

Programmier- handbuch

Motion Controller V3.0

Motion Control Systeme V3.0

```
35 :StartOfProgram
36 LET i = i + 1
37 LET a = 6
38 IF a < 5 THEN
39   LET c = 7
40 ELSE
41   LET c = 123456789
42   GOSUB GosubExample
43 END IF
44 IF (i % 10) = 0 THEN
45   LET j = j + 1
46 END IF
47 FOR k = (8 + 5) TO 15
48   LET l = k
49 NEXT k
50 REM Check Timer Abort
51 IF t = 1 THEN
52   SAVE i, j
53   DI_EVT
54   END
55 END IF
```

```
IF (i % 10) = 0 THEN
  LET j = j + 1
                                SAVE i, j
END IF
FOR k = (8 + 5) TO 15
  LET l = k
```

Impressum

Version:
4. Auflage, 28.02.2024

Copyright
by Dr. Fritz Faulhaber GmbH & Co. KG
Faulhaberstraße 1 · 71101 Schönaich

Alle Rechte, auch die der Übersetzung, vorbehalten.
Ohne vorherige ausdrückliche schriftliche Genehmigung
der Dr. Fritz Faulhaber GmbH & Co. KG darf kein Teil
dieser Beschreibung vervielfältigt, reproduziert, in einem
Informationssystem gespeichert oder verarbeitet oder in
anderer Form weiter übertragen werden.

Dieses Dokument wurde mit Sorgfalt erstellt.
Die Dr. Fritz Faulhaber GmbH & Co. KG übernimmt jedoch
für eventuelle Irrtümer in diesem Dokument und
deren Folgen keine Haftung. Ebenso wird keine Haftung
für direkte Schäden oder Folgeschäden übernommen,
die sich aus einem unsachgemäßen Gebrauch der Geräte
ergeben.

Bei der Anwendung der Geräte sind die einschlägigen
Vorschriften bezüglich Sicherheitstechnik und Funkentstörung
sowie die Vorgaben dieses Dokuments zu beachten.

Änderungen vorbehalten.

Die jeweils aktuelle Version dieses Dokuments
finden Sie auf der Internetseite von FAULHABER:
www.faulhaber.com

Inhalt

1	Zu diesem Dokument	4
1.1	Gültigkeit dieses Dokuments	4
1.2	Mitgeltende Dokumente	4
1.3	Abkürzungsverzeichnis	4
1.4	Symbole und Kennzeichnungen	5
2	Einführung	6
3	Eigenschaften der Programmiersprache	7
3.1	Befehlssatz	7
3.2	Operatoren und Sonderzeichen	12
3.3	Variablen	13
3.3.1	Globale Variablen mit fester Adresse	13
3.3.2	Allokierte globale und lokale Variablen	14
3.4	Hinweise zur Programmerstellung	14
4	Entwicklung von Ablaufprogrammen mit dem Motion Manager	15
4.1	Programm bearbeiten	17
4.1.1	Dateiverwaltung	17
4.1.2	Dateibearbeitung.....	18
4.2	Programm auf die Steuerung laden und ausführen	18
4.3	Programm debuggen	19
5	Steuerung von Ablaufprogrammen	20
5.1	Steuerung über die Schnittstelle	20
5.2	Fehlerbehandlung	21
5.3	Ablaufprogramm automatisch starten	22
5.4	Ablaufprogramme schützen	23
5.5	Datenaustausch mit Ablaufprogrammen	23
6	FAULHABER Motion Bibliothek	24
6.1	MotionParameters	24
6.2	MotionMacros	25
6.3	MotionFunctions	25
6.4	MyControlLib.bi	26
7	Beispielprogramme	28
7.1	Einfache zyklische Bewegung unter Nutzung der Bibliotheksfunktionen	28
7.2	Nutzung von Schrittketten zur Programmgestaltung	29
7.2.1	Referenzfahrt mit anschließendem automatischen Wechsel in den Positionierbetrieb	30
7.2.2	Referenzfahrt mit anschließendem automatischen Wechsel in den Positionierbetrieb und Start-Stop-Funktion	32
7.3	Ereignisbehandlung	35

Zu diesem Dokument

1 Zu diesem Dokument

1.1 Gültigkeit dieses Dokuments

Dieses Dokument beschreibt die Programmierung von Ablaufprogrammen für Steuerungen der Motion Controller und Motion Control Systeme Familie V3.0 mit dem FAULHABER Motion Manager 7.

Dieses Dokument richtet sich an Softwareentwickler mit Programmiererfahrung und an Projektingenieure der Antriebstechnik.

Alle Angaben in diesem Dokument beziehen sich auf Standardausführungen der Antriebe. Änderungen aufgrund kundenspezifischer Ausführungen dem Beilageblatt entnehmen.

1.2 Mitgeltende Dokumente

Für bestimmte Handlungsschritte bei der Inbetriebnahme und Bedienung der FAULHABER Produkte sind zusätzliche Informationen aus folgenden Handbüchern hilfreich:

Handbuch	Beschreibung
Motion Manager 7	Bedienungsanleitung zur FAULHABER Motion Manager PC Software
Schnellstartanleitung	Beschreibung der ersten Schritte zur Inbetriebnahme und Bedienung des FAULHABER Motion Controllers
Antriebsfunktionen	Beschreibung der Betriebsarten und Funktionen des Antriebs

Diese Handbücher können im PDF-Format von der Internetseite www.faulhaber.com heruntergeladen werden.

1.3 Abkürzungsverzeichnis

Abkürzung	Bedeutung
BASIC	Beginner's All-Purpose Symbolic Instruction Code
EEPROM	Electrically Erasable Programmable Read-Only Memory
Sxx	Datentyp Signed (negative und positive Zahlen) mit Bitgröße xx
Uxx	Datentyp Unsigned (positive Zahlen) mit Bitgröße xx

1.4 Symbole und Kennzeichnungen



HINWEIS!

Gefahr von Sachschäden.

- ▶ Maßnahme zur Vermeidung



Hinweise zum Verständnis oder zum Optimieren der Arbeitsabläufe

- ✓ Voraussetzung zu einer Handlungsaufforderung
- 1. Erster Schritt einer Handlungsaufforderung
 - ↳ Resultat eines Schritts
- 2. Zweiter Schritt einer Handlungsaufforderung
 - ↳ Resultat einer Handlung
- ▶ Einschrittige Handlungsaufforderung

2 Einführung

Ablaufprogramme können über den FAULHABER Motion Manager auf die Steuerung geladen werden und lassen sich direkt auf der Steuerung ausführen. Auf diese Weise ist z. B. ein Stand-Alone-Betrieb ohne übergeordnete Steuerung möglich oder eine teilautonome Abarbeitung kleinerer Programmsequenzen.

Die Programmierung von Ablaufprogrammen erfolgt in der Programmiersprache BASIC, mit FAULHABER-spezifischen Erweiterungen.

8 unabhängige Speicherbereiche für Anwenderprogramme stehen zur Verfügung. Ein Programm kann auch optional nach dem Hochlauf automatisch gestartet werden.

Eigenschaften der Programmiersprache

3 Eigenschaften der Programmiersprache

- BASIC-Interpreter mit FAULHABER-spezifischen Erweiterungen
- Funktionsaufrufe
- Keine Zeilennummern; Sprünge auf Sprungmarken
- Sprungmarken stehen am Anfang einer Zeile und beginnen mit einem Doppelpunkt
- Unterscheidung von Groß- und Kleinschreibung (Befehle immer in Großbuchstaben)
- Lese- und Schreibzugriff auf Objekte im Objektverzeichnis
- Möglichkeit, während des normalen Programmablaufs auf Ereignisse zu reagieren
- Timer für Zeitmessungen und Warteschleifen
- Arithmetische, Vergleichs- und Bit-Operatoren
- Sonderzeichen \$ für Werte in hexadezimaler Darstellung
- Maximale Länge aller Programme: 16 kByte
- 26 globale Standard 32-Bit Variablen a...z (dauerhaft speicherbar)
- 26 globale symbolische 32-Bit Variablen (frei benennbar)
- Lokale symbolische Variablen (frei benennbar)

3.1 Befehlssatz

Tab. 1: Standard BASIC-Befehlssatz

Befehl	Funktion	Beispiel
REM...	Kommentar. Steht am Anfang einer Zeile und gilt bis Zeilenende.	REM Kommentar
END	Programm beenden	END
GOTO...	Sprung nach angegebener Sprungmarke. Folgende Konstrukte dürfen nicht mit GOTO verlassen werden: <ul style="list-style-type: none"> ■ IF...THEN...ELSE...END IF ■ GOSUB...RETURN ■ FOR...TO...NEXT... In Unterfunktionen (FUNCTION...) werden GOTO-Sprünge nicht unterstützt.	GOTO Start
GOSUB... ... RETURN	In Unterprogramm an angegebener Sprungmarke springen. Nach Abarbeitung Rücksprung an Aufrufposition. Es ist kein GOTO-Sprung aus einem Unterprogramm möglich.	GOSUB Step1 ... :Step1 RETURN
FOR...TO... ... NEXT...	Programmierung einer Schleife. Es ist kein bedingter GOTO-Sprung aus einer FOR-Schleife möglich.	FOR i = 1 TO 10 ... NEXT i
DO ... LOOP UNTIL...	Schleife mit Prüfung der Schleifenbedingung am Ende der Schleife.	DO ... LOOP UNTIL a = 5

