Programmierhandbuch für Schrittmotor-steuerungen

Gültig ab Firmware 10.10.2009

NANOTEC ELECTRONIC GmbH & Co. KG
Gewerbestraße 11
D-85652 Landsham 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
Gewerbestraße 11
D-85652 Landsham / Pliening

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

Internet: www.nanotec.de

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

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 (verfügbar ab 02/10)
- SMCI33 *
- SMCI35 (verfügbar ab 12/09)
- SMCI47-S *
- SMCP33
- PD4-N (verfügbar ab 12/09)
- PD6-N

* bitte nachfolgenden Hinweis beachten!

Hinweis zu SMCI33 und SMCI47-S

Bei Steuerungen mit einer Firmware jünger 30.04.2009 kann das Update auf die neue Firmware, die in diesen 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. Die Adresse wird als '001' und nicht als '1' zurückgeschickt.

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

Beispiel

An die Steuerung übergebener Wert:	„#1G=10000000\r“
Antwort der Firmware:	„001G10000000\r“ (Wert wird ignoriert)

Beispiele

Setzen des Verfahrweges von Steuerung 1:	„#1s1000\r“ -> „001s1000\r“
Starten eines Satzes:	„#1A\r“ -> „001A\r“
Ungültiger Befehl:	„#1°\r“ -> „001°?\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“ -> „001Zs1000\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: 'Z5s' -> 'Z5s2000'

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	Para-meter	Siehe Abschnitt	Seite
Positionsmodus	'p'	*2.6.6 Positionierart setzen*	43
Verfahrweg	's'	*2.6.8 Verfahrweg einstellen*	47
Anfangsschrittfrequenz	'u'	*2.6.9 Minimalfrequenz einstellen*	47
Maximalschrittfrequenz	'o'	*2.6.10 Maximalfrequenz einstellen*	48
Zweite Maximalschritt-frequenz	'n'	*2.6.11 Maximalfrequenz 2 einstellen*	48
Beschleunigungs- und Bremsrampe	'b'	*2.6.12 Beschleunigungsrampe einstellen*	49
Drehrichtung	'd'	*2.6.15 Drehrichtung einstellen*	50
Drehrichtungsumkehr bei Wiederholungssätzen	't'	*2.6.16 Richtungsumkehr einstellen*	51
Wiederholungen	'W'	*2.6.17 Wiederholungen einstellen*	51
Pause zwischen Wiederholungen und Folgesätzen	'P'	*2.6.18 Satzpause einstellen*	52
Satznummer des Folgesatzes	'N'	*2.6.19 Folgesatz einstellen*	52

2.5 Allgemeine Befehle

2.5.1 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.

2.5.2 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.

2.5.3 Schrittmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'g'	0 bis 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 255, welche den Adaptiven Schrittmodus auswählt.

Auslesen

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

2.5.4 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 die neu gesetzte Adresse nicht bereits von einem anderen Motor belegt ist, sonst ist keine Kommunikation mehr möglich.

Adresse 0 und 255 sind für Fehlerfälle des EEPROMS reserviert.

Auslesen

Mit dem Befehl 'Zm' kann die aktuelle Adresse ausgelesen werden. Siehe auch Befehl *2.5.20 Motoradresse auslesen 'M'*.

2.5.5 Motormodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'!'	1 bis 101	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Wenn für Motormodus '!' und Positionierart 'p' ungültige Werte eingestellt werden, behält der Motormodus '!' seinen Wert und die Positionierart 'p' wird auf 1 gesetzt.

Beschreibung

Setzt den Motormodus. Es sind die folgenden Modi verfügbar:

Für altes Schema:

1: Positioniermodus

2: Drehzahlmodus

3: Flagpositioniermodus

4: Taktrichtungsmodus

5: Analogmodus

6: Joystickmodus

7: Analogpositioniermodus

8: HW-Referenzmodus

9: Drehmomentmodus

101: CL-Schnelltestmodus

101: CL-Testmodus

Weitere Informationen siehe Befehl *2.6.6 Positionierart setzen (altes Schema) 'p'*.

Für neues Schema:

10: Motormodus

Weitere Informationen siehe Befehl *2.6.7 Positionierart setzen (neues Schema) 'p'*.

Auslesen

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

2.5.6 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.

2.5.7 Endschaltertyp einstellen

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Gibt den Typ des Endschalters an:

- Wert 0: Öffner
- Wert 1: Schließer

Mit diesem Parameter wird der Firmware kenntlich gemacht, wann diese den externen Endschalter betätigt sieht. Der Endschalter ist dabei zwischen Versorgungsspannung (bei SMCIxx an +5V) und Eingang 6 zu schalten.

Deswegen bedeutet 'Öffner', dass im Normalfall ein High-Level an Eingang anliegt, da der Schalter im Normalfall geschlossen ist. Wird der Schalter betätigt, öffnet ("Öffner") dieser den Kontakt und am Eingang liegt keine Spannung mehr an.

Auslesen

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

2.5.8 Schrittwinkel einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'a'	0 bis 255	ja	u8 (integer)	18

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Zum Umrechnen der Drehgeber-Position in die Rotorlage benötigt die Firmware Kenntnis über den Schrittwinkel des Motors. Für 0,9° Motoren ist ein Wert von 9 und für 1,8° Motoren ist ein Wert von 18 einzustellen. Andere Werte werden nicht unterstützt.

Auslesen

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

2.5.9 Fehlerkorrekturmodus einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'U'	0 und 1	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

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

Auslesen

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

2.5.10 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.
Siehe Befehl *2.5.9 Fehlerkorrekturmodus einstellen 'U'*.

Auslesen

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

2.5.11 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.

2.5.12 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 *2.5.9 Fehlerkorrekturmodus einstellen 'U'*.

Zwischen Wiederholungs- oder Folgesätzen wird die Position nur geprüft, wenn die Pausezeit (siehe Befehl *2.6.18 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.

2.5.13 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.

2.5.14 Positionsfehler zurücksetzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'D'	-2147483648 bis +2147483647	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 *2.5.15* und *2.5.16*).

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

2.5.15 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 *2.5.21 Status auslesen '$'* ausgelesen werden.

2.5.16 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 *2.5.17 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.

2.5.17 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 $\pm 2^{31}$).

2.5.18 Position zurückstellen

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Position des Motors auf 0 zurück.

Die aktuelle Position des Motors ist danach die Referenzposition.

2.5.19 „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 *2.5.14 Positionsfehler zurücksetzen* und *2.5.18 Position zurückstellen*.

2.5.20 Motoradresse auslesen

Parameter

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

Antwort der Firmware

Liefert die Motoradresse zurück.

Beschreibung

Liefert die serielle Adresse zurück. Ist vor allem im Zusammenhang mit der Adressierungsart '*' sinnvoll, wenn die Motoradresse nicht bekannt ist.

2.5.21 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 bis 6 geben den aktuellen Modus als Integer an:

0: unbenutzt

1: Steuerung im Positioniermodus

2: Steuerung im Drehzahlmodus

3: Steuerung im Flagpositioniermodus

4: Steuerung Taktrichtungsmodus

5: Analogmodus

6: Joystickmodus

7: unbenutzt

Bit 7 ist unbelegt

2.5.22 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: 'PD4','PD4lc','PD2lc','SMCI32','SMCI47'

'_' Trennzeichen

Kommunikation: 'USB' oder 'RS485'

'_' Trennzeichen

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

Beispiel einer kompletten Antwort

„001v PD4_RS485_26-09-2007\r“

2.5.23 Firmwareversion auslesen (alt)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
' ' (Space)	–	nein	–	–

Antwort der Firmware

String der Firmwareversion (const, da neuer Befehl 'v' die Funktion übernommen hat).

Beschreibung

Für Bootloader benötigt, sonst nutzlos.

2.5.24 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 *2.5.27 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 'ZL' 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 angepochen wird, also 4 + 16 = 20.

2.5.25 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.

2.5.26 Debounce-Zeit für Eingänge setzen (Entprellen)

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt die Zeit in ms, die nach einer Signaländerung an einem Eingang gewartet wird, bis das Signal sicher anliegt (so genanntes „Entprellen“).

Auslesen

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

2.5.27 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 *2.5.24 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.

2.5.28 EEPROM Reset durchführen

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

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

2.5.29 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 *2.5.21 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.

2.5.30 Bootloader starten

Parameter

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

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.

2.5.31 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.

2.5.32 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 Currentmode (da diese Modi generell keine Rampe verwenden).

Auslesen

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

2.5.33 Maximalen Ruck für Beschleunigungsrampe setzen

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Setzt den maximalen Ruck für die Beschleunigung.

Auslesen

Wird das Schlüsselwort ohne „= + Wert" gesendet, kann der aktuell eingestellte 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)

2.5.34 Maximalen Ruck für Bremsrampe setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':B'	1 - 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

Wird das Schlüsselwort ohne „= + Wert" gesendet, kann der aktuell eingestellte 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)

2.5.35 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

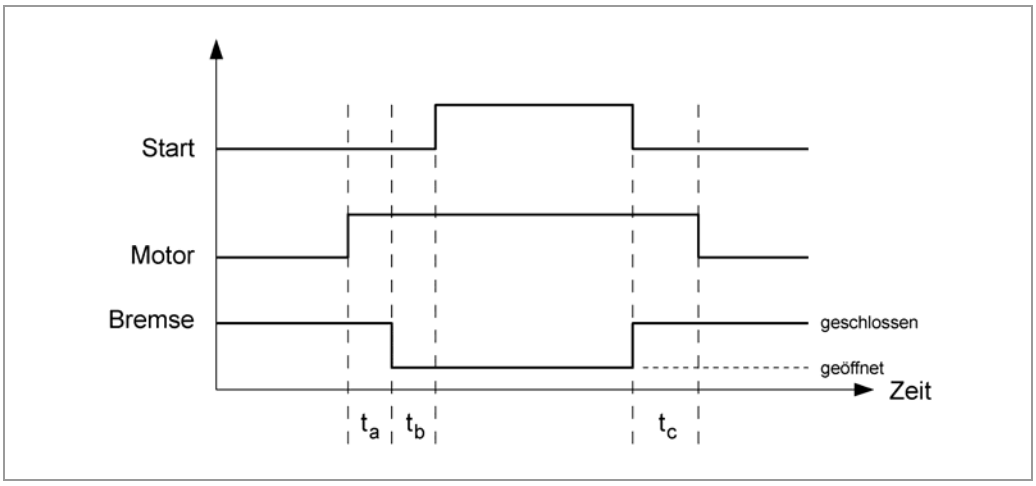
Bestätigt den Befehl durch Echo.

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.

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.

2.5.36 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 *2.5.35 Wartezeit für Abschalten der Bremsspannung setzen'ta'* für weitere Informationen.

Auslesen

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

2.5.37 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 *2.5.35 Wartezeit für Abschalten der Bremsspannung setzen'ta'* für weitere Informationen.

