FAULHABER

Programming Manual

Motion Controller V3.0

Motion Control Sys-
teme 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

Imprint

Version:
4th edition, 28.02.2024

This document has been prepared with care.
Dr. Fritz Faulhaber GmbH & Co. KG cannot accept any liability for any errors in this document or for the consequences of such errors. Equally, no liability can be accepted for direct or consequential damages resulting from improper use of the equipment.

The relevant regulations regarding safety engineering and interference suppression as well as the requirements specified in this document are to be noted and followed when using the software.

Subject to change without notice.

The respective current version of this technical manual is available on FAULHABER's internet site:
www.faulhaber.com

Content

1 About this document

1.1 Validity of this document

This document describes the programming of sequence programs for drive electronics of the Motion Controller and Motion Control Systems of the V3.0 family using the FAULHABER Motion Manager 7.

This document is intended for software developers with programming experience, and for drive technology project engineers.

All data in this document relate to the standard versions of the drives. Changes relating to customer-specific versions can be found in the attached sheet.

1.2 Associated documents

For certain actions during commissioning and operation of FAULHABER products additional information from the following manuals is useful:

Manual	Description
Motion Manager 7	Operating instructions for FAULHABER Motion Manager PC software
Quick start guide	Description of the first steps for commissioning and operation of FAULHABER Motion Controllers
Drive functions	Description of the operating modes and functions of the drive

These manuals can be downloaded in pdf format from the web page www.faulhaber.com.

1.3 List of abbreviations

Abbreviation	Meaning
BASIC	Beginner's All-Purpose Symbolic Instruction Code
EEPROM	Electrically Erasable Programmable Read-Only Memory
Sxx	Data type signed (negative and positive numbers) with bit size xx
Uxx	Data type unsigned (positive numbers) with bit size xx

1.4 Symbols and designations

NOTICE!
Risk of damage.
- Measures for avoidance

Instructions for understanding or optimizing the operational procedures

✓ Pre-requirement for a requested action

1. First step for a requested action
   - Result of a step
2. Second step of a requested action

Result of an action

- Request for a single-step action

2 Introduction

Sequence programs can be transferred to controller by the FAULHABER Motion Manager and can be executed directly by the controller. This enables e.g. stand-alone operation without a supervisory controller or semi-autonomous execution of smaller program sequences.

Sequence programs are programmed in the BASIC programming language, with FAULHABER-specific extensions.

8 independent memory areas for user programs are available. Optionally, one program can also be started automatically at boot-up.

3 Characteristics of the programming language

- BASIC interpreter with FAULHABER-specific extensions
- Function calls
- No line numbers; jumps are to jump labels
- Jump labels are placed at the beginning of a line and start with a colon
- Distinction between upper and lower case characters (commands always in upper case)
- Read and write access to objects in the object dictionary
- Capability to respond to events during normal execution of a program
- Timer for time measurement and wait loops
- Arithmetic, comparison and bit operators
- Special character **$** for values expressed as hexadecimal numbers
- Maximum length of all programs: 16 kByte
- 26 global standard 32-bit variables a…z (can be stored permanently)
- 26 global symbolic 32-bit variables (can be freely named)
- Local symbolic variables (can be freely named)

3.1 Command set

Tab. 1: Standard BASIC command set

Command	Function	Example
REM…	Comment. Placed at the beginning of a line and applies until the end of the line.	REM comment
END	Exit program	END
GOTO…	Jump to the specified jump label. The following constructs may not be exited with GOTO: ▪ IF…THEN…ELSE…END IF ▪ GOSUB…RETURN ▪ FOR…TO…NEXT… GOTO Jumps are not supported in sub-functions (FUNCTION…).	GOTO Start
GOSUB… … RETURN	Jump to a sub-program at the specified label. After execution, jump back to the calling position. No GOTO jump may be performed from a sub-program.	GOSUB Step1 … :Step1 RETURN
FOR…TO… … NEXT…	Programming a loop. No conditional GOTO jump may be performed from a FOR loop.	FOR i = 1 TO 10 … NEXT i
DO … LOOP UNTIL…	Loop with check of the loop condition at the end of the loop.	DO … LOOP UNTIL a = 5

Characteristics of the programming language

Command	Function	Example
DO … LOOP	Loop without check of a loop condition. Exit the loop with EXIT.	DO ... LOOP
DO … LOOP WHILE…	Loop with check of the loop condition at the end of the loop.	DO ... LOOP WHILE stop = 0
DO WHILE… … LOOP	Loop with check of the loop condition at the start of the loop.	DO WHILE speed > 500 ... LOOP
EXIT…	Jump from a loop without having reached the end of the loop. The keyword of the loop must be specified.	EXIT FOR EXIT DO

IF…THEN…
ELSEIF…THEN…
ELSE…
END IF
	Programming a branch.
	No GOTO jump may be performed from an IF instruction.
		IF a > 3 THEN
		  b = 1
		ELSE
		  b = 0
		END IF

IF…THEN GOTO… IF…THEN GOSUB… IF…THEN <Name>()	Conditional jump or branch into a sub-program. Used in a line without END IF. The following constructs may not be exited with GOTO: ▪ IF…THEN…ELSE…END IF ▪ GOSUB…RETURN ▪ FOR…TO…NEXT…	IF z=1 THEN GOSUB Step1
IF…THEN EXIT FOR IF…THEN EXIT EVT	Jump out of a FOR loop or an event routine. Used in a line without ENDIF. May not be used in the following constructs: ▪ IF…THEN…ELSE…END IF ▪ GOSUB…RETURN	FOR a = 1 TO 5 IF x = 1 THEN EXIT FOR NEXT a
IF…THEN EXIT GOSUB	Jump out of a sub-program. Used in a line without ENDIF. May not be used in the following constructs: ▪ IF…THEN…ELSE…END IF ▪ GOSUB…RETURN	:Sub1 IF x = 1 THEN EXIT GOSUB RETURN