Eigenschaften der Programmiersprache

Befehl	Funktion	Beispiel
DO ... LOOP	Schleife ohne Prüfung einer Schleifenbedingung. Verlassen der Schleife mit EXIT.	DO ... LOOP
DO ... LOOP WHILE...	Schleife mit Prüfung der Schleifenbedingung am Ende der Schleife.	DO ... LOOP WHILE stop = 0
DO WHILE... ... LOOP	Schleife mit Prüfung der Schleifenbedingung am Beginn der Schleife.	DO WHILE speed > 500 ... LOOP
EXIT...	Sprung aus einer Schleife, ohne das Schleifenende erreicht zu haben. Das Schlüsselwort der Schleife muss mit angegeben werden.	EXIT FOR EXIT DO
IF...THEN... ELSEIF...THEN... ELSE... END IF	Programmierung einer Verzweigung. Es ist kein GOTO-Sprung in einer IF-Anweisung möglich.	IF a > 3 THEN b = 1 ELSE b = 0 END IF
IF...THEN GOTO... IF...THEN GOSUB... IF...THEN <Name>()	Bedingter Sprung bzw. Verzweigung in ein Unterprogramm. Angabe in einer Zeile ohne END IF. Folgende Konstrukte dürfen nicht mit GOTO verlassen werden: <ul style="list-style-type: none"> IF...THEN...ELSE...END IF GOSUB...RETURN FOR...TO...NEXT... 	IF z=1 THEN GOSUB Step1
IF...THEN EXIT FOR IF...THEN EXIT EVT	Sprung aus einer FOR-Schleife oder eine Event-Routine. Angabe in einer Zeile ohne ENDIF. Darf in folgenden Konstrukten nicht verwendet werden: <ul style="list-style-type: none"> IF...THEN...ELSE...END IF GOSUB...RETURN 	FOR a = 1 TO 5 IF x = 1 THEN EXIT FOR NEXT a
IF...THEN EXIT GOSUB	Sprung aus einem Unterprogramm. Angabe in einer Zeile ohne ENDIF. Darf in folgenden Konstrukten nicht verwendet werden: <ul style="list-style-type: none"> IF...THEN...ELSE...END IF GOSUB...RETURN 	:Sub1 IF x = 1 THEN EXIT GOSUB RETURN

Eigenschaften der Programmiersprache

Befehl	Funktion	Beispiel
FUNCTION... ... RETURN... END FUNCTION	<p>Definition einer Unterfunktion. Die Funktion wird im Programmtext über ihren Namen aufgerufen.</p> <p>Lokale Variablen können über das Schlüsselwort DIM allokiert werden.</p> <p>Parameter können als Zahlen oder mit Variablen übergeben werden.</p> <p>Mit dem Schlüsselwort RETURN kann die Unterfunktion ein numerisches Ergebnis zurückliefern.</p> <p>GOTO-Sprünge werden nicht unterstützt.</p> <p>Jede Unterfunktion muss mit dem Schlüsselwort END FUNCTION beendet werden.</p>	<p>Definition</p> <p>FUNCTION <Name> (Parameter1, Parameter2)</p> <p>DIM result</p> <p>...</p> <p>RETURN result</p> <p>END FUNCTION</p> <p>Aufruf</p> <p><Name> (1, 5)</p>
DIM...	<p>Allokiert eine Variable mit symbolischem Namen.</p> <ul style="list-style-type: none"> Wird DIM außerhalb einer Unterfunktion (FUNCTION) genutzt, wird eine globale Variable allokiert, die unter diesem Namen auch aus allen Unterfunktionen nutzbar ist. Globale symbolische Variablen können in der im Motion Manager integrierten Entwicklungsumgebung über ihren Namen gelesen und verändert werden. Wird DIM innerhalb einer Unterfunktion (FUNCTION) genutzt, wird eine lokale Variable allokiert, die nur innerhalb der Unterfunktion gültig ist. Lokale Variablen können in der im Motion Manager integrierten Entwicklungsumgebung gelesen und verändert werden, solange die Unterfunktion selbst aktiv ist (z. B. angehalten an einem Breakpoint oder im Einzelschritt). 	DIM Statusword

Tab. 2: FAULHABER Befehlserweiterung

Befehl	Funktion	Beispiel
SETOBJ...	<p>Beschreibung eines Objekts im Objektverzeichnis.</p> <p>Syntax: SETOBJ <Index>.<Subindex> = <variable oder value></p>	SETOBJ \$6083.\$00 = 500
GETOBJ...	<p>Lesen eines Objekts im Objektverzeichnis.</p> <p>Syntax: <variable> = GETOBJ <Index>.<Subindex></p>	a = GETOBJ \$6083.\$00
DEF_EVT_VAR...	<p>Definiert eine Variable, die den Wert der Event-Zustands-Bitmaske zum Zeitpunkt des Events zurückgibt.</p>	DEF_EVT_VAR e
EN_EVT...	<p>Aktivierung einer Event-Routine, die bei Änderung des über Objekt 0x2324.01 signalisierten Gerätezustands angesprungen wird (Ereignisbehandlung).</p> <p>Hinweis: Es kann nur eine Event-Routine aktiv sein.</p> <p>Syntax: EN_EVT <bit mask>,<event mark></p>	EN_EVT \$ffffff, EvHandler
DI_EVT	<p>Deaktivierung aller Events für die nebenläufige Verarbeitung.</p> <p>Syntax: DI_EVT</p>	DI_EVT
RET_EVT	<p>Rücksprung aus einer Event-Routine.</p> <p>Syntax: RET_EVT</p>	: EvHandler RET_EVT

Eigenschaften der Programmiersprache

Befehl	Funktion	Beispiel
SAVE...	Dauerhaftes Speichern einer oder mehrerer Variablen im EEPROM (komma-separierte Liste). Syntax: SAVE <variable1<,variable2,...>>	SAVE a, b, z
LOAD...	Laden einer oder mehrerer zuvor abgespeicherten Variablen aus dem EEPROM (komma-separierte Liste). Syntax: LOAD <variable1<,variable2,...>>	LOAD a, b, z
DEF_TIM_VAR...	Definiert eine Variable zur Verwendung als Timer. Syntax: DEF_TIM_VAR <variable>	DEF_TIM_VAR t
START_TIM...	Startet den Timer mit einem Wert in ms (oder stoppt den Timer wenn Wert = 0). Syntax: START_TIM <variable oder value> Wenn die angegebene Zeit abgelaufen ist, ist die Timer-Variable 1, sonst 0 (Timer läuft noch).	START_TIM 3000 IF t = 1 THEN ENDIF
DEF_CYC_VAR...	Definiert eine Variable zur Verwendung als 1 ms Zykluszähler. Hiermit lassen sich z. B. Zeitmessungen durchführen. Der Zähler läuft max. 24 Tage und bleibt dann bei -1. Syntax: DEF_CYC_VAR <variable>	DEF_CYC_VAR z
START_CYC	Startet den Zykluszähler mit dem Wert 0. Syntax: START_CYC	START_CYC
STOP_CYC	Stoppt den Zykluszähler. Der aktuelle Zählerstand bleibt in der hierfür definierten Variable enthalten und kann weiterverarbeitet werden. Syntax: STOP_CYC	STOP_CYC
DELAY...	Wartezeit in ms. Das Programm wird während der Wartezeit nicht weiter bearbeitet. Syntax: DELAY <variable oder value>	DELAY 200
#DEFINE...	Weist einer symbolischen Bezeichnung einen Wert zu. Syntax: #DEFINE <symbol> <value> Über dieses Schlüsselwort können auch komplexere Makros definiert werden, die im Programmtext über den Makronamen verwendet werden können. Syntax: #DEFINE <macro> <expression> Wird dem Makronamen ein "MC." vorangestellt, steht das Makro in der Autovervollständigung des Motion Manager Editors zur Verfügung. Die Liste mit verfügbaren Makros erscheint nach Eingabe von "MC."	#DEFINE MaxSpeed 2000 #DEFINE MC.IsTargetReached ((GETOBJ \$6041.00 & \$400) = \$400) If MC.IsTargetReached THEN...

