für Microsoft Visual Studio realisiert. Alle Beispiele demonstrieren die Interaktion mit 2 Steuerungen an unterschiedlichen seriellen Schnittstellen. Es folgt eine kurze Auflistung der vorhandenen Beispiele.

CsharpExample

Dieses Beispiel ist in der Programmiersprache C# implementiert und als Visual Studio 2005 Projekt realisiert.

ManagedC++Example:

Dieses Beispiel ist in der Programmiersprache C++ unter Verwendung von Managed Code implementiert und als Visual Studio 2008 Projekt realisiert.

UnmanagedC++Example:

Dieses Beispiel ist in der Programmiersprache C++ unter Verwendung von Unmanaged Code implementiert und als Visual Studio 2008 Projekt realisiert. Im Unterschied zu den anderen Beispielen beinhaltet dieses Beispiel keine grafische Benutzeroberfläche.

VBExample:

Dieses Beispiel ist in der Programmiersprache Visual Basic implementiert und als Visual Studio 2005 Projekt realisiert.

5 Anhang: Berechnung der CRC-Prüfsumme

Zweck der CRC-Prüfsumme

Die CRC-Prüfsumme wird von Nanotec Schrittmotorsteuerungen, Plug & Drive Motoren und der Software NanoPro berechnet, um Übertragungsfehler auf dem RS485-Bus zu erkennen.

Funktion zur Berechnung der CRC-Prüfsumme

Die rs485_com.dll verwendet folgende C-Funktion zur Berechnung der CRC-Prüfsumme:

```
unsigned char mcrc8( unsigned char crc, char* str, int len ) {
    const unsigned char pol = 0x07;

    unsigned char c;
    unsigned char i;

    while(len--) {
        c = *str;
        for( i=0 ; i<8 ; i++ ) {
            if ( (crc & 0x80) != (c & 0x80) ) {
                crc = (crc << 1) ^ pol;
            } else {
                crc <<= 1;
            }
            c <<= 1;
        }
        str++;
    }

    return crc;
}
```

Argumente für die Funktion

- unsigned char crc: Startwert für Prüfsumme. Es wird immer 0 verwendet.
- char* str: Zeiger auf erstes Zeichen des zu sendenden char-Arrays.
- int len: Länge des zu sendenden Strings ohne Carriage return.

Anwendungsbeispiel

Senden

Zu sendender String	„#1v\r“
Argumente für den Aufruf der Funktion mcrc8	• unsigned char crc: 0 • char* str: Entsprechender Zeiger auf erstes Zeichen des zu sendenden char-Arrays. • int len: Länge des zu sendenden Strings ohne Carriage return = 3.
Rückgabewert der Funktion	87 (dezimal) = 0x57 (hexadezimal)
Tatsächlich zu sendender String (Die CRC-Prüfsumme wird durch ein Tabulator-Zeichen \t vom eigentlichen Befehlsstring getrennt und mitgesendet, die Steuerung berechnet dann zur Kontrolle ebenfalls die Prüfsumme des empfangenen Strings.)	„#1v\t57\r“

Empfangen

Antwort der Steuerung	„1v SMCI47-S_RS485_29-09-2010\t75\r“
Argumente für den Aufruf der Funktion mcrc8	• unsigned char crc: 0 • char* str: Entsprechender Zeiger auf erstes Zeichen des empfangenen char-Arrays. • int len: Länge des empfangenen Strings bis vor dem Tabulator-Zeiten = 28.
Rückgabewert der Funktion	117 (dezimal) = 0x75 (hexadezimal)

Wenn die berechnete Prüfsumme mit der empfangenen Prüfsumme übereinstimmt, war die Übertragung fehlerfrei.

6 Anhang: Motordaten

6.1 Default Werte für Schrittmotoren

Lastwinkel	Wert
1	16384
2	18384
3	20384
4	22384
5	24384
6	26384
7	28384

6.2 Default Werte für BLDC Motoren

Lastwinkel	Wert
1	16384
2	16500
3	17000
4	17500
5	18000
6	18500
7	19000

6.3 Schrittmotoren der Serie STxxxx

Die folgende Tabelle gilt für Schrittmotoren der Serie ST2018, ST3518, ST4118, ST4209, ST4218, ST5709, ST5909, ST5918, ST6018, ST6318, ST8918, ST11018.

Lastwinkel	Wert
1	16384
2	16500
3	17000
4	17500
5	18000
6	18500
7	19000

6.4 BLDC Motoren der Serie DB22

DB22L01

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17500
5	18000
6	18500
7	19000

DB22M01

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17500
5	18000
6	18500
7	18500

6.5 BLDC Motoren der Serie DB28

DB28M01

Lastwinkel	Wert
1	16000
2	17000
3	17000
4	17000
5	18000
6	18000
7	18000

DB28S01

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17500
5	18000
6	18500
7	18500

6.6 BLDC Motoren der Serie DB33

DB33S01

Lastwinkel	Wert
1	16000
2	16000
3	16500
4	16500
5	17000
6	17000
7	17000

6.7 BLDC Motoren der Serie DB42

DB42C01

Lastwinkel	Wert
1	16000
2	18000
3	20000
4	20000
5	20000
6	21000
7	20000

DB42C02

Lastwinkel	Wert
1	16000
2	18000
3	20000
4	20000
5	20000
6	21000
7	22000

DB42C03

Lastwinkel	Wert
1	16000
2	16500
3	16800
4	17100
5	17400
6	17700
7	17800

DB42L01

Lastwinkel	Wert
1	16000
2	17000
3	17500
4	17500
5	17700
6	18300
7	18400

DB42M01

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17500
5	18500
6	18750
7	19000

DB42M02

Lastwinkel	Wert
1	16000
2	18000
3	20000
4	20000
5	20000
6	21000
7	22000

DB42M03

Lastwinkel	Wert
1	16000
2	17000
3	17000
4	17000
5	18000
6	19000
7	19000

DB42S01

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17500
5	18000
6	18000
7	18500

DB42S02

Lastwinkel	Wert
1	16000
2	18000
3	18000
4	18000
5	18500
6	19000
7	19000

DB42S03

Lastwinkel	Wert
1	16000
2	18000
3	20000
4	20000
5	20000
6	21000
7	22000

6.8 BLDC Motoren der Serie DB57

DB57C01

Lastwinkel	Wert
1	16000
2	16500
3	16500
4	16500
5	17000
6	17000
7	17000

DB57L01

Lastwinkel	Wert
1	16000
2	17000
3	17000
4	17000
5	17000
6	17000
7	17000

DB57S01

Lastwinkel	Wert
1	16500
2	17000
3	17000
4	17000
5	17000
6	17500
7	17500

6.9 BLDC Motoren der Serie DB87

DB87L01-S

Lastwinkel	Wert
1	16384
2	17000
3	17000
4	17000
5	17000
6	17000
7	17000

DB87M01-S

Lastwinkel	Wert
1	16384
2	18384
3	20384
4	22384
5	24384
6	26384
7	28384

DB87S01-S

Lastwinkel	Wert
1	16000
2	16500
3	17000
4	17250
5	17500
6	17500
7	18000

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z