Command	Function	Example
FUNCTION… … RETURN… END FUNCTION	Definition of a sub-function. The function is called in the program text via its name. Local variables can be allocated via the key word DIM. Parameters can be passed as numbers or with variables. The sub-function can return a numeric result with the key word RETURN. GOTO jumps are not supported. Each sub-function must be ended with the key word END FUNCTION.	**Defintion** FUNCTION <Name> (Parameter1, Parameter2) DIM result … RETURN result END FUNCTION **Call** <Name> (1, 5)
DIM…	Allocates a variable with symbolic name. ▪ If DIM is used outside of a sub-function (FUNCTION), a global variable is allocated that can also be used under this name by all sub-functions. Global symbolic variables can be read and changed in the development environment integrated in the Motion Manager via their name. ▪ If DIM is used within a sub-function (FUNCTION), a local variable is allocated that is valid only within the sub-function. Local variables can be read and changed in the development environment integrated in the Motion Manager as long as the sub-function itself is active (e.g., stopped at a breakpoint or in single-step mode).	DIM Statusword

Tab. 2: FAULHABER command extension

Command	Function	Example
SETOBJ…	Write an object in the object dictionary. Syntax: SETOBJ <Index>.<Subindex> = <variable or value>	SETOBJ $6083.$00 = 500
GETOBJ…	Read an object in the object dictionary. Syntax: <variable> = GETOBJ <Index>.<Subindex>	a = GETOBJ $6083.$00
DEF_EVT_VAR…	Defines a variable which, when the event occurs, returns the value of the event status bit mask.	DEF_EVT_VAR e
EN_EVT…	Activation of an event routine which is triggered by the device state signalled by a change in the object 0x2324.01 (event handling). Note: Only one event routine can be active. Syntax: EN_EVT <bit mask>,<event mark>	EN_EVT $ffffffff, EvHandler
DI_EVT	Deactivation of all events for processing that is being performed in parallel. Syntax: DI_EVT	DI_EVT
RET_EVT	Jump back from an event routine. Syntax: RET_EVT	: EvHandler RET_EVT
SAVE…	Permanent saving of one or more variables in the EEPROM (comma-separated list). Syntax: SAVE <variable1<,variable2,...>>	SAVE a, b, z

Characteristics of the programming language

Command	Function	Example
LOAD…	Loading one or more previously saved variables from the EEPROM (comma-separated list). Syntax: LOAD <variable1<,variable2,...>>	LOAD a, b, z
DEF_TIM_VAR…	Defines a variable to be used as a timer. Syntax: DEF_TIM_VAR <variable>	DEF_TIM_VAR t

START_TIM…	Starts the timer with a value in ms (or stops the timer if the value = 0).
	Syntax: START_TIM <variable or value>
	When the specified time has elapsed, the timer variable is 1, otherwise it is 0 (timer still running).
		START_TIM 3000
		
		IF t = 1 THEN
		ENDIF

DEF_CYC_VAR…	Defines a variable to be used as a 1 ms cycle counter. This can be used to, e.g., perform time measurements. The counter runs a maximum of 24 days and then stops with –1. Syntax: DEF_CYC_VAR <variable>	DEF_CYC_VAR z
START_CYC	Starts the cycle counter with the value 0. Syntax: START_CYC	START_CYC
STOP_CYC	Stops the cycle counter. The current counter state is retained in the variable defined for this purpose and can be processed further. Syntax: STOP_CYC	STOP_CYC
DELAY…	Waiting time in ms. The program is not processed further during the waiting time. Syntax: DELAY <variable or value>	DELAY 200
#DEFINE…	Assigns a value to a symbolic designation. Syntax: #DEFINE <symbol> <value> This key word can also be used to define more complex macros that can be used in the program text via the macro names. Syntax: #DEFINE <macro> <expression> If a macro name is prefixed with "MC.", the macro is available in the autocompletion of the Motion Manager editor. The list of available macros appears after entering "MC."	#DEFINE MaxSpeed 2000 #DEFINE MC.IsTargetReached ((GETOBJ $6041.00 & $400) = $400) If MC.IsTargetReached THEN…
#INCLUDE…	Instructs the development environment to link another file to the program (Basic-Include file *.bi). Sets of pre-defined symbolic names or function libraries can thereby be linked and reused. If no path is specified, include files are searched for in either the Motion Manager ProgramData directory or in the same folder as the corresponding .bas file. Include files from other folders can be referenced by specifying an absolute path. Syntax: #INCLUDE <[path] filename>	#INCLUDE "MotionParameters.bi"

Command	Function	Example
EVENT…	Reports an event to the event broker (see drive functions manual). This can be used either to set the event flag in device status word 0x2324.01 or to start a recording of the trace recorder. Any 16 bit value is used as a parameter passed as event code that contains the event identifies. Event code 0 resets the event. Events can only be set again if the event has been canceled with code 0 in the meantime or via different event codes. The set event code cannot be read back via the object dictionary.	EVENT 1 EVENT 0
ERROR…	This command can be used to set bit 8 in the FAULHABER error word 0x2320.00. The parameter passed on is a 16-bit error code, which can be read out in object 0x2322.01. Program processing continues normally after the error is signaled in the FAULHABER error word. An unexpected situation can be signaled using the ERROR keyword. Depending on the error handling setting (masks below 0x2321.xx), the drive can also be stopped automatically. By default, an EMCY message with the error code 0xFF30 is generated via the error monitoring unit. Error code 0 resets the error status.	ERROR $1234
RESET	This command can be used to completely reinitialize the Motion Controller from a sequence program. All control and communication is immediately interrupted. The device restarts similar to after a power cycle.	RESET