Eigenschaften der Programmiersprache

Befehl	Funktion	Beispiel
#INCLUDE...	<p>Weist die Entwicklungsumgebung an, eine weitere Datei (Basic-Include-Datei *.bi) in das Programm einzubinden.</p> <p>Damit können Sätze an vordefinierten symbolischen Namen oder Funktionsbibliotheken eingebunden und wiederverwendet werden.</p> <p>Ohne Pfad-Angabe werden Include-Dateien entweder im Motion Manager ProgramData-Verzeichnis oder im selben Ordner wie die zugehörige .bas-Datei gesucht. Include-Dateien aus anderen Ordnern können über eine absolute Pfad-Angabe referenziert werden.</p> <p>Syntax: #INCLUDE <[path] filename></p>	#INCLUDE "MotionParameters.bi"
EVENT...	<p>Meldet einen Event an den Eventbroker (siehe Handbuch Antriebsfunktionen).</p> <p>Darüber kann entweder das Eventflag im Gerätestatuswort 0x2324.01 gesetzt oder eine Aufzeichnung des Trace-Recorders gestartet werden.</p> <p>Als Parameter wird ein beliebiger 16 Bit Wert als Eventcode übergeben, der den Event kennzeichnet.</p> <p>Ein Eventcode 0 setzt den Event zurück.</p> <p>Events können nur erneut gesetzt werden, wenn zwischenzeitlich der Event mit Code 0 zurückgenommen worden war oder über verschiedene Eventcodes.</p> <p>Der gesetzte Eventcode kann nicht über das Objektverzeichnis zurückgelesen werden.</p>	<p>EVENT 1</p> <p>EVENT 0</p>
ERROR...	<p>Dieser Befehl kann genutzt werden, um im FAULHABER Fehlerwort 0x2320.00 das Bit 8 zu setzen.</p> <p>Als Parameter wird ein 16 Bit Fehlercode übergeben, der im Objekt 0x2322.01 ausgelesen werden kann.</p> <p>Die Programmbearbeitung wird nach Signalisierung des Fehlers im FAULHABER Fehlerwort normal fortgesetzt.</p> <p>Über das Schlüsselwort ERROR kann damit eine unerwartete Situation signalisiert werden. Je nach Einstellung im Fehlerhandling (Masken unter 0x2321.xx) kann der Antrieb dabei auch automatisch stillgesetzt werden.</p> <p>Standardmäßig wird über die Fehlerüberwachungseinheit eine EMCY Botschaft mit dem Fehlercode 0xFF30 erzeugt.</p> <p>Ein Fehlercode 0 setzt den Fehlerzustand wieder zurück.</p>	ERROR \$1234
RESET	<p>Dieser Befehl kann genutzt werden, um aus einem Ablaufprogramm heraus den Motion Controller komplett zu reinitialisieren. Dabei wird unmittelbar jede Regelung und Kommunikation abgebrochen. Das Geräte startet neu wie nach einem Power-Cycle.</p>	RESET

Eigenschaften der Programmiersprache

3.2 Operatoren und Sonderzeichen

Arithmetische Operatoren	
Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Modulo (Divisionsrest)	%
Logische Operatoren	
Und-Verknüpfung	AND
Oder-Verknüpfung	OR
Invertierung	NOT
Vergleichsoperatoren	
Größer als	>
Kleiner als	<
Gleich	=
Ungleich	<>
Größer gleich	>=
Kleiner gleich	<=
Bit-Operatoren	
Bitweises UND	&
Bitweises ODER	
Bitweises EXCLUSIV-ODER (XOR)	^
Bitweise Invertierung	~
Bitweises Schieben einer Variablen nach links	<<
Bitweises Schieben einer Variablen nach rechts	>>
Zuweisungsoperator	
Zuweisungsoperator	=
Sonderzeichen	Bedeutung
()	Verwendung für mathematische Operatoren
,	Verwendung in EN_EVT und SAVE/LOAD
.	Trennzeichen in SETOBJ /GETOBJ
\$	Hexadezimalzahlen
:	Sprungmarke, steht am Anfang der Zeile

Eigenschaften der Programmiersprache

3.3 Variablen

In Ablaufprogrammen stehen 3 Variablentypen zur Verfügung:

- Globale Variablen mit fester Adresse (siehe Kap. 3.3.1, S. 13)
- Allokierte globale Variablen (siehe Kap. 3.3.2, S. 14)
- Allokierte lokale Variablen (siehe Kap. 3.3.2, S. 14)

3.3.1 Globale Variablen mit fester Adresse

Variablen, die über die Buchstaben a...z referenziert werden, stehen global in allen Programmbereichen zur Verfügung. Jede Änderung ist global sichtbar.

Variablen mit fester Adresse haben folgende Eigenschaften:

- Sie können über die Befehle LOAD und SAVE im EEPROM gespeichert und geladen werden.
- Sie können als Timervariable (DEF_TIM_VAR) oder Eventvariable (DEF_EVT_VAR) verwendet werden (trifft nur auf Variablen mit fester Adresse zu).
- Sie können über das Objekt 0x3005 direkt über die Kommunikationsschnittstelle angesprochen werden.
- Sie können auf PDOs gemappt werden.
- Ihnen kann über #DEFINE ein symbolischer Namen zugewiesen werden. Diese symbolischen Namen können überall im Programmtext verwendet werden.

Beispiel:

```
#DEFINE RefSpeed a
#DEFINE MC.TagetVelocity SETOBJ $60FF.00
```

```
RefSpeed = 2000
MC.TagetVelocity = RefSpeed
...
SAVE RefSpeed
```

Eigenschaften der Programmiersprache

3.3.2 Allokierte globale und lokale Variablen

Mit dem Schlüsselwort `DIM` <Variablenname> kann eine Variable mit symbolischem Namen angelegt werden.

Beispiel:

```
DIM Statusword
```

- **Globale Variablen:**

Wenn `DIM` außerhalb einer Unterfunktion verwendet wird, wird dadurch eine Variable aus einem Satz von maximal 26 globalen Variablen allokiert. Die Variable kann von allen Funktionen genutzt werden. Der symbolische Name kann wie eine Zahl in allen Ausdrücken verwendet werden.

- **Lokale Variablen:**

Wenn `DIM` innerhalb einer Unterfunktion verwendet wird, wird dadurch eine nur innerhalb dieser Unterfunktion gültige lokale Variable allokiert. Übergabeparameter von Unterfunktionen werden wie lokale Variablen gehandhabt. Die maximale Anzahl von lokalen Variablen liegt bei 26 pro Unterfunktion.

Allokierte Variablen können nicht als Timervariable (`DEF_TIM_VAR`) oder Eventvariable (`DEF_EVT_VAR`) verwendet werden. Sie können nicht direkt über `LOAD` oder `SAVE` aus dem EEPROM initialisiert bzw. dort hin gespeichert werden.

Im Motion Manager kann auf allokierte Variablen über deren symbolische Namen zugegriffen werden. Der Zugriff auf lokale Variablen ist nur innerhalb der Funktion möglich, in der die Variable definiert ist.

3.4 Hinweise zur Programmerstellung

- Ein Ablaufprogramm sollte grundsätzlich als Schrittkette mit einer Hauptschleife über den gesamten Ausführungscode aufgebaut sein (siehe Kap. 7.2, S. 29). Warteschleifen mit bedingten Sprüngen auf bestimmte Ereignisse sind nicht möglich.
- Die Erstellung und Bearbeitung von Ablaufprogrammen erfolgt mit dem FAULHABER Motion Manager.
- Vor dem Download eines Ablaufprogramms auf die Steuerung führt der FAULHABER Motion Manager eine Vorverarbeitung durch, um z. B. die Adressen von Sprungmarken und den notwendigen Speicherbereich zu ermitteln.
- Mit dem FAULHABER Motion Manager können nicht nur Ablaufprogramme erstellt, bearbeitet und auf die Steuerung geladen werden, sondern auch Programmfehler gesucht und beseitigt werden (Debug-Möglichkeiten).
- Eigene Funktionsbibliotheken können entwickelt und als Basic-Include-Dateien (*.bi) in ein Ablaufprogramm eingebunden werden.
Es wird empfohlen, diese Bibliotheken zunächst als BAS-Datei zu erstellen und erst nach Fertigstellung von Entwicklung und Test die Funktionen in eine Include-Datei auszulagern. Der Motion Manager stellt keine Debug-Möglichkeiten für Include-Dateien zur Verfügung.