Auslesen

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

2.5.38 Baudrate der Steuerung setzen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':baud'	0 bis 255	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 'Z:baud' kann der aktuell gültige 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	Parameter	Siehe Abschnitt	Seite
Positionsmodus	'p'	*2.6.6 Positionierart setzen*	43
Verfahrweg	's'	*2.6.8 Verfahrweg einstellen*	47
Anfangsschrittfrequenz	'u'	*2.6.9 Minimalfrequenz einstellen*	47
Maximalschrittfrequenz	'o'	*2.6.10 Maximalfrequenz einstellen*	48
Zweite Maximalschritt-frequenz	'n'	*2.6.11 Maximalfrequenz 2 einstellen*	48
Beschleunigungs- und Bremsrampe	'b'	*2.6.12 Beschleunigungsrampe einstellen*	49
Drehrichtung	'd'	*2.6.15 Drehrichtung einstellen*	50
Drehrichtungsumkehr bei Wiederholungssätzen	't'	*2.6.16 Richtungsumkehr einstellen*	51
Wiederholungen	'w'	*2.6.17 Wiederholungen einstellen*	51
Pause zwischen Wieder-holungen und Folge-sätzen	'P'	*2.6.18 Satzpause einstellen*	52
Satznummer des Folge-satzes	'N'	*2.6.19 Folgesatz einstellen*	52

2.6.6 Positionierart setzen (altes Schema)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'p'	1 bis 17	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Wenn für Motormodus '!' und Positionierart 'p' ungültige Werte eingestellt werden, behält der Motormodus '!' seinen Wert und die Positionierart 'p' wird auf 1 gesetzt.

Beschreibung

Die Motormodi (Befehl '!') und Positionierarten (Befehl 'p') können sowohl nach dem alten Schema, als auch nach dem neuen Schema angewählt werden, siehe Abschnitt *2.6.7 Positionierart setzen (neues Schema)*.

Die Wertekombinationen des alten Schemas für Motormodus '!' und Positionierart 'p' sind:

Positioniermodus (!=1)	
p=1	Relative Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen* 's' gibt den Verfahrweg relativ zur aktuellen Position an. Der Befehl *2.6.15 Drehrichtung einstellen* 'd' gibt die Richtung an. Der Parameter *2.6.8 Verfahrweg einstellen* 's' muss positiv sein.
p=2	Absolute Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen 's'* gibt die Zielposition relativ zur Referenzposition an. Der Befehl *2.6.15 Drehrichtung einstellen 'd'* wird ignoriert.
p=3	Interne Referenzfahrt; Der Motor läuft mit der unteren Geschwindigkeit in die Richtung, die in Befehl *2.6.15 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 *2.5.6 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 *2.6.15 Drehrichtung einstellen 'd'* eingestellt ist, bis er den Endschalter erreicht hat. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe Befehl *2.5.6 Endschalterverhalten einstellen 'l'*.
Drehzahlmodus (!=2)	
p=1	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=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus

Flagpositioniermodus (!=3)	
p=1	Flagpositioniermodus; Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.9 Trigger auslösen 'T'* oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.8 Verfahrweg einstellen 's'*) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.11 Maximalfrequenz 2 einstellen 'n'*).
p=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Taktrichtungsmodus (!=4)	
p=1	Manuell links.
p=2	Manuell rechts.
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Analogmodus (!=5)	
	Nicht zutreffend
Joystickmodus (!=6)	
	Nicht zutreffend
Analogpositioniermodus (!=7)	
p=1	Analogpositioniermodus
p=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
HW-Referenzmodus (!=8)	
	Nicht zutreffend
Drehmomentmodus (!=9)	
	Nicht zutreffend
CL-Schnelltestmodus (!=101)	
p=1	CL-Schnelltestmodus
CL-Testmodus (!=101)	
p=2	CL-Testmodus

Auslesen

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

2.6.7 Positionierart setzen (neues Schema)

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'p'	1 bis 17	ja	u8 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Wenn für Motormodus '!' und Positionierart 'p' ungültige Werte eingestellt werden, behält der Motormodus '!' seinen Wert und die Positionierart 'p' wird auf 1 gesetzt.

Beschreibung

Die Motormodi (Befehl '!') und Positionierarten (Befehl 'p') können sowohl nach dem neuen Schema, als auch nach dem alten Schema angewählt werden, siehe Abschnitt *2.6.6 Positionierart setzen (altes Schema)*.

Im neuen Schema laufen alle Motormodi und Positionierarten des alten Schemas unter demselben Motormodus 10 ('!10') und der jeweiligen Positionierart ('p1' bis 'p17'). Dadurch ist es möglich, in den Sätzen den Motormodus und die Positionierart zu speichern.

Die Wertekombinationen des neuen Schemas für Motormodus '!' und Positionierart 'p' sind:

Positioniermodus (!=10)	
p=1	Relative Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen* 's' gibt den Verfahrweg relativ zur aktuellen Position an. Der Befehl *2.6.15 Drehrichtung einstellen* 'd' gibt die Richtung an. Der Parameter *2.6.8 Verfahrweg einstellen* 's' muss positiv sein.
p=2	Absolute Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen 's'* gibt die Zielposition relativ zur Referenzposition an. Der Befehl *2.6.15 Drehrichtung einstellen 'd'* wird ignoriert.
p=3	Interne Referenzfahrt; Der Motor läuft mit der unteren Geschwindigkeit in die Richtung, die in Befehl *2.6.15 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 *2.5.6 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 *2.6.15 Drehrichtung einstellen 'd'* eingestellt ist, bis er den Endschalter erreicht hat. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe Befehl *2.5.6 Endschalterverhalten einstellen 'l'*.
Drehzahlmodus (!=10)	
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 (!=10)	
p=6	Flagpositioniermodus; Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.9 Trigger auslösen 'T'* oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.8 Verfahrweg einstellen 's'*) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.11 Maximalfrequenz 2 einstellen 'n'*).
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Taktrichtungsmodus (!=10)	
p=7	Manuell links.
p=8	Manuell rechts.
p=9	Interne Referenzfahrt; siehe Positioniermodus
p=10	Externe Referenzfahrt; siehe Positioniermodus
Analogmodus (!=10)	
p=11	Analogmodus
Joystickmodus (!=10)	
p=12	Joystickmodus
Analogpositioniermodus (!=10)	
p=13	Analogpositioniermodus
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
HW-Referenzmodus (!=10)	
p=14	HW-Referenzmodus
Drehmomentmodus (!=10)	
p=15	Drehmomentmodus
CL-Schnelltestmodus (!=10)	
p=16	CL-Schnelltestmodus
CL-Testmodus (!=10)	
p=17	CL-Testmodus

Auslesen

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

2.6.8 Verfahrweg einstellen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
's'	-2147483648 bis +2147483647	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 *2.6.15 Drehrichtung einstellen 'd'* eingestellt.

Für die absolute Positionierung gibt dieser Befehl die Zielposition an. Negative Werte sind hier erlaubt. Die Drehrichtung aus Befehl *2.6.15 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 $\pm 2^{31}$).

Im Adaptiven Modus bezieht sich dieser Parameter auf Vollschritte.

Auslesen

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

2.6.9 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 *2.6.12 Beschleunigungsrampe einstellen 'b'*) bis zur Maximalgeschwindigkeit (Befehl *2.6.10 Maximalfrequenz einstellen 'o'*) hoch.

Auslesen

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

2.6.10 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.

2.6.11 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 Maximalgeschwindigkeit2 in Hertz (Schritte pro Sekunde) an.

Die Maximalgeschwindigkeit2 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 'p'*.

Auslesen

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

2.6.12 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.

2.6.13 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.

Auslesen

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

2.6.14 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.

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

Auslesen

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

2.6.15 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.

2.6.16 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 *2.6.17 Wiederholungen einstellen 'W'*.

Auslesen

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

2.6.17 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.

2.6.18 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.

2.6.19 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 die Anzahl der Samples eingestellt werden, über die der endgültige Wert gemittelt wird.

Auslesen

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

f = 8Bit (Bit 0-3: Anzahl der Samples; Bit 4-7: Größe der Hysterese) + 16

Bsp.: f=50: Glättung: rekursiver Filter über 4 Werte
f=84: starke Glättung: rekursiver Filter über 16 Werte

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 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.

2.7.6 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 abgesenktr wird.

Auslesen

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

2.7.7 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.

2.7.8 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.

2.7.9 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.

2.8 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 Geladenes Java-Programm verifizieren

Parameter

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

Antwort der Firmware

Als Antwort auf den Befehl meldet die Steuerung die Kennung des Programms „ECAFFE01“.

Beschreibung

Der Befehl lädt das aktuelle Programm aus dem EEPROM und initialisiert die VM. Diese Initialisierung erfolgt auch automatisch beim Einschalten der Steuerung sowie beim Übertragen des Programms mit dem PD4 Util.

2.8.5 Java-Programm beim Einschalten der Steuerung automatisch starten

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
'(JB'	0 bis 255	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

2.8.6 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.

2.8.7 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 und 1	ja	u8 (integer)	0

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Wird der Wert auf „1“ gesetzt, wird die Firmware angewiesen, den Regelkreis zu aktivieren. Dies ist nur möglich, wenn seit dem letzten Einschalten eine spezielle Referenzfahrt (Modus 8 „!8“) durchgeführt wurde.

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 *2.9.9 Polpaare des Motors einstellen*, *2.9.10 Anzahl der Inkremente einstellen* und *2.9.11 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 101 „!101“).

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 anabled
- „1“ = enabled

Beschreibung

Liest den Status des Closed-Loop-Modus aus.

2.9.3 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 *2.9.4 Zeit für Toleranzfenster der Endposition einstellen*.

Auslesen

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

2.9.4 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 *2.9.3 Toleranzfenster für Endposition einstellen*.

Auslesen

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

2.9.5 Maximal erlaubter 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 *2.9.6 Zeit für maximalen Schleppfehler einstellen*.

Auslesen

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

2.9.6 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 *2.9.5 Maximal erlaubter Schleppfehler einstellen*.

Auslesen

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

2.9.7 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 *2.9.8 Zeit für maximal erlaubte Drehzahlabweichung*.

Auslesen

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

2.9.8 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 *2.9.7 Maximal erlaubte Drehzahlabweichung*.

Auslesen

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

2.9.9 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 1/4 der Anzahl der Vollschritte pro Umdrehung. Die einstellbaren Werte betragen derzeit 50 und 100. Andere Werte haben zur Folge, dass der Regelkreis nicht funktioniert. Die Umrechnung für die Fehlerkorrektur ohne Regelkreis funktioniert aber auch dann.

Dieser Parameter korrespondiert mit dem Befehl *2.5.8 Schrittwinkel einstellen* 'a'. Wird der Parameter „:CL_motor_pp“ oder Parameter 'a' geändert, wird der jeweils korrespondierende Parameter ebenfalls geändert.

Die Umrechnung erfolgt nach folgender Formel:

$$\frac{\text{CL\_motor\_pp} = 900}{\text{COMM\_CMD\_SETSTEPANGLE}}$$

Auslesen

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

2.9.10 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 *2.9.11 Anzahl der Wellenumdrehungen einstellen*.

Derzeit werden die Werte 1600 und 2000 für den Regelkreis unterstützt. 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.

2.9.11 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 *2.9.10 Anzahl der Inkremente 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.

2.9.12 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.

2.9.13 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.

2.9.14 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.

2.9.15 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.

2.9.16 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.

2.9.17 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.

2.9.18 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.

2.9.19 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.

2.9.20 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.

2.9.21 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.

2.9.22 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.

2.9.23 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.

2.9.24 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.

2.9.25 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.

2.9.26 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.

2.9.27 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.

2.9.28 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.

2.9.29 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.

2.9.30 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.

2.9.31 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.

2.9.32 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.

2.9.33 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.

2.9.34 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.

2.9.35 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.