3.2 Operators and special characters

Arithmetic operators	
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo (remainder)	%

Logic operators	
And operation	AND
Or operation	OR
Inversion	NOT

Comparison operators	
Greater than	>
Less than	<
Equal to	=
Not equal to	<>
Greater than or equal to	>=
Less than or equal to	<=

Bit operators	
Bit-wise AND	&
Bit-wise OR	|
Bit-wise EXCLUSIVE OR (XOR)	^
Bit-wise inversion	~
Bit-wise moving of a variable to the left	<<
Bit-wise moving of a variable to the right	>>

Assignment operator	
Assignment operator	=

Special character	Meaning
()	Used for mathematical operators
,	Used in EN_EVT and SAVE/LOAD
.	Delimiting characters in SETOBJ /GETOBJ
$	Hexadecimal numbers
:	Jump label, placed at the start of the line

3.3 Variables

Three variable types are available in sequence programs:

- Global variables with fixed address (see chap. 3.3.1, p. 13)
- Allocated global variables (see chap. 3.3.2, p. 14)
- Allocated local variables (see chap. 3.3.2, p. 14)

3.3.1 Global variables with fixed address

Variables that are referenced via letters a…z are available globally in all program areas. Each change is visible globally.

Variables with fixed address have the following properties:

- They can be saved and loaded in the EEPROM using the LOAD and SAVE commands.
- They can be used as timer variables (DEF_TIM_VAR) or event variables (DEF_EVT_VAR) (only applies to variables with fixed address).
- They can be addressed via object 0x3005 directly via the communication interface.
- They can be mapped to PDOs.
- They can be assigned a symbolic name via #DEFINE. These symbolic names can be used everywhere in the program text.

Example:

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

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

3.3.2 Allocated global and local variables

With the key word DIM <variable name>, a variable can be created with a symbolic name.

Example:

```
DIM Statusword
```

- Global variables:
  If `DIM` is used outside of a sub-function, a variable from a set of a maximum of 26 global variables is thereby allocated. The variable can be used by all functions. The symbolic name can be used like a number in all expressions.
- Local variables:
  If `DIM` is used within a sub-function, only a local variable valid within this sub-function is thereby allocated. Transfer parameters of sub-functions are handled as local variables. The maximum number of local variables is 26 per sub-function.

Allocated variables cannot be used as timer variables (DEF_TIM_VAR) or event variables (DEF_EVT_VAR). They cannot be directly initialised via LOAD or SAVE from the EEPROM nor can they be stored there.

In the Motion Manager, it is possible to access allocated variables via their symbolic names. Local variables can only be accessed within the function in which the variable is defined.

3.4 Instructions for creating programs

- A sequence program should basically be structured as a step chain with a main loop covering the entire execution code (see chap. 7.2, p. 28). Wait loops with conditional jumps to specific events are not possible.
- Sequence programs are created and edited using the FAULHABER Motion Manager.
- Before downloading a sequence program to the controller, the FAULHABER Motion Manager performs a pre-processing step in order for instance to determine the addresses of the jump labels and the necessary memory size.
- FAULHABER Motion Manager offers the capability not only to create sequence programs, edit them and transfer them to the controller, but also to check for programming errors and correct them (debugging options).
- Own function libraries can be developed and linked as basic include files (*.bi) in a sequence program.
  It is recommended that these libraries first be created as a BAS file and the functions outsourced to an include file only after completion of development and testing. The Motion Manager offers no debugging options for include files.

4 Developing sequence programs using the Motion Manager

The FAULHABER Motion Manager 7 offers an integrated development environment for sequence programs in the **programming** area. The development environment offers the following facilities:

- Syntax highlighting
- Loading, displaying, and editing sequence programs from the device memory and from the PC memory
- Start individual sequence programs
- Stop the active sequence program
- Pause the active sequence program
- Single step execution
- Definition of a breakpoint
- Display the current program status and the current program line
- Monitor and change the contents of variables
- Read protection by means of an access code
- Code modules for use in your own programs
- Autocompletion
- Automatic syntax check in the background
- Setting an auto-start program