Entwicklung von Ablaufprogrammen mit dem Motion Manager

4 Entwicklung von Ablaufprogrammen mit dem Motion Manager

Der FAULHABER Motion Manager 7 bietet im Bereich **Programmierung** eine integrierte Entwicklungsumgebung für Ablaufprogramme. Die Entwicklungsumgebung hat folgende Eigenschaften:

- Syntaxhervorhebung
- Laden, Anzeigen und Bearbeiten von Ablaufprogrammen aus dem Gerätespeicher und aus dem PC-Speicher
- Starten einzelner Ablaufprogramme
- Stoppen des aktiven Ablaufprogramms
- Anhalten des aktiven Ablaufprogramms
- Einzelschrittausführung
- Definition eines Haltepunkts
- Anzeige des aktuellen Programmstatus und der aktuellen Programmzeile
- Beobachten und Ändern von Variablen-Inhalten
- Ausleseschutz über Zugangsschlüssel
- Code-Bausteine zur Verwendung in eigenen Programmen
- Autovervollständigung
- Automatische Syntaxprüfung im Hintergrund
- Festlegung eines Auto-Start-Programms

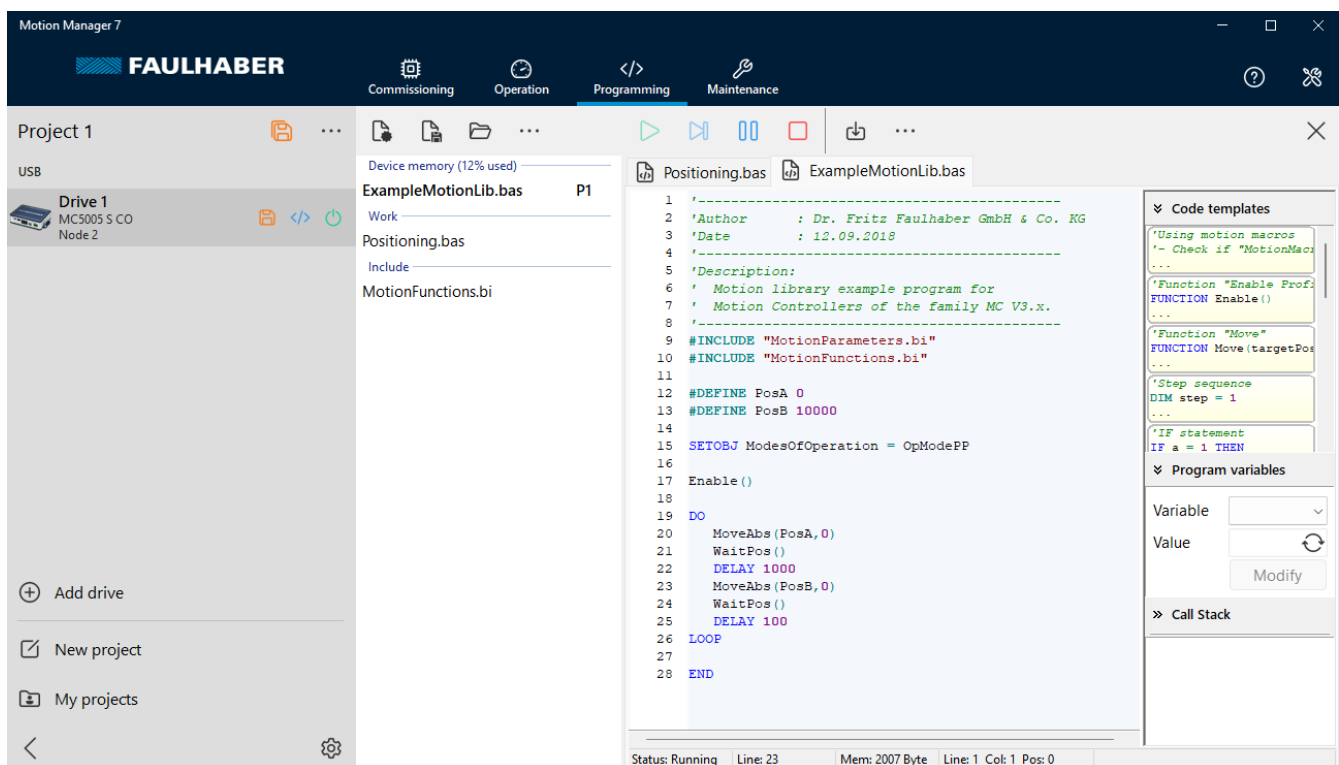







Abb. 1: Motion Manager 7 - Programmierung

Entwicklung von Ablaufprogrammen mit dem Motion Manager

Tab. 3: Schaltflächen-Funktionen des Editors

Schaltfläche	Benennung	Funktion
	Run	Ablaufprogramm auf die Steuerung hinunterladen und ausführen.
	Step	Ablaufprogramm im Einzelschritt ausführen oder fortsetzen.
	Halt	Ablaufprogramm anhalten.
	Stop	Ablaufprogramm beenden.
	Download	Ablaufprogramm auf die Steuerung hinunterladen und speichern, ohne es auszuführen.

Entwicklung von Ablaufprogrammen mit dem Motion Manager

4.1 Programm bearbeiten

4.1.1 Dateiverwaltung

- Ein neues Programm wird über die Schaltfläche **Neue Datei** und der Auswahl **Ablaufprogramm-Datei (*.bas)** angelegt und in einem neuen Reiter mit vorbereitetem Dateikopf angezeigt.
- Ein bereits existierendes Programm wird über die Schaltfläche **Datei öffnen** geladen und in einem neuen Reiter angezeigt.
- Von FAULHABER vorbereitete Beispiel-Programme können über das Erweiterungsmenü (...) der Dateiverwaltung und Auswahl von **Beispiele** geladen werden.
- Werden neu geladene oder angelegte Dateien gespeichert, ausgeführt oder auf die Steuerung hinunter geladen, wird ein Dateiname abgefragt, unter dem die Datei dann im Projektordner für den jeweiligen Antrieb abgelegt wird. Die Datei erscheint darauf in der Dateiverwaltung im linken Bereich der Programmierungsumgebung.
 - Über das Kontextmenü der Dateiverwaltung kann unter anderem die Programmnummer-Zuordnung geändert oder das Programm aus dem Gerätespeicher gelöscht werden.
 - Über die Speicher-Schaltfläche im Projektbereich wird ein heruntergeladenes Ablaufprogramm fest im Gerät und im Projekt-Ordner gespeichert und einer Programmnummer zugeordnet.
 - Im Gerät gespeicherte Ablaufprogramme werden beim Hinzufügen eines Antriebs zu einem Projekt automatisch ausgelesen. Falls im Projektordner eine zugehörige Quelldatei gefunden wird, wird diese in den Projekt-Ordner des Antriebs kopiert und in der Dateiverwaltung angezeigt. Andernfalls wird eine .out-Datei angezeigt, die den Programminhalt enthält, wie er im Gerät gespeichert ist.

Die Bearbeitung einer .out-Datei ist nicht möglich. Es kann aber eine Kopie mit der Endung .bas erstellt werden. Damit wird das Programm, soweit möglich, für eine Nachbearbeitung zurückübersetzt. Nachträglich kann der Gerätespeicher über das Erweiterungsmenü der Dateiverwaltung ausgelesen werden.

Die .out-Dateien können verwendet werden, um Ablaufprogramme unverändert auf andere Antriebe zu übertragen.

Entwicklung von Ablaufprogrammen mit dem Motion Manager

4.1.2 Dateibearbeitung

- Kommentarzeilen, die mit `'` beginnen, werden nur zu Dokumentationszwecken in der Datei gespeichert und nicht auf die Steuerung hinuntergeladen.
- Der Inhalt von Dateien, die über `#INCLUDE` referenziert sind, wird vor der Übertragung auf die Steuerung mit dem eigentlichen Programmcode zusammengefügt.
- Im Programm angegebene Makrobefehle und symbolische Variablen werden vor der Übertragung auf die Steuerung durch die mit `#DEFINE` hinterlegten Ausdrücke und die mit `DIM` zugeteilten internen Variablen ersetzt.
- Zur Anwenderunterstützung gibt es auf der rechten Seite des Editors Code-Vorlagen, die mit der Maus in das eigene Programm gezogen und angepasst werden können.
- Ist die automatische Syntaxprüfung aktiviert (Erweiterungsmenü des Datei-Reiters), werden erkannte Syntaxfehler rot unterstrichen angezeigt.
- Mit der rechten Maustaste kann ein Kontextmenü für weitere Editor-Funktionen geöffnet werden. Auf diese Weise kann z. B. eine im Programmcode angegebene Include-Datei geöffnet oder an die Stelle einer Funktions- oder Makro-Definition gesprungen werden.

4.2 Programm auf die Steuerung laden und ausführen

- Über die Schaltfläche **Run** wird ein fertiggestelltes Programm auf die Steuerung hinuntergeladen und anschließend sofort ausgeführt.
- Die Dateiverwaltung zeigt verschiedene Zustände des Programms an:

Anzeige	Beschreibung
*	Die Datei wurde übertragen, aber noch nicht fest im Gerät gespeichert (neues Programm im Arbeitsspeicher).
P1	Die Datei ist auf Speicherplatz 1 abgelegt.
P1*	Das Programm wurde neu übertragen, ist aber noch nicht neu gespeichert (geändertes Programm im Arbeitsspeicher)
Dateiname in fettem Schriftstil	Aktives Programm im Gerät.

- Nach dem Start der Ausführung schaltet der Editor-Bereich in den Debug-Modus (geänderte Hintergrundfarbe). In diesem Modus ist keine Programmbearbeitung möglich.
- Bei Fehlern während der Ausführung bricht die Programmausführung ab und die zuletzt ausgeführte Zeile wird rot hervorgehoben.
- Um wieder in den Programmbearbeitungs-Modus zu gelangen, muss die Programmausführung über die Schaltfläche **Stop** beendet werden.