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

Allgemeines

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

Diese Korrekturwerte sind mit NanoPro ausles- und speicherbar, um sie bei einem Steuerungswechsel wieder zurück schreiben 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 Lastwinkel des Motors 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 Daten des Lastwinkels des Motors (Closed-Loop load angle) werden ausgelesen:

- 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

2.10.3 Korrekturwerte des Geschwindigkeitsreglers 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 Daten des Lastwinkels des Geschwindigkeitsreglers (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

2.10.4 Korrekturwerte des Stromreglers 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 Daten des Lastwinkels des Stromreglers (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

2.10.5 Korrekturwerte des Positionsreglers 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 Daten des Lastwinkels des Positionsreglers (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

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 → Sende 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_iIn=1

:Capt_sCurr_BYTE

:Capt_iIn_BYTE_HI

:Capt_iIn_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. Der Wertebereich beträgt „Unsigned 16“

„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

:Capt_sPos=1 → die Sollposition wird gewählt

:Capt_sPos=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

:Capt_iPos=1 → die Istposition wird gewählt

:Capt_iPos=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_iIn'	0 und 1	ja	u8 (integer)	0

Priorität

5

Einheit

keine

Beschreibung

Liefert die Bitmaske der Eingänge.

Beispiel

:Capt_iIn=1 → die Bitmaske der Eingänge wird gewählt

:Capt_iIn=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-Bus in %.

Beispiel

:Capt_iBus=1 → die Auslastung des CAN-Bus wird gewählt

:Capt_iBus=0 → die Auslastung des CAN-Bus wird abgewählt

2.11.10 Temperatur der Steuerung auslesen

Parameter

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

Priorität

8

Einheit

Wertebereich 0 – 1023

295 = 75 °C

261 = 80 °C

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

2.11.11 Schleppfehler auslesen

Parameter

Zeichen	Erlaubte Werte	Beschreibbar	Datentyp	Default-Wert
':Capt_IFollow'	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 der Steuerung SMCP33 und PD4-N

2.12.1 P-Anteil des Stromreglers im Stillstand einstellen

Parameter

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

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der P-Anteil des Stromreglers der Steuerungen SMCP33 und PD-4N 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 65535	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der P-Anteil des Stromreglers der Steuerungen SMCP33 und PD-4N 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 65535	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 der Steuerungen SMCP33 und PD-4N 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 65535	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der I-Anteil des Stromreglers der Steuerungen SMCP33 und PD-4N 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 65535	ja	u16 (integer)	1

Antwort der Firmware

Bestätigt den Befehl durch Echo.

Beschreibung

Mit diesem Parameter kann der I-Anteil des Stromreglers der Steuerungen SMCP33 und PD-4N 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 65535	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 der Steuerungen SMCP33 und PD-4N 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 des aktuellen Entwicklungsstandes (Beta1) sowie der zugrunde liegenden Hardware weist die aktuelle VM folgende Einschränkungen auf:

- Das Programm darf nach dem linken maximal 4096 Byte groß sein.
- 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

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.

Einschränkungen:

- Senden ist über Java nicht möglich.
- Aus dem Java-Programm und über die serielle Schnittstelle sollten nicht die gleichen Funktionen zur gleichen Zeit verwendet werden (z.B. Schrittmodus ändern), da die Änderungen in der Firmware eine gewisse Zeit benötigen und es deshalb zu undefinierten Antworten kommen kann.

3.2 Befehlsübersicht

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

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:

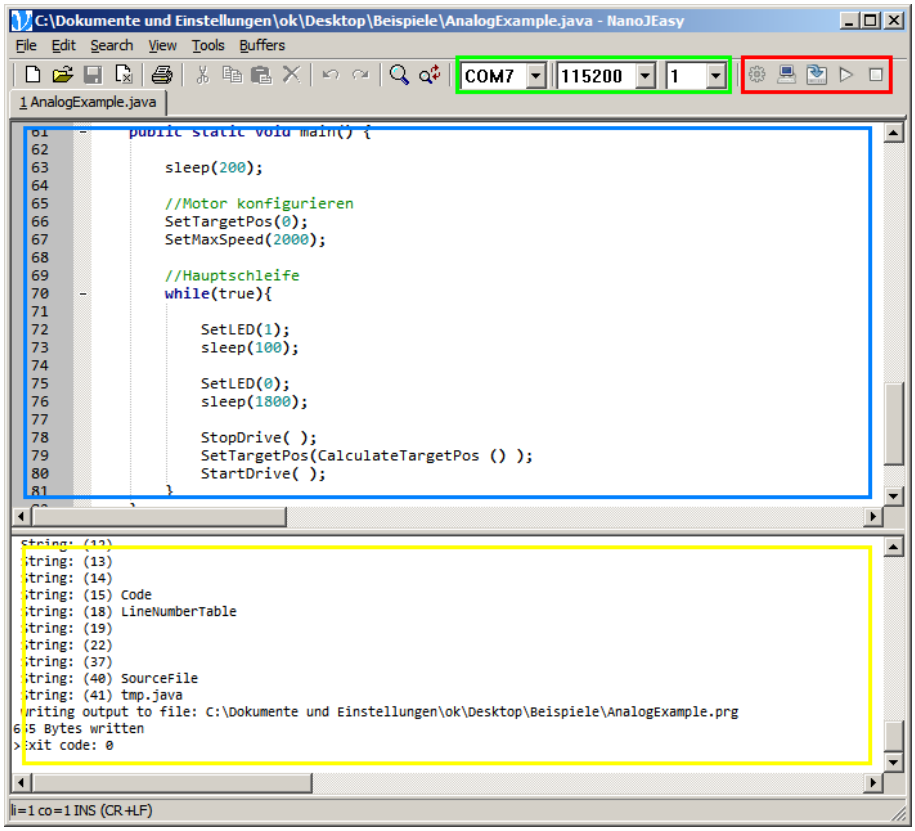
Schritt	Durchführung
1	Doppelklicken Sie auf die Datei setup.exe.
2	Wählen Sie die gewünschte Sprache aus.
3	Bestätigen Sie, dass Sie die Lizenzbestimmungen anerkennen.
4	Wählen Sie den Ordner aus, in dem NanoJEasy installiert werden soll.
5	Bestätigen oder Ändern Sie den vorgeschlagenen Startmenüeintrag 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 vier verschiedenen Klassen zusammengefasst, welche im Quellcode eingebunden werden können.

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

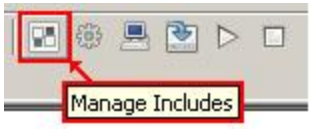
Einbinden einer Klasse

Die vier 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();

Einbinden einer einzelnen Funktion

Um Speicherplatz zu sparen, können die in den Klassen enthaltenen Funktionen auch einzeln verwendet werden.

Dazu muss jede Funktion, die verwendet werden soll, im Quellcode als Deklaration enthalten sein:

```
class Beispiel{
        //deklaration der Funktion
        static native void StartDrive( );

        //main Funktion
        //wird beim Programmstart aufgerufen
        public static void main() {

            //verwenden der Funktion
            StartDrive( );
        }
}
```

3.5 Klassen und Funktionen

3.5.1 Klasse „comm“

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.

3.5.2 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 Maximalgeschwindigkeit in Hertz (Schritte pro Sekunde) an.

Die Maximalgeschwindigkeit wird erst nach Durchfahren der Beschleunigungsrampe erreicht.

Die Funktion entspricht dem seriellen Befehl 'o<value>', siehe Befehl *2.6.10 Maximalfrequenz einstellen*.

drive.GetMaxSpeed

Deklaration:

static native int GetMaxSpeed( );

Liest den aktuell gültigen Wert der Maximalgeschwindigkeit in Hertz (Schritte pro Sekunde) aus.

Die Funktion entspricht dem seriellen Befehl 'Zo', siehe *2.3 Lesebefehl*.

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 *2.6.9 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.SetAccelaration

Deklaration:

static native void SetAccelaration(int value);

Gibt die Beschleunigungsrampe (und momentan noch gleichzeitig die Bremsrampe) an.

Zum Umrechnen der Parameters 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 *2.6.12 Beschleunigungsrampe einstellen*.

drive.GetAccelaration

Deklaration:

static native int GetAccelaration( );

Liest den aktuell gültigen Wert der Beschleunigungsrampe aus.

Die Funktion entspricht dem seriellen Befehl 'Zb', siehe *2.3 Lesebefehl*.

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 *2.6.8 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.SetDriveMode

Deklaration:

static native void SetDriveMode(int value);

Die Funktion entspricht dem seriellen Befehl '!<value>', siehe Befehl *2.5.5 Motormodus einstellen*.

Setzt den Motormodus. Es sind die folgenden Modi verfügbar:

Für altes Schema:

1: Positionsmodus

2: Drehzahlmodus

3: Flagpositioniermodus

4: Taktrichtungsmodus

5: Analogmodus

6: Joystickmodus

7: Analogpositioniermodus

8: HW-Referenzmodus

9: Drehmomentmodus

101: CL-Schnelltestmodus

101: CL-Testmodus

Weitere Informationen siehe Befehl *2.6.6 Positionierart setzen (altes Schema) 'p'*.

Für neues Schema:

10: Motormodus

Weitere Informationen siehe Befehl *2.6.7 Positionierart setzen (neues Schema) 'p'*.

drive.GetDriveMode

Deklaration:

static native int GetDriveMode( );

Liest den aktuellen Motormodus aus.

Die Funktion entspricht dem seriellen Befehl 'Z!', siehe *2.3 Lesebefehl*.

drive.SetMode

Deklaration:

static native void SetMode(int value);

Die Funktion entspricht dem seriellen Befehl 'p<value>', siehe Befehl *2.5.5 Motormodus einstellen*.

Die Wertekombinationen des alten Schemas für Motormodus '!' und Positionierart 'p' sind:

Positioniermodus (!=1)	
p=1	Relative Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen* 's' gibt den Verfahrweg relativ zur aktuellen Position an. Der Befehl *2.6.15 Drehrichtung einstellen* 'd' gibt die Richtung an. Der Parameter *2.6.8 Verfahrweg einstellen* 's' muss positiv sein.
p=2	Absolute Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen 's'* gibt die Zielposition relativ zur Referenzposition an. Der Befehl *2.6.15 Drehrichtung einstellen 'd'* wird ignoriert.
p=3	Interne Referenzfahrt; Der Motor läuft mit der unteren Geschwindigkeit in die Richtung, die in Befehl *2.6.15 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 *2.5.6 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 *2.6.15 Drehrichtung einstellen 'd'* eingestellt ist, bis er den Endschalter erreicht hat. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe Befehl *2.5.6 Endschalterverhalten einstellen 'l'*.
Drehzahlmodus (!=2)	
p=1	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=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Flagpositioniermodus (!=3)	

p=1	Flagpositioniermodus; Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.9 Trigger auslösen 'T'* oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.8 Verfahrweg einstellen 's'*) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.11 Maximalfrequenz 2 einstellen 'n'*).
p=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Taktrichtungsmodus (!=4)	
p=1	Manuell links.
p=2	Manuell rechts.
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Analogmodus (!=5)	
	Nicht zutreffend
Joystickmodus (!=6)	
	Nicht zutreffend
Analogpositioniermodus (!=7)	
p=1	Analogpositioniermodus
p=2	Nicht belegt
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
HW-Referenzmodus (!=8)	
	Nicht zutreffend
Drehmomentmodus (!=9)	
	Nicht zutreffend
CL-Schnelltestmodus (!=101)	
p=1	CL-Schnelltestmodus
CL-Testmodus (!=101)	
p=2	CL-Testmodus

Die Wertekombinationen des neuen Schemas für Motormodus '!' und Positionierart 'p' sind:

Positioniermodus (!=10)	
p=1	Relative Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen* 's' gibt den Verfahrweg relativ zur aktuellen Position an. Der Befehl *2.6.15 Drehrichtung einstellen* 'd' gibt die Richtung an. Der Parameter *2.6.8 Verfahrweg einstellen* 's' muss positiv sein.
p=2	Absolute Positionierung; Der Befehl *2.6.8 Verfahrweg einstellen 's'* gibt die Zielposition relativ zur Referenzposition an. Der Befehl *2.6.15 Drehrichtung einstellen 'd'* wird ignoriert.
p=3	Interne Referenzfahrt; Der Motor läuft mit der unteren Geschwindigkeit in die Richtung, die in Befehl *2.6.15 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 *2.5.6 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 *2.6.15 Drehrichtung einstellen 'd'* eingestellt ist, bis er den Endschalter erreicht hat. Danach wird je nach Einstellung eine Freifahrt durchgeführt. Siehe Befehl *2.5.6 Endschalterverhalten einstellen 'l'*.
Drehzahlmodus (!=10)	
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 (!=10)	
p=6	Flagpositioniermodus; Nach dem Start fährt der Motor auf die Maximaldrehzahl hoch. Nach Eintreffen des Trigger-Events (Befehl *2.7.9 Trigger auslösen 'T'* oder Trigger-Eingang) fährt der Motor noch den eingestellten Verfahrweg (Befehl *2.6.8 Verfahrweg einstellen 's'*) und verändert hierzu seine Geschwindigkeit auf die Maximalgeschwindigkeit2 (Befehl *2.6.11 Maximalfrequenz 2 einstellen 'n'*).
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
Taktrichtungsmodus (!=10)	
p=7	Manuell links.
p=8	Manuell rechts.
p=9	Interne Referenzfahrt; siehe Positioniermodus

p=10	Externe Referenzfahrt; siehe Positioniermodus
Analogmodus (!=10)	
p=11	Analogmodus
Joystickmodus (!=10)	
p=12	Joystickmodus
Analogpositioniermodus (!=10)	
p=13	Analogpositioniermodus
p=3	Interne Referenzfahrt; siehe Positioniermodus
p=4	Externe Referenzfahrt; siehe Positioniermodus
HW-Referenzmodus (!=10)	
p=14	HW-Referenzmodus
Drehmomentmodus (!=10)	
p=15	Drehmomentmodus
CL-Schnelltestmodus (!=10)	
p=16	CL-Schnelltestmodus
CL-Testmodus (!=10)	
p=17	CL-Testmodus

drive.GetMode

Deklaration:

static native int GetMode( );

Liest den aktuellen Positioniermodus 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 *2.5.1 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 *2.5.2 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

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 *2.6.15 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.GetEncoderPosition

Deklaration:

static native int GetEncoderPosition( );

Liest die aktuelle Position des Drehgebers aus.

Die Funktion entspricht dem seriellen Befehl 'I', siehe Befehl *2.5.16 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 *2.5.17 Position auslesen*.

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.

3.5.3 Klasse „io“

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).

3.5.4 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

3.6 Java Programmbeispiele

Es folgen einige kurze Beispielprogramme. Die Programme liegen sowohl als Quellcode, als auch in bereits compilierter Form in Verzeichnis „Beispiele“.

3.6.1 AnalogExample.java

```
/** liest alle 2 Sekunden den Analogwert und fährt eine
 *  daraus berechnete Position an
 *
 * */

import nanotec.io;
import nanotec.drive;
import nanotec.util;

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
        drive.SetTargetPos(0);
        drive.SetMaxSpeed(2000);

        //Hauptschleife
        while(true){

            io.SetLED(1);
            util.Sleep(100);

            io.SetLED(0);
            util.Sleep(1800);

            drive.StopDrive( );
            drive.SetTargetPos(CalculateTargetPos () );
            drive.StartDrive( );
        }
    }
}
```

3.6.2 DigitalExample.java

```
/** wenn Eingang 1 aktiv, wird die LED eingeschaltet
 * */

import nanotec.io;
import nanotec.util;


    class DigitalExample {

        public static void main() {

            util.Sleep(200);

            //Hauptschleife
            while(true){

                if( io.GetDigitalInput( ) == 65 ){
                    io.SetLED(1);
                } else {
                    io.SetLED(0);
                }

            }
        }
    }
```

3.6.3 TimerExample.java

```
/** Beispiel für einen mit GetMillis() realisierten Timer
 *
 *  Das Programm lässt die rote LED blinken
 * */

import nanotec.io;
import nanotec.util;


    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 Absolutpositionierung
 *  und fährt zwischen 2 Positionen mit verschiedenen Geschwindigkeiten
 *          hin und her
 * */

import nanotec.drive;
import nanotec.util;

        class ConfigDriveExample {

            public static void main() {

                //Motor konfigurieren
                drive.SetDriveMode(1);              //Positioniermodus
                drive.SetMode(2);                       //Absolut Positionierung
                drive.SetMinSpeed(100);
                drive.SetAccelaration(2000);    //Rampe
                drive.SetCurrent(10);               //Strom
                drive.SetCurrentReduction(1);  //Strom für Reduzierung

                //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 Sekunde warten


                }
            }
        }
```

3.6.5 DigitalOutput.java

```
/**setzt die Ausgänge und sendet den aktuellen Status
 * über die serielle Schnittstelle
 *
 * */


import nanotec.io;
import nanotec.comm;
import nanotec.util;


    class DigitalOutput {

        public static void main() {

            while( true ){
                io.SetDigitalOutput(0);
                comm.SendInt( GetDigitalOutput( ) );
                util.Sleep(1000);

                io.SetDigitalOutput(1);
                comm.SendInt( GetDigitalOutput( ) );
                util.Sleep(1000);

                io.SetDigitalOutput(2);
                comm.SendInt( GetDigitalOutput( ) );
                util.sleep(1000);
            }
        }
    }
```

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.

Java SE

Bei Java SE handelt es sich um die Standard Java Implementierung der Firma Sun. Von Sun werden zwei verschiedene Versionen angeboten: Das JRE (Java Runtime Environmont), mit dem ein fertiges Java Programm ausgeführt werden kann, sowie dem JDK (Java Development Kit), welches zum Entwickeln von Java Programmen benötigt wird.

Beides kann kostenlos von Sun (java.sun.com) heruntergeladen werden. Zum Entwickeln eines Programms für die Steuerung ist das JDK notwendig, welches auch die JRE enthält. Aktuell ist „JDK 6 Update 14“.

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.
3	Klicken Sie unter dem Fenster „Systemvariablen“ auf <Bearbeiten>.
4	Geben Sie unter „Wert der Variablen“ den Installationspfad des Programms ein.
5	Klicken Sie auf <OK>.

PD4-Utility

Das PD4 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 pd4_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 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 ohne Debug-Funktion starten

Geben Sie ein:

`ejvm_linker.exe Meinprogramm.class Meinprogramm.prg`

„*Meinprogramm*“ ist der Platzhalter für den Namen Ihres Programms.

ejvm_linker.exe mit Debug-Funktion starten

Für Debug-Zwecke kann das Programm mit dem Schalter '-debug' konvertiert werden. Das erzeugte Programm enthält dann noch zusätzliche Debug-Informationen und der Linker gibt detailliertere Informationen aus. Das erzeugte Programm wird dadurch jedoch größer.

Geben Sie ein:

`ejvm_linker.exe -debug Meinprogramm.class Meinprogramm.prg`

„*Meinprogramm*“ ist der Platzhalter für den Namen Ihres Programms.

Ergebnis

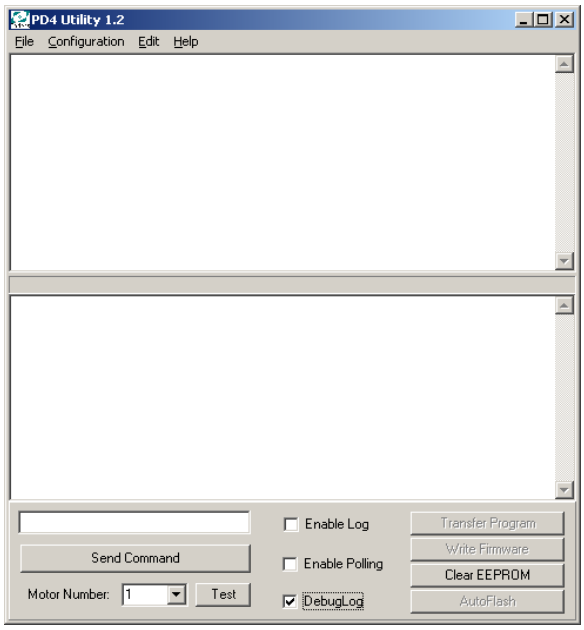
Das Ergebnis des Linkens und Konvertierens ist eine .prg-Datei, welche in die Steuerung geladen werden kann:

`Meinprogramm.prg`

3.7.4 Programm auf die Steuerung übertragen

Dialogfenster PD4-Utility

Die Übertragung auf die Steuerung erfolgt mit PD4-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 PD4 Utility wird ausgefüllt.
3	Klicken Sie zum Übertragen des Programms zur Steuerung auf die Schaltfläche <Transfer Program>.

3.7.5 Programm ausführen

PD4 Utility

Mit PD4 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:

(JI ... Geladenes Java-Programm verifizieren

Dieser Befehl lädt das aktuelle Programm aus dem EEProm und initialisiert die VM. Diese Initialisierung erfolgt auch automatisch beim Einschalten der Steuerung sowie beim Übertragen des Programms mit dem PD4 Util.

Als Antwort auf das Kommando erhält man die Kennung des Programms „ECAFFE01“. Siehe auch Abschnitt *2.8.4 Geladenes Java-Programm verifizieren*.

(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 *2.8.5 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 *2.8.6 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 *2.8.7 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).

Index	Fehlermeldung	Bedeutung
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).
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 *2.8.6 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 0.51.0.41 und SDK-Version 0.51.0.41 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 Kid) für „NanoPro“ sollte installiert sein. Durch dessen Installation wird die CommandsPD4I.dll registriert.
- Das .net-Framework 2.0 muss installiert sein.

Um die „Office-Beispiele“ mit Excel ausprobieren zu können, muss das VBA Add-On für Excel installiert sein. Für eine reibungslose Zusammenspiel der einzelnen Komponenten sollte das MS-Office 2003 und höher benutzt werden.

Programmierumgebungen

Als Programierumgebung kann VBA (Visual Basic) oder jede beliebige hochsprachige IDE wie z.B. Visual Studio verwendet werden. Den Beispielen ist einen Visual Studio Projektdatei beigefügt.

4.2 Befehlsübersicht

Nachfolgend finden Sie eine Auflistung der Befehle für die Programmierung über die COM-Schnittstelle:

4.3 Beschreibung der Funktionen

Methoden

Es gibt zwei Kategorien von Methoden:

- zum einen so genannte Set-Methoden, welche Informationen an die Steuerung übergeben,
- zum anderen Get-Methoden die Informationen von der Steuerung holen. Mit dem Rückgabewert bei den Set-Methoden kann geprüft werden, ob die Information auch zur Steuerung gesendet worden ist.

Abruf des Status der Ojekte

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

4.3.1 Allgemeine Funktionen

Errorflag

Definition:

bool **Errorflag**

Mit dieser Funktion kann abgefragt werden, ob ein Fehler aufgetreten ist.

ErrorNumber

Definition:

int **ErrorNumber**

Mit dieser Funktion kann die Fehlernummer abgefragt werden.

ErrorMessageString

Definition:

string **ErrorMessageString**

Mit dieser Funktion kann die Beschreibung des Fehlers abgefragt werden.