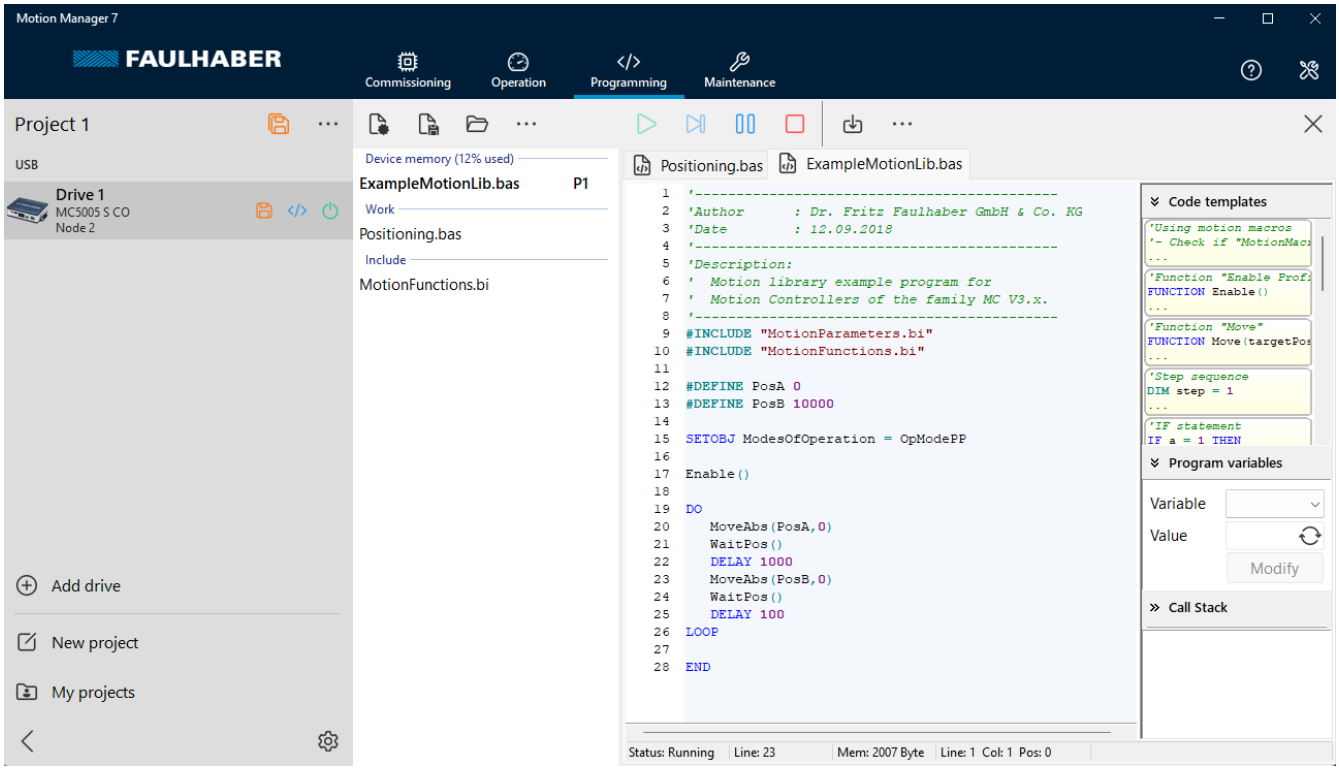


Fig. 1: Motion Manager 7 – Programming

Tab. 3: Editor button functions

Button	Designation	Function
	Run	Download sequence program to the controller and execute it.
	Step	Execute the sequence program in single steps or continue.
	Halt	Pause the sequence program.
	Stop	End the sequence program.
	Download	Download the sequence program to the controller and save it without executing it.

4.1 Edit program

4.1.1 File management

- A new program is created using the **New file** button and the selection **Sequence program file (*.bas)** and displayed in a new tab with a prepared file header.
- An existing program is loaded using the **Open file** button and displayed in a new tab.
- Sample programs prepared by FAULHABER can be loaded via the extension menu **(...)** of the file management and selection of **Examples**.
- If newly loaded or created files are saved, executed, or downloaded to the controller, a file name is asked for, under which the file is then stored in the project folder for the respective drive. The file then appears in the file management in the left area of the programming environment.
  - Using the file management context menu, you can, among other things, change the program number assignment or delete the program from the device memory.
  - Using the save button in the project area, a downloaded sequence program is permanently saved in the device and in the project folder and assigned to a program number.
  - Sequence programs stored in the device are automatically read out when a drive is added to a project. If an associated source file is found in the project folder, it will be copied to the drive's project folder and displayed in the file management. Otherwise, an .out file will be displayed containing the program content as stored in the device.

    Editing an .out file is not possible. However, a copy with the ending .bas can be created. This means that the program is translated back, if possible, for post-processing. The device memory can subsequently be read out using the file management extension menu.

    The .out files can be used to transfer sequence programs unchanged to other drives.

4.1.2 File editing

- Comment lines that start with ' are saved in the file only for purposes of documentation and are not downloaded to the controller.
- The content of the files that are referenced via `#INCLUDE` is merged with the actual program code prior to transfer to the controller.
- Macro commands and symbolic variables specified in the program are replaced with the expressions stored with `#DEFINE` and the internal variables allocated with `DIM` prior to transfer to the controller.
- For user support, there are code templates on the right side of the editor that can be dragged into your own program with the mouse and adapted.
- If the automatic syntax check is activated (extension menu of the file tab), detected syntax errors are displayed underlined in red.
- A context menu for further Editor functions can be opened with the right mouse button. In this way e.g., an include file specified in the program code can be opened or jumped to the location of a function or macro definition.

4.2 Load the program to the controller and execute it

- Pressing the **Run** button downloads the finished program to the controller and immediately executes it.
- The file management displays various states of the program:

Display	Description
*	The file has been transferred, but not yet permanently saved in the device (new program in the RAM).
P1	The file is stored in storage location 1.
P1*	The program has been re-transferred, but has not yet been re-saved (changed program in RAM)
File name in bold font style	Active program in the device.

- After execution has been started, the editor area switches into debug mode (with a different background colour). This mode does not allow program editing.
- If errors occur whilst running the program, execution is interrupted and the last line executed is highlighted in red.
- To return to program edit mode, program execution must be ended by pressing the **Stop** button.