Entwicklung von Ablaufprogrammen mit dem Motion Manager

4.3 Programm debuggen

Folgende Debug-Möglichkeiten zur Fehlersuche in Ablaufprogrammen stehen zur Verfügung:

- Programm an aktueller Ausführungsposition anhalten (Schaltfläche **Halt**):
 - Die aktive Zeile wird im Editor markiert. Befindet sich das Programm gerade bei der Abarbeitung einer Funktion in einer eingebundenen Datei, wird die aufrufende Stelle in der Hauptdatei markiert.
 - Der Edit-Bereich bleibt inaktiv.
 - Nach **Halt** kann das Programm entweder über **Run** fortgesetzt oder über **Step** im Einzelschritt weitergeführt werden. Mit **Stop** gelangt man wieder in den Programmbearbeitungsmodus.
- Programm im Einzelschritt weiterführen (Schaltfläche **Step**):
 - Nur die nachfolgende Programmzeile wird ausgeführt.
 - Die neue aktive Zeile wird im Editor markiert.
 - Der Edit-Bereich bleibt inaktiv.
 - Nach **Step** kann das Programm entweder über **Run** fortgesetzt oder über **Step** im Einzelschritt weitergeführt werden. Mit **Stop** gelangt man wieder in den Programmbearbeitungsmodus.
- Programm an Haltepunkt anhalten:
 - Mit einem Mausklick auf die gewünschte Zeilennummer am linken Fensterrand kann ein Haltepunkt angegeben werden.
 - Die Programmausführung wird bei Erreichen dieser Zeile angehalten und kann über **Run** oder **Step** fortgesetzt werden. Mit **Stop** gelangt man wieder in den Programmbearbeitungsmodus.
 - Mit einem Mausklick auf den Haltepunkt am linken Fensterrand wird der Haltepunkt gelöscht. Dies ist die Voraussetzung für das Setzen eines neuen Haltepunktes.
 - Ein Haltepunkt kann vor dem Start eines Programms und auch während der Programmausführung gesetzt werden.
- Untersuchen und Ändern von Variablen-Inhalten:
 - In den Programmier-Tools am rechten Editor-Rand gibt es den Bereich **Programmvariablen**. Aus der Variablenliste können jederzeit die global definierten symbolischen Variablen und die Standardvariablen a...z ausgewählt, angezeigt und verändert werden. Auf lokale Variablen kann nur zugegriffen werden, wenn das Programm in der jeweiligen Funktion angehalten hat.
- Untersuchen des Aufruf-Stacks:
 - In den Programmier-Tools am rechten Editor-Rand gibt es den Bereich **Aufruf-Stack**. Dort werden neben der aktuellen Zeile eines angehaltenen Programms auch die Nummern der Zeilen in der Aufruffolge von Funktionen angezeigt.

Steuerung von Ablaufprogrammen

5 Steuerung von Ablaufprogrammen

Ein gespeichertes Ablaufprogramm kann von einem Host-Rechner über die Schnittstelle oder automatisch beim Hochfahren der Steuerung gestartet werden.

5.1 Steuerung über die Schnittstelle

Durch Zugriff auf das Objekt 0x3001 kann die Ausführung von Ablaufprogrammen von einem übergeordneten Rechner gesteuert und überwacht werden.

Tab. 4: Current Control Parameter Set

Index	Subindex	Name	Typ	Attr.	Bedeutung
0x3001	0	Number of Entries	U8	ro	Anzahl Objekteinträge
	1	Program Control	U8	rw	Steuerung des über 0x3001.02 oder 0x3002.00 aktivierten Ablaufprogramms: <ul style="list-style-type: none"> 1: Aktiviertes Programm aus EEPROM laden (Load) 2: Geladenes Programm starten oder fortsetzen (Run) 3: Einzelne Programmzeile ausführen (Step) 4: Laufendes Programm anhalten (Break) 5: Laufendes Programm beenden (Terminate)
	2	Program Number	U8	rw	Ablaufprogramm an Programmnummer aktivieren
	3	Actual Position	U16	ro	Adresse der aktuell auszuführenden Zeile
	4	Actual Program State	U8	ro	Aktueller Programmstatus: <ul style="list-style-type: none"> 0: Keine Aktion (Idle) 1: Programm wird gerade aus dem EEPROM geladen (Reading) 2: Programm wird gerade ins EEPROM gespeichert (Saving) 3: Programm wird gerade gelöscht (Deleting) 4: Programm wird gerade ausgeführt (Running) 5: Programm angehalten (Halted)
	8	Error State	U8	ro	Fehlerstatus: <ul style="list-style-type: none"> 0: Kein Fehler (No Error) 1: Syntax Fehler (Parsing Error) 2: Fehler bei Zugriff auf EEPROM (EEPROM Access Error)
	9	Error Code	U16	ro	Detaillierter Fehlercode im Falle eines Syntax Fehlers (siehe Kap. 5.2, S. 21)

i Vor dem Laden eines neuen Programms muss ein bereits laufendes Programm beendet werden.

Pseudocode:

- Wenn 0x3001.04 = 4 (Running) oder 0x3001.04 = 5 (Halted), dann 0x3001.01 = 5 (Terminate)
- Warten, bis 0x3001.04 = 0 (Idle)

Steuerung von Ablaufprogrammen

Beispiel für das Laden und Ausführen eines Ablaufprogramms in Programmnummer 1:

1. Programm 1 wählen:
 - 0x3001.02 = 1 (P1)
 2. Programm laden:
 - 0x3001.01 = 1 (Load)
 - Warten, bis 0x3001.04 = 0 (nicht mehr Reading).
 3. Programm ausführen:
 - 0x3001.01 = 2 (Run)
- 🔗 Programm 1 ist geladen und wird ausgeführt.

5.2 Fehlerbehandlung

Fehler, die während der Programmausführung auftreten, werden als Fehlercode im Objekt 0x3001.09 zurückgemeldet.

Wenn ein Fehler auftritt, werden folgende Aktionen automatisch ausgelöst:

- Der Programmablauf wird beendet.
- Der detaillierte Fehlercode wird im Objekt 0x3001.09 zurückgemeldet (siehe Tab. 5).
- Im FAULHABER Fehlerwort 0x2320.00 wird ein CalculationError (Bit 12) gesetzt.

Beim Start eines neuen Programms werden der Fehlercode im Objekt 0x3001.09 und CalculationError im Fehlerwort 0x2320.00 zurückgesetzt.

i Über das Fehlerwort 0x2320.00 können im Fehlerfall weitere Aktionen automatisch ausgelöst werden. Zum Beispiel kann der Antrieb automatisch abgeschaltet werden.

Tab. 5: Fehlercodes zu 0x3001.09

Code	Fehler	Bedeutung	Behebung
0	No error	Kein Fehler	–
1	Generic error	Allgemeiner Fehler	Syntax prüfen.
3	Unexpected token	Das Zeichen war an der Stelle nicht erwartet worden.	Syntax prüfen.
4	Missing return value	Eine Funktion wurde ohne RETURN beendet, obwohl ein Rückgabewert erwartet worden war.	RETURN-Anweisung ergänzen.
6	End of parsing memory	Durch verschachtelte Funktionsaufrufe wird zu viel Speicher beansprucht.	<ul style="list-style-type: none"> ▪ Verschachtelungstiefe reduzieren. ▪ Kürzere aufrufende Zeilen verwenden.
7	Too many variables	Durch die Kette von aufgerufenen Funktionen werden zu viele lokale Variablen genutzt.	Programmkomplexität reduzieren
8	Illegal variable type	Für folgende Funktionen können keine über DIM angelegten Variablen verwendet werden: <ul style="list-style-type: none"> ▪ Sonderfunktionen des Timers ▪ Sonderfunktionen der Event-Bearbeitung ▪ SAVE ▪ LOAD 	Manuell angelegte Variablen a...z verwenden.

Steuerung von Ablaufprogrammen

Code	Fehler	Bedeutung	Behebung
9	Function stack overflow	Die Verschachtelungstiefe der Funktionsaufrufe ist zu hoch. Maximal werden 15 Aufrufebenen unterstützt.	Verschachtelungstiefe reduzieren.
10	Nested condition overflow	Die Verschachtelungstiefe von Bedingungen wie IF ist zu hoch. Maximal wird eine Tiefe von 15 Ebenen unterstützt.	Verschachtelungstiefe reduzieren.
11	Division by 0		
12	Event while in Event	Ein Event wurde ausgelöst, während die Event-Bearbeitung noch aktiv war. Der Event kann nicht bearbeitet werden.	Triggerhäufigkeit reduzieren.
13	RET_EVT while not in event	RET wurde außerhalb eines Events verwendet.	RET nur als Sprung aus Events heraus verwenden.
14	ELSE or END IF without IF	Der Befehl ELSE oder END IF wurde ohne das zugehörige IF erkannt.	Syntax prüfen.
15	Wrong variable used in NEXT token	Die für den Aufruf von NEXT genutzte Variable entspricht nicht der aus dem FOR-Aufruf.	Syntax prüfen.
16	GOTO not supported here	GOTO wird innerhalb von Funktionen nicht unterstützt.	GOTO nicht in Funktionen verwenden.
17	Illegal object	Das für GETOBJ oder SETOBJ verwendete Objekt existiert nicht oder lässt den Zugriff nicht zu.	Syntax prüfen.
18	RETURN while not in SUB or FUNCTION	Ein RETURN wurde erkannt, ohne vorher in eine Unterfunktion gesprungen zu sein.	Syntax prüfen.