SerialPorts

Definition:

string[] **SerialPorts**

Mit dieser Funktion kann eine Liste der vorhandenen seriellen Schnittstellen des Computersystems abgefragt werden.

Verwendung:

string[ ] ports = SelectedPort

SelectedPort

Definition:

string **SelectedPort**

Mit dieser Funktion kann die zu benutzende serielle Schnittstelle gesetzt oder abgefragt werden.

Verwendung:

string port = SelectedPort

port = „COM40“

Selectedport = port

Baudrate

Definition:

int **Baudrate**

Mit dieser Funktion kann die Übertragungsrate gesetzt oder abgefragt werden.

GetAvailableMotorAddresses

Definition:

IList<int> **GetAvailableMotorAddresses**

Mit dieser Funktion kann eine Liste der verfügbaren Motoradressen abgefragt werden.

MotorAddresse

Definition:

int **MotorAddresse**

Mit diese Funktion wird die Motoradresse des zuvor angelegten COM-Objekts gesetzt oder abgefragt.

4.3.2 Statusfunktionen für ältere Motoren

GetStatusByte

Definition:

byte **GetStatusByte()**

Mit dieser Funktion kann das Statusbyte der Steuerung abgefragt werden.

Die Funktion entspricht dem seriellen Befehl '$', siehe Befehl *2.5.21 Status auslesen*.

IsMotorReady()

Definition:

bool **IsMotorReady()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor ready ist.

IsAtReferencePosition()

Definition:

bool **IsAtReferencePosition()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor an der Referenceposition ist.

HasPositionError

Definition:

bool **HasPositionError()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor einen Positionsfehler hat.

HasEndedTravelProfileAndStartInputStillActive

Definition:

bool **HasEndedTravelProfileAndStartInputStillActive()**

Diese Funktion liefert true (wahr) zurück, wenn das Fahrprofil beendet ist und das Eingangssignal Start noch aktiv ist.

IsPositionModeActive

Definition:

bool **IsPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Positionsmodus aktiv ist.

IsSpeedModeActive

Definition:

bool **IsSpeedModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Drehzahlmodus aktiv ist.

IsFlagPositionModeActive

Definition:

bool **IsFlagPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Flagpositions-modus aktiv ist.

IsClockDirectionModeActive

Definition:

bool **IsFlagPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Takt-Richtungsmodus aktiv ist.

IsJoyStickModeActive

Definition:

bool **IsJoyStickModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Joystickmodus aktiv ist.

IsAnalogModeActive

Definition:

bool **IsAnalogModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Analogmodus aktiv ist.

IsTorqueModeActive

Definition:

bool **IsTorqueModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Drehmoment-modus aktiv ist.

4.3.3 Motorsteuerungsfunktionen für ältere Motoren

ResetPositionError

Definition:

bool **ResetPositionError()**

Mit dieser Funktion kann der Positionsfehler zurückgesetzt werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x44.

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 0x41.

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 0x53.

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 0x2b.

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 0x2d.

TriggerOn

Definition:

bool **TriggerOn()**

Diese Funktion schaltet den Trigger des Motors ein.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x54.

StoreRecord

Definition:

bool **StoreRecord(***int recordNumber***)**

Diese Funktion speichert die zuvor gesetzten Parameter in dem Satz mit der Nummer des übergebenen Wertes.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x3e.

WriteCurrentReduction

Definition:

bool **WriteCurrentReduction(***int currentReduction***)**

Diese Funktion setzt den Strom der Stromreduzierung bei Stillstand in Prozent. Dieser Wert ist wie der Phasenstrom relativ zum Endwert. 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 0x72.

WritePhaseCurrent

Definition:

bool **WritePhaseCurrent(***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 0x69.

WriteFinalySendStatus

Definition:

bool **WriteFinalySendStatus(***bool value***)**

Diese Funktion schaltet das selbständige Senden eines Status am Ende einer Fahrt:

- value = 0, senden ausgeschaltet
- value = 1, senden eingeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x4a.

WriteAnalogueMin

Definition:

bool **WriteAnalogueMin(***double min***)**

Diese Funktion setzt die untere Spannungsschwelle 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 0x51.

WriteAnalogueMax

Definition:

bool **WriteAnalogueMax(***double max***)**

Diese Funktion setzt die obere Spannungsschwelle 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 0x52.

WriteErrorCorrectionRecord

Definition:

bool **WriteErrorCorrectionRecord(***string value, bool enabled***)**

Diese Funktion schaltet die automatische Fehlerkorrektur der Steuerung ein:

- Value = der Satz, welcher im Fehlerfall ausgeführt wird
- enabled = 0, Fehlerkorrektur ausgeschaltet
- enabled = 1, Fehlerkorrektur eingeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x46.

WriteToleranceWidth

Definition:

bool **WriteToleranceWidth(***int value***)**

Diese Funktion setzt die Toleranzbreite der Steuerung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x58.

WriteSwingOutTime

Definition:

bool **WriteSwingOutTime(***int value, bool enabled***)**

Diese Funktion setzte die Ausschwingzeit der Steuerung:

- Value = Werte der Ausschwingzeit
- enabled = 0, Ausschwingzeit ausgeschaltet
- enabled = 1, Ausschwingzeit eingeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x4F.

WriteReverseEncoderRotatingDirection

Definition:

bool **WriteReverseEncoderRotatingDirection(***bool value***)**

Diese Funktion setzt die Umkehrung der Drehgeberrichtung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x71.

SetAddress

Definition:

bool **SetAddress(***int newMotoraddress***)**

Diese Funktion setzt eine neue Motoradresse.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x6d.

SetReverseClearance

Definition:

bool **SetReverseClearance(***int reverseClearence***)**

Diese Funktion setzt das Umkehrspiel in Schritte.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x7a.

ResetCounter

Definition:

bool **ResetCounter()**

Diese Funktion setzt den Positionszähler auf den Wert 0.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x63.

WriteStartFrequency

Definition:

bool **WriteStartFrequency(***double frequency***)**

Diese Funktion setzt die Startfrequenz 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 0x75.

WriteRamp

Definition:

bool **WriteRamp(***int ramp***)**

Diese Funktion setzt die Rampe 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 0x62.

WriteNumberOfPasses

Definition:

bool **WriteNumberOfPasses(***int numberOfPasses***)**

Diese Funktion setzt die Anzahl der Durchgänge für 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 0x57.

WriteNormalFrequency

Definition:

bool **WriteNormalFrequency(***double frequency***)**

Diese Funktion setzt die Zielfrequenz 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 0x6f.

WriteNextOperation

Definition:

bool **WriteNextOperation(***int operationNumber***)**

Diese Funktion setzt den nächsten aus zuführenden Satz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x4e.

WriteSteps

Definition:

bool **WriteSteps(***int steps***)**

Diese Funktion setzt die Anzahl der auszuführenden Schritte 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 0x73.

WriteMaximumFrequency

Definition:

bool **WriteMaximumFrequency(***double frequency***)**

Diese Funktion setzt die maximale 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 0x6e.

ReadChangeDirection

Definition:

bool ***ReadChangeDirection(****int operationNumber***)**

Diese Funktion liest die Richtungsumkehr des Motors:

- *changeDirection* = 0, Richtungsumkehr ausgeschaltet
- *changeDirection* = 1, Richtungsumkehr eingeschaltet

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil) aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

WriteChangeDirection

Definition:

bool **WriteChangeDirection(***bool changeDirection***)**

Diese Funktion schaltet die Richtungsumkehr des Motors ein:

- *changeDirection* = 0, Richtungsumkehr ausgeschaltet
- *changeDirection* = 1, Richtungsumkehr eingeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 0x74.

WriteBreak

Definition:

bool **WriteBreak(***double brakTime***)**

Diese Funktion setzt die Pausenzeit des Motors 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 0x50.

ReadRecord

Definition:

int **ReadRecord(***int operationNumber***)**

Diese Funktion liest einen Record (Datensatz) des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadAddress

Definition:

bool **ReadAddress(***int operationNumber***)**

Diese Funktion liest die Motoradresse des angeschlossenen Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x4d.

Achtung:
Bei Verwendung dieses Befehls sollte nur ein Motor angeschlossen sein.

ReadCounter

Definition:

bool **ReadCounter(***double operationNumber***)**

Diese Funktion liest die aktuelle Position des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x43.

ReadStartFrequency

Definition:

double **ReadStartFrequency(***int operationNumber***)**

Diese Funktion liest die Startfrequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadRamp

Definition:

int **ReadRamp(***int operationNumber***)**

Diese Funktion liest die Rampe des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadNumberOfPasses

Definition:

int **ReadNumberOfPasses(***int operationNumber***)**

Diese Funktion liest die Anzahl der Durchläufe des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadNormalFrequency

Definition:

double **ReadNormalFrequency(***int operationNumber***)**

Diese Funktion liest die Zielfrequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadNextOperation

Definition:

int **ReadNextOperation(***int operationNumber***)**

Diese Funktion liest die Nummer des nächsten auszuführenden Satzes (Fahrprofil).

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadSteps

Definition:

int **ReadSteps(***int operationNumber***)**

Diese Funktion liest die Anzahl der in der Steuerung eingestellten Schritte des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadMemory

Definition:

int **ReadMemory(***int speicherAddress***)**

Diese Funktion liest einen Wert aus einer bestimmten Speicheradresse. Die Speicheradresse wird als Parameter übergeben.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadMaximumFrequency

Definition:

int **ReadMaximumFrequency(***int operationNumber***)**

Diese Funktion liest maximale Frequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadBreak

Definition:

int **ReadBreak(***int operationNumber***)**

Diese Funktion liest die Pausenzeit in des Motors in Millisekunden.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadCurrentReduction

Definition:

int **ReadCurrentReduction(***int operationNumber***)**

Diese Funktion liest die Stromreduzierung bei Stillstand in Prozent.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadPhaseCurrent

Definition:

int **ReadPhaseCurrent(***int operationNumber***)**

Diese Funktion liest den Phasenstrom des Motors in Prozent.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

GetOperationMode

Definition:

int **GetOperationMode()**

Diese Funktion liest den gerade aktiven Operationsmode des Motors:

- Rückgabe = 1 entspricht Positionsmodus
- Rückgabe = 2 entspricht Drehzahlmodus
- Rückgabe = 3 entspricht Flagpositionmodus
- Rückgabe = 4 entspricht Taktrichtungsmodus

Die Funktion verwendet den seriellen Befehl 0x21.

WriteStepMode

Definition:

bool **WriteStepMode(***int stepMode***)**

Diese Funktion setzt den Schrittmodus des Motors:

- stepMode = 0 entspricht Vollschritt
- stepMode = 1 entspricht Halbschritt
- stepMode = 2 entspricht Viertelschritt
- stepMode = 3 entspricht Füntelschritt
- stepMode = 4 entspricht Achtelschritt
- stepMode = 5 entspricht Zehntelschritt
- stepMode = 6 entspricht 16tel Schritt
- stepMode = 7 entspricht 32igstel Schritt
- stepMode = 8 entspricht 64idstel Schritt
- stepMode = 9 entspricht Adaptiver Mikroschritt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x67.

WriteInternalNormalRunBehavior

Definition:

bool **WriteInternalNormalRunBehavior(***int normalRunBehavior***)**

Diese Funktion setzt das Endschalterverhalten bei interner Normalfahrt des Motors:

- *normalRunBehavior* = 0 entspricht Disable
- *normalRunBehavior* = 1 entspricht Frei Rückwärts
- *normalRunBehavior* = 2 entspricht Frei Vorwärts
- *normalRunBehavior* = 3 entspricht Stop

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x65.