4.3 Program Debug

The following debugging options are available for troubleshooting in sequence programs:

- Pause the program at the current execution position (**Pause** button):
  - The active line is highlighted in the editor. If the program is currently processing a function in a linked file, the calling location in the main file is marked.
  - The edit area remains inactive.
  - After a **Pause** the program can either be continued via **Run** or executed further in single steps via **Step**. Pressing **Stop** reverts to program edit mode.
- Executing the program further in single steps (**Step** button):
  - Only the next program line is executed.
  - The new active line is highlighted in the editor.
  - The edit area remains inactive.
  - After **Step** the program can either be continued via **Run** or executed further in single steps via **Step**. Pressing **Stop** reverts to program edit mode.
- Pausing the program at a breakpoint:
  - A breakpoint can be established by clicking on the desired line number at the left-hand edge of the window.
  - Program execution is paused when it reaches this line. It can then be continued via **Run** or **Step**. Pressing **Stop** reverts to program edit mode.
  - Clicking on the breakpoint at the left-hand edge of the window deletes the breakpoint. Until this has been done, no further breakpoint can be established.
  - A breakpoint can be established before a program is started and also during program execution.

- Investigating and changing the contents of variables:
  - In the programming tools on the right edge of the editor there is the **Program variables** area. The globally defined symbolic variables and the standard variables a…z can be selected at any time from the variable list and displayed and changed there. Local variables can only be accessed if the program has stopped in the respective function.
- Examining the call stack:
  - In the programming tools on the right edge of the editor there is the **Call stack** area. In addition to the current line of a stopped program, the numbers of the lines in the call sequence of functions are also displayed.

5 Control of sequence programs

A saved sequence program can be started by a host computer via the interface, or automatically when the controller is booted up.

5.1 Control via the interface

The execution of sequence programs can be controlled and monitored by a supervisory computer, via the object 0x3001.

Tab. 4: Current Control Parameter Set

Index	Subindex	Name	Type	Attr.	Meaning
0x3001	0	Number of Entries	U8	ro	Number of object entries
	1	Program Control	U8	rw	Control of the sequence program activated via 0x3001.02 or 0x3002.00: ▪ 1: Load the activated program from the EEPROM (Load) ▪ 2: Start or continue the loaded program (Run) ▪ 3: Execute the individual program line (Step) ▪ 4: Pause the running program (Break) ▪ 5: End the running program (Terminate)
	2	Program Number	U8	rw	Activate the sequence program at program number
	3	Actual Position	U16	ro	Address of the line currently being executed
	4	Actual Program State	U8	ro	Current status of the program: ▪ 0: No action (Idle) ▪ 1: Program is currently being loaded from the EEPROM (Reading) ▪ 2: Program is currently being saved to the EEPROM (Saving) ▪ 3: Program is currently being deleted (Deleting) ▪ 4: Program is currently being executed (Running) ▪ 5: Program paused (Halted)
	8	Error State	U8	ro	Error status: ▪ 0: No error (No Error) ▪ 1: Syntax error (Parsing Error) ▪ 2: Error accessing the EEPROM (EEPROM Access Error)
	9	Error Code	U16	ro	Detailed error code in the event of a syntax error (see chap. 5.2, p. 21)

Before a new program is loaded, any program already running must be ended.

Pseudo-code:

- If 0x3001.04 = 4 (Running) or 0x3001.04 = 5 (Halted), then 0x3001.01 = 5 (Terminate)
- Wait until 0x3001.04 = 0 (Idle)

Example for loading and running a sequence program in program number 1:

1. Select program 1:
   - 0x3001.02 = 1 (P1)
2. Load program:
   - 0x3001.01 = 1 (Load)
   - Wait until 0x3001.04 = 0 (no longer Reading).
3. Run program:
   - 0x3001.01 = 2 (Run)

↳ Program 1 is loaded and will be run.

5.2 Error handling

Errors that occur during program execution are returned as an error code in object 0x3001.09.

If an error occurs, the following actions are triggered automatically:

- Program execution is ended.
- The detailed error code is returned in object 0x3001.09 (see Tab. 5).
- A calculation error (bit 12) is set in FAULHABER error word 0x2320.00.

When starting a new program, the error code in object 0x3001.09 and calculation error in error word 0x2320.00 are reset.

Error word 0x2320.00 can be used to automatically trigger further actions in the event of an error. For example, the drive can be switched off automatically.

Tab. 5: Error codes for 0x3001.09

Code	Error	Meaning	Remedy
0	No error	No error	–
1	Generic error	General error	Check syntax.
3	Unexpected token	The character was not expected at this location.	Check syntax.
4	Missing return value	A function was ended without RETURN even though a return value was expected.	Add RETURN instruction.
6	End of parsing memory	Nested function calls using too much memory.	▪ Reduce nesting depth. ▪ Use shorter calling lines.
7	Too many variables	The chain of called functions results in too many local variables being used.	Reduce program complexity
8	Illegal variable type	No variables created via DIM can be used for the following functions: ▪ Special functions of the timer ▪ Special functions of event processing ▪ SAVE ▪ LOAD	Use manually created variables a…z.
9	Function stack overflow	The nesting depth of the function calls is too high. Maximum 15 call levels are supported.	Reduce nesting depth.