5.3 Ablaufprogramm automatisch starten

Über Objekt 0x3002.00 kann eine Programmnummer angegeben werden, die nach dem Hochfahren der Steuerung automatisch geladen und ausgeführt wird.

Tab. 6: Autostart Program Number

Index	Subindex	Name	Typ	Attr.	Bedeutung
0x3002	0	Autostart Program Number	U8	rw	Programmnummer eines Ablaufprogramms, das automatisch gestartet werden soll.



Diese Funktion steht auch im Kontextmenü der Dateiverwaltung im Motion Manager zur Verfügung (**Automatisch ausführen**).

Steuerung von Ablaufprogrammen

5.4 Ablaufprogramme schützen

Mit dem Objekt 0x3003.00 kann ein 32-Bit Schlüssel gesetzt werden, der die im Controller gespeicherten Programme gegen unbefugten Zugriff schützt.

Wenn ein Code 0x3003.00 $\neq 0$ gesetzt wurde und danach die Parameter des Controllers gespeichert wurden, können die gespeicherten Ablaufprogramme danach nur noch ausgelesen werden, wenn der Code erneut eingegeben wird.

Tab. 7: Access Code

Index	Subindex	Name	Typ	Attr.	Bedeutung
0x3003	0	Access Code	U32	ro	32-Bit Schlüssel zum Schutz von im Controller gespeicherten Programmen gegen unbefugten Zugriff.



Diese Funktion steht auch im Erweiterungsmenü der Dateiverwaltung im Motion Manager zur Verfügung (**Gerätespeicher sperren**).

5.5 Datenaustausch mit Ablaufprogrammen

Datenaustausch über Objekt 0x3004

Die Programmvariablen a bis z können auch zum Datenaustausch zwischen Ablaufprogramm und übergeordnetem Rechner verwendet werden. Über Objekt 0x3004.01 kann eine Variable ausgewählt und über Objekt 0x3004.02 deren Wert gelesen oder beschrieben werden.

Tab. 8: Variable Access

Index	Subindex	Name	Typ	Attr.	Bedeutung
0x3004	0	Number of entries	U8	ro	Anzahl Objekteinträge
	1	Variable index	U8	rw	Variablen-Index <ul style="list-style-type: none"> 0...25 = Standard-Variablen a...z 32 + (0...25) = Globale interne Variablen 128 + (0...25) = Lokale interne Variablen
	2	Variable value	S32	rw	Variablen-Wert

Datenaustausch über Objekt 0x3005

Über die Subindizes des Objekts 0x3005 kann auf einzelne Standard-Variablen direkt zugegriffen werden. Die Variablen können damit z. B. aufgezeichnet oder in ein PDO gemappt werden. Davon ausgenommen sind Variablen, die für Event-Handler, Timer oder Zähler verwendet werden.

Tab. 9: Debug User Program

Index	Subindex	Name	Typ	Attr.	Bedeutung
0x3005	0	Number of entries	U8	ro	Anzahl Objekteinträge
	1...26	User prog variable a...z	S32	rw	Werte der Variablen a...z

6 FAULHABER Motion Bibliothek

Mit dem FAULHABER Motion Manager 7 werden einige unterstützende Dateien ausgeliefert:

Systemdateien

Die Systemdateien sind im Installationsbereich des Motion Managers abgelegt und werden in neue Dateien automatisch eingebunden. Sie können nicht geändert werden.

Über die rechte Maustaste auf dem Dateinamen in der `#INCLUDE` Zeile des Programmcodes kann die Datei geöffnet werden.

Datei	Beschreibung
MotionParameters.bi	Vordefinierte Zuordnung von symbolischen Parameternamen mit den Werten, wie z. B. <code>#DEFINE Statusword \$6041.00</code> . Siehe Kap. 6.1, S. 24.
MotionMacros.bi	Vordefinierte Makros zum direkt Zugriff auf Parameter des Motion Controllers z.B. für Antriebssteuerung und Statusprüfung. Siehe Kap. 6.2, S. 25.

Beispieldateien

Die Beispieldateien sind unter **Öffentliche Dokumente** im Ordner `\Users\Public\Documents\Faulhaber\Motion Manager 7\Examples\MC Basic` abgelegt.

Über das Erweiterungs Menü der Dateiverwaltung im Motion Manager können die Dateien geladen und in den Projekt-Ordner des Antriebs zur Weiterverarbeitung gespeichert werden.

Datei	Beschreibung
MotionFunctions.bi	Vordefinierte Funktionen für typische Antriebsaufgaben der Motion Controller. Siehe Kap. 6.3, S. 25.
MyControlLib.bi	Erweiterung der MotionFunctions.bi für allgemeine Zwecke. Siehe Kap. 6.4, S. 26.

6.1 MotionParameters

Die Datei *MotionParameters.bi* enthält symbolische Definitionen für typische Parameter, wie z. B.:

```
#DEFINE Statusword $6041.00
```

Im Programm kann der Zugriff auf die Parameter dann über die symbolischen Namen erfolgen.

Beispiel:

```
DIM DeviceStatus
DeviceStatus = GETOBJ Statusword
```


FAULHABER Motion Bibliothek

6.2 MotionMacros

Die Datei *MotionMacros.bi* enthält vordefinierte Zugriffe auf Parameter des Antriebssystems.

Die Makros beginnen immer mit dem Kenner `MC`.

Beispiel:

```
#DEFINE MC.GetStatusword GETOBJ $6041.00  
...
```

```
DIM DeviceStatus  
DeviceStatus = MC.GetStatusword
```



Die Autovervollständigung des Motion Managers zeigt nach Eingabe von "MC. " (oder **Strg** + Leertaste nach dem Punkt) eine Liste verfügbarer Makro-Funktionen.

6.3 MotionFunctions

In der Datei *MotionFunctions.bi* befindet sich ein Satz von vorbereiteten Funktionen, die zum Aufbau eigener Abläufe verwendet werden können.

FUNCTION Enable ()

Die Funktion `Enable` versucht die Antriebszustandsmaschine in den Zustand *Operation Enabled* zu bringen.

Erst wenn der Zustand *Operation Enabled* erreicht ist, kehrt die Funktion zum aufrufenden Kontext zurück. Wenn der Zustand nicht erreicht werden kann, z. B. weil noch ein blockierender Fehler ansteht, kehrt die Funktion nicht zurück. Es handelt sich daher um einen blockierenden Aufruf.

FUNCTION Disable ()

Die Funktion `Disable` schaltet den Antrieb zunächst in den Zustand *Switched On*. Dadurch wird der Antrieb über die im Objekt *Disable Operation Option Code* (0x605C) eingestellte Rampe stillgesetzt. Danach wird in den Ausgangszustand *Switch On Disabled* zurückgeschaltet.

FUNCTION QuickStop ()

Die Funktion `QuickStop` schaltet den Antrieb aus dem Zustand *Operation Enabled* in den Zustand *Quick Stop Active*. Dadurch wird der Antrieb über die im Objekt *Quick Stop Option Code* (0x605A) eingestellte Rampe stillgesetzt.

FUNCTION MoveAbs (TargetPos, Immediate)

Die Funktion `MoveAbs` übergibt den Parameter *TargetPos* als neuen absoluten Sollwert.

Voraussetzung: Die Betriebsart PP ist eingestellt.

Der Parameter *Immediate* zwingt den Antrieb zur Übernahme des neuen Sollwerts auch bei noch laufender Positionierung.

Die Funktion kehrt sofort zum aufrufenden Kontext zurück und wartet nicht, bis die übergebene Zielposition erreicht ist.

FAULHABER Motion Bibliothek

FUNCTION MoveRel (TargetPos, Immediate)

Die Funktion `MoveRel` übergibt den Parameter *TargetPos* als neuen relativen Sollwert. Die neue Bewegung findet also relativ zur vorhergehenden Bewegung statt.

Voraussetzung: Die Betriebsart PP ist eingestellt.

Der Parameter *Immediate* zwingt den Antrieb zur Übernahme des neuen Sollwerts auch bei noch laufender Positionierung.

Die Funktion kehrt sofort zum aufrufenden Kontext zurück und wartet nicht, bis die übergebene Zielposition erreicht ist.

FUNCTION WaitPos ()

Die Funktion `WaitPos` wartet, bis der Antrieb über das Target-Reached-Bit im Statuswort ein erreichtes Ziel signalisiert.