WriteInternalReferenceRunBehavior

Definition:

bool **WriteInternalReferenceRunBehavior(***int refrenceBehavior***)**

Diese Funktion setzt das Endschalterverhalten bei interner Referenzfahrt des Motors:

- *refrenceBehavior* = 0 entspricht Frei Rückwärts
- *refrenceBehavior* = 1 entspricht Frei Vorwärts

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x65.

WriteExternalSwitchType

Definition:

int **WriteExternalSwitchType (***int switchType***)**

Diese Funktion setzt den externen Endschaltertyp des Motors:

- *switchType* = 0 entspricht Öffner
- *switchType* = 1 entspricht Schliesser

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x6c.

WriteExternalNormalRunBehavior

Definition:

bool **WriteExternalNormalRunBehavior(***int normalRunBehavior***)**

Diese Funktion setzt das Endschalterverhalten bei externer Normalfahrt des Motors:

- *normalRunBehavior* = 0 entspricht Disable
- *normalRunBehavior* = 1 entspricht Frei Rückwärts
- *normalRunBehavior* = 2 entspricht Frei Vorwärts
- *normalRunBehavior* = 3 entspricht Stop

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x65.

WriteExternalReferenceRunBehavior

Definition:

bool **WriteExternalReferenceRunBehavior(***int refrenceBehavior***)**

Diese Funktion setzt das Endschalterverhalten bei externer Referenzfahrt des Motors:

- *refrenceBehavior* = 0 entspricht Frei Rückwärts
- *refrenceBehavior* = 1 entspricht Frei Vorwärts

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x65.

WriteOperationType

Definition:

bool **WriteOperationType(***int operationType***)**

Diese Funktion setzt den Operationstypen des Motors:

- *operationType* = 0 entspricht relativ; abhängig vom Operationsmodus
- *operationType* = 1 entspricht absolut; abhängig vom Operationsmodus
- *operationType* = 3 entspricht interne Referenzfahrt
- *operationType* = 4 entspricht externe Referenzfahrt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 0x70.

ReadOperationType

Definition:

int **ReadOperationType(***int operationNumber***)**

Diese Funktion liest den eingestellten Operationstyp des Motors.

- *Rückgabe* = 0 entspricht relativ; abhängig vom Operationsmodus
- *Rückgabe* = 1 entspricht absolut; abhängig vom Operationsmodus
- *Rückgabe* = 3 entspricht interne Referenzfahrt
- *Rückgabe* = 4 entspricht externe Referenzfahrt

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

WriteDirection

Definition:

bool **WriteDirection(***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 verwendet den seriellen Befehl 0x64.

ReadDirection

Definition:

int **ReadDirection(***int operationNumber***)**

Diese Funktion liest die Drehrichtung des Motors:

- *Rückgabe* = 0 entspricht links
- *Rückgabe* = 1 entspricht rechts

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem gelesen werden soll.

Die Funktion verwendet den seriellen Befehl 0x5a.

ReadReverseClearance

Definition:

int **ReadReverseClearance()**

Diese Funktion liest das Umkehrspiel des Motors.

Die Funktion verwendet den seriellen Befehl 0x5a.

4.3.4 Statusfunktionen für neuere Motoren

GetNewStatus

Definition:

byte **GetNewStatus()**

Mit dieser Funktion kann der Status der Steuerung abgefragt werden.

IsNewMotorReady

Definition:

bool **IsNewMotorReady()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor ready ist.

IsAtNewReferencePosition

Definition:

bool **IsAtNewReferencePosition()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor an der Referenceposition ist.

HasNewPositionError

Definition:

bool **HasNewPositionError()**

Diese Funktion liefert true (wahr) zurück, wenn der Motor einen Positionsfehler hat.

HasNewEndedTravelProfileAndStartInputStillActive

Definition:

bool **HasNewEndedTravelProfileAndStartInputStillActive()**

Diese Funktion liefert true (wahr) zurück, wenn das Fahrprofil beendet ist und das Eingangssignal Start noch aktiv ist.

IsNewPositionModeActive

Definition:

bool **IsNewPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Positionsmodus aktiv ist.

IsNewSpeedModeActive

Definition:

bool **IsNewSpeedModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Drehzahlmodus aktiv ist.

IsNewFlagPositionModeActive

Definition:

bool **IsNewFlagPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Flagpositions-modus aktiv ist.

IsNewClockDirectionModeActive

Definition:

bool **IsNewFlagPositionModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Takt-Richtungsmodus aktiv ist.

IsNewNewJoyStickModeActive

Definition:

bool **IsNewJoyStickModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Joystickmodus aktiv ist.

IsNewAnalogModeActive

Definition:

bool **IsNewAnalogModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Analogmodus aktiv ist.

IsNewTorqueModeActive

Definition:

bool **IsNewTorqueModeActive()**

Diese Funktion liefert true (wahr) zurück, wenn in der Steuerung der Drehmoment-modus aktiv ist.

4.3.5 Motorsteuerungsfunktionen für neuere Motoren

SetNewSoftware

Definition:

bool **SetNewSoftware(***bool value***)**

Diese Funktion setzt für das COM-Objekt die Information, dass ein entsprechender (neuer) Befehlssatz für den Motor gewählt werden soll.

0 entspricht ja

1 entspricht nein

ResetAllSettings

Definition:

bool **ResetAllSettings()**

Diese Funktion setzt die Einstellungen der Steuerung auf Defaultwerte (Werkseinstellungen) zurück.

Die Funktion entspricht dem seriellen Befehl '~', siehe Befehl *2.5.28 EEPROM Reset*.

GetNewVersion

Definition:

string **GetNewVersion()**

Diese Funktion liest den Versionstext aus der Steuerung.

Die Funktion entspricht dem seriellen Befehl 'v', siehe Befehl *2.5.22 Firmwareversion auslesen*.

ResetNewPositionError

Definition:

bool **ResetNewPositionError()**

Mit dieser Funktion kann der Positionsfehler zurückgesetzt werden.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'D', siehe Befehl *2.5.14 Positionsfehler zurücksetzen*.

SetNewSendStatusWhenCompleted

Definition:

bool **SetNewSendStatusWhenCompleted(***bool sendStatus***)**

Diese Funktion schaltet das selbständige Senden eines Status am Ende einer Fahrt:

- *sendStatus* = 0, senden ausgeschaltet
- *sendStatus* = 1, senden eingeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'J', siehe Befehl *2.5.29 Automatisches Senden des Status einstellen*.

GetNewSendStatusWhenCompleted

Definition:

bool **GetNewSendStatusWhenCompleted()**

Diese Funktion liest, ob das selbständige Senden eines Status am Ende einer Fahrt eingeschaltet ist:

- *Rückgabe* = 0, senden ausgeschaltet
- *Rückgabe* = 1, senden eingeschaltet

Die Funktion entspricht dem seriellen Befehl 'J', siehe Befehl *2.5.29 Automatisches Senden des Status einstellen*.

GetNewPosition

Definition:

int **GetNewPosition()**

Diese Funktion liest die aktuelle Position des Motors.

Die Funktion entspricht dem seriellen Befehl 'C', siehe Befehl *2.5.17 Position auslesen*.

ResetNewPosition

Definition:

bool ResetNewPosition**()**

Diese Funktion setzt den Positionszähler auf den Wert 0.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'c', siehe Befehl *2.5.18 Position zurückstellen*.

StartNewTravelProfile

Definition:

bool **StartNewTravelProfile()**

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', siehe Befehl *2.6.1 Motor starten*.

StopNewTravelProfile

Definition:

bool **StopNewTravelProfile()**

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 'S', siehe Befehl *2.6.2 Motor stoppen*.

IncreaseNewFrequency

Definition:

bool **IncreaseNewFrequency()**

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 '+', siehe Befehl *2.7.7 Drehzahl erhöhen*.

DecreaseNewFrequency

Definition:

bool **DecreaseNewFrequency()**

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 '-', siehe Befehl *2.7.8 Drehzahl verringern*.

NewTriggerOn

Definition:

bool **NewTriggerOn()**

Diese Funktion schaltet den Trigger des Motors ein.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'T', siehe Befehl *2.7.9 Trigger auslösen*.

ChooseNewRecord

Definition:

bool **ChooseNewRecord(***int recordNumber***)**

Diese Funktion liest einen bestimmten Satz (Fahrprofil) des Motors.

Der Parameter *recordNumber* ist dabei die Satznummer (Fahrprofil), die gelesen 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', siehe Befehl *2.6.3 Satz aus EEPROM laden*.

SetNewRecord

Definition:

bool **SetNewRecord(***int recordNumber***)**

Diese Funktion schreibt einen neuen bestimmten Satz (Fahrprofil) des Motors.

Der Parameter *recordNumber* ist dabei die Satznummer (Fahrprofil), die geschrieben 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 '>', siehe Befehl *2.6.5 Satz speichern*.

SetNewPlay

Definition:

bool **SetNewPlay(***int play***)**

Diese Funktion setzt den Totbereich 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 '=', siehe Befehl *2.7.1 Totbereich Joystickmodus einstellen*.

GetNewPlay

Definition:

int **GetNewPlay()**

Diese Funktion liest den Totbereich des Motors.

Die Funktion entspricht dem seriellen Befehl '=', siehe Befehl *2.7.1 Totbereich Joystickmodus einstellen*.

SetNewSoftwareFilter

Definition:

bool **SetNewSoftwareFilter(***int softwareFilter***)**

Diese Funktion setzt den Filter 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 'f', siehe Befehl *2.7.2 Filter für Analog- und Joystickmodus einstellen*.

GetNewSoftwareFilter

Definition:

int **GetNewSoftwareFilter()**

Diese Funktion liest den Filter des Motors.

Die Funktion entspricht dem seriellen Befehl 'f', siehe Befehl *2.7.2 Filter für Analog- und Joystickmodus einstellen*.

SetNewStepMode

Definition:

bool **SetNewStepMode(***int stepMode***)**

Diese Funktion setzt den Schrittmodus des Motors:

- stepMode = 0 entspricht Vollschritt
- stepMode = 1 entspricht Halbschritt
- stepMode = 2 entspricht Viertelschritt
- stepMode = 3 entspricht Füntelschritt
- stepMode = 4 entspricht Achtelschritt
- stepMode = 5 entspricht Zehntelschritt
- stepMode = 6 entspricht 16tel Schritt

- stepMode = 7 entspricht 32igstel Schritt
- stepMode = 8 entspricht 64idstel Schritt
- stepMode = 9 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', siehe Befehl *2.5.3 Schrittmodus einstellen*.

GetNewStepMode

Definition:

int **GetNewStepMode()**

Diese Funktion liest den Schrittmodus des Motors:

- Rückgabe = 0 entspricht Vollschritt
- Rückgabe = 1 entspricht Halbschritt
- Rückgabe = 2 entspricht Viertelschritt
- Rückgabe = 3 entspricht Füntelschritt
- Rückgabe = 4 entspricht Achtelschritt
- Rückgabe = 5 entspricht Zehntelschritt
- Rückgabe = 6 entspricht 16tel Schritt
- Rückgabe = 7 entspricht 32igstel Schritt
- Rückgabe = 8 entspricht 64idstel Schritt
- Rückgabe = 9 entspricht Adaptiver Mikroschritt

Die Funktion entspricht dem seriellen Befehl 'g', siehe Befehl *2.5.3 Schrittmodus einstellen*.

SetNewMotorAddress

Definition:

bool **SetNewMotorAddress(***int motorNumber***)**

Diese Funktion setzt eine neue 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', siehe Befehl *2.5.4 Motoradresse einstellen*.