Code	Error	Meaning	Remedy
10	Nested condition overflow	The nesting depth of conditions such as IF is too high. A depth of max. 15 levels is supported.	Reduce nesting depth.
11	Division by 0		
12	Event while in Event	An event was triggered while event processing was still active. The event cannot be processed.	Reduce trigger frequency.
13	RET_EVT while not in event	RET was used outside of an event.	Only use RET to jump out of events.
14	ELSE or END IF without IF	The ELSE or END IF command was detected without the corresponding IF.	Check syntax.
15	Wrong variable used in NEXT token	The variable used to call NEXT does not correspond to that from the FOR call.	Check syntax.
16	GOTO not supported here	GOTO is not supported within functions.	Do not use GOTO in functions.
17	Illegal object	The object used for GETOBJ or SETOBJ does not exist or does not support the access.	Check syntax.
18	RETURN while not in SUB or FUNCTION	A RETURN was detected without first having entered in a sub-function.	Check syntax.

5.3 Start the sequence program automatically

Object 0x3002.00 allows input of a program number; when the controller is booted up this program will be started automatically.

Tab. 6: Autostart Program Number

Index	Subindex	Name	Type	Attr.	Meaning
0x3002	0	Autostart Program Number	U8	rw	Program number of the sequence program that will be started automatically.

This function is also available in the context menu of file management in Motion Manager (**Start automatically**).

5.4 Protecting sequence programs

With object 0x3003.00, a 32-bit key can be set that protects the programs stored in the controller against unauthorized access.

If a 0x3003.00 ≠ 0 code was set and the parameters of the controller then saved, the stored sequence programs can then only be read out if the code is entered again.

Tab. 7: Access Code

Index	Subindex	Name	Type	Attr.	Meaning
0x3003	0	Access Code	U32	ro	32-bit key for protecting the programs stored in the controller against unauthorized access.

This function is also available in the file management extension menu in Motion Manager (**Lock device memory**).

5.5 Data exchange with sequence programs

Data exchange via object 0x3004

The program variables a to z can also be used for data exchange between the sequence program and the supervisory computer. Object 0x3004.01 can be used to select a variable and object 0x3004.02 to read or write its value.

Tab. 8: Variable Access

Index	Subindex	Name	Type	Attr.	Meaning
0x3004	0	Number of entries	U8	ro	Number of object entries
	1	Variable index	U8	rw	Variable index ▪ 0…25 = standard variables a…z ▪ 32 + (0…25) = global internal variables ▪ 128 + (0…25) = local internal variables
	2	Variable value	S32	rw	Variable value

Data exchange via object 0x3005

Individual standard variables can be directly accessed via the sub-indices of object 0x3005. The variables can thereby be, e.g., recorded or mapped to a PDO. Not included here are variables that are used for event handlers, timers or counters.

Tab. 9: Debug User Program

Index	Subindex	Name	Type	Attr.	Meaning
0x3005	0	Number of entries	U8	ro	Number of object entries
	1…26	User prog variable a…z	S32	rw	Values of variables a…z

6 FAULHABER Motion library

Some supporting files are delivered with FAULHABER Motion Manager 7:

System files
The system files are stored in the Motion Manager installation area and are automatically integrated into new files. They cannot be changed.

The file can be opened by right-clicking on the file name in the `#INCLUDE` line of the program code.

File	Description
MotionParameters.bi	Pre-defined assignment of symbolic parameter names with values, e.g., #DEFINE Statusword $6041.00. See chap. 6.1, p. 24.
MotionMacros.bi	Pre-defined macros for directly accessing parameters of the Motion Controller, e.g., for drive control and status check. See chap. 6.2, p. 25.

Example files
The example files are stored under **Public Documents** in the folder **\Users\Public\Documents\Faulhaber\Motion Manager 7\Examples\MC Basic**.

The files can be loaded via the file management extension menu in the Motion Manager and saved in the drive's project folder for further processing.

File	Description
MotionFunctions.bi	Pre-defined functions for typical drive tasks of the Motion Controllers. See chap. 6.3, p. 25.
MyControlLib.bi	Extension of MotionFunctions.bi for general purposes. See chap. 6.4, p. 26.

6.1 MotionParameters

The *MotionParameters.bi* file contains symbolic definitions for typical parameters, e.g.:

```
#DEFINE Statusword $6041.00
```

The parameters can then be accessed in the program using the symbolic names.

Example:

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

6.2 MotionMacros

The *MotionMacros.bi* file contains pre-defined access to parameters of the drive system.

The macros always start with code `MC`.

Example:

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

DIM DeviceStatus
DeviceStatus = MC.GetStatusword
```

The Motion Manager autocomplete shows a list of available macro functions after entering "MC." (or **Ctrl** + Space bar after the period).

6.3 MotionFunctions

File *MotionFunctions.bi* contains a set of pre-defined functions that can be used to set up your own processes.

FUNCTION Enable ()
The `Enable` function tries to bring the drive state machine into the *Operation Enabled* state.

Only once the *Operation Enabled* state has been reached does the function return to the calling context. If the state cannot be achieved, e.g., because a blocking error is pending, the function does not return. This is, thus, a blocking call.

FUNCTION Disable ()
The `Disable` function first switches the drive to the *Switched On* state. This brings the drive to a stop via the ramp set in object *Disable Operation Option Code* (0x605C). Afterwards, it switches back to the initial state *Switch On Disabled*.

FUNCTION QuickStop ()
The `QuickStop` function switches the drive from the *Operation Enabled* state to the Quick Stop Active state. This brings the drive to a stop via the ramp set in object *Quick Stop Option Code* (0x605A).

FUNCTION MoveAbs (TargetPos, Immediate)
The `MoveAbs` function passes parameter *TargetPos* as new absolute set-point.

Prerequisite: Operating mode PP is set.

Parameter *Immediate* forces the drive to accept the new set-point even during running positioning.

The function immediately returns to the calling context and does not wait until the passed target position is reached.

FUNCTION MoveRel (TargetPos, Immediate)
The `MoveRel` function passes parameter *TargetPos* as new relative set-point. Thus, the new movement takes place relative to the previous movement.

Prerequisite: Operating mode PP is set.

Parameter *Immediate* forces the drive to accept the new set-point even during running positioning.

The function immediately returns to the calling context and does not wait until the passed target position is reached.

FUNCTION WaitPos ()
The `WaitPos` function waits until the drive signals via the Target Reached bit in the status word that a target has been reached.

Prerequisite: A new positioning operation was first started in operating mode PP with `MoveAbs` or `MoveRel`.

6.4 MyControlLib.bi

FUNCTION IsInput(pin)
The `IsInput` function returns the logical state of a digital input selected via the `pin` parameter:

Digital input	pin
DigIn1	`pin` = 1
DigIn2	`pin` = 2
...	...
DigIn8	`pin` = 8

FUNCTION SetOutput(pin,level)
The function `SetOutput` sets the digital output specified via the parameter `pin`:

Digital output	pin
DigOut1	`pin` = 1
DigOut2	`pin` = 2
DigOut3	`pin` = 3

The parameter `level` controls the logical signal of the digital output:

level	Digital output	LED
`level` = 0	low	lights up
`level` = 1	high	does not light up

FUNCTION StartHomingMethod(method)
The function `StartHomingMethod` sets the homing method specified via the parameter `method` and starts it. Any necessary I/O configuration must be made first.

FUNCTION StartHoming()
The function `StartHoming` starts the preconfigured or last set homing method. Any necessary I/O configuration must be made first.

FUNCTION isHomingFinished()
The `isHomingFinished` function checks in the drive status word whether a previously started homing was completed successfully. The function can only be used for the "moving" homings, i.e., for all except method 37.

There is no verification of whether a homing was started.

FUNCTION isInPos()
The `isInPos` function checks in the drive status word whether a previously started positioning process was successfully completed.

There is no verification of whether a positioning process was started.

7 Example programs

The example files are stored under **Public Documents** in the folder **\Users\Public\Documents\Faulhaber\Motion Manager 7\Examples\MC Basic**.

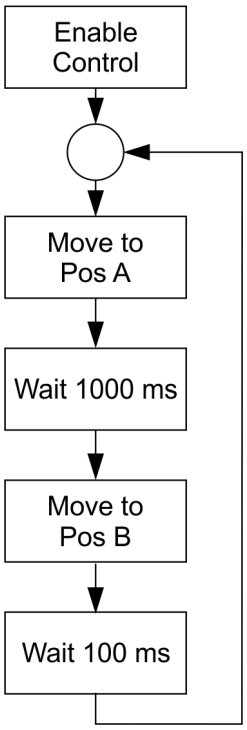
The files can be loaded via the file management extension menu in the Motion Manager and saved in the drive's project folder for further processing.

7.1 Simple cyclic movement using the library functions

In this example, the Profile Position Mode (PP) is first set. Two positions are defined symbolically. The motor control is started explicitly via the `Enable()` function.

With the library functions from *MotionFunctions.bi*, the position then alternates between the two positions.

Prerequisite: The motor was successfully commissioned and the control was adapted to the application.




```
'-----------------------------------------------
'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
```

7.2 Use of sequences of steps for program design

In many cases, sequences consist of individual steps that are to be processed in sequence or depending on other conditions.

Examples:

- **First,** start the control
- **Next,** execute a reference run
- **Then,** change to positioning operation
- **If** … is actuated, change to inching mode

The key words that identify the sequence of steps are in bold in the examples.

In all of these cases, it is useful to collect the steps in a table and organize them into a sequence. It must be noted for each step what is to occur during the step and what the condition is for advancing to the next step.

Implementation

```
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
```

7.2.1 Reference run with subsequent automatic change to positioning operation

In this example, the Motion Controller is configured so that the motor control is started automatically.

A check is first performed in the sequence program to determine whether the output stage is already activated. Afterwards, the previously configured reference run is started.

After the drive has been successfully referenced, it switches to active operation. DigIn1 can be used to switch between motion control with analogue setpoint specification and torque control.

Prerequisites

- The motor was successfully commissioned and the control adapted to the application.
- The desired type of reference run was configured in object 0x6098.00.
- The analogue setpoint specifications are appropriately scaled via objects 0x2313 and selected as sources for the position set value (0x2331.04) or the torque set value (0x2331.02) via object 0x2331.
- The motor control was automatically activated via bit 2 in object 0x233F.00.

Sequence

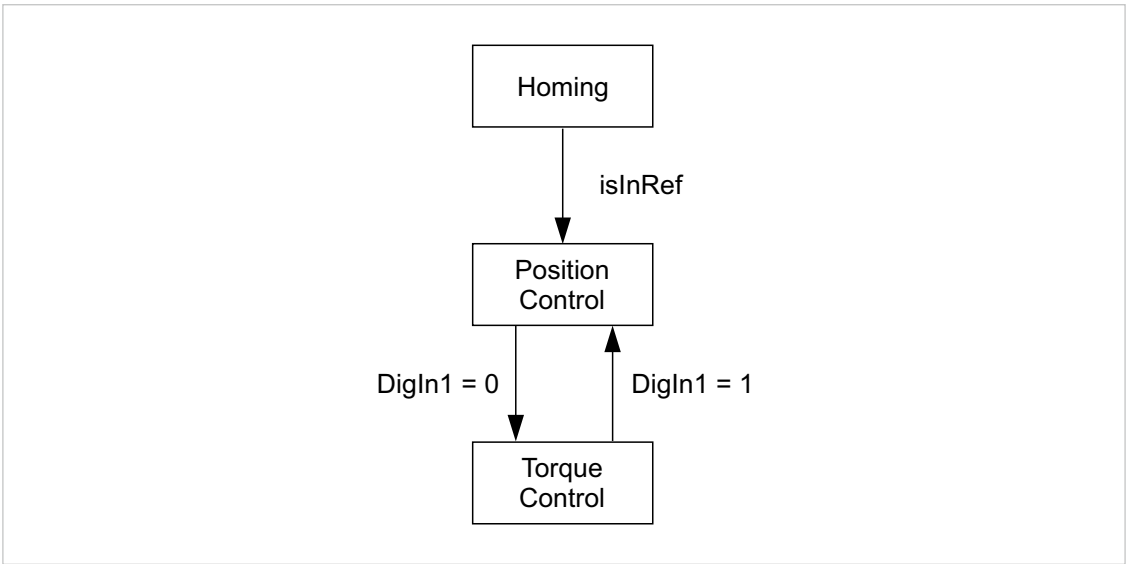


Fig. 2: Sequence for reference run with subsequent change to positioning operation

Implementation

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

#INCLUDE "MotionParameters.bi"
#INCLUDE "MotionFunctions.bi"

:Init
DIM StepCounter
DIM DigInStatus

StepCounter = 0

:MainLoop
DO
    DigInStatus = GETOBJ DigitalInputLogicalState

    IF StepCounter = 0 THEN
        IF isEnabled() THEN
```