Voraussetzung: Vorher wurde mit `MoveAbs` oder `MoveRel` eine neue Positionierung in der Betriebsart PP gestartet.

6.4 MyControlLib.bi

FUNCTION IsInput(pin)

Die Funktion `IsInput` gibt den logischen Zustand eines Digitaleingangs zurück, der über den Parameter `pin` ausgewählt wird:

Digitaleingang	pin
DigIn1	pin = 1
DigIn2	pin = 2
...	...
DigIn8	pin = 8

FUNCTION SetOutput(pin,level)

Die Funktion `SetOutput` setzt den über den Parameter `pin` angegebenen Digitalausgang:

Digitalausgang	pin
DigOut1	pin = 1
DigOut2	pin = 2
DigOut3	pin = 3

Mit dem Parameter `level` wird das logische Signal des digitalen Ausgangs gesteuert:

level	Digitalausgang	LED
level = 0	low	leuchtet
level = 1	high	leuchtet nicht

FUNCTION StartHomingMethod(method)

Die Funktion `StartHomingMethod` setzt die über den Parameter `method` angegebene Homingmethode und startet diese. Die ggf. nötige I/O-Konfiguration muss vorher eingestellt werden.

FUNCTION StartHoming()

Die Funktion `StartHoming` startet die vorkonfigurierte bzw. zuletzt gesetzte Homingmethode. Die ggf. nötige I/O-Konfiguration muss vorher eingestellt werden.

FUNCTION isHomingFinished()

Die Funktion `isHomingFinished` prüft im Drive Statuswort, ob ein vorher gestartetes Homing erfolgreich abgeschlossen wurde. Die Funktion kann nur für die "fahrenden" Homings verwendet werden, d. h. für alle außer Methode 37.

Es wird nicht geprüft, ob ein Homing gestartet wurde.

FUNCTION isInPos()

Die Funktion `isInPos` prüft im Drive Statuswort, ob ein vorher gestarteter Positioniervorgang erfolgreich abgeschlossen wurde.

Es wird nicht geprüft, ob ein Positioniervorgang gestartet wurde.

Beispielprogramme

7 Beispielprogramme

Die Beispieldateien sind unter **Öffentliche Dokumente** im Ordner `\Users\Public\Documents\Faulhaber\Motion Manager 7\Examples\MC Basic` abgelegt.

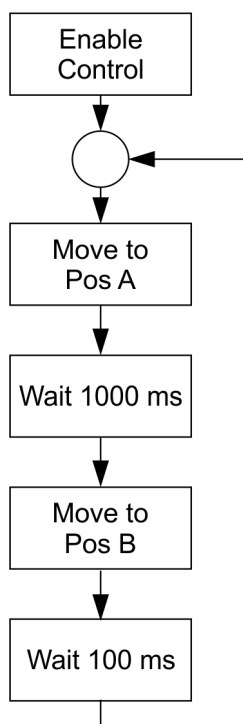
Über das Erweiterungs Menü der Dateiverwaltung im Motion Manager können die Dateien geladen und in den Projekt-Ordner des Antriebs zur Weiterverarbeitung gespeichert werden.

7.1 Einfache zyklische Bewegung unter Nutzung der Bibliotheksfunktionen

In diesem Beispiel wird zunächst die Betriebsart Profile Position Mode (PP) eingestellt. Zwei Positionen werden symbolisch definiert. Die Motorregelung wird explizit über die Funktion `Enable()` gestartet.

Mit den Bibliotheksfunktionen aus *MotionFunctions.bi* wird danach alternierend zwischen den beiden Positionen gewechselt.

Voraussetzungen: Der Motor wurde erfolgreich in Betrieb genommen und die Regelung wurde an die Anwendung angepasst.



```

'-----
'Author: MCSupport
'Date: 2018-09-14
'-----
'Description: Test of the Libs
'-----

#include "MotionParameters.bi"
#include "MotionFunctions.bi"

#define PosA 0
#define PosB 10000

SETOBJ ModesOfOperation = OpModePP

Enable ()

DO
    MoveAbs (PosA, 0)
    WaitPos ()
    DELAY 1000
    MoveAbs (PosB, 0)
    WaitPos ()
    DELAY 100
LOOP

END
  
```

Beispielprogramme

7.2 Nutzung von Schrittketten zur Programmgestaltung

In vielen Fällen bestehen Abläufe aus einzelnen Schritten, die nacheinander oder abhängig von weiteren Bedingungen bearbeitet werden sollen.

Beispiele:

- **Erst** die Regelung starten
- **Zunächst** eine Referenzfahrt ausführen
- **Dann** in den Positionierbetrieb wechseln
- **Wenn ...** betätigt, in den Tipp-Betrieb wechseln

In den Beispielen sind die Schlüsselwörter hervorgehoben, die die Schrittkette kennzeichnen.

In all diesen Fällen bietet es sich an, die Schritte in einer Tabelle zu sammeln und in eine Reihenfolge zu bringen. Zu jedem Schritt muss notiert werden, was während des Schritts passieren soll und was die Bedingung für den Übergang zum nächsten Schritt darstellt.

Implementierung

```
DIM StepCounter
StepCounter = 1
...
DO
    IF StepCounter = 1 THEN
        DoSomething ()
        IF FirstCondition THEN
            StepCounter = 2
        END IF
    ELSEIF StepCounter = 2 THEN
        DoWhatever ()
        IF NextCondition THEN
            StepCounter = 3
        END IF
    ELSEIF ... THEN
        ...
    END IF
LOOP
```

Beispielprogramme

7.2.1 Referenzfahrt mit anschließendem automatischen Wechsel in den Positionierbetrieb

In diesem Beispiel wird der Motion Controller so konfiguriert, dass die Motorregelung automatisch gestartet wird.

Im Ablaufprogramm wird zunächst geprüft, ob die Endstufe bereits aktiviert ist. Danach wird die vorab konfigurierte Referenzfahrt gestartet.

Nachdem der Antrieb erfolgreich referenziert wurde, wird in den aktiven Betrieb gewechselt. Über DigIn1 kann zwischen Positionsregelung mit analoger Sollwertvorgabe und Drehmomentenregelung gewechselt werden.

Voraussetzungen

- Der Motor wurde erfolgreich in Betrieb genommen und die Regelung an die Anwendung angepasst.
- Der gewünschte Typ der Referenzfahrt wurde in Objekt 0x6098.00 konfiguriert.
- Die analogen Sollwertvorgaben sind über die Objekte 0x2313 passend skaliert und über das Objekt 0x2331 als Quellen für den Positionssollwert (0x2331.04) bzw. den Drehmomentensollwert (0x2331.02) gewählt.
- Die Motorregelung wurde über Bit 2 im Objekt 0x233F.00 automatisch aktiviert.

Abfolge

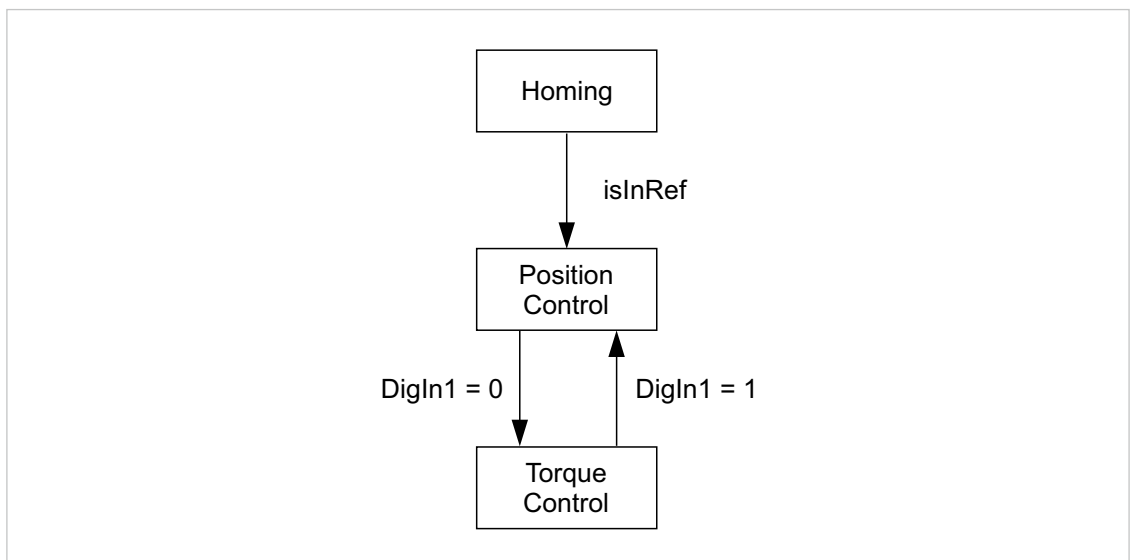


Abb. 2: Abfolge für Referenzfahrt mit anschließendem Wechsel in den Positionierbetrieb