GetNewMotorAddress

Definition:

int **GetNewMotorAddress()**

Diese Funktion liest die Motoradresse.

Die Funktion entspricht dem seriellen Befehl 'M', siehe Befehl *2.5.20 Motoradresse auslesen*.

Achtung:
Bei Verwendung dieses Befehls sollte nur ein Motor angeschlossen sein.

GetNewErrorAddress

Definition:

int **GetNewErrorAddress()**

Diese Funktion liest die Fehleradresse, an der sich der letzte Fehlercode befindet, aus der Steuerung.

Die Funktion entspricht dem seriellen Befehl 'E', siehe Befehl *2.5.15 Fehlerspeicher auslesen*.

GetNewError

Definition:

int **GetNewError(***int errorAddress***)**

Diese Funktion liest den Fehler (Status) an der übergebenen Adresse.

Die Funktion entspricht dem seriellen Befehl 'E', siehe Befehl *2.5.15 Fehlerspeicher auslesen*.

SetNewEnableAutoCorrect

Definition:

bool **SetNewEnableAutoCorrect(***string recordNumber, bool autocorrct***)**

Diese Funktion schaltet die automatische Fehlerkorrektur des Motors ein.

Der Parameter *recordNumber* ist dabei die Satznummer (Fahrprofil), bei der die Fehlerkorrektur eingeschaltet 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', siehe Befehl *2.5.10 Satz für Autokorrektur einstellen*.

SetNewSwingOutTime

Definition:

bool **WriteSwingOutTime(***int value***)**

Diese Funktion setzt die Ausschwingzeit der Steuerung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'O', siehe Befehl *2.5.12 Ausschwingzeit einstellen*.

GetNewSwingOutTime

Definition:

int **GetNewSwingOutTime()**

Diese Funktion liest die Ausschwingzeit der Steuerung.

Die Funktion entspricht dem seriellen Befehl 'O', siehe Befehl *2.5.12 Ausschwingzeit einstellen*.

SetNewNextRecord

Definition:

bool **SetNextRecord(***int recordNumber***)**

Diese Funktion setzt den nächsten auszuführenden Satz.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'N', siehe Befehl *2.6.19 Folgesatz einstellen*.

GetNewNextRecord

Definition:

int **GetNextRecord(***int recordNumber***)**

Diese Funktion liest den nächsten auszuführenden Satz.

Der Parameter *recordNumber* ist dabei die Satznummer (Fahrprofil), für die der nächste Satz gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'N', siehe Befehl *2.6.19 Folgesatz einstellen*.

SetNewOperationMode

Definition:

bool **SetNewOperationMode(***int operationMode***)**

Diese Funktion setzt den Operationsmodus:

- *operationMode* = 1 entspricht Positionsmodus
- *operationMode* = 2 entspricht Drehzahlmodus
- *operationMode* = 3 entspricht Flagpositionmodus
- *operationMode* = 4 entspricht Taktrichtungsmodus
- *operationMode* = 5 entspricht Analogmodus
- *operationMode* = 6 entspricht Joystickmodus
- *operationMode* = 7 entspricht AnalogPositionsmodus
- *operationMode* = *9* entspricht Drehmomentmodus

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl '!', siehe Befehl *2.5.5 Motormodus einstellen*.

GetNewOperationMode

Definition:

int **GetNewOperationMode()**

Diese Funktion liest den gerade aktiven Operationsmodus:

- Rückgabe = 1 entspricht Positionsmodus
- Rückgabe = 2 entspricht Drehzahlmodus
- Rückgabe = 3 entspricht Flagpositionmodus
- Rückgabe = 4 entspricht Taktrichtungsmodus
- Rückgabe = 5 entspricht Analogmodus
- Rückgabe = 6 entspricht Joystickmodus

- Rückgabe = 7 entspricht AnalogPositionsmodus
- Rückgabe = 9 entspricht Drehmomentmodus

Die Funktion entspricht dem seriellen Befehl '!', siehe Befehl *2.5.5 Motormodus einstellen*.

SetNewPhaseCurrent

Definition:

bool **SetNewDirection(***int phaseCurrent***)**

Diese Funktion setzt den Phasenstrom in Prozent. Werte über 100 sollten vermieden werden.

Die Funktion entspricht dem seriellen Befehl 'r', siehe Befehl *2.5.2 Phasenstrom im Stillstand einstellen*.

GetNewPhaseCurrent

Definition:

int **GetNewPhaseCurrent()**

Diese Funktion liest den Phasenstrom in Prozent.

Die Funktion entspricht dem seriellen Befehl 'i', siehe Befehl *2.5.1 Phasenstrom einstellen*.

SetNewCurrentReduction

Definition:

bool **SetNewCurrentReduction(***int currentReduktion***)**

Diese Funktion 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', siehe Befehl *2.5.2 Phasenstrom im Stillstand einstellen*.

GetNewCurrentReduction

Definition:

int **GetNewCurrentReductiont()**

Diese Funktion setzt den Strom der Stromreduzierung bei Stillstand in Prozent.

Die Funktion entspricht dem seriellen Befehl 'r', siehe Befehl *2.5.2 Phasenstrom im Stillstand einstellen*.

SetNewLimitSwitchType

Definition:

int **SetNewLimitSwitchType(***int switchType***)**

Diese Funktion setzt den externen Endschaltertyp des Motors:

- *switchType* = 0 entspricht Öffner
- *switchType* = 1 entspricht Schliesser

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion verwendet den seriellen Befehl 'e', siehe Befehl *2.5.7 Endschaltertyp einstellen*.

GetNewLimitSwitchType

Definition:

int **GetNewLimitSwitchType()**

Diese Funktion liest den externen Endschaltertyp des Motors:

- *switchType* = 0 entspricht Öffner
- *switchType* = 1 entspricht Schließer

Die Funktion verwendet den seriellen Befehl 'e', siehe Befehl *2.5.7 Endschaltertyp einstellen*.

SetNewReverseClearance

Definition:

bool **SetNewReverseClearance(***int reverseClearence***)**

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', siehe Befehl *2.5.31 Umkehrspiel einstellen*.

GetNewReverseClearance

Definition:

int **GetNewReverseClearance()**

Diese Funktion liest das Umkehrspiel in Schritten.

Die Funktion entspricht dem seriellen Befehl 'z', siehe Befehl *2.5.31 Umkehrspiel einstellen*.

SetNewMotorStepAngel

Definition:

bool **SetNewMotorStepAngele(***int stepAngle***)**

Diese Funktion setzt den Motorschrittwinkel:

- *stepAngle* = 9 entspricht 0,9°
- *stepAngle* = 18 entspricht 1,8°
- *stepAngle* = 375 entspricht 3,75°
- *stepAngle* = 75 entspricht 75°
- *stepAngle* = 150 entspricht 150°
- *stepAngle* = 180 entspricht 180°

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'a', siehe Befehl *2.5.8 Schrittwinkel einstellen*.

GetNewMotorStepAngel

Definition:

int **GetNewMotorStepAngel()**

Diese Funktion liest den Motorschrittwinkel:

- Rückgabe = 9 entspricht 0,9°
- Rückgabe = 18 entspricht 1,8°
- Rückgabe = 375 entspricht 3,75°
- Rückgabe = 75 entspricht 75°
- Rückgabe = 150 entspricht 150°
- Rückgabe = 180 entspricht 180°

Die Funktion entspricht dem seriellen Befehl 'a', siehe Befehl *2.5.8 Schrittwinkel einstellen*.

SetNewAnalogueMin

Definition:

bool **SetNewAnalogueMin(***double analoqueMin***)**

Diese Funktion setzt die minimale Analogspannung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'Q', siehe Befehl *2.7.3 Minimalspannung für Analogmodus einstellen*.

GetNewAnalogueMin

Definition:

double **GetNewAnalogueMin()**

Diese Funktion liest die minimale Analogspannung.

Die Funktion entspricht dem seriellen Befehl 'Q', siehe Befehl *2.7.3 Minimalspannung für Analogmodus einstellen*.

SetNewAnalogueMax

Definition:

bool **SetNewAnalogueMax(***double analoqueMin***)**

Diese Funktion setzt die maximale Analogspannung.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'R', siehe Befehl *2.7.4 Maximalspannung für Analogmodus einstellen*.

GetNewAnalogueMax

Definition:

double **GetNewAnalogueMax()**

Diese Funktion liest die maximale Analogspannung.

Die Funktion entspricht dem seriellen Befehl 'R', siehe Befehl *2.7.4 Maximalspannung für Analogmodus einstellen*.

SetNewAngelDeviationMax

Definition:

bool ***SetNewAngelDeviationMax(****int deviation***)**

Funktion setzt die maximale Winkelabweichung vom Motor.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'X', siehe Befehl *2.5.13 Maximale Abweichung Drehgeber einstellen*.

GetNewAngelDeviationMax

Definition:

double **GetNewAngelDeviationMax()**

Diese Funktion liest die maximale Winkelabweichung vom Motor.

Die Funktion entspricht dem seriellen Befehl 'X', siehe Befehl *2.5.13 Maximale Abweichung Drehgeber einstellen*.

SetNewPositionType

Definition:

bool **SetNewPositionType(***int positionType***)**

Diese Funktion setzt den Motorschrittwinkel:

- *positionType* = 1 entspricht relativ; abhängig vom Operationsmodus
- *positionType* = 2 entspricht absolut; abhängig vom Operationsmodus
- *positionType* = 3 entspricht internen Referenzfahrt
- *positionType* = 4 entspricht externen Referenzfahrt

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'p', siehe Befehle *2.6.6 Positionierart setzen (altes Schema)* und *2.6.7 Positionierart setzen (neues Schema)*.

GetNewPositionType

Definition:

int **GetNewPositionType(***int operationNumber***)**

Diese Funktion liest den Motorschrittwinkel:

- Rückgabe = 1 entspricht relativ; abhängig vom Operationsmodus
- Rückgabe = 2 entspricht absolut; abhängig vom Operationsmodus
- Rückgabe = 3 entspricht internen Referenzfahrt
- Rückgabe = 4 entspricht externen Referenzfahrt

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'p', siehe Befehle *2.6.6 Positionierart setzen (altes Schema)* und *2.6.7 Positionierart setzen (neues Schema)*.

SetNewSteps

Definition:

bool **SetNewSteps(***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', siehe Befehl *2.6.8 Verfahrweg einstellen*.

GetNewSteps

Definition:

int **GetNewSteps(***int operationNumber***)**

Diese Funktion liest die Anzahl der Schritte.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 's', siehe Befehl *2.6.8 Verfahrweg einstellen*.

SetNewStartFrequency

Definition:

bool **SetNewStartFrequency(***int startFreuqenz***)**

Diese Funktion setzt die Startfrequenz 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 'u', siehe Befehl *2.6.9 Minimalfrequenz einstellen*.

GetNewStartFrequency

Definition:

int **GetNewStartFrequency(***int operationNumber***)**

Diese Funktion liest die Startfrequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'u', siehe Befehl *2.6.9 Minimalfrequenz einstellen*.

SetNewMaxFrequency

Definition:

bool **SetNewMaxFrequency(***int maxFreuqenz***)**

Diese Funktion setzt die Zielfrequenz 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 'o', siehe Befehl *2.6.10 Maximalfrequenz einstellen*.

GetNewMaxFrequency

Definition:

int **GetNewMaxFrequency(***int operationNumber***)**

Diese Funktion liest die Zielfrequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'o', siehe Befehl *2.6.10 Maximalfrequenz einstellen*.