Example programs

```
            '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


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

7.2.2 Reference run with subsequent automatic change to positioning operation and start-stop function

This example builds on the example in chap. 7.2.1, p. 28.

Control is not activated automatically here. Control is activated from within the program if DigIn2 is active.

After the first switch-on procedure, the reference run is performed first followed by a change to normal operation. Control can also be deactivated again at any time via DigIn2. The reference run does, however, only occur after being powered on for the first time.

Prerequisites

- The motor was successfully commissioned and the control adapted to the application.
- The desired type of reference run was configured in object 0x6098.00.
- The analogue setpoint specifications are appropriately scaled via objects 0x2313 and selected as sources for the position set value (0x2331.04) or the torque set value (0x2331.02) via object 0x2331.

Sequence

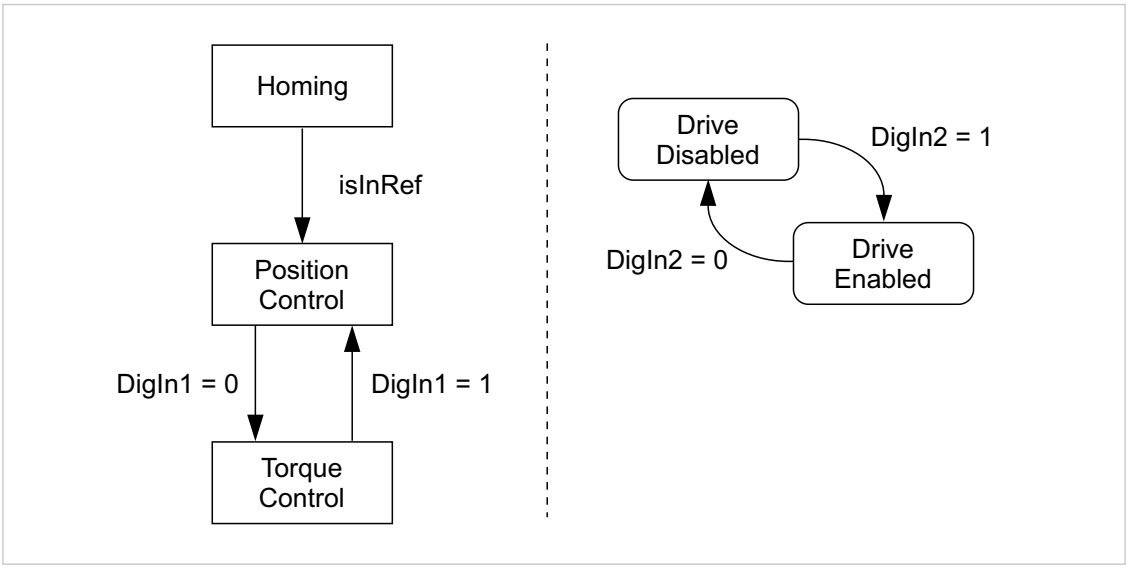


Fig. 3: Sequence for reference run with subsequent change to positioning operation and start-stop function

Implementation

The sequence for reference run with subsequent change to positioning operation (left side in Fig. 3) is taken over, including the sub-functions from the example in chap. 7.2.1, p. 28.

The check and reaction to DigIn2 is amended to the loop. Prerequisite for this is the implementation of the sequence as a sequence of steps without any blocking queries.

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

Example programs

```
#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

    'check DigIn2 for start/stop oft he 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
```

7.3 Event handling

The following program extract shows how the program can respond to the event **Temperature warning limit reached**.

```
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
```

FAULHABER

DR. FRITZ FAULHABER GMBH & CO. KG
Antriebssysteme

Faulhaberstraße 1
71101 Schönaich • Germany
Tel. +49(0)7031/638-0
Fax +49(0)7031/638-100
info@faulhaber.de
www.faulhaber.com

7000.05056, 4th edition, 28.02.2024