Implementierung

```

'-----
'Author: MCSupport
'Date: 2018-09-14
'-----
'Description: Test of the Libs
'-----

#include "MotionParameters.bi"
#include "MotionFunctions.bi"
  
```

Beispielprogramme

```
:Init
DIM StepCounter
DIM DigInStatus

StepCounter = 0

:MainLoop
DO
    DigInStatus = GETOBJ DigitalInputLogicalState

    IF StepCounter = 0 THEN
        IF isEnabled() THEN
            'Drive is enabled: start Homing
            StartHoming()
            StepCounter = 1
        END IF
    ELSEIF StepCounter = 1 THEN
        IF isInRef() THEN
            'can start the applicaton now1
            StepCounter = 2
        END IF
    ELSE
        Run()
    END IF
LOOP
END

'-----
'local functions

FUNCTION isEnabled()
    DIM DriveStatus

    'DriveStatus is the lower bits of the statusword
    DriveStatus = (GETOBJ Statusword) & $6F

    IF (DriveStatus = CiAStatus_OperationEnabled) THEN
        RETURN 1
    ELSE
        RETURN 0
    END IF
END FUNCTION

FUNCTION StartHoming()
    SETOBJ ModesOfOperation = OpModeHoming
    SETOBJ Controlword = (CiACmdEnableOperation | CiACmdStartBit)
END FUNCTION

FUNCTION isInRef()
    DIM DeviceStatus

    DeviceStatus = GETOBJ Statusword

    'check for IsInRef bit
    IF (DeviceStatus & $1000) > 0 THEN
        RETURN 1
    ELSE
        RETURN 0
    END IF
END FUNCTION
```

Beispielprogramme

```
END FUNCTION
```

```
FUNCTION Run()
  'check for DigIn1
  IF (DigInStatus & $01) > 0 THEN
    SETOBJ ModesOfOperation = OpModeAPC
  ELSE
    SETOBJ ModesOfOperation = OpModeATC
  END IF
END FUNCTION
```

7.2.2 Referenzfahrt mit anschließendem automatischen Wechsel in den Positionierbetrieb und Start-Stop-Funktion

Dieses Beispiel baut auf das Beispiel in Kap. 7.2.1, S. 30 auf.

Die Regelung wird hier nicht mehr automatisch aktiviert. Aus dem Programm heraus wird die Regelung aktiviert, wenn DigIn2 aktiv ist.

Nach dem ersten Einschaltvorgang wird zunächst die Referenzfahrt ausgeführt und danach in den normalen Betrieb gewechselt. Über DigIn2 kann die Regelung jederzeit auch wieder deaktiviert werden. Die Referenzfahrt erfolgt jedoch nur nach dem erstmaligen Einschalten.

Voraussetzungen

- Der Motor wurde erfolgreich in Betrieb genommen und die Regelung an die Anwendung angepasst.
- Der gewünschte Typ der Referenzfahrt wurde in Objekt 0x6098.00 konfiguriert.
- Die analogen Sollwertvorgaben sind über die Objekte 0x2313 passend skaliert und über das Objekt 0x2331 als Quellen für den Positionssollwert (0x2331.04) bzw. den Drehmomentsollwert (0x2331.02) gewählt.

Abfolge

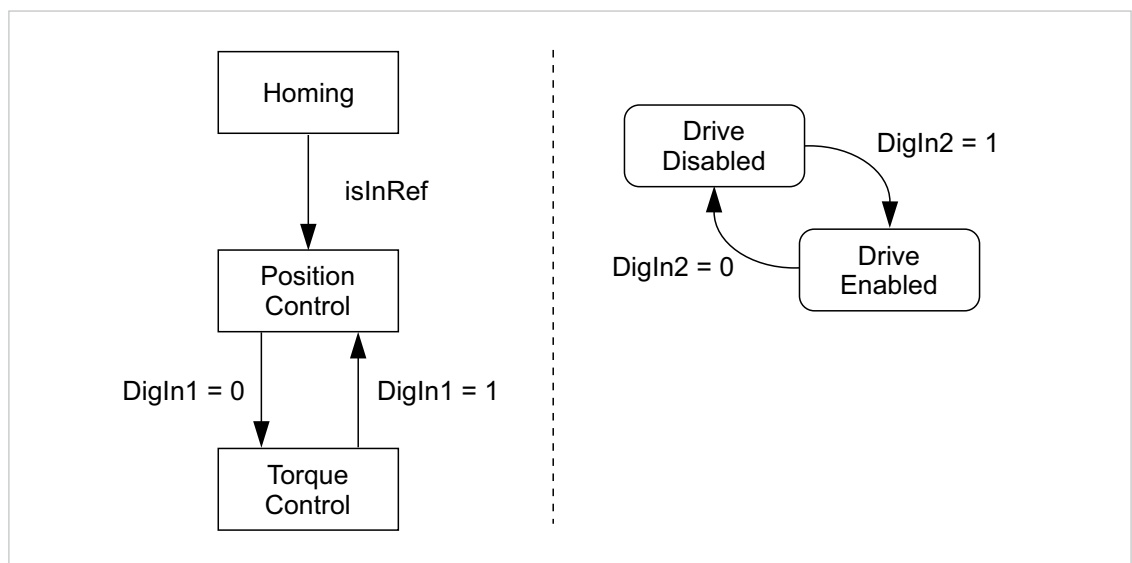


Abb. 3: Abfolge für Referenzfahrt mit anschließendem Wechsel in den Positionierbetrieb und Start-Stop-Funktion

Beispielprogramme

Implementierung

Die Abfolge zur Referenzfahrt mit anschließendem Wechsel in den Positionierbetrieb (linke Seite in Abb. 3) wird inklusive der Unterfunktionen aus dem Beispiel in Kap. 7.2.1, S. 30 übernommen.

Die Prüfung und Reaktion auf DigIn2 wird in der Schleife ergänzt. Die Voraussetzung dazu ist, den Ablauf als Schrittkette ohne blockierende Abfragen zu implementieren.

```
'-----  
'Author: MCSupport  
'Date: 2018-09-14  
'-----  
'Description: Test of the Libs  
'-----  
  
#INCLUDE "MotionParameters.bi"  
#INCLUDE "MotionFunctions.bi"  
  
:Init  
DIM StepCounter  
DIM DigInStatus  
DIM DriveStatus  
DIM isStarted  
  
StepCounter = 0  
isStarted = 0  
  
:MainLoop  
DO  
    'cyclic check of status  
    'DriveStatus is the lower bits of the statusword  
    DriveStatus = (GETOBJ Statusword) & $6Fe  
    DigInStatus = GETOBJ DigitalInputLogicalState  
  
    'check application status  
    IF StepCounter = 0 THEN  
        IF isEnabled() THEN  
            'Drive is enabled: start Homing  
            StartHoming()  
            StepCounter = 1  
        END IF  
    ELSEIF StepCounter = 1 THEN  
        IF isInRef() THEN  
            'can start the applicaton now1  
            StepCounter = 2  
        END IF  
    ELSE  
        Run()  
    END IF  
END DO
```

Beispielprogramme

```
'check DigIn2 for start/stop of the powerstage
IF isStarted THEN
    'check for stop command
    IF (DigInStatus & $02) = 0 THEN
        'drive shall be stopped
        IF StopDrive() THEN
            isStarted = 0
        END IF
    END IF
ELSE
    'check for start command
    IF (DigInStatus & $02) > 0 THEN
        'drive shall be started
        IF StartDrive() THEN
            isStarted = 1
        END IF
    END IF
END IF
LOOP
END

'-----
'local functions

FUNCTION isEnabled()
    IF (DriveStatus = CiAStatus_OperationEnabled) THEN
        RETURN 1
    ELSE
        RETURN 0
    END IF
END FUNCTION

FUNCTION StartHoming()
    SETOBJ ModesOfOperation = OpModeHoming
    SETOBJ Controlword = (CiACmdEnableOperation | CiACmdStartBit)
END FUNCTION

FUNCTION isInRef()
    DIM DeviceStatus
    DeviceStatus = GETOBJ Statusword

    'check for IsInRef bit
    IF (DeviceStatus & $1000) > 0 THEN
        RETURN 1
    ELSE
        RETURN 0
    END IF
END FUNCTION

FUNCTION Run()
    'check for DigIn1
    IF (DigInStatus & $01) > 0 THEN
        SETOBJ ModesOfOperation = OpModeAPC
    ELSE
        SETOBJ ModesOfOperation = OpModeATC
    END IF
END FUNCTION
```

Beispielprogramme

7.3 Ereignisbehandlung

Nachfolgender Programmausschnitt zeigt, wie auf das Ereignis **Temperatur Warngrenze erreicht** reagiert werden kann.

```
DEF_EVT_VAR e 'Define event mask
EN_EVT $00030000, EvtOverTemp 'activate event handling for over temperature

:EvtOverTemp
IF e & $00020000 THEN
    END
ELSE
    w = 1 'temperature warning, set variable w
END IF
RET_EVT
```