SetNewMaxFrequency2

Definition:

bool **SetNewMaxFrequency2(***int maxFreuqenz***)**

Diese Funktion setzt die maximale 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 'n', siehe Befehl *2.6.11 Maximalfrequenz 2 einstellen*.

GetNewMaxFrequency2

Definition:

int **GetNewMaxFrequency2(***int operationNumber***)**

Diese Funktion liest die maximale Frequenz des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'n', siehe Befehl *2.6.11 Maximalfrequenz 2 einstellen*.

NewSuppressResponse

Definition:

bool **NewSuppressResponse(***int suppress***)**

Diese Funktion aktiviert oder deaktiviert die Antwortunterdrückung beim Senden:

- *suppress* = 0 entspricht eingeschaltet
- *suppress* = 1 entspricht ausgeschaltet

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl '|', siehe Befehl *2.6.4 Aktuellen Satz auslesen*.

SetNewRamp

Definition:

bool **SetNewRamp(***int ramp***)**

Diese Funktion setzt die Rampe.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'b', siehe Befehl *2.6.12 Beschleunigungsrampe einstellen*.

GetNewRamp

Definition:

int **GetNewRamp(***int operationNumber***)**

Diese Funktion liest die Rampe.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'b', siehe Befehl *2.6.12 Beschleunigungsrampe einstellen*.

SetNewRotationMode

Definition:

bool **SetNewRotationMode(***int rotationMode***)**

Diese Funktion setzt den Drehgeberüberwachungsmodus:

- *rotationMode* = 0 entspricht ausgeschaltet
- *rotationMode* = 1 entspricht prüfen am Ende
- *rotationMode* = 2 entspricht prüfen dazwischen

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'U', siehe Befehl *2.5.9 Fehlerkorrekturmodus einstellen*.

GetNewRotationMode

Definition:

int **GetNewRotationMode()**

Diese Funktion liest den Drehgeberüberwachungsmodus:

- Rückgabe = 0 entspricht ausgeschaltet
- Rückgabe = 1 entspricht prüfen am Ende
- Rückgabe = 2 entspricht prüfen dazwischen

Die Funktion entspricht dem seriellen Befehl 'U', siehe Befehl *2.5.9 Fehlerkorrekturmodus einstellen*.

SetNewDirection

Definition:

bool **SetNewDirection(***int direction***)**

Diese Funktion setzt die Drehrichtung des Motors.

Mögliche Parameter sind:

- 0 entspricht der Drehrichtung links
- 1 entspricht der Drehrichtung 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', siehe Befehl *2.6.15 Drehrichtung einstellen*.

GetNewDirection

Definition:

int **GetNewDirection(***int operationNumber***)**

Diese Funktion liest die Drehrichtung des Motors.

Mögliche Rückgabewerte sind:

- 0 entspricht der Drehrichtung links
- 1 entspricht der Drehrichtung rechts

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'd', siehe Befehl *2.6.15 Drehrichtung einstellen*.

SetNewDirectionReverse

Definition:

bool **SetNewDirectionReverse(***bool directionReverse***)**

Diese Funktion setzt die Drehrichtungsumkehr 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 't', siehe Befehl *2.6.16 Richtungsumkehr einstellen*.

GetNewDirectionReverse

Definition:

bool **GetNewDirectionReverse(***int operationNumber***)**

Diese Funktion liest die Drehrichtungsumkehr des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 't', siehe Befehl *2.6.16 Richtungsumkehr einstellen*.

SetNewEncoderDirection

Definition:

bool **SetNewEncoderDirection(***bool encoderDirection***)**

Diese Funktion setzt, ob die Encoderdrehrichtung des Motors umgekehrt 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 'q', siehe Befehl *2.5.11 Encoderrichtung einstellen*.

GetNewEncoderDirection

Definition:

bool **GetNewEncoderDirection()**

Diese Funktion liest, ob die Encoderdrehrichtung des Motors umgekehrt wird.

Die Funktion entspricht dem seriellen Befehl 'q', siehe Befehl *2.5.11 Encoderrichtung einstellen*.

SetNewBreak

Definition:

bool **SetNewBreak(***double breakTime***)**

Diese Funktion setzt die Pausenzeit 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 'P', siehe Befehl *2.6.18 Satzpause einstellen*.

GetNewBreak

Definition:

int **GetNewBreak(***int operationNumber***)**

Diese Funktion liest die Pausenzeit des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'P', siehe Befehl *2.6.18 Satzpause einstellen*.

SetNewRepeat

Definition:

bool **SetNewRepeat(***int repeat***)**

Diese Funktion setzt die Wiederholungen 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 'W', siehe Befehl *2.6.17 Wiederholungen einstellen*.

GetNewRepeat

Definition:

int **GetNewRepeat(***int operationNumber***)**

Diese Funktion liest die Wiederholungen des Motors.

Der Parameter *operationNumber* ist dabei die Satznummer (Fahrprofil), aus dem der Positionstyp gelesen werden soll.

Die Funktion entspricht dem seriellen Befehl 'W', siehe Befehl *2.6.17 Wiederholungen einstellen*.

SetIO

Definition:

bool **SetIO(***int io***)**

Diese Funktion setzt den Status der Eingänge als integer Maske.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'Y', siehe Befehl *2.5.27 Ausgänge setzen*.

GetIO

Definition:

int **GetIO()**

Diese Funktion liest den Status der Eingänge als integer Maske.

Die Funktion entspricht dem seriellen Befehl 'Y', siehe Befehl *2.5.27 Ausgänge setzen*.

SetInputMask

Definition:

bool **SetInputMask(***int ioMask***)**

Diese Funktion setzt die Maske wie die Eingänge reagieren (fallende oder steigende Flanken) sollen.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'L', siehe Befehl *2.5.24 Eingänge maskieren und demaskieren*.

GetInputMask

Definition:

int **GetInputMask()**

Diese Funktion liest die Maske wie die Eingänge reagieren (fallende oder steigende Flanken).

Die Funktion entspricht dem seriellen Befehl 'L', siehe Befehl *2.5.24 Eingänge maskieren und demaskieren*.

SetInputMaskEdge

Definition:

bool **SetInputMaskEdge(***int ioMask***)**

Diese Funktion setzt die Maske wie die Eingänge reagieren (fallende oder steigende Flanken) sollen.

Über den Rückgabewert der Funktion kann geprüft werden, ob der Befehl korrekt von der Steuerung erkannt wurde.

Die Funktion entspricht dem seriellen Befehl 'h', siehe Befehl *2.5.25 Polarität der Ein- und Ausgänge umkehren*.

GetInputMaskEdge

Definition:

int **GetInputMaskEdge()**

Diese Funktion liest die Maske wie die Eingänge reagieren (fallende oder steigende Flanken).

Die Funktion entspricht dem seriellen Befehl 'h', siehe Befehl *2.5.25 Polarität der Ein- und Ausgänge umkehren*.

SetRampType

Definition:

bool **SetRampType(***int rampType***)**

Diese Funktion setzt den Rampentyp:

- *rampType* = 0 entspricht Trapezrampe
- *rampType* = 1 entspricht Sinusrampe

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', siehe Befehl *2.5.32 Rampe setzen*.

GetRampType

Definition:

int **GetRampType()**

Diese Funktion liest den Rampentyp:

- Rückgabe = 0 entspricht Trapezrampe
- Rückgabe = 1 entspricht Sinusrampe

Die Funktion entspricht dem seriellen Befehl 'ramp_mode', siehe Befehl *2.5.32 Rampe setzen*.

GetEncoderRotary

Definition:

int **GetEncoderRotary()**

Diese Funktion liest die Encoderposition.

Die Funktion entspricht dem seriellen Befehl 'I', siehe Befehl *2.5.16 Drehgeberposition auslesen*.

SetBraleTA

Definition:

bool **SetBraleTA(***uint32 brake***)**

Diese Funktion setzt den TA-Wert für die externe Bremse.

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', siehe Befehl *2.5.35 Wartezeit für Abschalten der Bremsspannung setzen*.

GetBrakeTA

Definition:

unit32 **GetBrakeTA()**

Diese Funktion liest den TA-Wert der externen Bremse.

Die Funktion entspricht dem seriellen Befehl 'brake_ta', siehe Befehl *2.5.35 Wartezeit für Abschalten der Bremsspannung setzen*.

SetBrakeTB

Definition:

bool **SetBrakeTB(***uint32 brake***)**

Diese Funktion setzt den TB-Wert für die externe Bremse.

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', siehe Befehl *2.5.36 Wartezeit für Motorbewegung setzen*.

GetBrakeTB

Definition:

unit32 **GetBrakeTB()**

Diese Funktion liest den TB-Wert der externen Bremse.

Die Funktion entspricht dem seriellen Befehl 'brake_tb', siehe Befehl *2.5.36 Wartezeit für Motorbewegung setzen*.

SetBrakeTC

Definition:

bool **SetBrakeTC(***uint32 brake***)**

Diese Funktion setzt den TC-Wert für die externe Bremse.

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', siehe Befehl *2.5.37 Wartezeit für Abschalten Motorstrom setzen*.

GetBrakeTC

Definition:

unit32 **GetBrakeTC()**

Diese Funktion liest den TC-Wert der externen Bremse.

Die Funktion entspricht dem seriellen Befehl 'brake_tc', siehe Befehl *2.5.37 Wartezeit für Abschalten Motorstrom setzen*.

SetRotencInc

Definition:

bool **SetRotencInc(**int *rotationInc***)**

Diese Funktion setzt die Drehgeberauflösung.

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 ', siehe Befehl *2.9.10 Anzahl der Inkremente einstellen*.

GetRotencInc

Definition:

int **GetRotencInc()**

Diese Funktion liest die Drehgeberauflösung.

Die Funktion entspricht dem seriellen Befehl 'CL_rotenc_inc', siehe Befehl *2.9.10 Anzahl der Inkremente einstellen*.

4.4 Programmbeispiele

Einleitung

Es folgt eine kurze Aufstellung an Beispielprogrammen. Die Programme selber liegen als Quellcode und in compilierter Form im Verzeichnis „Example“ der NanoPro-Software. Für die Beispiele gibt es auch eine Projektdatei für Visual Studio.

OfficeExample

Dieses Beispiel kann mittels VBA in Excel ausgeführt werden.

Um das Beispiel auszuführen, muss der Com-Port und die Übertragungsrate eingestellt werden.

Es gibt 2 Arten einen Motor Anzusteuern:

- eine Anzahl von Schritten fahren zu lassen
- einen vordefinierten Satz in der Steuerung ausführen zu lassen.

VBAExample

In diesem Beispiel gibt es zwei Modi:

- den Positionsmodus und
- den Drehzahlmodus.

Die Baudrate und der serielle Port muss auch eingestellt werden. Dieses Beispiel verwendet die CommandsPD4I.dll als Verweis im Projekt. Hier wird keine COM-Schnittstelle benutzt.

In diesem Beispiel wird von einer Motoradresse 1 ausgegangen.

VBACommandsPD4IExample-2Motor

Dieses Beispiel ist analog dem Beispiel VBAExample, jedoch werden hier zwei Motore angesteuert.

ManagedC++CommandsPD4I

Dieses ist genauso aufgebaut wie VBAExample, jedoch wird hier Managed C++ Code verwendet.

In diesem Beispiel wird von einer Motoradresse 1 ausgegangen.

UnManagedC++CommandsPD4I

Dieses ist genauso aufgebaut wie VBAExample, jedoch wird hier Unmanaged C++ Code verwendet.

In diesem Beispiel wird von einer Motoradresse 1 ausgegangen.

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z